Oracle® Database Graph Developer's Guide for Property Graph





Oracle Database Graph Developer's Guide for Property Graph, 23.1

F75606-07

Copyright © 2016, 2024, Oracle and/or its affiliates.

Primary Author: Lavanya Jayapalan

Contributors: Prashant Kannan, Chuck Murray, Melliyal Annamalai, Korbinian Schmid, Vlad Ioan Haprian, Albert Godfrind, Oskar van Rest, Jorge Barba, Ana Estrada, Steve Serra, Ryota Yamanaka, Bill Beauregard, Hector Briseno, Hassan Chafi, Eugene Chong, Souripriya Das, Juan Garcia, Florian Gratzer, Zazhil Herena, Sungpack Hong, Roberto Infante, Hugo Labra, Gabriela Montiel-Moreno, Eduardo Pacheco, Joao Paiva, Matthew Perry, Diego Ramirez, Siva Ravada, Carlos Reyes, Jane Tao, Edgar Vazquez, Zhe (Alan) Wu

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software, software documentation, data (as defined in the Federal Acquisition Regulation), or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software," "commercial computer software documentation," or "limited rights data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle®, Java, MySQL and NetSuite are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

	Preface				
	Audience	XXV			
	Documentation Accessibility				
	Related Documents				
	Conventions	XXV			
	Changes in This Release for This Guide				
Par	t Getting Started with Oracle Property Graphs				
1	Introduction to Property Graphs				
	1.1 What Are Property Graphs?	1-1			
	1.2 About the Property Graph Feature of Oracle Database	1-2			
	1.3 Overview of Property Graph Architecture	1-3			
	1.3.1 Architecture Model for Running Graph Queries in the Database	1-3			
	1.3.2 Architecture Model for Running Graph Analytics	1-4			
	1.3.3 Developing Applications Using Graph Server Functionality as a Library	1-6			
	1.4 Learn About the Graph Server (PGX)	1-6			
	1.4.1 Overview of the Graph Server (PGX)	1-7			
	1.4.1.1 Design of the Graph Server (PGX)	1-7			
	1.4.1.2 Usage Modes of the Graph Server (PGX)	1-8			
	1.5 Security Best Practices with Graph Data	1-9			
	1.6 About Oracle Graph Server and Client Accessibility	1-12			
2	Using Oracle Graph with the Autonomous Database				
	2.1 Two-Tier Deployments of Oracle Graph with Autonomous Database	2-2			
	2.2 Three-Tier Deployments of Oracle Graph with Autonomous Database	2-3			



Part II SQL Property Graphs

3.1 Qui	ck Start for Working with SQL Property Graphs	3-
SQL D	DL Statements for Property Graphs	
4.1 Cre	ating a SQL Property Graph	4-
4.1.1	About Vertex and Edge Graph Element Tables	4-
4.1.2	About Vertex and Edge Table Keys	4-
4.1.3	About Labels and Properties	4-
4.1.4	Using Graph Options to Create SQL Property Graphs	4-
4.1.5	Granting System and Object Privileges for SQL Property Graphs	4-1
4.1.6	Retrieving SQL Creation DDL Using the DBMS_METADATA Package	4-1
4.1.7	Limitations of Creating a SQL Property Graph	4-1
4.2 Rev	alidating a SQL Property Graph	4-1
4.3 Dro	pping a SQL Property Graph	4-1
4.4 JSC	ON Support in SQL Property Graphs	4-1
	ut Graph Patterns	5
5.1.1 5.1.2	Graph Element Variables	5· 5·
5.1.2	Label Expressions Accessing Label Properties	5·
	able-Length Path Patterns	5-
	nplex Path Patterns	5-
	ex and Edge Identifiers	5-
	ning GRAPH_TABLE Queries at a Specific SCN	5-1
	ileges to Query a SQL Property Graph	5-1
	mples for SQL Property Graph Queries	5-1
5.7.1	Setting Up Sample Data in the Database	5-2
	ported Features and Limitations for Querying a SQL Property Graph	5-2
	ing SQL Property Graph Queries	5-2
	pe Compatibility Rules for Determining Property Types	5-2
,		
Execut	ing PGQL Queries Against SQL Property Graphs	



Part III Property Graph Views

About F	Property Graph Views	
7.1 Crea	ating Property Graph Views on Oracle Database Tables	7-1
7.1.1	Retrieving Metadata for Property Graph Views	7-4
7.2 Crea	ating a PG View By Importing a GraphSON file	7-8
7.2.1	Additional Information on the GraphImporter Parameters	7-11
7.2.2	Mapping GraphSON Types to Oracle Database Data Types	7-13
7.3 Usir	ng JSON to Store Vertex and Edge Properties	7-13
Loadin	g a PG View into the Graph Server (PGX)	
8.1 Loa	ding a PG View Using the readGraphByName API	8-1
8.1.1	Specifying Options for the readGraphByName API	8-2
8.1.2	Specifying the Schema Name for the readGraphByName API	8-4
8.2 Loa	ding a Graph Using a JSON Configuration File	8-5
8.2.1	Configuring PARALLEL Hint when Loading a Graph	8-7
8.3 Loa	ding a Graph by Defining a Graph Configuration Object	8-8
8.4 Loa	ding a Subgraph from Property Graph Views	8-10
8.4.1	PGQL Based Subgraph Loading	8-11
8.4.2	Prepared PGQL Queries	8-14
8.4.3	Providing Database Connection Credentials	8-15
8.4.4	Dynamically Expanding a Subgraph	8-16
Quick S	Starts for Using Property Graph Views	
9.1 Usir	ng Sample Data for Graph Analysis	9-1
9.1.1	Importing Data from CSV Files	9-1
9.2 Quid	ck Start: Working with Property Graph Views	9-3
9.3 Quid	ck Start: Using the Python Client as a Module	9-10
9.4 Ora	cle LiveLabs Workshops for Graphs	9-13
Getting	Started with the Client Tools	
10.1 Inte	eractive Graph Shell CLIs	10-1
10.1.1	Starting the OPG4J Shell	10-2
10.1.2	Starting the OPG4Py Shell	10-4
10.2 Us	ing Autonomous Database Graph Client	10-6



	10.2.1 Prerequisites for Using Autonomous Database Graph Client	10-13
	10.3 Using the Graph Visualization Web Client	10-15
	10.4 Using the Jupyter Notebook Interface	10-15
	10.5 Additional Client Tools for Querying PG Views	10-16
	10.5.1 Using Oracle SQLcl	10-17
	10.5.2 Using Oracle SQL Developer	10-19
	10.5.2.1 About Property Graph Support in SQL Developer	10-19
	10.5.2.2 Working with Property Graph Views	10-20
11	Property Graph Query Language (PGQL)	
	11.1 Creating a Property Graph Using PGQL	11-1
	11.2 Pattern Matching with PGQL	11-4
	11.3 Edge Patterns Have a Direction with PGQL	11-5
	11.4 Vertex and Edge Labels with PGQL	11-5
	11.5 Variable-Length Paths with PGQL	11-6
	11.6 Aggregation and Sorting with PGQL	11-6
	11.7 Executing PGQL Queries Against Property Graph Views	11-7
	11.7.1 Supported PGQL Features and Limitations for PG Views	11-8
	11.7.1.1 Additional Information on Supported PGQL Features with Examples	11-10
	11.7.2 Performance Considerations for PGQL Queries	11-15
	11.7.3 Using the Java and Python APIs to Run PGQL Queries	11-19
	11.7.3.1 Creating a Property Graph View	11-19
	11.7.3.2 Executing PGQL SELECT Queries	11-22
	11.7.3.3 Executing PGQL Queries Using JDBC Driver	11-31
	11.7.3.4 Executing PGQL Queries to Modify Property Graph Views	11-33
	11.7.3.5 Dropping A Property Graph View	11-36
Part	IV Installing Oracle Graph Server (PGX) and Client	
12	Oracle Graph Server and Client Installation	
	12.1 Before You Begin	12-1
	12.1.1 Verifying Database Compatibility	12-2
	12.1.2 Downloading Oracle Graph Server and Client	12-2
	12.1.3 Installing PL/SQL Packages in Oracle Database	12-3
	12.2 Oracle Graph Server Installation	12-5
	12.2.1 Using the RPM Installation	12-5
	12.2.1.1 Prerequisites for Installing Oracle Graph Server	12-5
	12.2.1.2 Installing Oracle Graph Server	12-6
	12.2.1.3 Uninstalling Oracle Graph Server	12-8



	12.2.1.4	Upgrading Oracle Graph Server	12-8
12	2.2.2 Dep	loying Oracle Graph Server to a Web Server	12-9
	12.2.2.1	Deploying to Apache Tomcat	12-9
	12.2.2.2	Deploying to Oracle WebLogic Server	12-10
12	2.2.3 Use	r Authentication and Authorization	12-11
	12.2.3.1	Privileges and Roles in Oracle Database	12-12
	12.2.3.2	Basic Steps for Using an Oracle Database for Authentication	12-12
	12.2.3.3	Prepare the Graph Server for Database Authentication	12-14
	12.2.3.4	Store the Database Password in a Keystore	12-17
	12.2.3.5	Adding Permissions to Publish the Graph	12-22
	12.2.3.6	Token Expiration	12-22
	12.2.3.7	Advanced Access Configuration	12-23
	12.2.3.8	Customizing Roles and Permissions	12-24
	12.2.3.9	Revoking Access to the Graph Server	12-27
	12.2.3.10	Examples of Custom Authorization Rules	12-28
	12.2.3.11	Kerberos Enabled Authentication for the Graph Server (PGX)	12-30
12.3	Oracle Gr	aph Client Installation	12-33
12	2.3.1 Gra _l	ph Clients	12-34
	12.3.1.1	Oracle Graph Java Client	12-34
	12.3.1.2	Oracle Graph Python Client	12-39
12	2.3.2 Gra _l	ph Visualization Web Client	12-42
	12.3.2.1	Running the Graph Visualization Application in Standalone Mode	12-43
	12.3.2.2	Deploying the Graph Visualization Application	12-45
	12.3.2.3	Configuring Advanced Options for PGQL Driver Selection	12-46
12.4	Setting Up	Transport Layer Security	12-48
12	2.4.1 Usir	ng a Self-Signed Server Keystore	12-49
	12.4.1.1	Generating a Self-Signed Server Keystore	12-49
	12.4.1.2	Configuring the Graph Server (PGX) When Using a Server Keystore	12-50
	12.4.1.3	Configuring a Client to Trust the Self-Signed Keystore	12-51
12	2.4.2 Usir	ng a Self-Signed Server Certificate	12-52
	12.4.2.1	Generating a Self-Signed Server Certificate	12-52
	12.4.2.2	Configuring the Graph Server (PGX)	12-53
	12.4.2.3	Configuring a Client to Trust the Self-Signed Certificate	12-54
Gett	ing Start	ted with the Graph Server (PGX)	
13.1	Starting th	e Graph Server (PGX)	13-1
13	3.1.1 Star	ting and Stopping the Graph Server (PGX) Using the Command Line	13-1
13	3.1.2 Con	figuring the Graph Server (PGX)	13-2
13.2	Connectin	g to the Graph Server (PGX)	13-7
13	3.2.1 Con	necting with the Graph Client CLIs	13-7



13

	13.2.3	3 Connecting with Python	13-14
Par	t V Usi	ing the Graph Server (PGX)	
14	Develo	ping Applications with Graph Analytics	
	14.1 Ab	out Vertex and Edge IDs	14-2
	14.2 Gr	aph Management in the Graph Server (PGX)	14-4
	14.2.1	Reading Graphs from Oracle Database into the Graph Server (PGX)	14-4
	1	4.2.1.1 Reading Entity Providers at the Same SCN	14-5
	1	4.2.1.2 Progress Reporting and Estimation for Graph Loading	14-8
	14	4.2.1.3 API for Loading Graphs into Memory	14-10
	1	4.2.1.4 Graph Configuration Options	14-11
	14	4.2.1.5 Data Loading Security Best Practices	14-18
	1	4.2.1.6 Data Format Support Matrix	14-18
	14	4.2.1.7 Immutability of Loaded Graphs	14-19
	14.2.2	Storing a Graph Snapshot on Disk	14-19
	14.2.3	B Publishing a Graph	14-20
	14.2.4	Deleting a Graph	14-26
	14.3 Ke	eeping the Graph in Oracle Database Synchronized with the Graph Server	14-28
	14.3.1	L Synchronizing a PG View Graph	14-30
	14.3.2	2 Synchronizing a Published Graph	14-34
	14.4 Or	otimizing Graphs for Read Versus Updates in the Graph Server (PGX)	14-40
	14.5 Ex	recuting Built-in Algorithms	14-41
	14.5.1	About Built-In Algorithms in the Graph Server (PGX)	14-42
	14.5.2	2 Running the Triangle Counting Algorithm	14-42
	14.5.3	Running the PageRank Algorithm	14-43
	14.6 Us	sing Custom PGX Graph Algorithms	14-44
	14.6.1	L Writing a Custom PGX Algorithm	14-44
	14	4.6.1.1 Collections	14-45
	1	4.6.1.2 Iteration	14-46
	1	4.6.1.3 Reductions	14-46
	14.6.2	Compiling and Running a Custom PGX Algorithm	14-47
	14.6.3	B Example Custom PGX Algorithm: PageRank	14-49
	14.7 Cr	eating Subgraphs	14-50
	14.7.1	L About Filter Expressions	14-51
	14.7.2	2 Using a Simple Filter to Create a Subgraph	14-52
	14.7.3	3 Using a Complex Filter to Create a Subgraph	14-52
	14.7.4	Using a Vertex Set to Create a Bipartite Subgraph	14-53

13.2.2 Connecting with Java

Starting and Stopping the PGX Engine



13-13

13-13

	14.8 Usin	g Automatic Delta Refresh to Handle Database Changes	14-55
	14.8.1	Configuring the Graph Server (PGX) for Auto-Refresh	14-55
	14.8.2	Configuring Basic Auto-Refresh	14-56
	14.8.3	Reading the Graph Using the Graph Server (PGX) or a Java Application	14-57
	14.8.4	Checking Out a Specific Snapshot of the Graph	14-57
	14.8.5	Advanced Auto-Refresh Configuration	14-58
	14.8.6	Special Considerations When Using Auto-Refresh	14-59
	14.9 User	r-Defined Functions (UDFs) in PGX	14-59
	14.10 Usi	ng the Graph Server (PGX) as a Library	14-63
L5	Using the	e Machine Learning Library (PgxML) for Graphs	
	15.1 Usin	g the DeepWalk Algorithm	15-1
	15.1.1	Loading a Graph	15-2
	15.1.2	Building a Minimal DeepWalk Model	15-3
	15.1.3	Building a Customized DeepWalk Model	15-4
	15.1.4	Training a DeepWalk Model	15-5
	15.1.5	Getting the Loss Value For a DeepWalk Model	15-6
	15.1.6	Computing Similar Vertices for a Given Vertex	15-7
	15.1.7	Computing Similar Vertices for a Vertex Batch	15-8
	15.1.8	Getting All Trained Vertex Vectors	15-9
	15.1.9	Storing a Trained DeepWalk Model	15-10
	15.1	1.9.1 Storing a Trained Model in Another Database	15-11
	15.1.10	Loading a Pre-Trained DeepWalk Model	15-12
	15.1	L.10.1 Loading a Pre-Trained Model From Another Database	15-13
	15.1.11	Destroying a DeepWalk Model	15-15
	15.2 Usin	g the Supervised GraphWise Algorithm (Vertex Embeddings and Classification)	15-15
	15.2.1	Loading a Graph	15-16
	15.2.2	Building a Minimal GraphWise Model	15-18
	15.2.3	Advanced Hyperparameter Customization	15-19
	15.2.4	Building a GraphWise Model Using Heterogeneous Graphs	15-22
	15.2.5	Classification Versus Regression Models on Supervised GraphWise Models	15-25
	15.2.6	Setting a Custom Loss Function and Batch Generator (for Anomaly Detection)	15-26
	15.2.7	Training a Supervised GraphWise Model	15-28
	15.2.8	Getting the Loss Value For a Supervised GraphWise Model	15-28
	15.2.9	Inferring the Vertex Labels for a Supervised GraphWise Model	15-29
	15.2.10	Evaluating the Supervised GraphWise Model Performance	15-30
	15.2.11	Inferring Embeddings for a Supervised GraphWise Model	15-31
	15.2	2.11.1 Inferring Embeddings for a Model in Another Database	15-32
	15.2.12	Storing a Trained Supervised GraphWise Model	15-33
	15.2.13	Loading a Pre-Trained Supervised GraphWise Model	15-34



	15.2.14	Destroying a Supervised GraphWise Model	15-35
	15.2.15	Explaining a Prediction of a Supervised GraphWise Model	15-36
15	.3 Usinç	g the Unsupervised GraphWise Algorithm	15-40
	15.3.1	Loading a Graph	15-41
	15.3.2	Building a Minimal Unsupervised GraphWise Model	15-42
	15.3.3	Advanced Hyperparameter Customization	15-43
	15.3.4	Building an Unsupervised GraphWise Model Using Heterogeneous Graphs	15-45
	15.3.5	Training an Unsupervised GraphWise Model	15-48
	15.3.6	Getting the Loss Value for an Unsupervised GraphWise Model	15-49
	15.3.7	Inferring Embeddings for an Unsupervised GraphWise Model	15-49
	15.3.8	Storing an Unsupervised GraphWise Model	15-51
	15.3.9	Loading a Pre-Trained Unsupervised GraphWise Model	15-52
	15.3.10	Destroying an Unsupervised GraphWise Model	15-53
	15.3.11	Explaining a Prediction for an Unsupervised GraphWise Model	15-53
15	.4 Usinç	the Supervised EdgeWise Algorithm (Edge Embeddings and Classification)	15-57
	15.4.1	Loading a Graph	15-58
	15.4.2	Building a Minimal Supervised EdgeWise Model	15-60
	15.4.3	Advanced Hyperparameter Customization	15-61
	15.4.4	Applying EdgeWise for Partitioned Graphs	15-63
	15.4.5	Classification Versus Regression on Supervised EdgeWise Models	15-66
	15.4.6	Setting a Custom Loss Function and Batch Generator (for Anomaly Detection)	15-67
	15.4.7	Setting the Edge Embedding Production Method	15-69
	15.4.8	Training the Supervised EdgeWise Model	15-70
	15.4.9	Getting the Loss Value for a Supervised EdgeWise Model	15-71
	15.4.10	Inferring Edge Labels for a Supervised EdgeWise Model	15-71
	15.4.11	Evaluating Model Performance	15-73
	15.4.12	Inferring Embeddings for a Supervised EdgeWise Model	15-75
	15.4.13	Storing a Supervised EdgeWise Model	15-76
	15.4.14	Loading a Pre-Trained Supervised EdgeWise Model	15-77
	15.4.15	Destroying a Supervised EdgeWise Model	15-78
	15.4.16	Example: Predicting Ratings on the Movielens Dataset	15-79
15	.5 Usinç	g the Pg2vec Algorithm	15-82
	15.5.1	Loading a Graph	15-83
	15.5.2	Building a Minimal Pg2vec Model	15-85
	15.5.3	Building a Customized Pg2vec Model	15-86
	15.5.4	Training a Pg2vec Model	15-87
	15.5.5	Getting the Loss Value For a Pg2vec Model	15-88
	15.5.6	Computing Similar Graphlets for a Given Graphlet	15-88
	15.5.7	Computing Similars for a Graphlet Batch	15-90
	15.5.8	Inferring a Graphlet Vector	15-91
	15.5.9	Inferring Vectors for a Graphlet Batch	15-92



15.5.10 Storing a	Trained Pg2vec Model	1
15.5.11 Loading a	a Pre-Trained Pg2vec Model	1
15.5.12 Destroyir	ng a Pg2vec Model	1
15.6 Model Repositor	ry and Model Stores	1
15.6.1 Database-	-Backed Model Repository	1
Executing PGQL	Queries Against the Graph Server (PGX)	
16.1 Getting Started	with PGQL	
16.2 Creating Proper	ty Graphs Using Options	
16.3 Supported PGQ	L Features and Limitations on the Graph Server (PGX)	
16.3.1 Support for	or Selecting All Properties	
16.3.2 Unnesting	of Variable-Length Path Queries	
16.3.3 Using INT	ERVAL Literals in PGQL Queries	1
16.3.4 Limitations	s on Quantifiers	1
16.3.5 Limitations	s on WHERE and COST Clauses in Quantified Patterns	2
16.4 Java APIs for Ex	xecuting CREATE PROPERTY GRAPH Statements	2
16.5 Python APIs for	Executing CREATE PROPERTY GRAPH Statements	:
16.6 Java APIs for Ex	xecuting SELECT Queries	:
16.6.1 Executing	SELECT Queries Against a Graph in the Graph Server (PGX)	
16.6.2 Executing	SELECT Queries Against a PGX Session	
16.6.3 Iterating T	hrough a Result Set	:
16.6.4 Printing a	Result Set	
16.7 Java APIs for Ex	xecuting UPDATE Queries	
16.7.1 Updatabili	ty of Graphs Through PGQL	:
16.7.2 Executing	UPDATE Queries Against a Graph in the Graph Server (PGX)	
16.7.3 Executing	UPDATE Queries Against a PGX Session	
16.7.4 Altering th	e Underlying Schema of a Graph	
16.8 PGQL Queries v	with Partitioned IDs	:
16.9 Security Tools fo	or Executing PGQL Queries	
16.9.1 Using Bind	d Variables	:
16.9.2 Using Ider	ntifiers in a Safe Manner	
16.10 Best Practices	for Tuning PGQL Queries	
16.10.1 Memory	Allocation	
16.10.2 Parallelis	sm	
16.10.3 Query PI	an Explaining	,
REST Endpoints	for the Graph Server	
17.1 Login	<u> </u>	
17.2 List Graphs		



	17.3	Run a	a PGQL Query	17-3
	17.4	Get L	Jser	17-6
	17.5	Logo	ut	17-6
	17.6	Asyno	chronous REST Endpoints	17-6
	1	7.6.1	Run a PGQL Query Asynchronously	17-7
	1	7.6.2	Check a Query Completion	17-7
	1	7.6.3	Cancel a Query Execution	17-7
	1	7.6.4	Retrieve a Query Result	17-8
Part	VI	Grap	oh Visualization Application	
18	Abo	ut the	e Graph Visualization Application	
	18.1		does the Graph Visualization Application Work	18-1
	18.2		eros Enabled Authentication for the Graph Visualization Application	18-1
		8.2.1	Prerequisite Requirements for Kerberos Authentication	18-2
	18	8.2.2	Preparing the Graph Visualization Application for Kerberos Authentication	18-2
	18.3	Embe	edding the Graph Visualization Library in a Web Application	18-5
19	Usir	ng the	e Graph Visualization Application	
	19.1	Visua	alizing PGQL Queries on Graphs Loaded Into the Graph Server (PGX)	19-1
	19.2	Visua	alizing PGQL and SQL Graph Queries on Graphs in the Database	19-2
	19	9.2.1	Visualizing PGQL Queries on PG Views	19-3
	19	9.2.2	Visualizing Graph Queries on SQL Property Graphs	19-5
	19.3	Grapl	h Visualization Modes	19-6
	19.4	Grapl	h Visualization Settings	19-6
	19.5	Using	the Geographical Layout	19-9
	19.6	Using	g Live Search	19-11
	19.7	Using	URL Parameters to Control the Graph Visualization Application	19-12
Part	VII	Gra	ph Server (PGX) Advanced User Guide	
20	Con	figuri	ng the Graph Server (PGX) and the Graph Client	
	20.1	Confi	guration Parameters for the Graph Server (PGX) Engine	20-1
	20	0.1.1	Configuration of the Graph Server (PGX) Run-Time Parameters	20-12
	20	0.1.2	Passing the Configuration File to the Graph Server (PGX)	20-15
	20	0.1.3	Memory Consumption by the Graph Server (PGX)	20-17
		20.1	.3.1 Memory Management	20-17



	20.2 Configuration Parameters for Connecting to the Graph Server (PGX)20.3 Configuration Parameters for the Graph Client	20-19 20-19
21	Deploying Oracle Graph Server Behind a Load Balancer	
	21.1 Using HAProxy for PGX Load Balancing and High Availability	21-1
	21.2 Deploying Graph Server (PGX) Using OCI Load Balancer	21-3
	21.3 Health Check in the Load Balancer	21-6
22	Namespaces and Sharing	
	22.1 Defining Graph Names	22-1
	22.2 Retrieving Graphs by Name	22-1
	22.3 Checking Used Names	22-2
	22.4 Property Name Resolution and Graph Mutations	22-2
23	PGX Programming Guides	
	23.1 Design of the Graph Server (PGX) API	23-3
	23.2 Data Types and Collections in the Graph Server (PGX)	23-4
	23.2.1 Using Collections and Maps	23-7
	23.2.1.1 Collection Data Types	23-7
	23.2.1.2 Map Data Types	23-12
	23.2.2 Using Datetime Data Types	23-17
	23.2.2.1 Loading Datetime Data	23-18
	23.2.2.2 Specifying Custom Datetime Formats	23-20
	23.2.2.3 APIs for Accessing Datetime Data	23-21
	23.2.2.4 Querying Datetime Data Using PGQL	23-22
	23.2.2.5 Accessing Datetimes from PGQL Result Sets	23-24
	23.3 Handling Asynchronous Requests in Graph Server (PGX)	23-26
	23.3.1 Blocking Operation	23-26
	23.3.2 Chaining Operation	23-27
	23.3.3 Cancelling Operation	23-28
	23.3.4 Handling Concurrent Asynchronus Operations	23-28
	23.4 Graph Client Sessions	23-29
	23.5 Graph Mutation and Subgraphs	23-31
	23.5.1 Altering Graphs	23-31
	23.5.1.1 Loading Or Removing Additional Vertex or Edge Providers	23-32
	23.5.2 Simplifying and Copying Graphs	23-40
	23.5.3 Transposing Graphs	23-42
	23.5.4 Undirecting Graphs	23-43
	23.5.5 Advanced Multi-Edge Handling	23-44



	23.5	5.5.1	Picking	23-44
	23.5	5.5.2	Merging	23-45
	23.5	5.5.3	StrategyBuilder in General	23-46
	23.5.6	Creat	ting a Subgraph	23-47
	23.5.7	Creat	ting a Bipartite Subgraph	23-47
	23.5.8	Creat	ting a Sparsified Subgraph	23-48
23.	6 Grap	h Build	der and Graph Change Set	23-49
	23.6.1	Build	ing Graphs Using GraphBuilder Interface	23-49
	23.6	5.1.1	Creating a Simple Graph	23-49
	23.6	5.1.2	Adding a Vertex Property	23-51
	23.6	5.1.3	Using Strings as Vertex Identifiers	23-53
	23.6	5.1.4	Referencing a Vertex for Creating Edges	23-54
	23.6	5.1.5	Adding an Edge Property and a Label	23-56
	23.6	5.1.6	Using Graph Builder with Implicit IDs	23-57
	23.6.2	Modif	fying Loaded Graphs Using ChangeSet	23-59
	23.6	5.2.1	Modifying Vertices	23-59
	23.6	5.2.2	Adding Edges	23-60
	23.6	5.2.3	GraphChangeSet with Partitioned IDs	23-61
	23.6	5.2.4	Error Handling when Using a ChangeSet	23-62
23.	7 Man	aging 7	Transient Data	23-64
	23.7.1	Mana	aging Transient Properties	23-64
	23.7.2	Mana	aging Collections and Scalars	23-66
23.	8 Grap	oh Vers	sioning	23-68
	23.8.1	Confi	iguring the Snapshots Source	23-68
	23.8.2	Creat	ting a Snapshot via Refreshing	23-69
	23.8.3	Creat	ting a Snapshot via ChangeSet	23-71
	23.8.4	Chec	king Out the Latest Snapshots of a Graph	23-73
	23.8.5	Chec	king Out Different Snapshots of a Graph	23-74
	23.8.6	Direc	tly Loading a Specific Snapshot of a Graph	23-75
23.	9 Labe	els and	Properties	23-77
	23.9.1	Settir	ng and Getting Property Values	23-77
	23.9.2	Gettir	ng Label Values	23-79
23.	10 Filt	er Expı	ressions	23-79
	23.10.1	Syn	tax	23-80
	23.10.2	Тур	e System	23-85
	23.10.3	Path	n Finding Filters	23-85
	23.10.4	Sub	graph Filters	23-85
	23.10.5	Ope	erations on Filter Expressions	23-86
	23.1	10.5.1	Defining Filter Expressions	23-86
	23.2	L0.5.2	Defining Result Set Filters	23-87
	23.1	L0.5.3	Creating a Subgraph from PGQL Result Set	23-89



23.10	U.5.4 Defining Collection Filters	23-90
23.10	0.5.5 Creating a Subgraph from Collection Filters	23-91
23.10	0.5.6 Combining Filter Expressions	23-92
23.10	0.5.7 Creating a Subgraph Using Filter Expressions with Partitioned IDs	23-94
23.11 Adva	anced Task Scheduling Using Execution Environments	23-95
23.11.1	Enterprise Scheduler Configuration Guide	23-95
23.11.2	Enabling Enterprise Scheduler Features	23-98
23.11.3	Retrieving and Inspecting the Execution Environment	23-98
23.11.4	Modifying and Submitting Tasks Under an Updated Environment	23-100
23.11.5	Using Lambda Syntax	23-101
23.12 Adm	nin API	23-102
23.12.1	Get a Server Instance	23-102
23.12.2	Get Inspection Data	23-102
23.12.3	Get Active Sessions	23-104
23.12.4	Get Cached Graphs	23-106
23.12.5	Get Published Graphs	23-107
23.12.6	Get Currently Loading Graphs	23-107
23.12.7	Get Tasks	23-108
23.12.8	Get Available Memories	23-108
23.13 Pgx	Frames Tabular Data-Structure	23-108
23.13.1	Converting PgqlResultSet to a PgxFrame	23-109
23.13.2	Storing a PgxFrame to a Database	23-111
23.13.3	Storing a PgxFrame to a CSV File	23-113
23.13.4	Union of PGX Frames	23-114
23.13.5	Joining PGX Frames	23-114
23.13.6	Printing the Content of a PgxFrame	23-115
23.13.7	Destroying a PgxFrame	23-116
23.13.8	Loading and Storing Vector Properties	23-117
23.13.9	Flattening Vector Properties	23-119
23.13.10	PgxFrame Helpers	23-119
23.13.11	Converting a PgxFrame to PgqlResultSet	23-123
23.13.12	PgxFrame to Pandas DataFrame Conversions	23-123
23.13.13	Loading a PgxFrame from a Database	23-124
23.13.14	Loading a PgxFrame from a CSV File	23-127
23.13.15	Loading a PgxFrame from Client-Side Data	23-128
23.13.16	Creating a Graph from Multiple PgxFrame Objects	23-133
Working	with Files Using the Graph Server (PGX)	
24.1 Loadi	ng Graph Data from Files	24-1
24.1.1	Graph Configuration for Loading from File	24-3



24

	24.1.2	Specifying the File Path	24-7
	24.1.3	Supported File Access Protocols	24-7
	24.1.4	Plain Text Formats	24-8
	24.	.1.4.1 Comma-Separated Values (CSV)	24-10
	24.	.1.4.2 Adjacency List (ADJ_LIST)	24-13
	24.	.1.4.3 Edge List (EDGE_LIST)	24-13
	24.	.1.4.4 Two Tables (TWO_TABLES)	24-15
	24.1.5	XML File Formats	24-16
	24.1.6	Binary File Formats	24-17
	24.2 Loa	ading Graph Data in Parallel from Multiple Files	24-22
	24.3 Exp	porting Graphs Into a File	24-24
	24.3.1	Exporting a Graph to Disk	24-26
	24.4 Exp	porting a Graph into Multiple Files	24-28
25	Log Mai	nagement in the Graph Server (PGX)	
	25.1 Cor	nfiguring Logback Logging	25-1
Part A		Supplementary Information for Property Graph Suppor	<u>t</u>
-	A.1 Prop	perty Graph Schema Objects for Oracle Database	A-2
	A.1.1	Property Graph Tables (Detailed Information)	A-3
	A.1.2	Default Indexes on Vertex (VT\$) and Edge (GE\$) Tables	A-6
	A.1.3	Flexibility in the Property Graph Schema	A-6
	A.2 Data	a Access Layer	A-7
	A.3 Getti	ing Started with Property Graphs	A-7
	A.3.1	Required Privileges for Database Users	A-8
	A.4 Usin	g Java APIs for Property Graph Data	A-8
	A.4.1	Overview of the Java APIs	A-8
	A.4	4.1.1 Oracle Graph Property Graph Java APIs	A-9
	A.4	4.1.2 Oracle Database Property Graph Java APIs	A-9
	A.4.2	Parallel Retrieval of Graph Data	A-9
	A.4.3	Using an Element Filter Callback for Subgraph Extraction	A-11
	A.4.4	Using Optimization Flags on Reads over Property Graph Data	A-14
	A.4.5	Adding and Removing Attributes of a Property Graph Subgraph	A-16
	A.4.6	Getting Property Graph Metadata	A-21
	A.4.7	Merging New Data into an Existing Property Graph	A-22
	A.4.8	Opening and Closing a Property Graph Instance	A-24
	A.4.9	Creating Vertices	A-26



A.4.10	Crea	iting Edges	A-26
A.4.11	Dele	ting Vertices and Edges	A-27
A.4.12	Rea	ding a Graph from a Database into an Embedded Graph Server (PGX)	A-27
A.4.13	Spec	cifying Labels for Vertices	A-28
A.4.14	Build	ling an In-Memory Graph	A-28
A.4.15	Drop	pping a Property Graph	A-30
A.4.16	Exec	cuting PGQL Queries	A-30
Acce	ss Cor	trol for Property Graph Data (Graph-Level and OLS)	A-30
A.5.1	Apply	ing Oracle Label Security (OLS) on Property Graph Data	A-31
SQL-	Based	Property Graph Query and Analytics	A-36
A.6.1	Simpl	e Property Graph Queries	A-37
A.6.2	Text C	Queries on Property Graphs	A-40
A.6.3	Navig	ation and Graph Pattern Matching	A-45
A.6.4	Navig	ation Options: CONNECT BY and Parallel Recursion	A-50
A.6.5	Pivot		A-53
A.6.6	SQL-I	Based Property Graph Analytics	A-54
A.6	.6.1	Shortest Path Examples	A-55
A.6	.6.2	Collaborative Filtering Overview and Examples	A-58
' Crea	ting Pr	operty Graph Views on an RDF Graph	A-64
-		Interactively Analyze Graph Data Stored in Property Graph Schema	
-			A-67
A.8.1	-		A-67
A.8	.1.1	Create and Query a Graph in the Database	A-68
A.8	.1.2	Load the Graph into Memory and Run Graph Analytics	A-72
A.8.2	Quick	Start: Create, Query, and Analyze a Graph in Graph Server (PGX)	A-75
) Work	ing wit	h Property Graph Objects in SQL Developer	A-80
.0 Exe	cuting	PGQL Queries Against Property Graph Schema Tables	A-83
A.10.1	PGÇ	L Features Supported in Property Graph Schema	A-85
A.1	0.1.1	Temporal Types	A-87
A.1	0.1.2	Type Casting	A-87
A.1	0.1.3	CONTAINS Built-in Function	A-88
A.10.2	Crea	ating Property Graphs through CREATE PROPERTY GRAPH Statements	A-89
A.10.3	Drop	ping Property Graphs through DROP PROPERTY GRAPH Statements	A-95
A.10.4	Usin	g the oracle.pg.rdbms.pgql Java Package to Execute PGQL Queries	A-96
A.1	0.4.1	Basic Query Execution	A-99
A.1	0.4.2	Security Techniques for PGQL Queries	A-108
A.1	0.4.3	Using a Text Index with PGQL Queries	A-114
A.1	0.4.4	Obtaining the SQL Translation for a PGQL Query	A-117
A.1	0.4.5	Additional Options for PGQL Translation and Execution	A-125
A.1	0.4.6	Querying Another User's Property Graph	A-143
A.1	0.4.7	Using Query Optimizer Hints with PGQL	A-145
	A.4.11 A.4.12 A.4.13 A.4.14 A.4.15 A.4.16 A.4.16 A.6.1 A.6.2 A.6.3 A.6.4 A.6.5 A.6.6 A.6.6 A.6.6 A.6.7 Crea B. Quick Object A.8.1 A.8 A.8.2 A.9 A.10.1 A.1 A.1 A.1 A.1 A.1 A.1 A.1 A.1 A.1 A	A.4.11 Dele A.4.12 Read A.4.13 Special A.4.14 Build A.4.15 Drop A.4.16 Execis A.5.1 Apply S SQL-Based A.6.1 Simpli A.6.2 Text C A.6.3 Navig A.6.4 Navig A.6.5 Pivot A.6.6 SQL-F A.6.6.1 A.6.6.2 Creating Property Cobjects A.8.1 Quick Serve A.8.1.1 A.8.1.2 A.8.2 Quick Company Serve A.8.1.1 A.8.1.2 A.8.1 Quick Company Serve	A.4.11 Deleting Vertices and Edges A.4.12 Reading a Graph from a Database into an Embedded Graph Server (PGX) A.4.13 Specifying Labels for Vertices A.4.14 Building an In-Memory Graph A.4.15 Dropping a Property Graph A.4.16 Executing PGQL Queries A.4.17 Access Control for Property Graph Data (Graph-Level and OLS) A.5.1 Applying Oracle Label Security (OLS) on Property Graph Data SQL-Based Property Graph Query and Analytics A.6.1 Simple Property Graph Query and Analytics A.6.2 Text Queries on Property Graphs A.6.3 Navigation and Graph Pattern Matching A.6.4 Navigation Options: CONNECT BY and Parallel Recursion A.6.5 Pivot A.6.6 SQL-Based Property Graph Analytics A.6.6.1 Shortest Path Examples A.6.6.2 Collaborative Filtering Overview and Examples Creating Property Graph Views on an RDF Graph Quick Start: Interactively Analyze Graph Data Stored in Property Graph Schema Objects A.8.1 Quick Start: Create and Query a Graph in the Database, Load into Graph Server (PGX) for Analytics A.8.1.1 Create and Query a Graph in the Database A.8.1.2 Load the Graph into Memory and Run Graph Analytics A.8.1.1 Create, Query, and Analyze a Graph in Graph Server (PGX) Working with Property Graph Objects in SQL Developer D. Executing PGQL Queries Against Property Graph Schema A.10.1.1 Temporal Types A.10.1.2 Type Casting A.10.1.3 CONTAINS Built-in Function A.10.2 Creating Property Graphs through CREATE PROPERTY GRAPH Statements A.10.4 Using the oracle.pg.rdbms.pgql Java Package to Execute PGQL Queries A.10.4.1 Basic Query Execution A.10.4.2 Security Techniques for PGQL Queries A.10.4.3 Using a Text Index with PGQL Queries A.10.4.4 Obtaining the SQL Translation for a PGQL Query A.10.4.5 Additional Options for PGQL Translation and Execution A.10.4.6 Querying Another User's Property Graph



	A.10).4.8	Statements	A-148
	A.10.5	Hein	g the Python Client to Execute PGQL Queries	A-140 A-161
			Creating a Property Graph Using the Python Client	A-161 A-161
).5.2	Dropping a Property Graph Using the Python Client	A-161 A-162
			Basic Query Execution	A-162
).5.4	Iterating a Query Result Set	A-162
			ormance Considerations for PGQL Queries	A-103 A-167
A.:			S Package Subprograms	A-167 A-168
Λ		_	APIS.ANALYZE PG	A-169
	A.11.1		APIS.CF	A-109 A-171
	A.11.2 A.11.3		APIS.CF_CLEANUP	A-171 A-174
	A.11.3		APIS.CF PREP	A-174 A-176
	A.11.4 A.11.5		APIS.CLEAR PG	A-170 A-177
	A.11.5 A.11.6		APIS.CLEAR_PG_INDICES	A-177 A-178
	A.11.0 A.11.7		APIS.CLONE GRAPH	A-178 A-178
	A.11.7 A.11.8		APIS.COUNT TRIANGLE	A-176 A-179
	A.11.0 A.11.9		APIS.COUNT_TRIANGLE APIS.COUNT_TRIANGLE_CLEANUP	A-179 A-180
	A.11.10		G_APIS.COUNT_TRIANGLE_CLEANOP	A-180 A-181
	A.11.10 A.11.11		G_APIS.COUNT_TRIANGLE_FREF G_APIS.COUNT_TRIANGLE_RENUM	A-181 A-183
	A.11.11 A.11.12		G_APIS.CREATE_EDGES_TEXT_IDX	A-183 A-184
	A.11.12 A.11.13		G_AFIS.CREATE_EDGES_TEXT_IDX G_APIS.CREATE_PG	A-184 A-185
	A.11.13 A.11.14		G_AFIS.CREATE_FG G_APIS.CREATE_PG_SNAPSHOT_TAB	A-185 A-186
	A.11.14 A.11.15		G_APIS.CREATE_PG_SNAFSHOT_TAB	A-188
	A.11.15 A.11.16		G_APIS.CREATE_FG_TEXTIDA_TAB G_APIS.CREATE_STAT_TABLE	A-189
	A.11.10 A.11.17		G_APIS.CREATE_STAT_TABLE G_APIS.CREATE_SUB_GRAPH	A-109 A-190
	A.11.17 A.11.18		G_APIS.CREATE_SUB_GRAPH G_APIS.CREATE_VERTICES_TEXT_IDX	A-190 A-191
	A.11.19		G_APIS.DROP_EDGES_TEXT_IDX	A-191 A-193
	A.11.19 A.11.20		G_AFIS.DROP_EDGES_TEXT_IDX G_APIS.DROP_PG	A-193 A-193
	A.11.20 A.11.21		G_AFIS.DROP_FG G APIS.DROP PG VIEW	A-193 A-194
	A.11.21		G_APIS.DROP_FG_VIEW G_APIS.DROP_VERTICES_TEXT_IDX	A-194 A-194
	A.11.22 A.11.23		G_AFIS.DROP_VERTICES_TEXT_IDX G_APIS.ESTIMATE_TRIANGLE_RENUM	A-194 A-195
	A.11.24		G_APIS.ESTIMATE_TRIANGEE_RENOW G_APIS.EXP_EDGE_TAB_STATS	A-193 A-197
	A.11.25		G_APIS.EXP_VERTEX_TAB_STATS	A-197 A-198
	A.11.26		G_APIS.FIND_CC_MAPPING_BASED	A-190 A-199
	A.11.27		G_APIS.FIND_CC_MAPFING_BASED G_APIS.FIND_CLUSTERS_CLEANUP	A-199 A-200
	A.11.27 A.11.28		G_APIS.FIND_CLUSTERS_CLEANOP G_APIS.FIND_CLUSTERS_PREP	A-200 A-201
	A.11.20 A.11.29		G_APIS.FIND_CLOSTERS_PREP G_APIS.FIND_SP	A-201 A-203
	A.11.29 A.11.30		G_APIS.FIND_SP_CLEANUP	A-203 A-204
	A.11.30 A.11.31		G_APIS.FIND_SP_CLEANUP G_APIS.FIND_SP_CLEANUP	A-204 A-205
			G_APIS.FIND_SP_PREP G_APIS.GET_BUILD_ID	A-205 A-206



A.1	11.33	OPG_APIS.GET_GEOMETRY_FROM_V_COL	A-206
A.1	11.34	OPG_APIS.GET_GEOMETRY_FROM_V_T_COLS	A-207
A.1	11.35	OPG_APIS.GET_LATLONG_FROM_V_COL	A-209
A.1	11.36	OPG_APIS.GET_LATLONG_FROM_V_T_COLS	A-210
A.1	11.37	OPG_APIS.GET_LONG_LAT_GEOMETRY	A-211
A.1	11.38	OPG_APIS.GET_LATLONG_FROM_V_COL	A-211
A.1	11.39	OPG_APIS.GET_LONGLAT_FROM_V_T_COLS	A-212
A.1	11.40	OPG_APIS.GET_OPG_VERSION	A-213
A.1	11.41	OPG_APIS.GET_SCN	A-214
A.1	11.42	OPG_APIS.GET_VERSION	A-215
A.1	11.43	OPG_APIS.GET_WKTGEOMETRY_FROM_V_COL	A-215
A.1	11.44	OPG_APIS.GET_WKTGEOMETRY_FROM_V_T_COLS	A-216
A.1	11.45	OPG_APIS.GRANT_ACCESS	A-217
A.1	11.46	OPG_APIS.IMP_EDGE_TAB_STATS	A-218
A.1	11.47	OPG_APIS.IMP_VERTEX_TAB_STATS	A-220
A.1	11.48	OPG_APIS.PR	A-221
A.1	11.49	OPG_APIS.PR_CLEANUP	A-223
A.1	11.50	OPG_APIS.PR_PREP	A-224
A.1	11.51	OPG_APIS.PREPARE_TEXT_INDEX	A-225
A.1	11.52	OPG_APIS.RENAME_PG	A-226
A.1	11.53	OPG_APIS.SPARSIFY_GRAPH	A-226
A.1	11.54	OPG_APIS.SPARSIFY_GRAPH_CLEANUP	A-228
A.1	11.55	OPG_APIS.SPARSIFY_GRAPH_PREP	A-229
A.12	OPG_	GRAPHOP Package Subprograms	A-230
A.1	12.1	OPG_GRAPHOP.POPULATE_SKELETON_TAB	A-231
Марр	ping	Graph Server Roles to Default Privileges	
Disal	bling	Transport Layer Security (TLS) in Graph Server	
Migra	ating	Property Graph Applications from Before Release 21c	
Upgr	adin	g From Graph Server and Client 20.4.x to 21.x	
Third	l-Par	ty License Information for Oracle Graph Server and Client	





List of Figures

1-1	Simple Property Graph Example	1-2
1-2	Property Graph Architecture for Running Graph Queries	1-4
1-3	Property Graph Architecture for Running Graph Analytics	1-5
1-4	Graph Server (PGX) Design	1-7
1-5	Remote Server Mode	1-8
1-6	PGX as a Library	1-9
1-7	Enabling Accessibility in the Graph Visualization Application	1-11
3-1	Using SQL Developer to Create a SQL Property Graph	3-1
3-2	Visualizing GRAPH_TABLE Query	3-3
6-1	PGQL on SQL Property Graphs in Oracle Database	6-1
7-1	PROPERTY_GRAPH_METADATA Graph Design	7-5
7-2	Financial Transactions Graph	7-14
8-1	Subgraph Visualization	8-12
8-2	Expanding a Subgraph	8-18
10-1	Creating a PG View in Jupyter Notebook	10-16
10-2	Running Graph Algorithms in Jupyter Notebook	10-16
10-3	Property Graph Support in SQL Developer	10-20
10-4	Create a Property Graph View	10-21
10-5	Running a PGQL INSERT Query	10-22
10-6	Running a PGQL SELECT Query	10-22
10-7	Dropping a Property Graph View	10-23
11-1	PGQL on Property Graph Views in Oracle Database	11-7
12-1	Graph Visualization Login	12-44
12-2	PGQL on Graph Server (PGX)	12-47
12-3	PGQL on Database	12-48
14-1	Edges Matching src.prop == 10	14-51
14-2	Graph Created by the Simple Filter	14-51
14-3	Edges Matching the outDegree Filter	14-53
14-4	Graph Created by the outDegree Filter	14-53
15-1	Pg2vec - Visualization of Two Similar Graphlets	15-90
16-1	Visualizing Unnesting of Variable-Length Path Queries	16-10
19-1	Query Visualization	19-2
19-2	Creating a PG View	19-3
19-3	Updating an Edge in a PG View	19-4
19-4	Deleting an Edge in a PG View	19-4



19-5	Querying a PG View	19-4
19-6	Dropping a PG View	19-4
19-7	GRAPH_TABLE Query on SQL Property Graph	19-5
19-8	Graph Visualization Settings Window	19-7
19-9	Highlights Options for Vertices	19-8
19-10	Geographical Layout	19-9
19-11	Setting Geographical Layout	19-10
19-12	Selecting the Coordinates for the Geographical layout	19-11
21-1	Configuring Load Balancer Details	21-4
21-2	Adding Backends to Load Balancer	21-4
21-3	Configuring a Listener for the Load Balancer	21-5
21-4	Enabling Session Persistence	21-6
23-1	Picking Strategy	23-45
23-2	Merging Strategy	23-46
A-1	Phones Graph for Collaborative Filtering	A-59
A-2	Creating a Property Graph Object	A-81
A-3	Updating a Property Graph Object	A-81
A-4	Running a PGQL SELECT Query	A-82
A-5	Dropping a Property Graph Object	A-83
A-6	PGQL on Property Graph Schema Tables in Oracle Database (RDBMS)	A-84



List of Tables

1-1	Graph Size Estimator	1-5
4-1	System Privileges for SQL Property Graph Objects	4-11
4-2	Object Privileges for SQL Property Graphs	4-11
5-1	Arrow Tokens for Edge Patterns	5-2
5-2	Supported Vertex and Edge Label Expressions	5-4
5-3	Quantifier Support for Variable-Length Graph Patterns	5-8
6-1	Supported PGQL Functionalities and Limitations for SQL Property Graphs	6-3
7-1	Metadata Tables for PG Views	7-2
7-2	Database Connection Parameters	7-11
7-3	GraphImporter Configuration Parameters	7-11
7-4	SQL Storage Parameters	7-12
7-5	PGQL Supported Parameters	7-12
7-6	Mapping GraphSON Types to Oracle Database Types	7-13
8-1	Parameters for the readGraphByName method	8-1
8-2	PARALLEL_HINT_DEGREE values	8-7
11-1	CREATE PROPERTY GRAPH Statement Support	11-4
11-2	Supported PGQL Functionalities and Limitations for PG Views	11-8
11-3	Supported Quantifiers in PGQL SELECT Queries	11-11
12-1	Workflow for Installing Oracle Graph Server and Client	12-1
12-2	Components in the Oracle Graph Server and Client Deployment	12-2
12-3	Privileges and Roles in Oracle Database	12-12
12-4	Advanced Access Configuration Options	12-23
12-5	API for Checking Graph Permissions	12-24
12-6	Allowed Permissions	12-28
13-1	Configuration Parameters for the Graph Server (PGX)	13-2
14-1	Valid values for "as_of" Key in Graph Configuration	14-5
14-2	Example Scenario Using "as_of"	14-7
14-3	Asynchronous Graph Loading APIs	14-8
14-4	Graph Config JSON Fields	14-11
14-5	Provider Configuration JSON file Options	14-14
14-6	Property Configuration	14-15
14-7	Loading Configuration	14-17
14-8	Error Handling Configuration	14-17
14-9	Data Format Support Matrix	14-19
14-10	Overview of Built-In Algorithms	14-41



14-11	Fields for Each UDF	14-63
16-1	Graph Optimization Options	16-3
16-2	Supported PGQL Functionalities and Limitations on the Graph Server (PGX)	16-5
16-3	Valid values for fields in INTERVAL values	16-12
17-1	Parameters	17-1
17-2	Query Parameters	17-3
18-1	Location of WEB-INF/web.xml file	18-3
19-1	Available URL Parameters	19-12
20-1	Configuration Parameters for the Graph Server (PGX) Engine	20-1
20-2	Graph Server (PGX) Run-Time Parameters	20-12
20-3	Configuration Parameters for the Graph Client	20-20
23-1	PGX API Interface	23-1
23-2	Overview of Data types	23-5
23-3	Overview of Datetime Data Types in PGX	23-18
23-4	Default Property Values	23-52
23-5	Default Temporal Formats	23-81
23-6	Session Information Options	23-105
23-7	Graph Information	23-106
23-8	Mapping between In-Place and Out-Place Operations	23-108
24-1	Loading from File - Graph Configuration Options	24-3
24-2	CSV Specific Options	24-5
24-3	Type Encoding	24-17
24-4	File Layout	24-17
24-5	Integer Vertex Keys	24-19
24-6	Long Vertex Keys	24-19
24-7	String Vertex Keys	24-19
24-8	String Key Element Layout	24-19
24-9	Primitive Type Layout	24-19
24-10	Vector Property Layout	24-20
24-11	String Type Layout	24-20
24-12	String Dictionary Layout	24-20
24-13	String Dictionary Element Layout	24-20
24-14	Vertex Labels Layout	24-21
24-15	Shared Pools Layout	24-21
24-16	Type == Enum	24-21
24-17	Type == Prefix	24-22
24-18	String Table for Shared Pools	24-22



24-19	Property Names Layout	24-22
24-20	Files CompressionScheme	24-25
24-21	Graph Configuration when Exporting Graph into Multiple Files	24-25
A-1	Supported PGQL Features and Limitations for PG Schema Graphs	A-85
A-2	Type Casting Support in PGQL (From and To Types)	A-88
A-3	PGQL Translation and Execution Options	A-125
A-4	PGQL Statement Modification Options	A-157
B-1	Mapping Graph Server Roles to Default Privileges	B-1



Preface

This document provides conceptual and usage information about Oracle Database support for working with property graph data.

- Audience
- Documentation Accessibility
- Related Documents
- Conventions

Audience

This document is intended for database and application developers in an Oracle Database environment.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Documents

For more information, see the following documents:

- Oracle Spatial Developer's Guide
- Oracle Database Graph Developer's Guide for RDF Graph
- Oracle Spatial GeoRaster Developer's Guide
- Oracle Spatial Topology and Network Data Model Developer's Guide
- Oracle Big Data Spatial and Graph User's Guide and Reference

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
monospace	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.



Changes in This Release for This Guide

The following changes apply to property graph support that is shipped with Oracle Graph Server and Client.

Oracle Graph Server and Client is required for using the property graph feature of Oracle Database (see Oracle Graph Server and Client Installation), and is released four times a year.

New Features

Significant New Features in Oracle Graph Server and Client 23.1 That Work With Oracle Database 23ai

Oracle Graph Server and Client Release 23.1 works with the following property graph features of Oracle Database 23ai:

- Added support for creating property graph SQL-schema object in Oracle Database.
 - See Introduction to SQL Property Graphs for more information.
- Added support for running graph queries on SQL property graphs.
 See SQL GRAPH_TABLE Queries for more information.
- Added support for running PGQL SELECT queries against SQL property graphs.
 See Executing PGQL Queries Against SQL Property Graphs for more information.
- Added support for visualizing SQL GRAPH_TABLE queries on graphs in the database.
 - See Visualizing Graph Queries on SQL Property Graphs for more information.

Significant New Features in Oracle Graph Server and Client 23.1 (Applies for 23ai and Prior Oracle Database Releases)

- Added support to create property graph views by importing graph data from a GraphSON file.
 - See Creating a PG View By Importing a GraphSON file for more information.
- Added support for explicitly specifying the schema name when:
 - Loading a property graph view into the graph server (PGX) using the readGraphByName API.
 - See Specifying the Schema Name for the readGraphByName API for more information.
 - Loading a subgraph into memory in the graph server (PGX).
 See Loading a Subgraph by Explicitly Specifying the Schema Name for more information.
 - Expanding an in-memory subgraph .
 See Expanding a Subgraph by Explicitly Specifying the Schema Name for more information.



- Added a new release artifact, oracle-graph-visualization-library-23.1.0.zip, to the
 Oracle Graph Server and Client deployment. This contains a JavaScript library for the
 Graph Visualization interface which you can embed in your web application.
 See Embedding the Graph Visualization Library in a Web Application for more
 information.
- Enhanced the graph server (PGX) ServerInstance#getServerState API to provide the session user information.
 See Get Inspection Data for more information.
- Added v2 for the Graph Visualization GET /graphs REST endpoint.
 See List Graphs for more information.
- Added support for Supervised EdgeWise graph algorithm in the PgxML Library.
 See Using the Supervised EdgeWise Algorithm (Edge Embeddings and Classification) for more information.
- Added support for regression tasks on Supervised GraphWise Models.
 See Classification Versus Regression Models on Supervised GraphWise Models for more information.

Deprecated Features

GraphServer#getInstance API

The following GraphServer#getInstance APIs are deprecated:

- GraphServer.getInstance(ClientConfig clientConfig, String username, char[] password, int refreshTimeBeforeTokenExpiry)
- GraphServer.getInstance(String baseUrl, String username, char[] password, int refreshTimeBeforeTokenExpiry)
- GraphServer.getInstance(String baseUrl, String kerberosTicketPath, int refreshTimeBeforeTokenExpiry)

Instead, configure the $refresh_time_before_token_expiry_seconds$ parameter in the pgx.conf file.

opg_apis.get_version()

The <code>OPG_APIS.GET_VERSION()</code> function is deprecated and will be desupported in a future release. Instead, use <code>OPG_APIS.GET_OPG_VERSION</code>.

Apache HDFS support for Cloudera CDH6

Support for Apache HDFS on Cloudera CDH6 is deprecated.

Methods deprecated for PgglViewGraphExpander

PgqlViewGraphExpander.schema(String) and PgqlViewGraphExpander.owner(String) are deprecated. Instead, use PgqlViewGraphExpander.fromPgView(String, String).

Graph Server (PGX) Configuration Fields

The graph server configuration fields, server_cert and server_private_key are deprecated. Instead, use server keystore.

PyPGX

- The following attributes on Operation are now deprecated: graph_id, operation_type, cost_estimate, total_cost_estimate, cardinality_estimate, pattern_info, and children. Instead, use the corresponding getter methods, such as get graph id(), get operation type(), and so on.
- The pgx_version attribute in ServerInstance class is deprecated. Instead, use get version().



- The attribute pg_view_name in PartitionedGraphConfig is deprecated.
 Instead, use source name and source type (set to pg view).
- set_standarize in GraphWiseModelConfig is deprecated. Instead, use set standardize.
- The return value of PgqlResultSet.get_vertex_labels may or may not be a list

Subgraph Loading

Creating Subgraphs using filter expressions is deprecated. Instead, use Loading a Subgraph from Property Graph Views.

PgxML: inferAndGetExplanation Function

GraphWiseModel.inferAndGetExplanation() is deprecated. Instead, use GraphWiseModel.gnnExplainer() to obtain a GnnExplainer object for the model and use GnnExplainer.inferAndExplain().

- Pg2vecModelBuilder.setUseGraphletSize(java.lang.Boolean useGraphletSize) method in oracle.pgx.api.mllib API is deprecated. Instead, use the Pg2vecModelBuilder.setUseGraphletSize(boolean useGraphletSize) method.
- PgxML: SupervisedGraphWiseModelBuilder.setLossFunction Function
 SupervisedGraphWiseModelBuilder.setLossFunction(SupervisedGraphWiseModelConfig.LossFunction ...) is deprecated. Instead, use
 SupervisedGraphWiseModelBuilder.setLossFunction(LossFunction ...)
 function.

PL/SQL API OPG_APIS.GET_SCN Function

The PL/SQL API OPG_APIS.GET_SCN function is deprecated. Instead, to retrieve the current SCN (system change number), use the DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER function:

SELECT dbms flashback.get system change number FROM DUAL;

Desupported Features

- Groovy support for using the Java API in Apache Zeppelin client is desupported.
- Oracle Linux 6 is desupported.
- Oracle Text with property graph schema graphs is desupported.
- Apache HBase is desupported.
- Support for mixed case string arguments in PyPGX for cases where there are a fixed, enumerated list of possible values (such as, ['linear', 'tanh', 'relu']) are desupported. Only lower case arguments are now supported.
- The two-table format is desupported.
- The following Java API classes are desupported:
 - oracle.pg.rdbms.OraclePgqlColumnDescriptor.java
 - oracle.pg.rdbms.OraclePgqlColumnDescriptorImpl.java
 - oracle.pg.rdbms.OraclePgqlExecution.java
 - oracle.pg.rdbms.OraclePgqlExecutionFactory.java



- oracle.pg.rdbms.OraclePgqlPreparedStatement.java
- oracle.pg.rdbms.OraclePgqlResult.java
- oracle.pg.rdbms.OraclePgqlResultElement.java
- oracle.pg.rdbms.OraclePgqlResultElementImpl.java
- oracle.pg.rdbms.OraclePgqlResultImpl.java
- oracle.pg.rdbms.OraclePgqlResultIterable.java
- oracle.pg.rdbms.OraclePgqlResultIteratorImpl.java
- oracle.pg.rdbms.OraclePgqlResultSet.java
- oracle.pg.rdbms.OraclePgqlResultSetImpl.java
- oracle.pg.rdbms.OraclePgqlResultSetMetaData.java
- oracle.pg.rdbms.OraclePgqlResultSetMetaDataImpl.java
- oracle.pg.rdbms.OraclePgqlSqlTrans.java
- oracle.pg.rdbms.OraclePgqlSqlTransImpl.java
- oracle.pg.rdbms.OraclePgqlStatement.java
- The following Java API methods, objects and fields in oracle.pgx.api are no longer supported:

Desupported Methods:

- PgxCollection methods:
 - * addAllAsync(Collection<E> source)
 - * removeAllAsync(Collection<E> source)
 - * addAll(ID...ids)
 - * removeAll(ID...ids)
- PgqlResultSet methods:
 - * getResults(): instead, use PgqlResultSet to directly iterate the result set
 - * destroy()
- User-defined pattern matching semantic methods:
 - * PgxGraph#queryPgql(String, PatternMatchingSemantic): instead, use PgxGraph#queryPgql(String)
 - * PgxSession.setPatternMatchingSemantic(..)
- GraphMetaData constructors and related methods:
 - * GraphMetaData()
 - * GraphMetaData(GraphMetaData other, java.net.URI baseUri)
 - * GraphMetaData(IdType vertexIdType)
 - * GraphMetaData.setVertexIdType()
 - * GraphMetaData.setEdgeIdType()
- PgxSession#getAvailableSnapshots(GraphConfig): instead, use PgxSession#getAvailableSnapshots(PgxGraph)



 All Analyst#filteredBfs and Analyst#filteredDfs methods that accepts filter parameter: instead, use the navigator parameter

Desupported Objects

- PgqlResult(a result of resultSet.getResults().iterator().next():
 instead, use PgxResult as returned from resultSet.iterator().next()

Desupported Fields

- pattern matching semantic configuration field
- The Java API method AbstractGraphConfigBuilder#setNodeIdType in oracle.pgx.config is desupported. Instead, use AbstractGraphConfigBuilder#setVertexIdType().
- The following PyPGX classes are desupported in pypgx package. Instead, use pypgx.api.filters subpackage to access these classes:
 - EdgeFilter
 - GraphFilter
 - VertexFilter
- The following PyPGX classes are desupported in pypgx.api package. Instead, use pypgx.api.frames subpackage to access these classes:
 - PgxCsvFrameReader
 - PgxCsvFrameStorer
 - PgxDbFrameReader
 - PgxDbFrameStorer
 - PgxFrame
 - PgxFrameBuilder
 - PgxFrameColumn
 - PgxGenericFrameReader
 - PgxGenericFrameStorer
 - PgxPgbFrameReader
 - PgxPgbFrameStorer
- The following Python API packages are no longer supported:
 - common: This internal package is desupported. Few of the classes from this package are moved to the public package pypgx.api.
 - utils: This internal package is renamed to utils.
- Graph property text search based on Apache Solr/Lucene is desupported. Instead, use PGQL query expressions.
- The PGX property type DATE is desupported. Instead, use LOCAL_DATE or TIMESTAMP.
- Property Graph support for data stored in Oracle NoSQL Database is desupported.
- Support for Gremlin Groovy shell is desupported.



- Apache Tinkerpop API support for Oracle Database is desupported.
- Loading data from flat file formats into the property graph schema is desupported.
- Support for the Apache Groovy-based shell was deprecated in 19c and is now desupported.
- Support for Apache HBase and Apache HDFS on Cloudera CDH5 is desupported.



Part I

Getting Started with Oracle Property Graphs

Part I provides the fundamental information to get you started on the property graph feature of Oracle Database.

This part covers the following:

- Introduction to Property Graphs
 Property graphs give you a different way of looking at your data.
- Using Oracle Graph with the Autonomous Database
 Oracle Graph with the Autonomous Database allows you to create property graphs from data in your Autonomous Database.



1

Introduction to Property Graphs

Property graphs give you a different way of looking at your data.

You can model your data as a graph by making data entities **vertices** in the graph, and relationships between them as **edges** in the graph. For example, in a bank, customer accounts can be vertices, and cash transfer relationships between them can be edges.

When you view your data as a graph, you can analyze your data based on the connections and relationships between them. You can run graph analytics algorithms like PageRank to measure the relative importance of data entities based on the relationships between them (for instance, links between web pages).

What Are Property Graphs?

A property graph consists of a set of objects or **vertices**, and a set of arrows or **edges** connecting the objects.

- About the Property Graph Feature of Oracle Database
 - The Property Graph feature delivers advanced graph query and analytics capabilities in Oracle Database.
- Overview of Property Graph Architecture
 - The property graph feature of Oracle Database supports the following architecture models.
- Learn About the Graph Server (PGX)
 - The in-memory graph server layer enables you to analyze property graphs using parallel in-memory execution.
- Security Best Practices with Graph Data
 Several security-related best practices apply when working with graph data.
- About Oracle Graph Server and Client Accessibility
 This section provides information on the accessibility features for Oracle Graph Server and Client.

1.1 What Are Property Graphs?

A property graph consists of a set of objects or **vertices**, and a set of arrows or **edges** connecting the objects.

Vertices and edges can have multiple properties, which are represented as key-value pairs.

Each vertex has a unique identifier and can have:

- A set of outgoing edges
- A set of incoming edges
- A collection of properties

Each edge has a unique identifier and can have:

- An outgoing vertex
- An incoming vertex

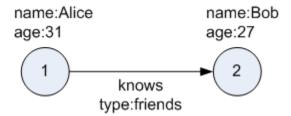


- A text label that describes the relationship between the two vertices
- A collection of properties

For vertices and edges, each property is identified with a unique name.

The following figure illustrates a very simple property graph with two vertices and one edge. The two vertices have identifiers 1 and 2. Both vertices have properties name and age. The edge is from the outgoing vertex 1 to the incoming vertex 2. The edge has a text label knows and a property type identifying the type of relationship between vertices 1 and 2.

Figure 1-1 Simple Property Graph Example



A property graph can have self-edges (that is, an edge whose source and destination vertex are the same), as well as multiple edges between the same source and destination vertices.

A property graph can also have different types of vertices and edges in the same graph. For example a graph can have a set of vertices with label Person and a set of vertices with label Place, with different properties relevant to these two sets of vertices.

The property graph data model is similar to the W3C standards-based Resource Description Framework (RDF) graph data model; however, the property graph data model is simpler and less precise than RDF.

The property graph data model features and analytic APIs make property graphs a good candidate for use cases such as these:

- Identifying influencers in a social network
- Predicting trends and customer behavior
- Discovering relationships based on pattern matching
- Identifying clusters to customize campaigns

1.2 About the Property Graph Feature of Oracle Database

The Property Graph feature delivers advanced graph query and analytics capabilities in Oracle Database.

This feature supports graph operations, indexing, queries, search, and in-memory analytics.

Graphs manage networks of linked data as vertices, edges, and properties of the vertices and edges. Graphs are commonly used to model, store, and analyze



relationships found in social networks, cybersecurity, utilities and telecommunications, life sciences and clinical data, and knowledge networks.

Typical graph analyses encompass graph traversal, recommendations, finding communities and influencers, and pattern matching. Industries including telecommunications, life sciences and healthcare, security, media, and publishing can benefit from graphs.

The property graph features of Oracle Database support those use cases with the following capabilities:

- A scalable graph database
- Developer-based APIs based upon PGQL and Java graph APIs
- · A parallel, in-memory graph server (PGX) for running graph queries and graph analytics
- A fast, scalable suite of social network analysis functions that include ranking, centrality, recommender, community detection, and path finding
- Parallel bulk load and export of property graph data in Oracle-defined flat files format
- A powerful Graph Visualization application
- Notebook support through integration with Jupyter

1.3 Overview of Property Graph Architecture

The property graph feature of Oracle Database supports the following architecture models.

- Architecture Model for Running Graph Queries in the Database
 Using any of the supported client tools, you can directly interact with the graph data
 stored in the relational tables in the database.
- Architecture Model for Running Graph Analytics
 You can load your property graph into the graph server (PGX) in order to perform specialized graph computations.
- Developing Applications Using Graph Server Functionality as a Library
 The graph functions available with the graph server (PGX) can be used as a library in
 your application.

1.3.1 Architecture Model for Running Graph Queries in the Database

Using any of the supported client tools, you can directly interact with the graph data stored in the relational tables in the database.

This approach runs graph queries, as shown in the following figure.



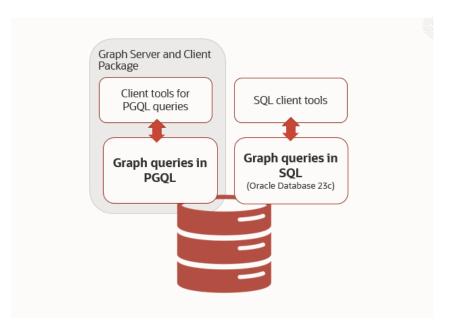


Figure 1-2 Property Graph Architecture for Running Graph Queries

This model allows you to create a property graph using any one of the following supported options:

- Create a SQL property graph directly over existing database schema objects using SQL DDL statement. See SQL Property Graphs for more information.
- Create a property graph view directly over the graph data in the tables. See Property Graph Views for more information.

You can directly query the graphs, without loading the graphs into the graph server (PGX), using PGQL. Additionally, you can also run graph pattern matching queries on SQL property graphs using the <code>GRAPH_TABLE</code> operator. See SQL GRAPH_TABLE Queries for more information.

However, if you want to run graph analytics algorithms, then you must load this graph into the graph server (PGX). You can configure the graph server to periodically fetch data updates from the database to keep the graph synchronized. Note that loading the graph into the graph server (PGX) and performing graph synchronization operations are supported only for property graph views.

1.3.2 Architecture Model for Running Graph Analytics

You can load your property graph into the graph server (PGX) in order to perform specialized graph computations.



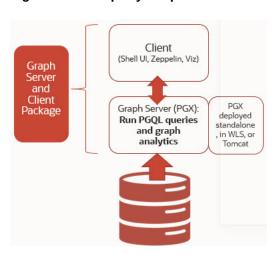


Figure 1-3 Property Graph Architecture for Running Graph Analytics

As seen in the preceding architecture design, the graph server (PGX) is a mid-tier server that can run as a standalone, or in a container like Oracle WebLogic Server or Apache Tomcat. Using this approach, you can load your property graph into the graph server (PGX). This allows you to run graph queries and analytical operations in memory in the graph server.

The graph can be created directly from the relational tables, or loaded from a property graph view which stores the graph in the database. You can modify the graph in memory (insert, update, and delete vertices and edges, and create new properties for results of executing an algorithm). The graph server does not write the modifications back to the relational tables.

Property Graph Sizing Recommendations

You can compute the memory required by the graph server (PGX) by using this calculator, Graph Size Estimator.

For example, the following table shows the memory estimated by the calculator for the given input:

Table 1-1 Graph Size Estimator

Number of vertices	of	Properties per Vertex	Properties per Edge	Estimated graph size
10M	100M	• 4 - Integer Type • 1 - String Type(15 characters)	• 4 - Integer Type • 1 - String Type(15 characters)	15 GB
100M	1B	• 4 - Integer Type • 1 - String Type(15 characters)	• 4 - Integer Type • 1 - String Type(15 characters)	140 GB



Note:

- Reading a graph into memory can take upto twice the amount of memory needed to represent it in memory. So when you calculate the memory required for running PGX it is recommended that you double the amount of memory of the estimated graph size.
- CPU Processors: The recommended number of CPU processors for a graph with 10M vertices and 100M edges is 2-4 processors, and up to 16 processors for more compute-intensive workloads. Increasing CPU processors will improve performance.

1.3.3 Developing Applications Using Graph Server Functionality as a Library

The graph functions available with the graph server (PGX) can be used as a library in your application.

After the rpm install of the graph server, all the jar files can be found in <code>/opt/oracle/graph/lib</code>. In this case, the server installation and the client user application are in the same machine.

For such use cases, development and testing can be done using the interactive Java shell or the Python shell in embedded (local) mode. This means a local PGX instance is created and runs in the same JVM as the client. If you start the shell without any parameters it will start a local PGX instance and run in embedded mode.

See Using the Graph Server (PGX) as a Library for more information to obtain reference to a local PGX instance.

1.4 Learn About the Graph Server (PGX)

The in-memory graph server layer enables you to analyze property graphs using parallel in-memory execution.

It provides over 60 analytic functions. Examples of the categories and specific functions include:

- Centrality Degree Centrality, Eigenvector Centrality, PageRank, Betweenness Centrality, Closedness Centrality
- Component and Community Strongly Connected Components (Tarjan's and Kosaraju's). Weakly Connected Components
- Twitter's Who-To-Follow, Label Propagation.
- Path Finding Single source all destination (Bellman-Ford), Dijsktra's shortest path, Hop Distance (Breadth-first search)
- Community Evaluation Coefficient (Triangle Counting), Conductance, Modularity, Adamic-Adar counter.



Overview of the Graph Server (PGX)

The Graph Server (PGX) is an in-memory accelerator for fast, parallel graph query and analytics. The server uses light-weight in-memory data structures to enable fast execution of graph algorithms.

Related Topics

- Installing Oracle Graph Server
- Getting Started with the Graph Server (PGX)
 Once you have installed the graph server (PGX), you can start and connect to a graph server instance.

1.4.1 Overview of the Graph Server (PGX)

The Graph Server (PGX) is an in-memory accelerator for fast, parallel graph query and analytics. The server uses light-weight in-memory data structures to enable fast execution of graph algorithms.

There are multiple options to load a graph into the graph server either from Oracle Database or from files.

The graph server can be deployed standalone (it includes an embedded Apache Tomcat instance), or deployed in Oracle WebLogic Server or Apache Tomcat.

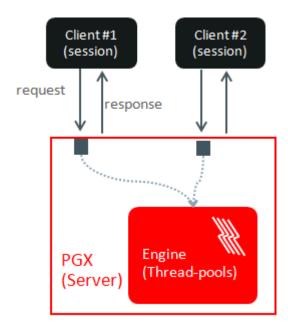
- Design of the Graph Server (PGX)
- Usage Modes of the Graph Server (PGX)

1.4.1.1 Design of the Graph Server (PGX)

The design of the graph server (PGX) is based on a Server-Client usage model. See Usage Modes of the Graph Server (PGX) for more details on the different graph server (PGX) execution modes.

The following figure shows the graph server (PGX) design:

Figure 1-4 Graph Server (PGX) Design





The core concepts of the graph server (PGX) design are as follows:

- Multiple graph clients can connect to the graph server at the same time.
- Each client request is processed by the graph server asynchronously. The client requests are queued up first and processed later, when resources are available. The client can poll the server to check if a request has been finished.
- Internally, the server maintains its own engine (thread pools) for running parallel graph algorithms and queries. The engine tries to process each analytics request concurrently with as many threads as possible.

Isolation Between Concurrent Clients

The graph server (PGX) supports data isolation between concurrent clients. Each client has its own private workspace, called session. Sessions are isolated from each other. Each client can load a graph instance into its own session, independently from other clients. Therefore, each client can load a graph instance (as well as its properties) into its own session, independently from other clients.

1.4.1.2 Usage Modes of the Graph Server (PGX)

This section presents an overview of the different usage modes of the graph server (PGX). The graph server can be executed in one of the following usage modes.

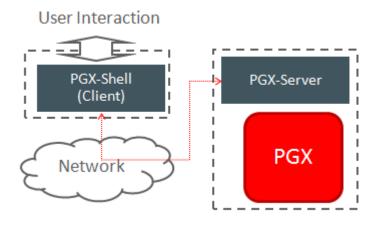
Remote Server Mode

In the remote server mode, the main PGX execution engine is deployed as a RESTful application on a powerful server machine, and you can connect to it remotely from your machine using graph shell. Also, multiple clients can connect to the same graph server (PGX) at the same time and therefore the graph server is time-shared among these clients.

See Interactive Graph Shell CLIs for more information on the graph shell.

The following figure shows the graph server (PGX) in a remote execution mode:

Figure 1-5 Remote Server Mode



The remote server mode is useful for the following situations where you want to:



- Perform graph analysis on a large data set with a powerful server-class machine that has many cores and a large memory.
- The server-class machine is shared by multiple clients.

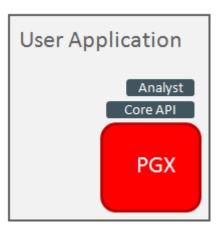
See Starting the Graph Server (PGX) for instructions on how to start the graph server (PGX) in remote server mode.

Using Graph Server (PGX) as a Library

You can also include the graph server (PGX) as a normal Java library in your application.

The following figure shows the graph server (PGX) used as a library in an application:

Figure 1-6 PGX as a Library



The embedded mode is useful when you want to build an application having graph analysis as a part of its functionality.

See Using the Graph Server (PGX) as a Library for more information.

Deploying Graph Server (PGX) as Servlet Web Application

You can deploy the graph server (PGX) as a web application using Apache Tomcat or Oracle WebLogic Server.

See Deploying Oracle Graph Server to a Web Server for instructions to deploy the graph server (PGX) in Apache Tomcat or Oracle WebLogic Server.

1.5 Security Best Practices with Graph Data

Several security-related best practices apply when working with graph data.

Sensitive Information

Graph data can contain sensitive information and should therefore be treated with the same care as any other type of data. Oracle recommends the following considerations when using a graph product:

 Avoid storing sensitive information in your graph if that information is not required for analysis. If you have existing data, only model the relevant subset you need for analysis

- as a graph, either by applying a preprocessing step or by using subgraph and filtering techniques that are part of graph product.
- Model your graph in a way that vertex and edge identifiers are not considered sensitive information.
- Do not deploy the product into untrusted environments or in a way that gives access to untrusted client connections.
- Make sure all communication channels are encrypted and that authentication is always enabled, even if running within a trusted network.

Least Privilege Accounts

The database user account that is being used by the graph server (PGX) to read data should be a low-privilege, read-only account. PGX is an in-memory accelerator that acts as a read-only cache on top of the database, and it does not write any data back to the database.

If your application requires writing graph data and later analyzing it using PGX, make sure you use two different database user accounts for each component.

Public Health Endpoint Security

Unless you run multiple graph servers behind a load balancer (Deploying Oracle Graph Server Behind a Load Balancer), it is a good security practice to disable the public endpoint of the graph server, which load balancers need to determine the health of the graph servers.

To disable the endpoint:

- Locate the WAR file of the graph server. If you installed the graph server via RPM, then the file is located at /opt/oracle/graph/pgx/server/pgx-webapp-<version>.war.
- 2. Unzip the .war file into a location of your choice and then edit the WEB-INF/web.xml file inside the unzipped directory with a text editor of your choice.
- 3. Locate the pgx.auth.exceptions parameter in the file. The list of public endpoints can be seen as shown:

4. Remove the isReady endpoint from the list of public endpoints as shown:

- **5.** Save your changes, repackage the WAR file and redeploy the file to its original location.
- 6. Restart the graph server.

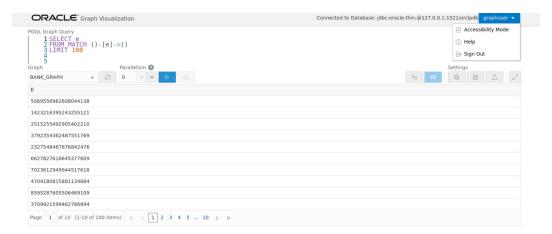


1.6 About Oracle Graph Server and Client Accessibility

This section provides information on the accessibility features for Oracle Graph Server and Client.

- For information on addressing accessibility for the Java and Python command line interfaces, which are installed on Oracle Linux, see Working With Accessibility Features in Oracle Linux 7.
- For information on keyboard shortcuts for the Java command line interface, which is built on top of the Java Shell (JShell), see Keyboard Shortcuts for JShell.
- For information on addressing accessibility for the Graph Visualization Application, which
 is based on Oracle JET, see About Oracle JET and Accessibility.
- You can enable accessibility in the Graph Visualization application by selecting the
 Accessibility Mode check box option from the user account drop-down menu on the top right of the user interface. Once enabled, the query output is always displayed in a
 tabular layout as shown:

Figure 1-7 Enabling Accessibility in the Graph Visualization Application





2

Using Oracle Graph with the Autonomous Database

Oracle Graph with the Autonomous Database allows you to create property graphs from data in your Autonomous Database.

When using Autonomous Database in a shared deployment, you can use Graph Studio, a fully managed service with a powerful user interface for developing applications that use graph analysis. Using Graph Studio, you can automate the modeling of graphs from tables in Autonomous Database. You can interactively analyze and visualize the graph queries using advanced notebooks with multiple visualization options. You can execute over 60 built-in graph algorithms in Graph Studio to gain useful insights on your graph data. See Using Graph Studio in Oracle Autonomous Database for more information.

You can also access few Graph Studio features using the Autonomous Database Graph Client API using the client shell CLIs or through your Java or Python application. See Using Autonomous Database Graph Client for more information.

Alternatively, you can use any version of Oracle Graph Server and Client with the family of Oracle Autonomous Database to create and work with property graphs. This includes any version of Oracle Autonomous Database (shared) or Oracle Autonomous Database (dedicated). You can always upgrade to the latest version of Graph Server and Client regardless of the version of your Autonomous Database. Note that the graph server is managed by the application in this case.

You can connect in two-tier mode (connect directly to Autonomous Database) or three-tier mode (connect to PGX on the middle tier, which then connects to Autonomous Database).

The database schema storing the graph must have the privileges listed in Required Privileges for Database Users.

- Two-Tier Deployments of Oracle Graph with Autonomous Database
 In two-tier deployments, the client graph application connects directly to the Autonomous Database.
- Three-Tier Deployments of Oracle Graph with Autonomous Database
 In three-tier deployments, the client graph application connects to PGX in a middle tier,
 and PGX connects to the Autonomous Database.

Related Topics

Using Autonomous Database Graph Client
Using the AdbGraphClient API, you can access Graph Studio features in Autonomous
Database programmatically using the Oracle Graph Client or through your Java or
Python application.



2.1 Two-Tier Deployments of Oracle Graph with Autonomous Database

In two-tier deployments, the client graph application connects directly to the Autonomous Database.

- 1. Install Oracle Graph Client, as explained in Installing the Java Client From the Graph Server and Client Downloads.
- Establish a JDBC connection, as described in the Oracle Autonomous Warehouse documentation.

You must download the wallet and unzip it to a secure location. You can then reference it when establishing the connection as shown in Example 2-1.

3. Start the Java Shell as shown in the code:

```
/bin/opg4j --no_connect
```

4. Connect to your database as shown in Example 2-1.

Note:

If you need to use the Graph Visualization Application, you must additionally install the Oracle Graph Server.

- See Installing Oracle Graph Server for more details.
- See Deploying the Graph Visualization Application for more details on deploying the Graph Visualization Application in Tomcat or Oracle WebLogic Server.

Example 2-1 Creating a Database Connection in a Two-Tier Graph Deployment with Autonomous Database

```
opg4j> var jdbcUrl = "jdbc:oracle:thin:@<tns_alias>?
TNS_ADMIN=<wallet_location>" // jdbc url to the DB
opg4j> var user = "<user>"
opg4j> var pass = "<password>"
opg4j> var conn = DriverManager.getConnection(jdbcUrl, user, pass) //
connecting to the DB
conn ==> oracle.jdbc.driver.T4CConnection@57e6cb01
```

In the preceding example:

- <tns_alias>: TNS alias used in tnsnames.ora file
- <wallet location>: Path to the directory where the wallet is stored
- <user>: Name of the database user
- <password>: Password for the user



2.2 Three-Tier Deployments of Oracle Graph with Autonomous Database

In three-tier deployments, the client graph application connects to PGX in a middle tier, and PGX connects to the Autonomous Database.

The wallets downloaded from the Oracle Cloud Console are mainly *routing wallets*, meaning they are used to route the connection to the right database and to encrypt the connection. In most cases, they are not auto-login wallets, so they do not contain the password for the actual connection. The password usually needs to be provided separately to the wallet location.

The graph server does not support a wallet stored on the client file system or provided directly by remote users. The high level implications of this are:

- The server administrator provides the wallet and stores the wallet securely on the server's file system.
- Similar to Java EE connection pools, remote users will use that wallet when connecting.
 This means the server administrator trusts all remote users to use the wallet. As with any
 production deployments, the PGX server must be configured to enforce authentication
 and authorization to establish that trust.
- Remote users still need to provide a user name and password when sending a graph read request, just as with non-autonomous databases.
- You can only configure one wallet for each PGX server.

Having the same PGX server connecting to multiple Autonomous Databases is not supported. If you have that use case, start one PGX server for each Autonomous Database.

Pre-loaded graphs

To read a graph from Autonomous Database into PGX at server startup, follow the steps described in Store the Database Password in a Keystore to:

- Create a Java Keystore containing the database password
- Create a PGX graph configuration file describing the location and properties of the graph to be loaded
- 3. Update the /opt/oracle/graph/pgx.conf file to reference the graph configuration file

As root user, edit the service file at /etc/systemd/system/pgx.service and specify the environment variable under the [Service] directive:

Environment="JAVA OPTS=-Doracle.net.tns admin=/etc/oracle/graph/wallets"

Make sure that the directory (/etc/oracle/graph/wallets in the preceding code) is readable by the Oracle Graph user, which is the user that starts up the PGX server when using systemd.



In addition, edit the ${\tt ExecStart}$ command to specify the location of the keystore containing the password:

ExecStart=/bin/bash start-server --secret-store /etc/keystore.p12



Please note that /etc/keystore.p12 must not be password protected for this to work. Instead protect the file via file system permission that is only readable by oraclegraph user.

After the file is edited, reload the changes using:

```
systemctl daemon-reload
```

Finally start the server:

```
sudo systemctl start pgx
```

On-demand graph loading

To allow remote users of PGX to read from the Autonomous Database on demand, you can choose from two approaches:

Provide the path to the wallet at server startup time via the <code>oracle.net.tns_admin</code> system property. Remote users have to provide the TNS address name, username and keystore alias (password) in their graph configuration files. The wallet is stored securely on the graph server's file system, and the server administrator trusts all remote users to use the wallet to connect to an Autonomous Database.

For example, the server administrator edits the service file at /etc/systemd/system/pgx.service and specifies the environment variable the under the [Service] directive:

```
Environment="JAVA_OPTS=-Doracle.net.tns_admin=/etc/oracle/graph/
wallets"
```

and then start the server using

```
systemctl start pgx
```

The /etc/oracle/graph/wallets/tnsnames.ora file contains an address as follows:

```
sombrero_medium = (description= (retry_count=20) (retry_delay=3)
(address=(protocol=tcps) (port=1522) (host=adb.us-
ashburn-1.oraclecloud.com))
(connect_data=(service_name=181gholga0ujxsa_sombrero_medium.adwc.ora
clecloud.com)) (security=(ssl server cert dn="CN=adwc.uscom-
```



```
east-1.oraclecloud.com,OU=Oracle BMCS US,O=Oracle Corporation,L=Redwood City,ST=California,C=US")))
```

Now remote users can read data into the server by sending a graph configuration file with the following connection properties:

```
"jdbc_url": "jdbc:oracle:thin:@sombrero_medium",
"username": "hr",
    "keystore_alias": "database1",
...
}
```

Note that the keystore still lives on the client side and should contain the password for the hr user referenced in the config object, as explained in Store the Database Password in a Keystore. A similar approach works for Tomcat or WebLogic Server deployments.

Use Java EE connection pools in your web application server. Remote users only have to
provide the name of the datasource in their graph configuration files. The wallet and the
connection credentials are stored securely in the web application server's file system, and
the server administrator trusts all remote users to use a connection from the pool to
connect to an Autonomous Database.

You can find instructions how to set up such a data source at the following locations:

- WebLogic Server: Configuring a WebLogic Data Source to use ATP
- Tomcat: https://www.oracle.com/technetwork/database/application-development/jdbc/documentation/atp-5073445.html#Tomcat

If you gave the data source the name adb_ds , you can the reference them by sending a graph configuration file with the following connection properties:

```
{
    ...
    "datasource_id": "adb_ds",
    ...
}
```



Part II

SQL Property Graphs

Learn and work with SQL property graphs.

Effective with Oracle Database 23ai, you can create and query SQL property graphs.

The following chapters provide in-depth information on SQL property graphs:

- Introduction to SQL Property Graphs
 You can work with SQL property graphs in any SQL based interface (such as SQL Developer, SQLPLUS, or SQLcl) or from a Java program using JDBC.
- SQL DDL Statements for Property Graphs
 You can create, revalidate, and drop SQL property graphs using SQL data definition
 language (DDL) statements.
- SQL GRAPH_TABLE Queries
 You can query a SQL property graph using the GRAPH_TABLE operator to express graph pattern matching queries.
- Executing PGQL Queries Against SQL Property Graphs
 You can directly run PGQL gueries against a SQL property graph in the database.



Introduction to SQL Property Graphs

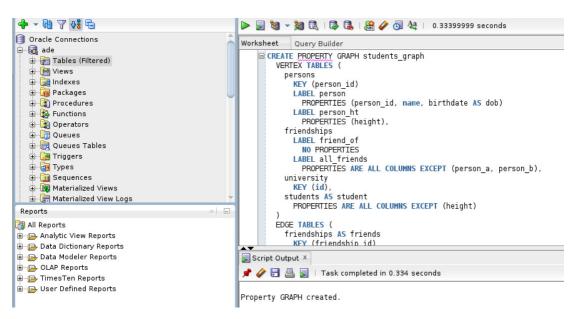
You can work with SQL property graphs in any SQL based interface (such as SQL Developer, SQLPLUS, or SQLcl) or from a Java program using JDBC.

Using SQL statements, you can perform the following:

- Create a SQL property graph from existing database objects in your schema, such as:
 - Tables (with some exceptions as listed in Limitations of Creating a SQL Property Graph)
 - Materialized views
 - External tables
 - Synonyms for any of the preceding database objects
- Create a synonym for a SQL property graph.
- Revalidate a SQL property graph.
- Run graph pattern matching queries on a SQL property graph.
- Drop a SQL property graph.

For example, the following figure shows the creation of a SQL property graph using the SQL Developer tool.

Figure 3-1 Using SQL Developer to Create a SQL Property Graph



Quick Start for Working with SQL Property Graphs
 This tutorial helps you get started on creating and querying a SQL property graph.

3.1 Quick Start for Working with SQL Property Graphs

This tutorial helps you get started on creating and querying a SQL property graph.

In order to try this tutorial, ensure that you meet the following requirements:

- Load the sample bank graph data provided with the graph server installation in the database tables. See Using Sample Data for Graph Analysis for more information.
- You have the required privileges to create and drop a SQL property graph. See Granting System and Object Privileges for SQL Property Graphs for more information.

In the following tutorial, the examples in Step 1, Step 2, and Step 4 are performed using the SQLcl tool. However, you can run these examples using any SQL based interface.

1. Create a SQL property graph using the CREATE PROPERTY GRAPH DDL statement.

```
SQL> CREATE PROPERTY GRAPH bank sql pg
 2 VERTEX TABLES (
 3
      bank accounts
  4
        KEY (id)
 5
       LABEL account
       PROPERTIES ALL COLUMNS
 7
    EDGE TABLES (
 8
 9
      bank txns
10
          KEY (txn id)
11
          SOURCE KEY (from acct id) REFERENCES bank accounts (id)
12
          DESTINATION KEY (to acct id) REFERENCES bank accounts
(id)
13
          LABEL transfer
14
          PROPERTIES ALL COLUMNS
15*
      );
```

Property created.

On execution, the $bank_sql_pg$ graph schema object is created in the database. The graph is made up of one vertex graph element table ($bank_accounts$) and one edge graph element table ($bank_txns$).

See Creating a SQL Property Graph to learn the concepts of graph element tables, keys, labels and properties.

2. Run a GRAPH_TABLE query, on the newly created graph, to list all the transactions from the account with id value 816.

```
SQL> SELECT * FROM GRAPH_TABLE (bank_sql_pg
2    MATCH
3    (a IS account WHERE a.id = 816) -[e IS transfer]-> (b IS account)
4    COLUMNS (a.id AS acc_a, e.amount AS amount, b.id AS acc_b)
5* );
```



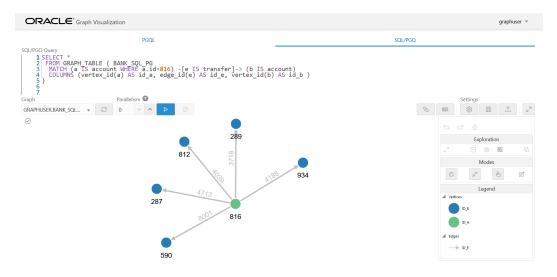
ACC_A	AMOUNT	ACC_B
816	4713	287
816	8001	590
816	4186	934
816	3718	289
816	4039	812

See SQL GRAPH_TABLE Queries to understand more about GRAPH_TABLE queries...

3. Optionally, if you have installed the graph server (PGX), then you can also visualize the preceding GRAPH TABLE query, using the graph visualization tool.

The only difference is that you must return the vertex and edge IDs in order to visualize the vertices and edges of the <code>GRAPH_TABLE</code> query together with their IDs and all their labels and properties. Note that the <code>COLUMNS</code> clause in the following example uses the <code>VERTEX_ID</code> and <code>EDGE_ID</code> operators:

Figure 3-2 Visualizing GRAPH_TABLE Query



- See Vertex and Edge Identifiers to learn more about the VERTEX_ID and EDGE_ID operators.
- See Visualizing Graph Queries on SQL Property Graphs for more details.
- 4. Drop the SQL property graph after running the graph queries.

```
SQL> drop property graph bank_sql_pg;
```

Property dropped.



SQL DDL Statements for Property Graphs

You can create, revalidate, and drop SQL property graphs using SQL data definition language (DDL) statements.

- Creating a SQL Property Graph
 - Using the CREATE PROPERTY GRAPH DDL statement, you can create a property graph schema object directly in an Oracle Database.
- Revalidating a SQL Property Graph
 Using the ALTER PROPERTY CRAPH COMPLIE DDL statement your
 - Using the ALTER PROPERTY GRAPH COMPILE DDL statement, you can revalidate an existing property graph schema object in the database.
- Dropping a SQL Property Graph
 Using the DROP PROPERTY GRAPH DDL statement, you can remove a property graph
- Using the DROP PROPERTY GRAPH DDL statement, you can remove a property graph schema object in Oracle Database.
- JSON Support in SQL Property Graphs
 When creating a SQL property graph, you can define a label property over a JSON data
 type column using simplified dot notation. You can later access this property inside the
 GRAPH TABLE query.

4.1 Creating a SQL Property Graph

Using the CREATE PROPERTY GRAPH DDL statement, you can create a property graph schema object directly in an Oracle Database.

Example 4-1 Creating a SQL Property Graph Using the CREATE PROPERTY GRAPH DDL Statement

This example creates a SQL property graph, students_graph, using persons, university, friendships, and students as the underlying database tables for the graph.

In order to run this example, ensure the following:

- Set up the sample tables in the database as explained in Setting Up Sample Data in the Database.
- 2. See Granting System and Object Privileges for SQL Property Graphs to ensure you have the required privileges to create a SQL property graph.

```
CREATE PROPERTY GRAPH students_graph

VERTEX TABLES (

persons KEY (person_id)

LABEL person

PROPERTIES (person_id, name, birthdate AS dob)

LABEL person_ht

PROPERTIES (height),

friendships

LABEL friend_of

NO PROPERTIES

LABEL all friends
```

```
PROPERTIES ARE ALL COLUMNS EXCEPT (person_a, person_b),
university KEY (id),
students AS student
PROPERTIES ARE ALL COLUMNS EXCEPT (height)

)

EDGE TABLES (
friendships AS friends
KEY (friendship_id)
SOURCE KEY (person_a) REFERENCES persons(person_id)
DESTINATION KEY (person_b) REFERENCES persons(person_id)
PROPERTIES (friendship_id, meeting_date),
students AS student_of
SOURCE KEY (s_person_id) REFERENCES persons(person_id)
DESTINATION KEY (s_univ_id) REFERENCES university(id)
PROPERTIES (subject)
);
```

On execution, the preceding example creates a SQL property graph object that uses the tables in your schema to define its graph element tables. Note that the creation of the new property graph schema object, results only in the storage of the property graph metadata, and there is no copying of data from the underlying database objects into the graph element tables. This implies that when querying a SQL property graph, all <code>GRAPH_TABLE</code> queries are performed on the current graph data in the database. You may also specify another schema to contain the SQL property graph provided that you have sufficient privileges.

The graph definition in the example creates a graph that comprises:

- Four vertex graph element tables:
 - persons: The table has an explicitly defined unique key, person_id, and it is associated with two labels:
 - * person: This label exposes person id, name and birthdate as properties.
 - * person ht: This label exposes only the height property.
 - friendships: The unique key is implicitly inferred from the underlying database object and the table is associated with two labels:
 - * friend of: This label exposes no properties.
 - * all_friends: This label exposes all columns of the underlying database table as properties, except person_a and person_b.
 - university: The label for the table is implicitly inferred and by default all visible columns of the underlying database table are exposed as properties.
 - students: The table uses a table alias student and exposes all columns of the underlying database table as properties, except height.
- Two edge graph element tables:
 - friends: The edge table references persons as the underlying database table for both the source and destination vertex tables. The source and destination keys (person_a and person_b) for the edge table correspond to the unique key of the source and destination vertex tables respectively. The label for the edge table is automatically inferred from the name of the graph element table



(friends, in this case) and exposes friendship id and meeting date as properties.

- student_of: The edge table references persons and university as the underlying database tables for the source and destination vertex tables respectively. The source and destination keys (s_person_id and s_univ_id) for the edge table correspond to the unique key of the source and destination vertex tables respectively. The label for the edge table is automatically inferred from the name of the graph element table (student of, in this case) and exposes subject as the property.

It is important to note that once a SQL property graph is created, you cannot alter the graph definition. However, you can redefine a SQL property graph using the OR REPLACE clause in the CREATE PROPERTY GRAPH DDL statement. You can use this clause to change the definition of an existing SQL property graph without dropping, re-creating, and regranting object privileges that were earlier granted on it.



CREATE PROPERTY GRAPH in Oracle Database SQL Language Reference

The following sections explain more on the concepts of the graph element tables, keys, labels and properties:

- About Vertex and Edge Graph Element Tables
 The vertices and edges of a SQL property graph defined from the underlying database objects are stored in the graph element tables.
- About Vertex and Edge Table Keys
 Each vertex and edge table used in a SQL property graph definition must have a key in order to identify a unique vertex or an edge in a SQL property graph.
- About Labels and Properties
 Labels can be associated to one or more graph element tables and they enrich the graph definition. A label can be defined with or without properties.
- Using Graph Options to Create SQL Property Graphs
 You can use graph options to control the behavior of a SQL property graph at the time of its creation.
- Granting System and Object Privileges for SQL Property Graphs
 Learn about the new system and object privileges for performing operations on SQL property graphs.
- Retrieving SQL Creation DDL Using the DBMS_METADATA Package
- Limitations of Creating a SQL Property Graph
 This section lists a few restrictions that apply when creating a SQL property graph.

4.1.1 About Vertex and Edge Graph Element Tables

The vertices and edges of a SQL property graph defined from the underlying database objects are stored in the graph element tables.

A graph element table can either be a vertex table or an edge table.

Refer to the graph definition in Example 4-1 to easily understand the following sections:



Vertex graph element table

- A vertex table is defined using the VERTEX TABLES clause.
- Each row in a vertex table corresponds to a vertex of the graph.
- A vertex graph element table has a name that is independent from the name of the underlying database object.
- By default, the name of the vertex graph element table is the same as the name of the underlying database object.
- A vertex table name must be unique for a graph. In case you want to define a SQL property graph with multiple graph element tables from the same database object, then you must specify an alternate graph element table name using the AS clause.

Edge graph element table

- An edge table is defined using the EDGE TABLES clause.
- It specifies a direct relationship between the source vertex table and the destination vertex table using the SOURCE and DESTINATION keywords that REFERENCES the respective vertex tables.
- Each row in an edge table corresponds to an edge of the graph.
- An edge graph element table has a name that is independent from the name of the underlying database object.
- By default, the name of the edge graph element table is the same as the name of the underlying database object.
- The edge table name must be unique for a graph. An edge table name cannot be shared with a vertex table or another edge table. For instance, in Example 4-1, since the source friendships table is used both as a vertex and an edge table, a table alias friends is declared using the AS clause when defining the edge table.

4.1.2 About Vertex and Edge Table Keys

Each vertex and edge table used in a SQL property graph definition must have a key in order to identify a unique vertex or an edge in a SQL property graph.

The key is defined from one or more columns of the underlying table. The key may be implicitly inferred based on an existing primary key or a unique constraint defined on the underlying table, or explicitly defined. The key should be unique.

However, note that the uniqueness constraint for the key column is required if you create the graph in ENFORCED MODE. Otherwise, you can create the graph in TRUSTED MODE using key columns that do not have a uniqueness constraint. See Using Graph Options to Create SQL Property Graphs for more information on the different modes that can be applied during graph creation.

Vertex or edge table keys can be defined for any of the following built-in data type columns:

- VARCHAR2
- NVARCHAR2
- NUMBER



- BINARY FLOAT
- BINARY DOUBLE
- CHAR
- NCHAR
- DATE
- INTERVAL (both YEAR TO MONTH and DAY TO SECOND)
- TIMESTAMP

Note that the TIMESTAMP WITH TIME ZONE data type is not supported.

Refer to the SQL property graph definition in Example 4-1 to easily understand the following sections:

Vertex Table Key

- By default, the key for a vertex table is automatically identified from a single PRIMARY KEY or UNIQUE key constraint on the underlying database object. If both exist, then the PRIMARY KEY constraint takes precedence over the UNIQUE key constraint.
- If the vertex table key is automatically inferred based on a single UNIQUE key, then the set of columns in that UNIQUE key must also be NOT NULL.
- If the underlying database object does not contain a unique constraint to enforce uniqueness, then you must explicitly define the KEY subclause in the VERTEX TABLES clause, to identify the columns that define a unique key for the vertex table. Note that the column names must match the column names of the underlying database object.
- Composite vertex table keys are also supported.

Edge Table Key

- By default, the key for an edge table is automatically identified from a single PRIMARY KEY or UNIQUE key constraint on the underlying database object. If both exist, then the PRIMARY KEY constraint takes precedence over the UNIQUE key constraint.
- If the edge table key is automatically inferred based on a single UNIQUE key, then the set of columns in that UNIQUE key must also be NOT NULL.
- If the underlying database object does not contain a unique constraint to enforce uniqueness, then you must explicitly define the KEY subclause in the EDGE TABLES clause, to identify the columns that define a unique key for the edge table. Note that the column names must match the column names of the underlying database object.
- By default, the SOURCE and DESTINATION table keys are automatically obtained from a single FOREIGN KEY constraint between the edge table and the underlying source and destination tables respectively.
- However, you must explicitly specify the KEY subclause for the SOURCE and DESTINATION vertex tables, if any of the following applies:
 - There is no FOREIGN KEY constraint between the edge and the referenced vertex tables.
 - There are multiple FOREIGN KEY constraints between the edge and the referenced vertex tables.



 The underlying database objects for the edge table and its source and destination vertex tables are materialized views or external tables.



All restrictions that apply for primary key constraints on a database object also apply on vertex and edge table keys.

4.1.3 About Labels and Properties

Labels can be associated to one or more graph element tables and they enrich the graph definition. A label can be defined with or without properties.

You can optionally define LABELS and PROPERTIES for the vertex and edge tables in your graph. When not specified, the graph element tables are automatically assigned a label with the name of the graph element table, and all visible columns are exposed as properties, using the column name as property name.

Refer to the SQL property graph definition in Example 4-1 to easily understand the following sections:

Labels

- By default, the vertex and edge tables are automatically assigned a label with the name of the respective graph element tables.
- The DEFAULT LABEL subclause can also be used to explicitly apply the preceding rule.
- You can explicitly assign a new label name to a vertex or an edge graph element table using the LABEL subclause.
- Multiple labels can be associated with the same graph element table.
- The same label can be **shared** with multiple graph element tables.
 A label can be associated with more than one graph element table (shared label) provided the following conditions apply:
 - All graph element tables that share this label declare the same set of property names. Note that the property order does not matter in the label definition.
 - Different columns or value expression exposed by the same property name have *union compatible* types.
- Also, refer to Type Compatibility Rules for Determining Property Types for more information.

Properties

- By default, all the visible columns of a vertex or an edge table are automatically exposed as properties if there is no label declaration or if the DEFAULT LABEL subclause is used in the property graph definition. The property names are the same as the column names of the underlying database object.
- Columns of any Oracle built-in data types can be exposed as properties of labels in a SQL property graph. This includes virtual columns, JSON data type columns, CLOB and BLOB data types.
 - However, the following are not supported:



- XMLType and SDO GEOMETRY type columns are not supported.
- SQL/XML value expressions over XMLType column stored as binary XML, and SDO_GEOMETRY built-in functions over SDO_GEOMETRY object datatype column are allowed as long as they return a value of a type supported for properties. Any general object data type and user defined data type and their corresponding SQL operator value expression over them are not supported.
- Columns of type ANYTYPE cannot be exposed as property.
- At the time of the SQL property graph creation, the data type of a vertex or edge property is determined as follows:
 - Distinct properties associated with distinct labels have the same data type as the underlying database columns.
 - Properties with the same name coming from different labels have the same data type as the underlying database columns. However, you must use the ALLOW MIXED PROPERTY TYPES option when creating the SQL property graph.
 See Using Graph Options to Create SQL Property Graphs for an example using a shared property name.
 - Properties with the same name coming from the same label will have the UNION ALL compatible type of the underlying database columns. In addition, you must use the ALLOW MIXED PROPERTY TYPES option when creating the SQL property graph:
 - * See Using Graph Options to Create SQL Property Graphs for an example using a shared property name in a shared label.
 - * See Type Compatibility Rules for Determining Property Types for more information on the type rules that determine the property type.
- If you want to explicitly define the vertex or edge properties for a label, then the following property declarations are supported:
 - PROPERTIES [ARE] ALL COLUMNS: To expose all the visible columns of the graph element table as label properties. However, if any columns are added or deleted in the source database object, after the creation of the SQL property graph, then these will not be reflected on the graph.
 - PROPERTIES [ARE] ALL COLUMNS EXCEPT (<column_names_list>): To expose all the visible columns of the graph element table as label properties except those that are explicitly listed.
 - PROPERTIES (<1ist_of_column_names>): To expose only those columns of the graph element table that are explicitly listed as label properties. The property name defaults to the column name.
 - PROPERTIES (<column_name AS property_name,...>): Same as the preceding option. However, if AS property_name is appended to the column_name, then property name is used as the property name.
 - PROPERTIES (<column_expressions AS property_name,...>): To declare a property which is an expression over columns. The AS clause is mandatory in this case. A value expression can either be a SQL operator expression defined over scalar data type columns or JSON expression. See JSON Support in SQL Property Graphs for an example using JSON expressions.
 - NO PROPERTIES: No columns are exposed for a label.
- Peudo-columns cannot be exposed as a label property.



4.1.4 Using Graph Options to Create SQL Property Graphs

You can use graph options to control the behavior of a SQL property graph at the time of its creation.

Graph options can be specified at the end of the CREATE PROPERTY GRAPH DDL statement using the OPTIONS clause. You can use either the MODE or MIXED PROPERTY TYPES option, or both as required.

Using an Option to Specify the Mode of the Graph

You can specify the MODE of the graph by using one of the following option values at the time of creating the SQL property graph:

- ENFORCED MODE: This ensures that there is a dependency to the unique key
 constraint on the underlying database tables. If used when creating a SQL
 property graph, the CREATE PROPERTY GRAPH statement will throw an error if any of
 the following conditions apply:
 - The specified vertex or edge table KEY for the graph element table is neither a PRIMARY KEY nor a UNIQUE key defined on NOT NULL columns.
 - There is no explicit vertex or edge table KEY defined for the graph element table and also the system is unable to automatically identify the default vertex or edge key, as there is no single PRIMARY KEY or a single UNIQUE key constraint on NOT NULL columns on the underlying database table.
 - For a specified edge source key and corresponding source vertex key or for a specified edge destination key and corresponding destination vertex key, there does not exist a corresponding FOREIGN KEY between the underlying tables.
 - An edge table has no explicit keys for the source or for the destination and the system is unable to implicitly infer the keys, as there is no single FOREIGN KEY constraint between the edge table and the referenced source (or destination) vertex table.

For example, consider the following t1 table in the database that does not have any primary key, unique key or a NOT NULL constraint.

```
SQL> CREATE TABLE t1 (id NUMBER, name VARCHAR2(10));

INSERT INTO t1 (id, name) VALUES (1, 'John');

INSERT INTO t1 (id, name) VALUES (2, 'Mary');
```

Create a SQL property graph using OPTIONS (ENFORCED MODE) as shown:

```
CREATE PROPERTY GRAPH g

VERTEX TABLES (

t1 KEY (id)

LABEL t PROPERTIES ARE ALL COLUMNS
) OPTIONS (ENFORCED MODE);
```

The graph creation fails with the following error as there are no key constraints to enforce uniqueness:

 ${\tt ORA-42434:}$ Columns used to define a graph element table key must be NOT NULL in ENFORCED MODE

If you omit the \mathtt{KEY} clause in the preceding graph definition, then the following error is thrown:

```
ORA-42402: cannot infer key for graph element table T1
```

• TRUSTED MODE (default): There is no dependency to the unique key constraint on the underlying database tables when using the TRUSTED mode. Therefore, the preceding example when run in TRUSTED mode will not throw any error. This implies that if you choose to use this option, then you must guarantee the uniqueness of primary keys on each of the graph element tables, as well as valid foreign key references between an edge table and its source and destination tables. Otherwise, your graph query results may be incorrect as the expected guarantees are not met.

Using an Option to Allow or Disallow Different Property Types for Shared Property Names

You can specify the MIXED PROPERTY TYPES options using one of the following values:

- ALLOW MIXED PROPERTY TYPES: This ensures that:
 - If two properties with the same name belong to different labels, then they can have completely different types.

For example, in addition to the sample tables persons and students (see Setting Up Sample Data in the Database), create the following additional table:

```
CREATE TABLE t2 (id NUMBER, name VARCHAR2(10), height VARCHAR2(4), CONSTRAINT t2_pk PRIMARY KEY (id));

INSERT INTO t2 (id, name, height) VALUES (1,'John', '1.80');
INSERT INTO t2 (id, name, height) VALUES (2,'Mary','1.65');
```

Run the following CREATE PROPERTY GRAPH DDL statement which uses three distinct labels for the same property name, height.

```
CREATE PROPERTY GRAPH g1

VERTEX TABLES (

persons

LABEL person PROPERTIES (name, height),

students

LABEL student PROPERTIES (height),

t2

LABEL t2 PROPERTIES (height)

) OPTIONS (ALLOW MIXED PROPERTY TYPES);
```

When the graph is created, the property type for height in the vertex tables associated with:

- * LABEL person is FLOAT
- * LABEL student is BINARY DOUBLE
- * LABEL t2 **is** VARCHAR



However, when querying this graph, the property type for height is dependent on the label constraint used in the GRAPH_TABLE query. See Accessing Label Properties for more information.

 If you are sharing property names inside shared labels, then they should be all union compatible types.

For example, run the following CREATE PROPERTY GRAPH DDL statement where the property name height is used inside the shared label t:

```
CREATE PROPERTY GRAPH g2

VERTEX TABLES (

persons

LABEL t PROPERTIES (height),

t2

LABEL t PROPERTIES (height)

) OPTIONS (ALLOW MIXED PROPERTY TYPES);
```

The graph creation fails as the column <code>height</code> in the tables <code>persons</code> and t2 has the data type <code>FLOAT</code> and <code>VARCHAR</code> respectively which are union incompatible. Therefore, the following error is thrown:

```
ORA-42414: cannot use mixed type for property HEIGHT of label T
```

However, the following graph will get created successfully as FLOAT and BINARY DOUBLE belong to the numeric group and are union compatible.

```
CREATE PROPERTY GRAPH g3

VERTEX TABLES (

persons

LABEL t PROPERTIES (height),

students

LABEL t PROPERTIES (height)

) OPTIONS (ALLOW MIXED PROPERTY TYPES);
```

See Type Compatibility Rules for Determining Property Types for more information.

• DISALLOW MIXED PROPERTY TYPES (default): This ensures that a property with the same name should strictly be the same data type. This applies to all labels irrespective of whether they are associated with a single or multiple graph element tables.

For example, run the following DDL statement using persons and t2 as the underlying database tables:

```
CREATE PROPERTY GRAPH g4

VERTEX TABLES (

persons

LABEL person PROPERTIES (name, height),

t2

LABEL t2 PROPERTIES (height)
);
```

The preceding code uses the default DISALLOW MIXED PROPERTY TYPES graph option and therefore throws an error as mixed property types are used in the graph definition:

ORA-42414: cannot use mixed type for property HEIGHT of label T2

The following table summarizes compatibility rules with respect to the MIXED PROPERTY TYPES options

Description	ALLOW	DISALLOW
Properties with the same name exposed by shared labels ¹	Union-compatible	Types must match
Shared properties ²	Any	Types must match

¹ A label with the same name can be associated with more than one graph element table.

4.1.5 Granting System and Object Privileges for SQL Property Graphs

Learn about the new system and object privileges for performing operations on SQL property graphs.

Table 4-1 System Privileges for SQL Property Graph Objects

System Privileges	Description
CREATE PROPERTY GRAPH	To create a SQL property graph in the grantee's schema
CREATE ANY PROPERTY GRAPH	To create a SQL property graph in any schema except ${\tt SYS}$ and ${\tt AUDSYS}$
ALTER PROPERTY GRAPH	To alter a SQL property graph in the grantee's schema
ALTER ANY PROPERTY GRAPH	To alter a SQL property graph in any schema except SYS and AUDSYS
READ PROPERTY GRAPH	To query a SQL property graph in the grantee's schema
READ ANY PROPERTY GRAPH	To query a SQL property graph in any schema except SYS and AUDSY
SELECT PROPERTY GRAPH	To query a SQL property graph in the grantee's schema
DROP ANY PROPERTY GRAPH	To drop a SQL property graph in any schema except SYS and AUDSYS

Table 4-2 Object Privileges for SQL Property Graphs

Object Privileges	Description
ALTER	To alter a SQL property graph
READ	To query a SQL property graph with a SQL graph query
¹ SELECT	To query a SQL property graph with a SQL graph query

Note that the SELECT privilege behaves exactly as the READ privilege for the SQL property graph object. It is mainly present for compatibility with the SQL standards for a property graph object.



² A property with the same name can be exposed by different labels.

The following shows the examples for granting and revoking the SQL property graph related privileges. Ensure you have SYSDBA access to grant and revoke these privileges:

```
GRANT CREATE PROPERTY GRAPH, CREATE ANY PROPERTY GRAPH,
ALTER ANY PROPERTY GRAPH, DROP ANY PROPERTY GRAPH,
READ ANY PROPERTY GRAPH TO <graphuser>;

REVOKE CREATE PROPERTY GRAPH, CREATE ANY PROPERTY GRAPH,
ALTER ANY PROPERTY GRAPH, DROP ANY PROPERTY GRAPH,
READ ANY PROPERTY GRAPH FROM <graphuser>;
```

You can share your SQL property graph in the database with another user as shown.

```
GRANT SELECT ON PROPERTY GRAPH <graph name> TO <schema user>;
```

4.1.6 Retrieving SQL Creation DDL Using the DBMS_METADATA Package

You can retrieve the creation DDL for you property graph schema object using the DBMS_METADATA package.

The following example displays the DDL for the graph created in Creating a SQL Property Graph using the DBMS METADATA package.

```
SQL> SELECT DBMS METADATA.GET DDL('PROPERTY GRAPH', 'STUDENTS GRAPH')
FROM DUAL:
 CREATE PROPERTY GRAPH "GRAPHUSER". "STUDENTS GRAPH"
 VERTEX TABLES (
   "GRAPHUSER". "FRIENDSHIPS" AS "FRIENDSHIPS" KEY ("FRIENDSHIP ID")
     LABEL ALL FRIENDS PROPERTIES ("FRIENDSHIP ID", "MEETING DATE")
     LABEL FRIEND OF NO PROPERTIES,
   "GRAPHUSER"."PERSONS" AS "PERSONS" KEY ("PERSON ID")
      LABEL PERSON PROPERTIES ("PERSON ID", "NAME", "BIRTHDATE" AS
"DOB")
     LABEL PERSON HT PROPERTIES ("HEIGHT"),
   "GRAPHUSER"."STUDENTS" AS "STUDENT" KEY ("S ID")
      PROPERTIES ("S ID", "S UNIV ID", "S PERSON ID", "SUBJECT"),
   "GRAPHUSER"."UNIVERSITY" AS "UNIVERSITY" KEY ("ID")
      PROPERTIES ("ID", "NAME") )
  EDGE TABLES (
   "GRAPHUSER"."FRIENDSHIPS" AS "FRIENDS" KEY ("FRIENDSHIP ID")
      SOURCE KEY("PERSON A") REFERENCES PERSONS ("PERSON ID")
      DESTINATION KEY("PERSON B") REFERENCES PERSONS ("PERSON ID")
     PROPERTIES ("FRIENDSHIP ID", "MEETING DATE"),
   "GRAPHUSER"."STUDENTS" AS "STUDENT OF" KEY ("S ID")
      SOURCE KEY("S PERSON ID") REFERENCES PERSONS ("PERSON ID")
      DESTINATION KEY("S UNIV ID") REFERENCES PERSONS ("ID")
```



PROPERTIES ("SUBJECT"))
OPTIONS (TRUSTED MODE, DISALLOW MIXED PROPERTY TYPES)

4.1.7 Limitations of Creating a SQL Property Graph

This section lists a few restrictions that apply when creating a SQL property graph.

- Hybrid partitioned tables, as well as views derived from these tables, cannot be used as graph element tables in a SQL property graph.
- Database links, as well as views defined using these links, cannot be used as graph element tables in a SQL property graph.
- Object tables (that is, table created with CREATE TABLE x OF myObjectType) and object views cannot be used as graph element tables in a SQL property graph.
- XMLType table (that is, table created with CREATE TABLE x OF XMLTYPE ...) cannot be used as graph element tables in a SQL property graph. However SQL/XML operators, XMLExists(), XMLCast(XMLQuery()) over XMLType column stored as binary XML to define property as SQL value expression is supported.
- Columns of type ANYTYPE cannot be exposed as properties or as keys for graph element tables.
- Pseudo-columns cannot be exposed as properties or as keys for graph element tables.
- Column expressions that comprise invocations to PL/SQL functions cannot be exposed
 as properties. Similarly, virtual columns defined over column expressions that comprise
 invocations to PL/SQL functions cannot be exposed as properties.
- SQL property graph are not editionable.
- A SQL property graph definition cannot be modified once the graph is created. However, you can redefine a SQL property graph using the OR REPLACE clause in the CREATE PROPERTY GRAPH DDL statement.
- SQL property graph creation is not supported in a shard catalog. However, you can create a property graph over sharded tables in the local shards.

4.2 Revalidating a SQL Property Graph

Using the ALTER PROPERTY GRAPH COMPILE DDL statement, you can revalidate an existing property graph schema object in the database.

A SQL property graph schema may become invalid due to the alteration of the underlying database objects. For instance, adding or dropping a column from the underlying database tables, used in the graph definition, can cause the graph to become invalid. Any invalidation of the graph will also invalidate cursors depending on the graph object. In such a case, you can recover your property graph from an **invalid** state as shown in the following example. Also, refer to Granting System and Object Privileges for SQL Property Graphs to ensure you have the required privilege to perform the ALTER PROPERTY GRAPH operation.

Example 4-2 Revalidating a SQL Property Graph

ALTER PROPERTY GRAPH students graph COMPILE;



See Also:

ALTER PROPERTY GRAPH in Oracle Database SQL Language Reference

4.3 Dropping a SQL Property Graph

Using the DROP PROPERTY GRAPH DDL statement, you can remove a property graph schema object in Oracle Database.

See Granting System and Object Privileges for SQL Property Graphs to ensure you have the required privilege to drop a SQL property graph.

Example 4-3 Dropping a SQL Property Graph

The following example removes the SQL property graph, students_graph, in the database.

DROP PROPERTY GRAPH students graph;

Similar to database views, dropping a property graph schema object does not remove the underlying database tables.

See Also:

DROP PROPERTY GRAPH in Oracle Database SQL Language Reference

4.4 JSON Support in SQL Property Graphs

When creating a SQL property graph, you can define a label property over a JSON data type column using simplified dot notation. You can later access this property inside the <code>GRAPH_TABLE</code> query.

The label property defined over a JSON data type column can be of common SQL scalar data types, such as:

- VARCHAR
- NUMBER
- BINARY FLOAT
- BINARY DOUBLE
- DATE
- TIMESTAMP
- raw JSON data converted to a SQL data type
 via .string(), .number(), .float(), .double(), .date(), .timestamp(), .binar
 y() or their equivalent using the JSON VALUE operator

Therefore, you can use either a JSON dot notation or the JSON_VALUE operator to select a scalar value in the JSON data to define a SQL property graph label property.

This also applies when accessing a label property defined over the JSON data type column inside a <code>GRAPH_TABLE</code> query.

Example 4-4 Defining a SQL Property Graph Using JSON Dot Notation and JSON Expressions for Label Properties

The following example creates a SQL property graph that contains label properties defined over a JSON data type column. The graph is created using the sample database tables (persons and friendships) defined in Setting Up Sample Data in the Database. The example uses both the JSON dot notation and the JSON_VALUE expression to define the label property.

```
CREATE PROPERTY GRAPH friends_graph

VERTEX TABLES (

persons AS p KEY (person_id)

LABEL person

PROPERTIES (name, birthdate AS dob,

p.person_data.department.string() AS "works_in",

JSON_VALUE(person_data, '$.role') AS "works_as")
)

EDGE TABLES (

friendships AS friends

KEY (friendship_id)

SOURCE KEY (person_a) REFERENCES p(person_id)

DESTINATION KEY (person_b) REFERENCES p(person_id)

PROPERTIES (meeting_date)
);
```

The graph gets created successfully and you can query the graph as shown in the following example:

Example 4-5 Querying a SQL Property Graph and Accessing Label Properties Defined As SQL/JSON Expressions

The following example queries the SQL property graph created in the preceding example to access the label properties created over a JSON data type column.

The guery produces the following output:



Example 4-6 Creating and Querying a SQL Property Graph with JSON Data Type Label Property

The following example creates a SQL property graph with JSON data type label property:

```
CREATE PROPERTY GRAPH friends_graph_new
  VERTEX TABLES (
        persons AS p KEY (person_id)
        LABEL person
        PROPERTIES (name, birthdate AS dob, p.person_data AS
"p_data")
)
EDGE TABLES (
    friendships AS friends
        KEY (friendship_id)
        SOURCE KEY (person_a) REFERENCES p(person_id)
        DESTINATION KEY (person_b) REFERENCES p(person_id)
        PROPERTIES (meeting_date)
);
```

You can then query the graph using a JSON VALUE expression as shown:

Technical Consultant 10-JUL-01 Mary



Bob IT

SQL GRAPH_TABLE Queries

You can query a SQL property graph using the GRAPH_TABLE operator to express graph pattern matching queries.

Graph pattern matching allows you to define a path pattern and match it against a graph to obtain a set of solutions. You must provide the graph to be queried as an input to the GRAPH_TABLE operator along with the MATCH clause containing the graph patterns to be searched as shown:

```
SELECT * FROM GRAPH_TABLE (students_graph
   MATCH
   (a IS person) -[e IS friends]-> (b IS person WHERE b.name = 'Mary')
   WHERE a.name='John'
   COLUMNS (a.name AS person_a, b.name AS person_b)
);
```

A basic Graph table query is made up of the following components:

- FROM clause: It includes the GRAPH_TABLE operator which takes the input graph name as the first parameter.
- MATCH clause: It expresses the graph element patterns (vertex or edge pattern) to be searched on the SQL property graph. It can optionally include an element pattern WHERE clause as seen in the preceding example ((b IS person WHERE b.name = 'Mary')) query. This in-line WHERE clause can access any matched variable.
- WHERE clause: This is an optional out-of-line WHERE clause. Similar to the element pattern
 WHERE clause, it has access to all the graph pattern variables and expresses a predicate
 that applies to the entire pattern in the MATCH clause.
- **COLUMNS clause:** This contains the query output columns.

See Also:

GRAPH_TABLE Operator in Oracle Database SQL Language Reference

The following sections explain GRAPH TABLE queries in detail:

- About Graph Patterns
 A SQL GRAPH_TABLE query is composed of graph patterns.
- Variable-Length Path Patterns
 Variable-length graph patterns provide advanced querying support for SQL property graphs.
- Complex Path Patterns
 You can query a SQL property graph using complex path patterns.

Vertex and Edge Identifiers

timestamp value.

You can uniquely identify each vertex and edge in a SQL property graph with the VERTEX ID and EDGE ID operators, respectively, in a GRAPH TABLE query.

Running GRAPH_TABLE Queries at a Specific SCN
 You can run a GRAPH TABLE query at a given System Change Number (SCN) or

Privileges to Query a SQL Property Graph
 You must have the READ or SELECT object privilege to guery a SQL property graph.

• Examples for SQL Property Graph Queries

This section contains a few examples for querying a SQL property graph with fixed-length and variable-length graph pattern matching queries.

- Supported Features and Limitations for Querying a SQL Property Graph
 This section provides the list of supported and unsupported features for querying a SQL Property Graph.
- Tuning SQL Property Graph Queries
 You can tune a SQL GRAPH TABLE query using the EXPLAIN PLAN statement.
- Type Compatibility Rules for Determining Property Types
 When using shared property names that are union compatible, the property type is determined by certain type compatibility rules.

5.1 About Graph Patterns

A SQL GRAPH TABLE query is composed of graph patterns.

Graph patterns are expressed between the input graph name and the COLUMNS clause in a GRAPH-TABLE query.

Graph patterns are made up of one or more vertex and edge patterns. For example, the following graph pattern has two vertex patterns and one edge pattern:

$$(v1) - [e] -> (v2)$$

A vertex pattern is enclosed in parentheses and specifies how to match a single vertex. An edge pattern is enclosed by a square bracket with delimiters on the left and right side of the edge pattern and specifies how to match a single edge.

Also, the available arrow tokens for edge patterns are summarized in the following table:

Table 5-1 Arrow Tokens for Edge Patterns

Directionality	Bracketed Syntax	Abbreviated Syntax ¹	
Directed to the right	-[]->	->	
Directed to the left	<-[]-	->	
Any directed edge (right or left)	<-[]-> or -[]-	-	

- There are no brackets for the arrows in the "abbreviated syntax" column.
 - All edge labels will be considered as no edge label is specified. Hence, filtering on a specific edge is not supported.



A graph element pattern (which can either be a vertex or an edge pattern) may in turn optionally include:

- An element variable.
- A label expression which is that part in an element pattern that starts with the keyword IS
 and is followed by a list of one or more label names. If there is more than one label name,
 then these are separated by vertical bars.
- An element pattern where clause which expresses a search condition on the element variable declared by the element pattern.

The following sections explain the graph pattern concepts more in detail:

- Graph Element Variables
 Vertex and edge pattern variables ranges over vertices and edges respectively.
- Label Expressions
 A label expression in a vertex or an edge element pattern is introduced by the keyword
 TS
- Accessing Label Properties
 You can access a property inside a graph element pattern, in the out-of-line WHERE clause
 or in the COLUMNS clause.

5.1.1 Graph Element Variables

Vertex and edge pattern variables ranges over vertices and edges respectively.

For example, consider the following graph pattern which contains three graph element variables.

```
(v1) - [e] -> (v2)
```

In the preceding graph pattern, v1 and v2 are two vertex pattern variables and e is an edge pattern variable.

Ensure that you apply the following rules for the graph pattern variables:

- You cannot use the same variable name for both a vertex and an edge.
- You can use the same variable name in two different vertex patterns as shown:
 MATCH (a IS person) -> (a IS person)

In the preceding example, the vertex variable a is used in two vertex patterns - (a IS person) and (a IS person). This implies that the two vertex patterns that declare the same vertex variable must bind to the same vertex. Thus the vertex variable binds to a unique vertex but the vertex pattern can appear multiple times in the same graph pattern.

- You can use the same variable name in two different edge patterns.
- Anonymous (that is, omitted) vertex and edge variables are supported. See Example 5-8.

5.1.2 Label Expressions

A label expression in a vertex or an edge element pattern is introduced by the keyword IS.

For example, in the following graph pattern, the vertex pattern associated with the graph element variable v1 has the label person. Also, the edge pattern associated with the graph element variable e contains the label friendof:



(v1 IS person)-[e IS friendOf]->(v2)

If the label is omitted in a graph element pattern, then the default is to query all vertices or edges.

A label expression can also include an optional in-line SQL search condition that can access any matched variable. When accessing a property, you must specify a graph pattern variable.

The supported vertex and edge label expressions are described in the following table:

Table 5-2 Supported Vertex and Edge Label Expressions

Vertex Label Expression	Edge Label Expression	Description
(a)	[e]	 The vertex graph pattern variable a may match a vertex with any label. The edge graph pattern variable e may match an edge with any label.
()	[]	 The vertex pattern has no label and can match any vertex. The edge pattern has no label and can match any edge. When a graph pattern variable is not specified, a unique vertex or edge variable name is internally generated by the system. Therefore, you cannot reference the vertex or edge elsewhere in the query, as it is unknown.
(IS person)	[IS friend_of]	 The vertex pattern has only the person label. The edge pattern has only the friend_of label. When a graph pattern variable is not specified, a unique vertex or edge variable name is internally generated by the system. Therefore, you cannot reference the vertex or edge elsewhere in the query, as it is unknown.
(IS person place thing)	[IS friend_of student_of]	 The vertex pattern has an alternation of three labels, person, place and thing. This implies that the vertex pattern can match any vertex having those labels. The edge pattern has an alternation of two labels, friend_of and student_of. This implies that the edge pattern can match any edge having those labels. As there is no explicit graph pattern variable in the vertex or edge pattern, you cannot reference this vertex or edge elsewhere in the query.



Table 5-2 (Cont.) Supported Vertex and Edge Label Expressions

Vertex Label Expression	Edge Label Expression	Description	
(a IS person place thing)	<pre>[e IS friend_of student_of]</pre>	Same as the preceding table entry. However, the vertex and edge patterns contain a and e as vertex and edge grap pattern variables respectively. Therefore you can reference the vertex or edge using the respective graph pattern variables elsewhere in the query. See Example 5-12 which describes a GRAPH_TABLE query that uses label disjunction in the vertex pattern.	
(a IS person), (a IS car)	(a)-[e IS L1]->(b), (a)-[e is L2]->(b)	 The vertex pattern a IS person implies that a must match vertices having the label person, and the vertex pattern a IS car implies that a must match vertices having the label car. Therefore, this represents that a must match vertices having both person and car as labels, effectively an AND of these two conditions. Also, you can reference a vertex as a elsewhere in the query. The edge pattern e IS L1 implies that e must match edges having the label L1, and the edge pattern e IS L2 implies that e must match edges having the label L2. Therefore, this represents that e must match edges having both L1 and L2 as labels, effectively an AND of these two conditions. Also, you can reference an edge as e elsewhere in the query. See Example 5-13 which describes a GRAPH_TABLE query that uses conjunction of labels in the vertex pattern. 	
<pre>(a IS person WHERE a.name = 'Fred')</pre>	<pre>[e IS student_of WHERE e.subject = 'Arts']</pre>	 The vertex pattern has a label person and a vertex graph pattern variable a, which is qualified in the element pattern WHERE clause. The edge pattern has a label student_of and an edge graph pattern variable e, which is qualified in the element pattern WHERE clause. The only graph pattern variable that is visible within an element pattern is the graph pattern variable defined locally by the element pattern. Graph pattern variables from another element patterns cannot be accessed. See Example 5-5. 	



5.1.3 Accessing Label Properties

You can access a property inside a graph element pattern, in the out-of-line where clause or in the COLUMNS clause.

Consider the following graph element pattern where a is a graph element variable and name is a property name:

```
(a IS person WHERE a.name='John')
```

You can then reference the property in the WHERE clause inside the graph element pattern as a.name. This means a.name references the property name of the graph element bound to the graph pattern variable a.

Also, the following conditions apply when accessing a property:

- The property name is part of at least one table that satisfies the label expression.
- A graph variable name must always be used to access a property.
- At the time of the GRAPH_TABLE query compilation, certain type checking rules
 apply for the vertex or edge table properties. See Type Compatibility Rules for
 Determining Property Types for more information.

The following examples describe a few scenarios for determining property types when querying SQL property graphs. Note that Example 5-1 to Example 5-3 refer to the SQL property graph definition for g1 which contains height as a shared property across different labels.

Example 5-1 Determining the Property Type for a Single Label

The data type for a height in the following query is FLOAT:

```
SELECT * FROM GRAPH_TABLE (g1
MATCH
  (a IS person)
COLUMNS (a.height)
);
```

The query output is as shown:

```
1.8
1.65
1.75
```



Example 5-2 Determining Union Compatible Property Type for Two Different Labels

The data type for a.height in the following query is the union compatible type between FLOAT and BINARY DOUBLE:

```
SELECT * FROM GRAPH_TABLE (g1
MATCH
(a IS person|student)
COLUMNS (a.height)
);
```

The query output is as shown:

```
HEIGHT
-----
1.8E+000
1.65E+000
1.75E+000
1.7E+000
1.8E+000
1.65E+000
1.75E+000
```

In the SQL property graph g1, the property type for height associated with the labels person and student is FLOAT and BINARY_DOUBLE respectively. BINARY_DOUBLE takes precedence over FLOAT and hence the resulting output property type for a.height is BINARY_DOUBLE.

Example 5-3 No Union Compatible Property Type for Two Different Labels

Error is thrown for the following query as the data type for a.height is not union compatible across the tables, person (FLOAT) and t2 (VARCHAR):

```
SELECT * FROM GRAPH_TABLE (g1
   MATCH
    (a IS person|t2)
   COLUMNS (a.height)
);
```

On execution. the preceding query throws the error - ORA-01790: expression must have same datatype as corresponding expression

Example 5-4 Determining Union Compatible Property Type for Shared Labels

Consider the SQL property graph definition for g3 which uses a shared label (t) that is associated with a shared property name (height).

When querying g3, the data type for a height in the following GRAPH_TABLE query is BINARY DOUBLE:

```
SELECT * FROM GRAPH_TABLE (g3 MATCH (a IS t)
```



```
COLUMNS (a.height)
);
```

The query output is a union of the property columns across all the graph element tables sharing the label. Also, the property type is BINARY_DOUBLE as per the Type Compatibility Rules for Determining Property Types:

HEIGHT ----1.8E+000 1.65E+000 1.75E+000 1.7E+000 1.8E+000 1.65E+000 1.75E+000

5.2 Variable-Length Path Patterns

Variable-length graph patterns provide advanced querying support for SQL property graphs.

Variable-length graph patterns require recursion such that there is a variable number of joins when translated into a relational query.

Bounded recursive path patterns that include one or more of the following quantifiers are supported:

Table 5-3 Quantifier Support for Variable-Length Graph Patterns

Quantifier	Description
{n}	Exactly n
{n, m}	Between n and m (inclusive)
{, m}	Between 0 and m (inclusive)
?	0 or 1

Note that the maximum upper bound limit for the quantifiers in the preceding table is

See Example 5-14 for sample GRAPH_TABLE queries using the quantifiers described in the preceding table.

5.3 Complex Path Patterns

You can query a SQL property graph using complex path patterns.

Cyclic Path Patterns

Vertex and edge path patterns can form cycles. For instance, consider the following graph pattern:

```
MATCH (a IS person) -[IS friends]-> (a IS person)
```

The preceding graph pattern describes a single path pattern, and it contains the vertex variable a twice. Thus, this finds cycles in the graph such that a binds to a person that has a friends edge to itself.

Also, note the following:

- The label person for the vertex variable a need not be repeated twice. The result is the same with or without repeating the label expression.
- You can use multiple in-line WHERE clauses to add conditions on the same pattern variable.
- Using the same edge variable twice in a path pattern also has the semantics that the edges must be the same.

Cycles can be longer than a single edge. See Example 5-11.

Multiple Path Patterns

A MATCH clause may have more than one path pattern, in a comma-separated list. For instance, the following example shows two path patterns:

```
MATCH (a IS person WHERE a.name='John') -[IS student_of]-> (b IS university),
  (a IS person WHERE a.name='John') -[IS friends]-> (c IS person)
```

Any graph pattern variables in common between two path patterns denotes an overlap between the path patterns. In the preceding example, the vertex variable a is shared. Note that the variable a must bind to the same graph element table in each element pattern of the graph pattern, and thus there is an implicit natural inner join on such repeated graph pattern variables.

If there are no shared variables between the two path patterns, then the resulting output set is a cross product of the outputs of the individual path patterns. See Example 5-9 and Example 5-10.

5.4 Vertex and Edge Identifiers

You can uniquely identify each vertex and edge in a SQL property graph with the <code>VERTEX_ID</code> and <code>EDGE_ID</code> operators, respectively, in a <code>GRAPH_TABLE</code> query.

Graph element identifiers are based on the key value defined for the graph element tables. Therefore, it is important to note the following:

- Graphs in TRUSTED mode may produce duplicate identifiers for different vertices if some key columns do not have a UNIQUE constraint.
- Graphs in ENFORCED mode are guaranteed to always produce unique identifiers.

The <code>VERTEX_ID</code> and <code>EDGE_ID</code> operators can be used in any expression appearing in the <code>COLUMNS</code> or <code>WHERE</code> clause in a <code>GRAPH TABLE</code> query.



Note:

In order to use the <code>VERTEX_ID</code> and <code>EDGE_ID</code> operators, you must ensure that you have the <code>READ</code> or <code>SELECT</code> privilege on both the property graph object and its underlying database tables.

The input to the <code>VERTEX_ID</code> operator is a single vertex graph pattern variable coming from a matched vertex pattern as shown:

```
MATCH (v) COLUMNS (VERTEX ID (v) AS v id)
```

Similarly, the $EDGE_ID$ operator takes as input a single edge graph pattern variable coming from a matched edge pattern as shown:

```
MATCH (v1)-[e]->(v2) COLUMNS (EDGE ID (e) AS e id)
```

The output of these operators is a vertex or an edge identifier of JSON data type. The following shows an example of a JSON output describing the vertex identifier:

```
{
  "GRAPH_OWNER": "GRAPHUSER",
  "GRAPH_NAME": "STUDENTS_GRAPH",
  "ELEM_TABLE": "PERSONS",
  "KEY_VALUE": {
     "PERSON_ID": 1
  }
}
```

In the preceding JSON output:

- GRAPH OWNER: Owner of the property graph object
- GRAPH NAME: Name of the property graph object
- ELEM TABLE: Name of the vertex table
- KEY VALUE: Name and value of the key column

The same list of JSON output fields apply to an edge identifier also. However, the ${\tt ELEM_TABLE}$ field represents the name of an edge table. Also, all operations that can be performed on a JSON data type can be performed on the vertex and edge identifiers.

See Example 5-19 for more information.

VERTEX EQUAL and EDGE EQUAL Predicates

The <code>VERTEX_EQUAL</code> and <code>EDGE_EQUAL</code> predicates can be used to, respectively, compare two vertex and edge identifiers and return <code>TRUE</code> if they are equal.

The inputs to the <code>VERTEX_EQUAL</code> predicate are two vertex graph pattern variables. Similarly for <code>EDGE_EQUAL</code>, both inputs must be edge graph pattern variables. These predicates can be used in the <code>WHERE</code> clause in a <code>GRAPH TABLE</code> query.

See Example 5-20 for more information.



5.5 Running GRAPH_TABLE Queries at a Specific SCN

You can run a GRAPH_TABLE query at a given System Change Number (SCN) or timestamp value.

The graph name, which is the first operand of a GRAPH_TABLE query, can be associated with either of the following clauses:

- AS OF SCN: Refer to Example 5-17
- AS OF TIMESTAMP: Refer to Example 5-18

5.6 Privileges to Query a SQL Property Graph

You must have the READ or SELECT object privilege to query a SQL property graph.

If you are the graph creator, then you can allow other graph users to query your graph by granting any one of the following privileges:

```
GRANT READ ON PROPERTY GRAPH  graph_name> TO <schema_user>;
GRANT SELECT ON PROPERTY GRAPH  graph_name> TO <schema_user>;
```

It is important to note that granting the preceding privileges allows access only to the property graph object and not to its underlying database tables.

This allows the graph user to successfully run <code>GRAPH_TABLE</code> queries on your graph without having access to the underlying tables. For example:

```
GRANT READ ON PROPERTY GRAPH students_graph TO hr;

SQL> conn hr/<password_for_hr>;
Connected.

SQL> SELECT * FROM GRAPH_TABLE (graphuser.students_graph MATCH (a IS person)
COLUMNS (a.name AS person_a));

PERSON_A
______
John
Mary
Bob
Alice
```

However, to perform <code>GRAPH_TABLE</code> queries with <code>VERTEX_ID</code> and <code>EDGE_ID</code> operators, the graph user must additionally have <code>READ</code> or <code>SELECT</code> privilege on the underlying database tables.

5.7 Examples for SQL Property Graph Queries

This section contains a few examples for querying a SQL property graph with fixed-length and variable-length graph pattern matching queries.

All the queries shown in the examples are run on the SQL property graph, students_graph, created in Example 4-1:

Example 5-5 GRAPH TABLE Query Using An Edge Pattern Directed Left-To-Right

The following example shows a GRAPH_TABLE query containing an edge pattern (-[e IS friends]->) which is directed from left-to-right:

```
SELECT * FROM GRAPH_TABLE (students_graph
   MATCH
   (a IS person) -[e IS friends]-> (b IS person WHERE b.name='Alice')
   WHERE a.name='Mary'
   COLUMNS (a.name AS person_a, b.name AS person_b)
);
```

The code produces the following output:

Example 5-6 GRAPH TABLE Query Using An Edge Pattern Directed Right-To-Left

The following example shows a GRAPH_TABLE query containing an edge pattern (<- [e IS friends] -) which is directed from right-to-left:

```
SELECT * FROM GRAPH_TABLE (students_graph
   MATCH
   (a IS person) <-[e IS friends]- (b IS person WHERE b.name='Mary')
   WHERE a.name='Alice'
   COLUMNS (a.name AS person_a, b.name AS person_b)
);</pre>
```

The code produces the following output:

Example 5-7 GRAPH TABLE Query Using Any-Directed Edge Pattern

The following example shows a GRAPH_TABLE query which contains any-directed edge pattern (-[e IS friends]-):

```
SELECT * FROM GRAPH_TABLE (students_graph
   MATCH
   (a IS person) -[e IS friends] - (b IS person WHERE b.name='Alice' OR
b.name='Mary')
   WHERE (a.name='Alice' OR a.name='Mary')
   COLUMNS (a.name AS person_a, b.name AS person_b)
);
```



The code produces the following output:

Example 5-8 GRAPH TABLE Query Using an Anonymous Edge Variable

The following example shows a GRAPH_TABLE query where the edge element variable is omitted:

```
SELECT * FROM GRAPH_TABLE (students_graph
  MATCH
  (a IS person) -[]-> (b IS person)
  COLUMNS (a.name AS person_a, b.name AS person_b)
);
```

Alternatively, you can replace the bracketed syntax for the edge pattern (-[]->) in the preceding query with an abbreviated syntax ->.

The code produces the following output:

Example 5-9 GRAPH TABLE Query Using Multiple Path Patterns

The following example shows a GRAPH_TABLE query containing two path patterns (a) \rightarrow (b), (a) \rightarrow (c)) which have a common vertex as shown:

```
SELECT * FROM GRAPH_TABLE (students_graph
   MATCH
   (a IS person WHERE a.name = 'John') -> (b IS person), (a IS person WHERE
a.name = 'John') -> (c IS university)
   COLUMNS (a.name AS person_a, b.name AS person_b,c.name as university)
);
```

The preceding code produces the following output:



Example 5-10 GRAPH TABLE Query Using Disjoint Path Patterns

The following example shows a GRAPH_TABLE query containing two disjoint path patterns:

```
SELECT * FROM GRAPH_TABLE (students_graph
MATCH (a IS person WHERE a.name='John') -[IS student_of]-> (b IS
university),
(x IS person) -[IS friends]-> (y IS person)
COLUMNS (a.name AS a, b.name as university, x.name AS x, y.name as y)
);
```

The resulting output is as shown:

A	UNIVERSITY	X	Υ
John	ABC	Mary	John
John	ABC	Bob	Mary
John	ABC	John	Bob
John	ABC	Mary	Alice

Example 5-11 GRAPH TABLE Query Using Cyclic Path Patterns

The following example uses a cyclic path pattern (MATCH (a)-[]->(b)-[]->(c)-[]->(a)) as shown. Note that the example uses the same vertex pattern variable name a (which is bound to person) twice. Thus, this finds cycles in the graph containing three edges that finally bind to a itself.

```
SELECT * FROM GRAPH_TABLE (students_graph
   MATCH
   (a IS person) -[IS friends]-> (b IS person) -[IS friends]->
   (c IS person) -[IS friends]-> (a)
   COLUMNS (a.name AS person_a, b.name AS person_b, c.name AS person_c)
);
```

The preceding code produces the following output:

PERSON_A	PERSON_B	PERSON_C
Bob	Mary	John
John	Bob	Mary
Mary	John	Bob

Example 5-12 GRAPH_TABLE Query Using Label Disjunction

The following example uses label disjunction in the vertex label expression:

```
SELECT * FROM GRAPH_TABLE (students_graph
MATCH
(a is person|university)
COLUMNS (a.name, a.dob)
);
```



The code produces the following output:

NAME	DOB
John	13-JUN-63
Mary	25-SEP-82
Bob	11-MAR-66
Alice	01-FEB-87
ABC	NULL
XYZ	NULL

6 rows selected.

Example 5-13 GRAPH TABLE Query Using Label Conjunction

The following example uses label conjunction in the vertex label expression:

```
SELECT * FROM GRAPH_TABLE (students_graph
MATCH
(a IS person), (a IS person_ht)
COLUMNS (a.name as name, a.dob as dob, a.height as height )
):
```

The code produces the following output:

NAME	DOB	HEIGHT
John	13-JUN-63	1.8
Mary	25-SEP-82	1.65
Bob	11-MAR-66	1.75
Alice	01-FEB-87	1.7

Example 5-14 GRAPH_TABLE Queries Using Recursive Path Patterns with Bounded Quantifiers

The following example uses a recursive path pattern to retrieve all friends within two hops:

```
SELECT * FROM GRAPH_TABLE (students_graph
MATCH (a is person WHERE a.name='Mary') -[is friends]->{2} (b is person)
COLUMNS (a.name AS a , b.name AS b)
);
```

The preceding code produces the following output:

```
A B -----
Mary Bob
```



The following example uses a recursive path pattern to retrieve all friends between one and two hops (inclusive):

```
SELECT * FROM GRAPH_TABLE (students_graph
MATCH (a is person WHERE a.name='Mary') -[is friends]->{1, 2} (b is
person)
COLUMNS (a.name AS a , b.name AS b)
);
```

The preceding code produces the following output:

```
A B
-----
Mary Alice
Mary John
Mary Bob
```

The following example uses a recursive path pattern to retrieve all friends by performing from zero to two iterations:

```
SELECT * FROM GRAPH_TABLE (students_graph
MATCH (a is person WHERE a.name='Mary') -[is friends]->{,2} (b is
person)
COLUMNS (a.name AS a , b.name AS b)
);
```

The preceding code produces the following output:

A	В
Mary	Mary
Mary	Alice
Mary	John
Marv	Bob

Note that in the first line of the preceding output, Mary is bound to both the element pattern variables, a and b. This is because the $GRAPH_TABLE$ query includes a zero hop iteration and therefore, the vertex pattern to the left and the vertex pattern to the right must bind to the same graph element.

Example 5-15 GRAPH TABLE Query Using Bind Variables

The example declares a bind variable, name and assigns a value as shown:

```
SQL> variable name VARCHAR2(10);
SQL> BEGIN
   2 :name := 'Bob';
   3 END;
   4 /
```

PL/SQL procedure successfully completed.



Using this bind variable, the following GRAPH TABLE query is performed:

The code produces the following output:

```
A B MET_ON
-----
John Bob 01-SEP-00
```

Example 5-16 GRAPH_TABLE Query Invoking a PL/SQL function Inside an Expression and in the COLUMNS Clause

The example declares a user defined function(UDF) as shown:

```
CREATE OR REPLACE FUNCTION get_age(
    id NUMBER
)
RETURN NUMBER
AS
    age NUMBER := 0;
BEGIN
    -- get age
        SELECT (EXTRACT(YEAR from SYSDATE) - EXTRACT(YEAR from birthdate))
        INTO age
        FROM persons
        WHERE person_id=id;
        -- return age
        RETURN age;
END;
/
Function created.
```

The following GRAPH_TABLE query invokes the UDF inside an expression in the WHERE clause and again in the COLUMNS clause:



```
e.meeting_date AS met_on)
);
```

The code produces the following output:

A	AGE	В	MET_ON
Bob	56	Mary	10-JUL-01
John	59	Bob	01-SEP-00

Example 5-17 GRAPH_TABLE Query Using SCN

Determine the current SCN value of the database as shown:

```
SQL> SELECT TIMESTAMP_TO_SCN(SYSDATE) FROM DUAL;

TIMESTAMP_TO_SCN(SYSDATE)

2117789
```

The following GRAPH TABLE query using the preceding SCN value as shown:

```
SELECT * FROM GRAPH_TABLE (students_graph AS OF SCN 2117789
   MATCH
   (a IS person) -[e]-> (b IS person)
   COLUMNS (a.name AS a, b.name AS b, e.meeting_date AS met_on)
);
```

The query produces the following output:

A	В	MET_ON
Mary	John	19-SEP-00
Bob	Mary	10-JUL-01
John	Bob	01-SEP-00
Mary	Alice	19-SEP-00

Example 5-18 GRAPH TABLE Query Using TIMESTAMP

The following GRAPH TABLE query uses a TIMESTAMP value as shown:

```
SQL> SELECT * FROM GRAPH_TABLE (students_graph AS OF TIMESTAMP
SYSTIMESTAMP
  MATCH
  (a IS person WHERE a.name='John') -[e]-> (b IS person)
COLUMNS (a.name AS a, b.name AS b, e.meeting_date AS met_on)
);
```



The query produces the following output:

```
A B MET_ON
-----
John Bob 01-SEP-00
```

Example 5-19 GRAPH_TABLE Query Using the VERTEX_ID and EDGE_ID Identifiers

```
SELECT * FROM GRAPH_TABLE (students_graph
MATCH
  (a IS person ) -[e IS friends]-> (b IS person)
COLUMNS (JSON_SERIALIZE(VERTEX_ID(a)) AS id_a , JSON_SERIALIZE(EDGE_ID(e))
AS id_e)
);
```

The query produces a JSON data type output that includes the graph owner, graph name and graph element table name and the key value as shown:

```
ID A
{"GRAPH OWNER": "GRAPHUSER", {"GRAPH OWNER": "GRAPHUSER", "GR
"GRAPH NAME": "STUDENTS GRAP APH NAME": "STUDENTS GRAPH", "EL
H", "ELEM TABLE": "PERSONS", " EM TABLE": "FRIENDS", "KEY VALUE
KEY VALUE":{"PERSON ID":1}} ":{"FRIENDSHIP ID":1}}
{"GRAPH OWNER": "GRAPHUSER", {"GRAPH OWNER": "GRAPHUSER", "GR
"GRAPH NAME": "STUDENTS GRAP APH NAME": "STUDENTS GRAPH", "EL
H", "ELEM TABLE": "PERSONS", " EM TABLE": "FRIENDS", "KEY VALUE
KEY VALUE":{"PERSON ID":2}} ":{"FRIENDSHIP ID":2}}
{"GRAPH OWNER": "GRAPHUSER", {"GRAPH OWNER": "GRAPHUSER", "GR
"GRAPH NAME": "STUDENTS GRAP APH NAME": "STUDENTS GRAPH", "EL
H", "ELEM TABLE": "PERSONS", " EM TABLE": "FRIENDS", "KEY VALUE
KEY VALUE":{"PERSON ID":2}} ":{"FRIENDSHIP ID":3}}
{"GRAPH OWNER": "GRAPHUSER", {"GRAPH OWNER": "GRAPHUSER", "GR
"GRAPH NAME": "STUDENTS GRAP APH NAME": "STUDENTS GRAPH", "EL
H", "ELEM TABLE": "PERSONS", " EM TABLE": "FRIENDS", "KEY VALUE
KEY VALUE":{"PERSON ID":3}} ":{"FRIENDSHIP ID":4}}
```

Example 5-20 GRAPH_TABLE Query Using the VERTEX_EQUAL Predicate

```
SELECT * FROM GRAPH_TABLE (students_graph
MATCH
  (a IS person WHERE a.name='John') -[e IS friends]->{,1} (b IS person)
WHERE VERTEX_EQUAL(a,b)
COLUMNS (JSON_SERIALIZE(VERTEX_ID(a)) AS id_a , JSON_SERIALIZE(VERTEX_ID(b))
AS id_b)
);
```



The query produces a JSON data type output that includes the graph owner, graph name and graph element table name and the key value as shown:

```
ID_B

{"GRAPH_OWNER":"GRAPHUSER", {"GRAPH_OWNER":"GRAPHUSER",
"GRAPH_NAME":"STUDENTS_GRAP "GRAPH_NAME":"STUDENTS_GRAP
H","ELEM_TABLE":"PERSONS"," H","ELEM_TABLE":"PERSONS","
KEY_VALUE":{"PERSON_ID":1}} KEY_VALUE":{"PERSON_ID":1}}
```

Setting Up Sample Data in the Database

5.7.1 Setting Up Sample Data in the Database

In order to create the SQL property graph, students_graph, shown in Creating a SQL Property Graph, the following sample tables with data need to be set up in the database.

- Connect to the database as the schema user.
- 2. Run the following SQL script to create the university, persons, students, and friendships tables with sample data in the database.

```
CREATE TABLE university (
   id NUMBER GENERATED ALWAYS AS IDENTITY (START WITH 1 INCREMENT
BY 1),
   name VARCHAR2(10),
   CONSTRAINT u pk PRIMARY KEY (id));
INSERT INTO university (name) VALUES ('ABC');
INSERT INTO university (name) VALUES ('XYZ');
CREATE TABLE persons (
    person id NUMBER GENERATED ALWAYS AS IDENTITY (START WITH 1
INCREMENT
    BY 1),
    name VARCHAR2(10),
    birthdate DATE,
    height FLOAT DEFAULT ON NULL 0,
    person data JSON,
    CONSTRAINT person pk PRIMARY KEY (person id)
   );
INSERT INTO persons (name, height, birthdate, person data)
       VALUES ('John', 1.80, to date('13/06/1963', 'DD/MM/YYYY'),
'{"department":"IT", "role": "Software Developer"}');
INSERT INTO persons (name, height, birthdate, person data)
       VALUES ('Mary', 1.65, to date('25/09/1982', 'DD/MM/YYYY'),
'{"department":"HR", "role":"HR Manager"}');
INSERT INTO persons (name, height, birthdate, person data)
       VALUES ('Bob', 1.75, to date('11/03/1966', 'DD/MM/YYYY'),
'{"department":"IT", "role": "Technical Consultant"}');
```



```
INSERT INTO persons (name, height, birthdate, person data)
       VALUES ('Alice', 1.70, to date('01/02/1987', 'DD/MM/YYYY'),
'{"department":"HR", "role":"HR Assistant"}');
CREATE TABLE students (
      s id NUMBER GENERATED ALWAYS AS IDENTITY (START WITH 1 INCREMENT BY
1),
      s univ id NUMBER,
      s person id NUMBER,
      subject VARCHAR2(10),
     height BINARY DOUBLE,
     CONSTRAINT stud pk PRIMARY KEY (s id),
     CONSTRAINT stud fk person FOREIGN KEY (s person id) REFERENCES
persons (person id),
     CONSTRAINT stud fk univ FOREIGN KEY (s univ id) REFERENCES
university(id)
   );
INSERT INTO students(s univ id, s person id, subject, height) VALUES
(1,1,'Arts',1.80);
INSERT INTO students(s univ id, s person id, subject, height) VALUES
(1,3,'Music',1.65);
INSERT INTO students(s univ id, s person id, subject, height) VALUES
(2,2,'Math',1.75);
INSERT INTO students(s univ id, s person id, subject, height) VALUES
(2,4,'Science',1.70);
CREATE TABLE friendships (
   friendship id NUMBER GENERATED ALWAYS AS IDENTITY (START WITH 1
INCREMENT BY 1),
   person a NUMBER,
   person b NUMBER,
   meeting date DATE,
   CONSTRAINT fk person a id FOREIGN KEY (person a) REFERENCES
persons (person id),
    CONSTRAINT fk person b id FOREIGN KEY (person b) REFERENCES
persons (person id),
   CONSTRAINT fs pk PRIMARY KEY (friendship id)
);
INSERT INTO friendships (person a, person b, meeting date) VALUES (1, 3,
to date('01/09/2000', 'DD/MM/YYYY'));
INSERT INTO friendships (person a, person b, meeting date) VALUES (2, 4,
to date('19/09/2000', 'DD/MM/YYYY'));
INSERT INTO friendships (person a, person b, meeting date) VALUES (2, 1,
to date('19/09/2000', 'DD/MM/YYYY'));
INSERT INTO friendships (person a, person b, meeting date) VALUES (3, 2,
to date('10/07/2001', 'DD/MM/YYYY'));
```



5.8 Supported Features and Limitations for Querying a SQL Property Graph

This section provides the list of supported and unsupported features for querying a SQL Property Graph.

Supported Features

- Single label, no label, label disjunction and label conjunction are supported in label expressions inside a graph pattern. For more information, see:
 - Table 5-2 in Label Expressions
 - Examples for SQL Property Graph Queries
- Any directed edge patterns (MATCH (a) [e] (b) are supported.
 See Example 5-7.
- Anonymous vertex (MATCH ()-[e]->()) and edge (MATCH (a)-[]->(b)) variables are supported.
 See Example 5-8.
- Complex path pattern queries are supported.
 See Example 5-9, Example 5-10 and Example 5-11.
- Bounded recursive path pattern queries are supported.
 See Example 5-14.
- Bind variables are supported inside a WHERE clause.
 See Example 5-15.
- VERTEX_ID and EDGE_ID operators that uniquely identify a vertex and an edge respectively can be used within a SQL GRAPH TABLE query.
 - See Vertex and Edge Identifiers.
 - See Example 5-19.
- VERTEX_EQUAL and EDGE_EQUAL predicates for matching vertex and edge identifiers are supported.
 - See Vertex and Edge Identifiers.
 - See Example 5-20.
- SQL and JSON expressions are supported inside WHERE and COLUMNS clauses. See Example 4-6.
- JSON simplified syntax is supported to access properties of type JSON.
 See Example 4-6.
- PL/SQL functions are supported inside a WHERE or COLUMNS clause.
 See Example 5-16.
- Single line and multi-line comments are supported within a graph query.
- All identifiers within a GRAPH_TABLE query, such as graph names, alias names, graph element pattern variable names, labels and property names follow the standard SQL rules about case sensitivity:
 - Identifiers within double quotes are case sensitive.



- Identifiers not enclosed in double quotes are implicitly converted to uppercase and enclosed in double quotes.
- SQL hints are supported inside and outside the GRAPH_TABLE query for tuning.
 See Tuning SQL Property Graph Queries for more information.
- You can query a graph defined in another schema if you have the required privileges.
 See Granting System and Object Privileges for SQL Property Graphs for more information.

Limitations

- Variable-length pattern matching goals (such as ANY, ALL, ALL SHORTEST, ANY CHEAPEST, and so on) are not supported.
- Path pattern variables (MATCH p = (n) [e] -> (m)) are not supported.
- Clauses such as COST and TOTAL COST are not supported.
- Inline subqueries and LATERAL inline views are not supported.
- SQL Macros are not supported.

5.9 Tuning SQL Property Graph Queries

You can tune a SQL GRAPH TABLE query using the EXPLAIN PLAN Statement.

A GRAPH_TABLE query is internally translated into equivalent SQL. You can therefore generate the EXPLAIN PLAN for the property graph query as shown:

```
SQL> EXPLAIN PLAN FOR SELECT * FROM GRAPH_TABLE (students_graph
MATCH (a is person)-[e is friends]-> (b is person)
COLUMNS (a.name AS a , b.name AS b)
);
Explained.
```

The EXPLAIN PLAN can be viewed as shown:

0 SELECT STATEMENT			4	264	10	(10)
00:00:01						
* 1 HASH JOIN			4	264	10	(10)
00:00:01						
* 2 HASH JOIN			4	184	7	(15)
00:00:01						
3 TABLE ACCESS FU	JLL PERSONS		4	80	3	(0)
00:00:01						
4 TABLE ACCESS FU	JLL FRIENDSHIPS	1	4	104	3	(0)
00:00:01						



You can tune the preceding <code>GRAPH_TABLE</code> query by using optimizer hints. For instance, the following example uses the <code>PARALLEL</code> hint and the hint usage can be seen in the following execution plan:

```
SQL> EXPLAIN PLAN FOR SELECT /*+ PARALLEL(4) */ * FROM GRAPH TABLE
(students graph
MATCH (a is person) - [e is friends] -> (b is person)
COLUMNS (a.name AS a , b.name AS b)
);
Explained.
SQL> SELECT * FROM TABLE(DBMS XPLAN.DISPLAY(format=>'ALL'));
Plan hash value: 1486901074
______
| Id | Operation
                     | Name | Rows | Bytes
| Cost (%CPU) | Time | TQ | IN-OUT | PQ Distrib |
4 | 264
                                           264
  4 (0) | 00:00:01 | Q1,00 | P->S | QC (RAND) |
 264
                                       4 |
                                           264
      NESTED LOOPS
  5 I
                                       4 |
                                           184
  3 (0) | 00:00:01 | Q1,00 | PCWP |
  6 | PX BLOCK ITERATOR |
       7 |
        TABLE ACCESS FULL | FRIENDSHIPS |
                                       4 |
                                           104
  2 (0) | 00:00:01 | Q1,00 | PCWP |
 8 | TABLE ACCESS BY INDEX ROWID| PERSONS |
                                       1 | 20
  0 (0) | 00:00:01 | Q1,00 | PCWP | |
      INDEX UNIQUE SCAN | PERSON PK |
 9 |
0 (0) | 00:00:01 | Q1,00 | PCWP |
|* 10 | INDEX UNIQUE SCAN | PERSON PK |
  0 (0) | 00:00:01 | Q1,00 | PCWP | |
```



```
| 11 | TABLE ACCESS BY INDEX ROWID | PERSONS
                                            1 | 20 |
  (0) | 00:00:01 | Q1,00 | PCWP |
  ._____
Query Block Name / Object Alias (identified by operation id):
  1 - SEL$B92C7F25
  7 - SEL$B92C7F25 / "E"@"SEL$213F43E5"
  8 - SEL$B92C7F25 / "A"@"SEL$213F43E5"
  9 - SEL$B92C7F25 / "A"@"SEL$213F43E5"
 10 - SEL$B92C7F25 / "B"@"SEL$213F43E5"
 11 - SEL$B92C7F25 / "B"@"SEL$213F43E5"
Hint Report (identified by operation id / Query Block Name / Object Alias):
Total hints for statement: 1
 0 - STATEMENT
PLAN TABLE OUTPUT
                     _____
        PARALLEL (4)
  - dynamic statistics used: dynamic sampling (level=2)
  - Degree of Parallelism is 4 because of hint
```

5.10 Type Compatibility Rules for Determining Property Types

When using shared property names that are union compatible, the property type is determined by certain type compatibility rules.

The following summarizes the rules for determining the type of a property for union compatible properties at the time of DDL creation and also during query compilation:

- If expressions exposed by a same property of a shared label are character data, then the data type of the property is determined as follows:
 - If all expressions are of data type CHAR of equal length, then the property has a data type CHAR of that length. If the expression are all of data type CHAR, but with different lengths, then the property type is VARCHAR2 with the length of the larger CHAR type.
 - If any, or all of the expressions are of data type VARCHAR2, then the property has data type VARCHAR2. The length of the VARCHAR2 is the maximum length size of the input columns.
- If expressions exposed by a same property of a shared label are numeric data, then the data type of the property is determined by numeric precedence:
 - If any expression exposed by a property is of data type BINARY DOUBLE, then the
 property has the data type BINARY DOUBLE.
 - If no expression defining the property are of data type BINARY DOUBLE, but any expression is of type BINARY FLOAT, then the property has data type BINARY FLOAT.
 - If all expressions defining the property are of data type NUMBER, then the property has data type NUMBER.

- If expressions exposed by a same property of a shared label are date and timestamp data, then the data type of the property is determined as follows:
 - If all expressions are of data type DATE, then property has data type DATE.
 - If any, or all of the expressions are of data type TIMESTAMP, then the property
 has data type TIMESTAMP.



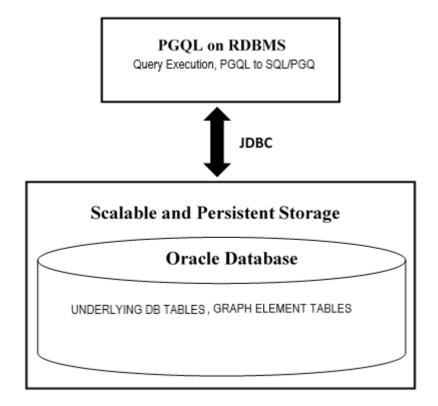
6

Executing PGQL Queries Against SQL Property Graphs

You can directly run PGQL queries against a SQL property graph in the database.

The PGQL query execution flow is shown in the following figure:

Figure 6-1 PGQL on SQL Property Graphs in Oracle Database



The basic execution flow is:

- The PGQL query is performed on a SQL property graph through a Java API.
- 2. The PGQL query is translated to SQL/PGQ (GRAPH TABLE query).
- 3. The translated SQL/PGQ is submitted to Oracle Database by JDBC.
- 4. The SQL/PGQ result set is wrapped as a PGQL result set and returned to the caller.

See Supported PGQL Features and Limitations for SQL Property Graphs for a complete list of supported and unsupported features.

- Executing PGQL SELECT Queries on a SQL Property Graph
 You can execute PGQL SELECT queries, on a SQL property graph, using the Java
 API in the oracle.pg.rdbms.pgql package.
- Supported PGQL Features and Limitations for SQL Property Graphs
 Learn about the supported PGQL features and limitations for SQL property
 graphs.

6.1 Executing PGQL SELECT Queries on a SQL Property Graph

You can execute PGQL SELECT queries, on a SQL property graph, using the Java API in the oracle.pg.rdbms.pgql package.

The following example shows a PGQL SELECT query execution:

- JShell
- Java
- Python

JShell

```
opg4j> var jdbcUrl="jdbc:oracle:thin:@<host name>:<port>/<db service>"
opq4j> var conn =
DriverManager.getConnection(jdbcUrl,"<username>","<password>")
opq4j> conn.setAutoCommit(false)
opg4j> var pgqlConn = PgqlConnection.getConnection(conn)
opg4j> var pgqlStmt = pgqlConn.createStatement()
opg4j> String query = "SELECT n.name FROM MATCH (n:person) ON
STUDENTS GRAPH"
opg4j> var rs = pgqlStmt.executeQuery(query)
opg4j> rs.print()
+----+
| NAME |
+----+
| John |
| Mary |
| Bob
| Alice |
+----+
```

Java



Python

```
>>> pgql_conn = opg4py.pgql.get_connection("<username>","<password>",
"<jdbcUrl>")
>>> pgql_statement = pgql_conn.create_statement()
>>> query = "SELECT n.name FROM MATCH (n:person) ON STUDENTS_GRAPH"
>>> rs = pgql_statement.execute_query(query)
>>> rs.print()
+-----+
| NAME |
+-----+
| John |
| Mary |
| Bob |
| Alice |
+------+
```

6.2 Supported PGQL Features and Limitations for SQL Property Graphs

Learn about the supported PGQL features and limitations for SQL property graphs.

The following table provides the complete list of supported and unsupported PGQL functionalities for SQL property graphs:

Table 6-1 Supported PGQL Functionalities and Limitations for SQL Property Graphs

Features	PGQL on SQL Property Graphs
CREATE PROPERTY GRAPH	Supported
DROP PROPERTY GRAPH	Supported
Fixed-length pattern matching	Supported
Variable-length pattern matching goals	Not Supported
Variable-length pattern matching quantifiers	Not Supported
Variable-length path unnesting	Not Supported
GROUP BY	Supported
HAVING	Supported
Aggregations	Supported: COUNT MIN, MAX, AVG, SUM LISTAGG Not supported: ARRAY_AGG



Table 6-1 (Cont.) Supported PGQL Functionalities and Limitations for SQL Property Graphs

Features	PGQL on SQL Property Graphs			
DISTINCT	Supported			
• SELECT DISTINCT				
 Aggregation with DISTINCT (such as, COUNT (DISTINCT e.prop)) 				
SELECT v.*	Not Supported			
ORDER BY (+ASC/DESC), LIMIT, OFFSET	Supported			
Data Types	All available Oracle RDBMS data types supported			
JSON	Supported: JSON storage: JSON strings (VARCHAR2) JSON objects JSON functions: Any JSON function call that follows the syntax, json_function_name (arg1, arg2,). For example: json_value (department_data, '\$.department') Limitations: Simple Dot Notation Any optional clause in a JSON function call (such as RETURNING, ERROR, and so on) is not supported. For example: json_value (department_data, '\$.employees[1].hireDate' RETURNING DATE)			
Operators	Supported:			
	 Relational: +, -, *, /, %, - (unary minus) Arithmetic: =, <>, <, >, <=, >= 			
	• Logical: AND, OR, NOT			
	String: (concat)			
Functions and predicates	Supported are all available functions in the Oracle RDBMS that take the form function_name (arg1, arg2,) with optional schema and package qualifiers. Supported PGQL functions/predicates: IS NULL, IS NOT NULL LOWER, UPPER SUBSTRING ABS, CEIL/CEILING, FLOOR, ROUND EXTRACT CAST CASE IN and NOT IN			
	Unsupported PGQL functions/predicates are all vertex/edge functions			



Table 6-1 (Cont.) Supported PGQL Functionalities and Limitations for SQL Property Graphs

Features	PGQL on SQL Property Graphs		
User-defined functions	Supported: PL/SQL functions Functions created via the Oracle Database Multilingual Engine (MLE)		
Subqueries: Scalar subqueries EXISTS and NOT EXISTS subqueries	Supported subqueries: EXISTS NOT EXISTS Not supported: Scalar subqueries		
INSERT/UPDATE/DELETE	Not supported		
INTERVAL literals and operations	Not supported		



Part III

Property Graph Views

Learn and work with property graph views.

You can work with property graph views if you are using Oracle Database 23ai or earlier.

- About Property Graph Views
 - You can create property graph views (PG Views) over data stored in Oracle Database. You can perform various graph analytics operations using PGQL on these views.
- Loading a PG View into the Graph Server (PGX)
 There are several ways to load a property graph view (PG View) into the graph server (PGX).
- Quick Starts for Using Property Graph Views
 This chapter contains quick start tutorials and other resources to help you get started on working with property graph views.
- Getting Started with the Client Tools
 You can use multiple client tools to interact with the graph server (PGX) or directly with
 the graph data in the database.
- Property Graph Query Language (PGQL)
 PGQL is a SQL-like query language for property graph data structures that consist of *vertices* that are connected to other vertices by *edges*, each of which can have key-value pairs (properties) associated with them.



7

About Property Graph Views

You can create property graph views (PG Views) over data stored in Oracle Database. You can perform various graph analytics operations using PGQL on these views.

The following sections explain PG Views in detail:

- Creating Property Graph Views on Oracle Database Tables
 The CREATE PROPERTY GRAPH statement in PGQL can be used to create a view-like object that contains metadata about the graph. This graph can be queried using PGQL.
- Creating a PG View By Importing a GraphSON file
 Using the GraphImporterBuilder API, you can create a property graph view (PG View)
 by importing graph data from a GraphSON file.
- Using JSON to Store Vertex and Edge Properties
 You can adopt a flexible schema approach in a property graph view (PG View) by
 encoding the vertex and edge properties as a single JSON value. You can then map this
 to a property value in a PG View.

7.1 Creating Property Graph Views on Oracle Database Tables

The CREATE PROPERTY GRAPH statement in PGQL can be used to create a view-like object that contains metadata about the graph. This graph can be queried using PGQL.

The property graph views are created directly over data that exists in the relational database tables. Since the graph is stored in the database tables it has a schema. This is unlike the graphs created with a flexible schema, where the data is copied from the source tables to property graph schema tables as described in Property Graph Schema Objects for Oracle Database.

One of the main benefits of property graph views, is that all updates to the database tables are immediately reflected in the graph.

Metadata Tables for PG Views

Each time a CREATE PROPERTY GRAPH statement is executed, metadata tables are created in the user's own schema.

The following table describes the set of metadata tables that are created for each graph on executing CREATE PROPERTY GRAPH statement.

All columns shown underlined in the Table 7-1 are part of the primary key of the table. Also all columns have a NOT NULL constraint.

Table 7-1 Metadata Tables for PG Views

Table Name	Description				
graphName_ ELEM_TAB LE \$	Metadata for graph element (vertex/edge) tables (one row per element table): ET_NAME: the name of the element table (the "alias")				
	ET_TYPE: either "VERTEX" or "EDGE" COURT NAME: the page of the selection of the underlying table.				
	SCHEMA_NAME: the name of the schema of the underlying table TABLE_NAME: the name of underlying table				
	TIDEL_NAME. WE HAVE OF GREENING GABLE				
graphName_ LABEL\$	Metadata on labels of element tables (one row per label; one label per element table): <u>LABEL_NAME</u> : the name of the label				
	• ET_NAME: the name of the element table (the "alias")				
	ET_TYPE: either "VERTEX" or "EDGE"				
<pre>graphName_PROPERTY</pre>	Metadata describing the columns that are exposed through a label (one row per property)				
\$	PROPERTY NAME: the name of the property				
	ET_NAME: the name of the element table (the "alias")				
	ET_TYPE: either "VERTEX" or "EDGE"				
	 LABEL_NAME: the name of the label that this property belongs to 				
	 COLUMN_NAME: the name of the column (initially, only the case where property 				
	names equal column names is allowed)				
graphName_ KEY \$	Metadata describing a vertex/edge key (one row per column in the key)				
	COLUMN_NAME: the name of the column in the key				
	COLUMN_NUMBER: the number of the column in the key				
	For example, in KEY (a, b, c), "a" has number 1, "b" has number 2 and "c" has number 3.				
	KEY TYPE: either "VERTEX" or "EDGE"				
	ET_NAME: the name of the element table (the "alias")				
graphName SRC DST	Metadata describing the edge source/destination keys (one row per column of a key):				
KEY\$	• ET NAME: the name of the element table (the "alias"), which is always an edge table				
	VT_NAME: the name of the vertex table				
	KEY TYPE: either "EDGE_SOURCE" or "EDGE_DESTINATION"				
	• ET COLUMN NAME: the name of the key column				
	ET COLUMN NUMBER: the number of the column in the key.				
	For example, in KEY (a, b, c), "a" has number 1, "b" has number 2 and "c" has number 3.				
	Note:				
	A Hotel				
	Currently, support is only for SOURCE KEY () REFERENCES T1 . So only the edge source/destination key				

Example 7-1 To create a Property Graph View

Consider the following CREATE PROPERTY GRAPH statement:

is stored.

CREATE PROPERTY GRAPH student_network
 VERTEX TABLES(



```
person
   KEY ( id )
   LABEL student
   PROPERTIES ( name ),
  university
   KEY ( id )
    PROPERTIES ( name )
EDGE TABLES (
  knows
    key (person1, person2)
    SOURCE KEY ( person1 ) REFERENCES person (id)
    DESTINATION KEY ( person2 ) REFERENCES person (id)
    NO PROPERTIES,
  person AS studentOf
    key (id, university)
    SOURCE KEY ( id ) REFERENCES person (id)
    DESTINATION KEY (university) REFERENCES university (id)
   NO PROPERTIES
OPTIONS (PG VIEW)
```

The OPTIONS clause allows the creation of a property graph view instead of the creation of property graph schema graph. You must simply pass the CREATE PROPERTY GRAPH statement to the execute method:

Note:

- You can create property graph views using the RDBMS Java API or through SQLcl.
- You can query property graph views using the graph visualization tool or SQLcl.

```
stmt.execute("CREATE PROPERTY GRAPH student network ...");
```

This results in the creation of the following metadata tables:

SQL> SELECT * FROM STUDENT NETWORK ELEM TABLE\$;

ET_NAME	ET_TYPE	SCHEMA_NAME	TABLE_NAME			
PERSON UNIVERSITY	VERTEX VERTEX	SCOTT SCOTT	PERSON UNIVERSITY			
KNOWS STUDENTOF	EDGE EDGE	SCOTT SCOTT	KNOWS PERSON			
SQL> SELECT * FROM STUDENT_NETWORK_LABEL\$;						
LABEL_NAME	ET_NAME	ET_TYPE				



KNOWS	UNIVERSIT	Y VERT EDGE	EDGE				
SQL> SELECT * FROM STUDENT_NETWORK_PROPERTY\$;							
PROPERTY_NAM	ME ET_NAME	ET_1	YPE	LABEL_NAME	COLUMN_NAME		
	UNIVERSIT	Y VERI	EX.	STUDENT UNIVERSITY	NAME NAME		
SQL> SELECT * FROM STUDENT_NETWORK_KEY\$; COLUMN_NAME COLUMN_NUMBER KEY_TY ET_NAME							
ID ID PERSON1 PERSON2 ID UNIVERSITY		1 VERTEX 1 VERTEX 1 EDGE 2 EDGE 1 EDGE 2 EDGE	Z PERSO Z UNIVE KNOWS KNOWS STUDE	ON CRSITY S S CNTOF			
SQL> SELECT * FROM STUDENT_NETWORK_SRC_DST_KEY\$; ET_NAME							
KNOWS PERSON1 KNOWS PERSON2		EDGE_SOUF 1 EDGE_DEST)N			
STUDENTOF ID	PERSON UNIVERSITY	EDGE_SOUF		NO			

You can now run PGQL queries on the property graph view student network.

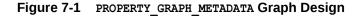
See Executing PGQL Queries Against Property Graph Views for more details to create, query and drop property graph views.

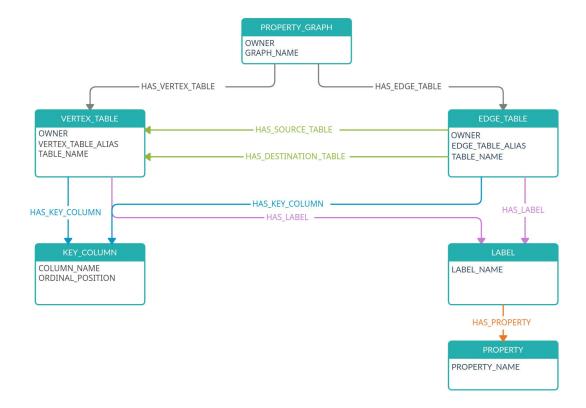
• Retrieving Metadata for Property Graph Views
You can retrieve the metadata of property graph views created in the database using the built-in PROPERTY GRAPH METADATA graph in your PGQL queries.

7.1.1 Retrieving Metadata for Property Graph Views

You can retrieve the metadata of property graph views created in the database using the built-in $PROPERTY_GRAPH_METADATA$ graph in your PGQL queries.

The PROPERTY GRAPH METADATA graph structure including properties is as shown:





The following describes the preceding design of the metadata graph:

```
PROPERTY_GRAPH -[:HAS_VERTEX_TABLE]-> VERTEX_TABLE
-[:HAS_EDGE_TABLE]-> EDGE_TABLE

VERTEX_TABLE -[:HAS_KEY_COLUMN]-> KEY_COLUMN
-[:HAS_LABEL]-> LABEL

EDGE_TABLE -[:HAS_KEY_COLUMN]-> KEY_COLUMN
-[:HAS_LABEL]-> LABEL
-[:HAS_SOURCE_TABLE]-> VERTEX_TABLE
-[:HAS_DESTINATION_TABLE]-> VERTEX_TABLE

LABEL -[:HAS_PROPERTY]-> PROPERTY
```

It is is important to note the following when using PROPERTY_GRAPH_METADATA in PGQL queries:

- PROPERTY_GRAPH_METADATA is automatically created and updated the first time you attempt to access it in a PGQL query.
- When running PGQL queries using the Java API, you must disable autocommit on the JDBC connection (conn.setAutoCommit(false)). This ensures that PROPERTY GRAPH METADATA graph gets created automatically.

The following examples show using PROPERTY_GRAPH_METADATA in PGQL queries to retrieve the required metadata.

You can retrieve the list of graphs to which you have access as shown:

- JShell
- Java
- Python

JShell

```
opg4j> String pgql =
...> "SELECT g.graph_name "
...> +"FROM MATCH (g:property_graph) ON property_graph_metadata "
...> +"ORDER BY g.graph_name"
pgql ==> "SELECT g.graph_name FROM MATCH (g:property_graph) ON
property_graph_metadata ORDER BY g.graph_name"
opg4j> pgqlStmt.executeQuery(pgql).print()
```

Java

```
String pgql = "SELECT g.graph_name "+
"FROM MATCH (g:property_graph) ON property_graph_metadata "+
"ORDER BY g.graph_name";
PgqlResultSet rs = pgqlStmt.executeQuery(pgql);
rs.print();
```

Python

```
>>> pgql = '''
... SELECT g.graph_name
... FROM MATCH (g:property_graph) ON property_graph_metadata
... ORDER BY g.graph_name
... '''
>>> pgql statement.execute query(pgql).print()
```

On execution, the preceding guery produces the following result:

```
+-----+
| GRAPH_NAME |
+-----+
| BANK_GRAPH_VIEW |
| FINANCIAL_TRANSACTIONS |
| FRIENDS |
```



You can retrieve the vertex properties of a graph as shown:

- JShell
- Java
- Python

JShell

```
opg4j> String pgql =
...> "SELECT p.property_name "
...> +"FROM MATCH(g:property_graph)-[:has_vertex_table]->(v)-[:has_label]-
>(l:label)-[:has_property]->(p:property) "
...> +"ON property_graph_metadata "
...> +"WHERE g.graph_name = 'FRIENDS' "
pgql ==> "SELECT p.property_name FROM MATCH(g:property_graph)-
[:has_vertex_table]->(v)-[:has_label]->(l:label)-[:has_property]-
>(p:property) ON property_graph_metadata WHERE g.graph_name = 'FRIENDS' "
opg4j> pgqlStmt.executeQuery(pgql).print()
```

Java

```
String pgql = "SELECT p.property_name "+
"FROM MATCH(g:property_graph)-[:has_vertex_table]->(v)-[:has_label]-
>(1:label)-[:has_property]->(p:property) "+
"ON property_graph_metadata "+
"WHERE g.graph_name = 'FRIENDS' ";
PgqlResultSet rs = pgqlStmt.executeQuery(pgql);
rs.print();
```

Python

```
>>> pgql = '''
... SELECT p.property_name
... FROM MATCH(g:property_graph) - [:has_vertex_table] -> (v) - [:has_label] -
>(l:label) - [:has_property] -> (p:property)
... ON property_graph_metadata
... WHERE g.graph_name = 'FRIENDS'
... '''
>>> pgql statement.execute query(pgql).print()
```

On execution, the preceding query produces the following result:

```
+----+
| PROPERTY_NAME |
+----+
| BIRTHDATE |
```



```
| HEIGHT |
| NAME |
+-----
```

7.2 Creating a PG View By Importing a GraphSON file

Using the GraphImporterBuilder API, you can create a property graph view (PG View) by importing graph data from a GraphSON file.

This import functionality consists of the following steps:

- 1. Parsing of the GraphSON to a data structure.
- 2. Creating the SQL tables from the data structure and inserting the data.
- 3. Generating and running the CREATE PROPERTY GRAPH statement.

The following example show using the <code>GraphImporterBuilder</code> API to create a PG View from a GraphSON file.

- JShell
- Java
- Python

JShell

Java

```
import oracle.pg.imports.*;
GraphImporter importer = new GraphImporter.Builder()
    .setFilePath("<path_to_graphson_file>")
    .setBatchSize(2)
    .setInputFormat(GraphImportInputFormat.GRAPHSON)
    .setOutputFormat(GraphImportOutputFormat.PG_VIEW)
    .setThreads(4)
    .setDbJdbcUrl("<jdbc url>")
```



```
.setDbUsername("<username>")
.setDbPassword("<password>")
.setGraphName("mygraph")
.build();
```

Python

```
>>> from opg4py.graph_importer import GraphImporter
>>> config = {
... 'jdbc_url' : '<jdbc_url>',
... 'username' : '<username>',
... 'password' : '<password>',
... 'file_path' : '<path_to_graphson_file>',
... 'graph_name' : 'mygraph',
... 'output_format': 'pg_view',
... 'input_format' : 'graphson'
... }
>>> importer = GraphImporter(config)
>>> importer.import_graph()
```

The preceding example sets up the required SQL tables in the database, generates and runs the DDL statement to create *mygraph*. For instance, this example generates the following CREATE PROPERTY GRAPH DDL statement:

```
"CREATE PROPERTY GRAPH mygraph
          VERTEX TABLES (
             software
              KEY (id)
              LABEL software
               PROPERTIES ARE ALL COLUMNS,
             person
              KEY (id)
               LABEL person
               PROPERTIES ARE ALL COLUMNS
           EDGE TABLES (
             created
              KEY (id)
               SOURCE KEY (sid) REFERENCES person (id)
               DESTINATION KEY (did) REFERENCES software (id)
               LABEL created
               PROPERTIES ARE ALL COLUMNS,
             knows
               KEY (id)
               SOURCE KEY (sid) REFERENCES person (id)
               DESTINATION KEY (did) REFERENCES person (id)
               LABEL knows
               PROPERTIES ARE ALL COLUMNS
           ) OPTIONS ( pg view )"
```



Alternatively, you can also create a connection to the database by using a data source to connect to the database as shown in the following example:

- JShell
- Java

JShell

```
opg4j> import oracle.pg.imports.*
opg4j> import oracle.jdbc.pool.OracleDataSource
opg4j> var ds = new OracleDataSource() // setup the data source
ds ==> oracle.jdbc.pool.OracleDataSource@4154ecd3
ds.setURL("<jdbc url>")
ds.setUser("<username>")
ds.setPassword("<password>")
opg4j> var importer = new GraphImporter.Builder().
...> setFilePath("<path to graphson file>").
...>
     setBatchSize(2).
     setInputFormat(GraphImportInputFormat.GRAPHSON).
setOutputFormat(GraphImportOutputFormat.PG_VIEW).
...>
...>
     setThreads(4).
...>
     setDataSource(ds).
        setGraphName("mygraph").
...>
       build()
...>
importer ==> oracle.pq.imports.GraphImporter@5d957cf0
opg4j> var ddl = importer.importGraph()
```

Java

```
import oracle.pg.imports.*;
import oracle.jdbc.pool.OracleDataSource;
//Setup the datasource
var ds = new OracleDataSource();
ds.setURL(<jdbc url>)
ds.setUser(<username>);
ds.setPassword(<password>);
//Setup the GraphImporter
GraphImporter importer = new GraphImporter.Builder()
     .setFilePath("<path to graphson file>")
     .setBatchSize(2)
     .setInputFormat(GraphImportInputFormat.GRAPHSON)
     .setOutputFormat(GraphImportOutputFormat.PG VIEW)
     .setThreads(4)
     .setDataSource(ds)
     .setGraphName("mygraph")
     .build();
var ddl = importer.importGraph();
```



Also, note the following:

- The GraphImporterBuilder API supports GraphSON file format version 3.0 only.
- Only GraphSON data types listed in Table 7-6 are supported.

The following sections provide more details on the GraphImporter parameters and the data type mapping between GraphSON and Oracle Database.

- Additional Information on the GraphImporter Parameters
 Learn more about the parameters used by the GraphImporter.
- Mapping GraphSON Types to Oracle Database Data Types
 The GraphSON data types can be mapped to their corresponding Oracle Database data types.

7.2.1 Additional Information on the GraphImporter Parameters

Learn more about the parameters used by the GraphImporter.

Table 7-2 Database Connection Parameters

Parameter	Description	Setter in API	Default Value	Optional
dataSource	Data source for the database	setDataSource	NULL	Only if passing dbJdbcUrl, dbUsername and dbPassword
dbJdbcUrl	JDBC url of the database	setDbJdbcUrl	""	Only if passing a dataSouce
dbPassword	Database password	setDbPassword	""	Only if passing a dataSouce
dbUsername	Database user name	setDbUsername	11 11	Only if passing a dataSouce

Table 7-3 GraphImporter Configuration Parameters

Parameter	Description	Setter in API	Default Value	Optional
pathName	Path to the GraphSON file	setPathname	""	No
graphName	Resulting graph name	setGraphName	""	Yes
inFormat	Input format for the importer	setInputFormat	GraphIm portInp utForma t.GRAPH SON	Yes
outFormat	Output format for the importer	setOutputFormat	GraphIm portOut putForm at.PG_V IEW	Yes



Table 7-3 (Cont.) GraphImporter Configuration Parameters

Parameter	Description	Setter in API	Default Value	Optional
batchSize	Number of rows read before inserting data to the database	setBatchSize	1000	Yes
threads	Number of threads to be used to insert to the database		1	Yes

Table 7-4 SQL Storage Parameters

Parameter	Description	Setter in API	Default Value	Optional
stringFieldSi ze	GraphSON String data type is translated as VARCHAR2 in the database. This parameter represents the VARCHAR2 size for the data storage.	setStringFiel dsSize	100	Yes
fractionalSec ondsPrecision	The fractional seconds precision parameter found in TIMESTAMP data type in the Oracle Database.	setFractional SecondsPrecis ion	6	Yes

 Table 7-5
 PGQL Supported Parameters

Parameter	Description	Setter in API	Default Value	Optional
parallel	Degree of parallelism to use for query and update operations	setPathname	0	Yes
dynamicSampli ng	Dynamic sampling value	setGraphName	2	Yes
matchOptions	Additional options used to influence query translation and execution	setMatchOptio ns	NULL	Yes



Table 7-5 (Cont.) PGQL Supported Parameters

Parameter	Description	Setter in API	Default Value	Optional
options	Additional options used to influence modify translation and execution	setOptions	NULL	Yes

7.2.2 Mapping GraphSON Types to Oracle Database Data Types

The GraphSON data types can be mapped to their corresponding Oracle Database data types.

The following table shows GraphSON data types mapping to Oracle Database data types:

Table 7-6 Mapping GraphSON Types to Oracle Database Types

GraphSON Type	Oracle Database Type	
String	VARCHAR2 ¹	
g:Int32	NUMBER (10)	
g:Int64	NUMBER (10)	
g:Float	FLOAT	
g:Double	FLOAT	
g:Date	DATE	
g:Timestamp	TIMESTAMP ²	
g:UUID	CHAR (36)	

¹ You can use the <code>stringFieldSize</code> parameter to determine the string size for the database to store on the <code>String</code> columns.

7.3 Using JSON to Store Vertex and Edge Properties

You can adopt a flexible schema approach in a property graph view (PG View) by encoding the vertex and edge properties as a single JSON value. You can then map this to a property value in a PG View.

PG Views do not provide schema flexibility by nature since adding a new label requires adding a new vertex or edge table, and adding a new property requires adding a new column, both of which are schema update operations. However, through the use of JSON you can model schema flexibility on top of PG Views.

For example, consider the following graph which represents financial transactions between two Account vertices. The Account can be owned either by a Person or a Company.



You can use the fractionalSecondsPrecision parameter to specify the precision on the columns of type Timestamp.

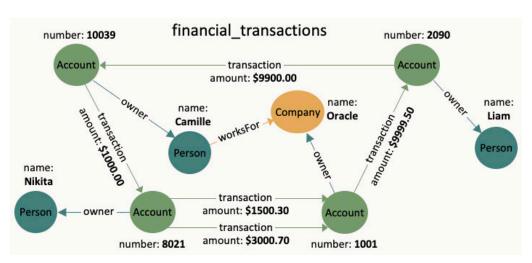


Figure 7-2 Financial Transactions Graph

You can create a single table for storing all the vertices and another single table for storing all the edges, as shown:

```
CREATE TABLE fin vertex table (
  id NUMBER PRIMARY KEY,
  properties VARCHAR2 (2000)
);
INSERT INTO fin vertex table VALUES ( 1,
'{"type": "Person", "name": "Nikita"}');
INSERT INTO fin vertex table VALUES ( 2,
'{"type":"Person", "name": "Camille"}');
INSERT INTO fin vertex table VALUES ( 3,
'{"type":"Person", "name": "Liam"}');
INSERT INTO fin vertex table VALUES ( 4,
'{"type":"Company", "name":"Oracle"}');
INSERT INTO fin vertex table VALUES ( 5,
'{"type": "Account", "number": 10039}');
INSERT INTO fin vertex table VALUES ( 6,
'{"type": "Account", "number": 2090}');
INSERT INTO fin vertex table VALUES ( 7,
'{"type": "Account", "number": 8021}');
INSERT INTO fin vertex table VALUES ( 8,
'{"type": "Account", "number": 1001}');
CREATE TABLE fin edge table (
  id NUMBER PRIMARY KEY,
  src NUMBER REFERENCES fin vertex table ( id ),
  dst NUMBER REFERENCES fin vertex table ( id ),
  properties VARCHAR2 (2000)
);
INSERT INTO fin edge table VALUES ( 1, 7, 1, '{"type":"owner"}');
INSERT INTO fin edge table VALUES ( 2, 5, 2, '{"type":"owner"}');
INSERT INTO fin edge table VALUES ( 3, 6, 3, '{"type":"owner"}');
INSERT INTO fin edge table VALUES ( 4, 8, 4, '{"type":"owner"}');
```



```
INSERT INTO fin_edge_table VALUES ( 5, 2, 4, '{"type":"worksFor"}');
INSERT INTO fin_edge_table VALUES ( 6, 5, 7,
'{"type":"transaction", "amount":1000.00}');
INSERT INTO fin_edge_table VALUES ( 7, 7, 8,
'{"type":"transaction", "amount":1500.30}');
INSERT INTO fin_edge_table VALUES ( 8, 7, 8,
'{"type":"transaction", "amount":3000.70}');
INSERT INTO fin_edge_table VALUES ( 9, 8, 6,
'{"type":"transaction", "amount":9999.50}');
INSERT INTO fin_edge_table VALUES ( 10, 6, 5,
'{"type":"transaction", "amount":9900.00}');
```

As seen in the preceding code, each vertex and edge is represented by a single row in the respective tables. The first column is the unique key of the vertex or the edge. The second and third columns of the edge table are its source key and destination key respectively. The last column of the vertex and edge tables encodes all the properties as well as the labels in a JSON object. A JSON is an unordered set of name and value pairs. Here, you can use such pairs to encode the property names and their values as well as the label's value. In case of the label, you can choose an arbitrary name such as "type" or "label". In this example we use "type".

Because all the labels and properties of a vertex or an edge are encoded as a single JSON value, you do not need to update the schema when new labels or properties are added to the graph. Instead, you can add new labels and properties by inserting additional vertices and edges or by updating the JSON value in the underlying table through SQL.

The following two examples demonstrate how you can extract labels and property values from JSON objects for PGQL on RDBMS and PGQL on PGX respectively.

Example 7-2 Extracting JSON properties using JSON_VALUE (PGQL on RDBMS)

The following code creates a PG View using the fin_vertex_table and fin_edge_table tables and executes a PGQL SELECT query:

```
PgglStatement pgglStmnt = pgglConn.createStatement();
/* Create the property graph */
pgglStmnt.execute(
  "CREATE PROPERTY GRAPH financial transactions " +
  " VERTEX TABLES ( " +
       fin vertex table PROPERTIES ( properties ) ) " +
    EDGE TABLES ( " +
       fin edge table " +
         SOURCE KEY ( src ) REFERENCES fin vertex table (id) " +
         DESTINATION KEY ( dst ) REFERENCES fin vertex table (id) " +
         PROPERTIES ( properties ) ) " +
    OPTIONS ( PG VIEW )");
/* Set the name of the graph so that we can omit the ON clause from queries
pgqlConn.setGraph("FINANCIAL TRANSACTIONS");
/* PGQL query: find all outgoing transactions from account 8021. Output the
   transaction amount and the destination account number. */
PgqlResultSet rs = pgqlStmnt.executeQuery(
  "SELECT JSON VALUE(trans.properties, '$.amount') AS transaction amount, " +
```

```
"     JSON_VALUE(account2.properties, '$.number') AS
account_number " +
    "FROM MATCH (account1) -[trans]-> (account2) " +
    "WHERE JSON_VALUE(account1.properties, '$.number') = 8021 " +
    "    AND JSON_VALUE(trans.properties, '$.type') = 'transaction'");
rs.print();
rs.close();
pgqlStmnt.close();
```

In the preceding code, the CREATE PROPERTY GRAPH statement maps the JSON column into a property named "properties". This property will thus contain all the labels and properties of the vertex or the edge. The PGQL SELECT query extracts these labels and properties using JSON VALUE.

For example, instead of account1.number = 8021, you must use JSON_VALUE(account1.properties, '\$.number') = 8021. This causes the query to become a bit lengthier.

The output of the Java code is:

```
+-----+
| AMOUNT | ACCOUNT_NUMBER |
+------+
| 1500.3 | 1001 |
| 3000.7 | 1001 |
```

Example 7-3 Using a UDF to extract a JSON property value (PGQL on PGX)

This example consists of two parts. The first part shows the creation of a UDF and the second part shows loading of the graph into the graph server (PGX) followed by the execution of a PGQL query using the UDF.

Since the Graph Server (PGX) does not have a built-in JSON_VALUE function like in PGQL on RDBMS, you can create a Java UDF instead.

Create the Java class (MyJsonUtils.java) that implements the UDF:

```
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;

public class MyJsonUtils {
   private final static ObjectMapper mapper = new ObjectMapper();
   public static String get_prop(String json_string, String prop_name)
throws JsonProcessingException {
    JsonNode node = mapper.readTree(json_string);
    return node.path(prop_name).asText();
   }
}
```



Compile the class with the JARs from $\protect\protec$

```
mkdir ./target
javac -classpath .:/opt/oracle/graph/pgx/server/lib/* -d ./target *.java
cd target
jar cvf MyJsonUtils.jar *
```

Using the following UDF JSON configuration file ($my_udfs.json$), you can now register the Java UDF on the graph server (PGX) by following step-3 to step-6 in User-Defined Functions (UDFs) in PGX:

```
"user defined functions": [
      "namespace": "my",
      "function name": "get prop",
      "language": "java",
      "implementation reference": "MyJsonUtils",
      "return type": "string",
      "arguments": [
        {
          "name": "json string",
          "type": "string"
        },
          "name": "prop name",
          "type": "string"
      1
    }
  1
}
```

On implementing the UDF for extracting property values from the JSON, you can now load the graph into the Graph Server (PGX) and issue a PGQL query:

```
/* Load the graph into the Graph Server (PGX) */
ServerInstance instance = GraphServer.getInstance("http://localhost:7007",
username, password.toCharArray());
session = instance.createSession("my-session");
PgxGraph g = session.readGraphByName("FINANCIAL_TRANSACTIONS",
GraphSource.PG_VIEW);

/* PGQL query: find all shortest paths from account 10039 to account 2090
following only outgoing transaction
  edges. Output the list of transaction amounts along each path as well as
the total amount of the transactions
  along each path. */
g.queryPgql(
  " SELECT LISTAGG(my.get_prop(e.properties, 'amount'), ' + ') || ' = ' AS
amounts_along_path, " +
```



```
" SUM(CAST(my.get_prop(e.properties, 'amount') AS DOUBLE))
AS total_amount " +
   " FROM MATCH ALL SHORTEST (a) (-[e]-> WHERE
my.get_prop(e.properties, 'type') = 'transaction')* (b) " +
   " WHERE my.get_prop(a.properties, 'number') = '10039' AND " +
   " my.get_prop(b.properties, 'number') = '2090' " +
   "ORDER BY total amount").print().close();
```

The output of the Java code is:



Loading a PG View into the Graph Server (PGX)

There are several ways to load a property graph view (PG View) into the graph server (PGX).

- Loading a PG View Using the readGraphByName API
 You can load a graph into the graph server (PGX) from a property graph view (PG View)
 by name.
- Loading a Graph Using a JSON Configuration File
 In order to load a property graph view into the graph server (PGX), you can create a graph configuration file, which contains the metadata of the graph to be loaded.
- Loading a Graph by Defining a Graph Configuration Object
 You can load a graph from Oracle Database by first defining the graph configuration
 object using the GraphConfigBuilder class and then reading the graph into the graph
 server (PGX).
- Loading a Subgraph from Property Graph Views
 You can create a subgraph from a property graph view and load it into memory in the
 graph server (PGX).

8.1 Loading a PG View Using the readGraphByName API

You can load a graph into the graph server (PGX) from a property graph view (PG View) by name.

You can use the PqxSession#readGraphByName API to load a graph from a PG View:

 $\label{lem:condition} readGraphByName\,(String\ schemaName,\ String\ graphName,\ GraphSource\ source,\ ReadGraphOption\ options)$

The arguments used in the method are described in the following table:

Table 8-1 Parameters for the readGraphByName method

Parameter	Description	Optional
schemaName	Schema owner	Yes
graphName	Name of the PG View	No
source	Source format for the graph (GraphSource.PG_VIEW)	No
options	Represents the graph optimization options	Yes

The <code>readGraphByName()</code> method reads the PG View metadata tables and internally generates the graph configuration to load the graph. You must have <code>PGX_SESSION_NEW_GRAPH</code> permission to use this API.

For example you can load the graph from a property graph view as shown:

- JShell
- Java
- Python

JShell

```
opg4j> var graph = session.readGraphByName("BANKDATAVIEW",
GraphSource.PG_VIEW)
$12 ==> PgxGraph[name=bankdataview, N=1000, E=5001, created=1625730942294]
```

Java

```
PgxGraph graph = session.readGraphByName("BANKDATAVIEW",
GraphSource.PG_VIEW);
Graph: PgxGraph[name=bankdataview, N=1000, E=5001, created=1625732149262]
```

Python

```
>>> graph = session.read_graph_by_name('BANKDATAVIEW', 'pg_view')
>>> graph
PgxGraph(name: bankdataview, v: 1000, e: 5001, directed: True,
memory(Mb): 0)
```

- Specifying Options for the readGraphByName API You can specify graph optimization options, OnMissingVertexOption or both when using the readGraphByName API for loading a property graph view (PG View).
- Specifying the Schema Name for the readGraphByName API
 You can specify the schema name when using the readGraphByName API for
 loading a property graph view (PG View).

8.1.1 Specifying Options for the readGraphByName API

You can specify graph optimization options, OnMissingVertexOption or both when using the readGraphByName API for loading a property graph view (PG View).

The ReadGraphOption interface supports an additional options parameter when loading a PG View by name.

The following sections explain the various options supported by the ReadGraphOption interface.

Using the Graph Optimization Options

You can optimize the read or update performance when loading a PG View by name by using one of the following options:



- ReadGraphOption.optimizeFor(GraphOptimizedFor.READ): Specifies that the loaded graph is optimized for READ.
- ReadGraphOption.optimizeFor(GraphOptimizedFor.UPDATES): Specifies that the loaded graph is optimized for UPDATE.
- ReadGraphOption.synchronizable(): Specifies that the loaded graph can be synchronized.

It is important to note the following:

- synchronizable() option can be used in combination with UPDATE and READ. However, the UPDATE and READ options cannot be used at the same time.
- If you are loading a PG View for SYNCHRONIZABLE option, then ensure that the vertex and edge keys are numeric and non-composite.

The following example loads a PG View for READ and SYNCHRONIZABLE options:

- JShell
- Java

JShell

```
opg4j> var graph = session.readGraphByName("BANK_GRAPH_VIEW",
GraphSource.PG_VIEW,
...>
ReadGraphOption.optimizeFor(GraphOptimizedFor.READ),
...> ReadGraphOption.synchronizable())
graph ==>
PgxGraph[name=BANK GRAPH VIEW 2,N=1000,E=5001,created=1648457198462]
```

Java

```
PgxGraph graph = session.readGraphByName("BANKDATAVIEW",
GraphSource.PG_VIEW, "BANK_GRAPH_VIEW", GraphSource.PG_VIEW,
ReadGraphOption.optimizeFor(GraphOptimizedFor.READ),
ReadGraphOption.synchronizable());
```

Using the OnMissingVertex Options

If either the source or destination vertex or both are missing for an edge, then you can use the <code>OnMissingVertexOption</code> which specifies the behavior for handling the edge with the missing vertex. The following values are supported for this option:

• ReadGraphOption.onMissingVertex(OnMissingVertex.ERROR): This is the default option and this specifies that an error must be thrown for edges with missing vertices.



- ReadGraphOption.onMissingVertex(OnMissingVertex.IGNORE_EDGE): Specifies that the edge for a missing vertex must be ignored.
- ReadGraphOption.onMissingVertex(OnMissingVertex.IGNORE_EDGE_LOG):
 Specifies that the edge for a missing vertex must be ignored and all ignored edges must be logged.
- ReadGraphOption.onMissingVertex(OnMissingVertex.IGNORE_EDGE_LOG_ONCE):
 Specifies that the edge for a missing vertex must be ignored and only the first ignored edge must be logged.

The following example loads the PG View by ignoring the edges with missing vertices and logging only the first ignored edge. Note, to view the logs, you must update the default Logback configuration file in /etc/oracle/graph/logback.xml and the graph server (PGX) logger configuration file in /etc/oracle/graph/logback-server.xml to log the DEBUG logs. You can then view the ignored edges in /var/opt/log/pgx-server.log file.

- JShell
- Java

JShell

```
opg4j> session.readGraphByName("REGIONS", GraphSource.PG_VIEW,
...>
ReadGraphOption.onMissingVertex(OnMissingVertex.IGNORE_EDGE_LOG_ONCE))
$7 ==> PgxGraph[name=REGIONVIEW 3,N=27,E=18,created=1655903219910]
```

Java

```
PgxGraph graph = session.readGraphByName("REGIONS",
GraphSource.PG_VIEW,
ReadGraphOption.onMissingVertex(OnMissingVertex.IGNORE EDGE LOG ONCE));
```

8.1.2 Specifying the Schema Name for the readGraphByName API

You can specify the schema name when using the readGraphByName API for loading a property graph view (PG View).

This feature allows you load a PG View from another user schema into the graph server (PGX). However, ensure that you have READ permission on all the underlying metadata and data tables when loading a PG View from another schema.

The following example loads a PG View from the GRAPHUSER schema:

JShell



Java

JShell

```
opg4j> var graph = session.readGraphByName("GRAPHUSER", "FRIENDS",
GraphSource.PG_VIEW)
graph ==> PgxGraph[name=FRIENDS, N=6, E=4, created=1672743474212]

Java

PgxGraph graph = session.readGraphByName("GRAPHUSER", "FRIENDS",
GraphSource.PG VIEW);
```

8.2 Loading a Graph Using a JSON Configuration File

In order to load a property graph view into the graph server (PGX), you can create a graph configuration file, which contains the metadata of the graph to be loaded.

The following shows a sample JSON configuration file:

```
"name": "BANK_GRAPH",
"source name": "BANK GRAPH",
"source type": "pg view",
"jdbc url": "jdbc:oracle:thin:@localhost:1521/orclpdb",
"username": "graphuser",
"keystore alias": "database1",
"vertex providers":[
          "name": "Accounts",
          "format": "rdbms",
          "database table name": "BANK ACCOUNTS",
          "key column":"ID",
          "key type": "integer",
          "parallel hint degree": 3,
           "props":[
                       "name":"ID",
                       "type": "integer"
               },
                       "name": "NAME",
                       "type": "string"
      }
```



```
],
    "edge providers":[
            "name": "Transfers",
            "format": "rdbms",
             "database table name": "BANK TXNS",
            "key column":"ID",
            "parallel hint degree": 3,
             "source column": "FROM ACCT ID",
             "destination column": "TO ACCT ID",
            "source vertex provider": "Accounts",
             "destination vertex provider": "Accounts",
             "props":[
                 {
                         "name": "FROM ACCT ID",
                         "type":"integer"
                 },
                 {
                         "name": "TXN AMOUNT",
                         "type": "float",
                         "column": "AMOUNT"
                 },
                 {
                         "name": "DESCRIPTION",
                         "type": "string"
                 },
                 {
                          "name": "TO ACCT ID",
                         "type":"integer"
            ]
        }
    ]
}
```

The preceding configuration uses a Java keystore alias to reference the database password that is stored in a keystore file. See Store the Database Password in a Keystore for more information.

Also, the edge property AMOUNT is renamed to $\texttt{TXN}_\texttt{AMT}$. This implies that when loading a graph into the graph server (PGX), you can optionally rename vertex or edge properties to have different names other than the names of the underlying columns in the database.

See Also:

- Configuring PARALLEL Hint when Loading a Graph
- Graph Configuration Options for more details on the graph configuration options.

You can now read the graph into the graph server as shown:



- JShell
- Java

JShell

```
./bin/opg4j --secret_store /etc/oracle/graph/keystore.p12
enter password for keystore /etc/oracle/graph/keystore.p12:
For an introduction type: /help intro
Oracle Graph Server Shell 23.1.0
Variables instance, session, and analyst ready to use
opg4j> var g =
session.readGraphWithProperties("<path_to_json_configuration>")
g ==> PgxGraph[name=BANK_GRAPH_NEW,N=999,E=4993,created=1675960224397]
```

Java

Configuring PARALLEL Hint when Loading a Graph

8.2.1 Configuring PARALLEL Hint when Loading a Graph

You can also optimize the graph loading performance by configuring a specific parallel hint value using the GraphConfig field, PARALLEL_HINT_DEGREE, which will be used by the underlying SQL queries. This can be applied when loading a graph using a JSON configuration file or through the GraphConfigBuilder API.

The following table describes how the internal queries are configured based on the specified $PARALLEL\ HINT\ DEGREE\ values.$

Table 8-2 PARALLEL HINT DEGREE values

PARALLEL_HINT_DEGREE Value	Parallel hint used in the SQL Statement		
Positive integer(n)	Uses the given n degree: SELECT /*+ PARALLEL(n) */		
Zero	Uses a plain hint: SELECT /*+ PARALLEL */		
Negative integer (Default value: -1)	No PARALLEL hint: SELECT		



See Also:

- Loading a Graph Using a JSON Configuration File for an example using parallel hint configuration.
- Loading a Graph by Defining a Graph Configuration Object for an example using parallel hint configuration.

8.3 Loading a Graph by Defining a Graph Configuration Object

You can load a graph from Oracle Database by first defining the graph configuration object using the <code>GraphConfigBuilder</code> class and then reading the graph into the graph server (PGX).

The following example loads a property graph view into memory, authenticating as <database user>/<database password> with the database:

- JShell
- Java

JShell

```
opg4j> var vertexConfig = new RdbmsEntityProviderConfigBuilder().
                                              setName("Account").
...>
                                              setKeyColumn("ID").
...>
                                              setParallelHintDegree(3).
...>
setDatabaseTableName("BANK ACCOUNTS").
                                              addProperty("ID",
PropertyType.INTEGER).
                                              build()
opq4j> var edgeConfig = new RdbmsEntityProviderConfigBuilder().
                                         setName("Transfer").
...>
                                         setKeyColumn("TXN ID").
setSourceColumn("FROM ACCT ID").
setDestinationColumn("TO ACCT ID").
setSourceVertexProvider("Account").
setDestinationVertexProvider("Account").
                                         setParallelHintDegree (3).
                                         createKeyMapping(true).
...>
```



```
setDatabaseTableName("BANK TXNS").
...>
                                         addProperty("FROM ACCT ID",
...>
PropertyType.INTEGER).
...>
                                         addProperty("TO ACCT ID",
PropertyType.INTEGER).
                                         addProperty("AMOUNT",
...>
PropertyType.FLOAT).
...>
                                         build()
opg4j> var cfg = GraphConfigBuilder.forPartitioned().
...>
                      setJdbcUrl("jdbc:oracle:thin:@localhost:1521/orclpdb").
...>
                      setUsername("graphuser").
                      setPassword("<password>").
...>
                      setName("bank graph").
...>
                      setSourceName("bank graph").
...>
                      setSourceType(SourceType.PG VIEW).
...>
                      setVertexIdType(IdType.INTEGER).
...>
                      addVertexProvider(vertexConfig).
...>
                      addEdgeProvider(edgeConfig).
...>
...>
                      build()
opg4j> var g = session.readGraphWithProperties(cfg)
g ==> PgxGraph[name=bank graph, N=999, E=4993, created=1676806306348]
Java
// Build the vertex provider
RdbmsEntityProviderConfig vertexConfig = new
RdbmsEntityProviderConfigBuilder()
                                               .setName("Account")
                                                .setKeyColumn("ID")
                                                .setParallelHintDegree(3)
                                                .setDatabaseTableName("BANK ACC
OUNTS")
                                                .addProperty("ID",
PropertyType.INTEGER)
                                                .build();
// Build the edge provider
RdbmsEntityProviderConfig edgeConfig = new RdbmsEntityProviderConfigBuilder()
                                                .setName("Transfer")
                                                .setKeyColumn("TXN ID")
                                                .setSourceColumn("FROM ACCT ID"
)
                                                .setDestinationColumn("TO ACCT
ID")
                                                .setSourceVertexProvider("Accou
nt")
                                                .setDestinationVertexProvider("
Account")
                                                .setParallelHintDegree(3)
                                                .createKeyMapping(true)
                                                .setDatabaseTableName("BANK TXN
S")
```

.addProperty("FROM_ACCT_ID",

```
PropertyType.INTEGER)
                                               .addProperty("TO ACCT ID"
, PropertyType.INTEGER)
                                               .addProperty("AMOUNT",
PropertyType.FLOAT)
                                               .build();
// Build the graph
GraphConfig cfg = GraphConfigBuilder.forPartitioned()
                            .setJdbcUrl("jdbc:oracle:thin:@localhost:152
1/orclpdb")
                           .setUsername("graphuser")
                            .setPassword("<password>")
                            .setName("bank graph")
                            .setSourceName("bank graph")
                            .setSourceType(SourceType.PG VIEW)
                            .setVertexIdType(IdType.INTEGER)
                            .addVertexProvider(vertexConfig)
                            .addEdgeProvider(edgeConfig)
                            .build();
PgxGraph g = session.readGraphWithProperties(cfg);
```

See Also:

Configuring PARALLEL Hint when Loading a Graph

8.4 Loading a Subgraph from Property Graph Views

You can create a subgraph from a property graph view and load it into memory in the graph server (PGX).

Instead of loading a full graph into memory, you can load a subgraph. This would consume less memory.

The following sections explain in detail on loading and expanding of subgraphs:

- PGQL Based Subgraph Loading
 - You can use the PgViewSubgraphReader#fromPgView API to create an in-memory subgraph from a property graph view (PG View) using a set of PGQL queries.
- Prepared PGQL Queries
 - You can also use prepared queries when loading a subgraph from a property graph view.
- Providing Database Connection Credentials
 - You can specify the database connection credentials with the PgViewSubgraphReader#fromPgView API instead of using the default credentials of the current user.



Dynamically Expanding a Subgraph

You can expand an in-memory subgraph by loading another subgraph into memory and merging it with the current in-memory subgraph.

8.4.1 PGQL Based Subgraph Loading

You can use the PgViewSubgraphReader#fromPgView API to create an in-memory subgraph from a property graph view (PG View) using a set of PGQL queries.

These PGQL queries define the vertices and edges that are to be loaded into the subgraph. You can also use multiple PGQL queries and the resulting output graph is a union of the subgraphs, each being loaded independently by each PGQL query.

Note:

- Only non-composite vertex and edge keys are supported.
- Only numeric edge keys are supported.
- PGQL queries with GROUP BY or ORDER BY clauses are not supported for loading of subgraphs from a property graph view.

The following example creates a subgraph from a PG View using multiple PGQL queries:

- JShell
- Java
- Python

JShell

Java



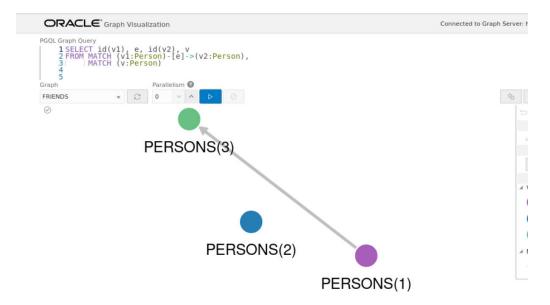
```
'PERSONS(2)'") .load();
```

Python

```
>>> graph = session.read_subgraph_from_pg_view("FRIENDS", ["MATCH
(v1:Person)-[e:FRIENDOF]->(v2:Person) WHERE id(v1) = 'PERSONS(1)'",
... "MATCH (v:Person) WHERE id(v) =
'PERSONS(2)'"])
>>> graph
PgxGraph(name: FRIENDS, v: 3, e: 1, directed: True, memory(Mb): 0)
```

The following displays the output for the preceding PGQL query using the graph visualization tool.

Figure 8-1 Subgraph Visualization



Loading Subgraphs with Custom Names

By default, the new subgraph gets created with the same name as the PG View graph. Alternatively, if you want to load a subgraph with a custom name, then you can configure the subgraph name as shown:

- JShell
- Java
- Python



JShell

Java

Python

```
>>> graph = session.read_subgraph_from_pg_view("FRIENDS",
... ["MATCH (v1:Person)-[e:FRIENDOF]->(v2:Person) WHERE
id(v1) = 'PERSONS(1)'",
... "MATCH (v:Person) WHERE id(v) = 'PERSONS(2)'"],
... graph_name="friends_network")
>>> graph
PgxGraph(name: friends_network, v: 3, e: 1, directed: True, memory(Mb): 0)
```

Loading a Subgraph by Explicitly Specifying the Schema Name

If you want to load a subgraph by reading a PG View from another schema, you can additionally provide the schema name as an argument to the PgViewSubgraphReader#fromPgView API. You must also ensure that you have READ permission on all the underlying metadata and data tables for the PG View.

For example:

- JShell
- Java
- Python



JShell

... ["MATCH (v:Person) WHERE id(v) = PERSONS(2)"],

8.4.2 Prepared PGQL Queries

... schema="GRAPHUSER")

You can also use prepared queries when loading a subgraph from a property graph view.

You can pass bind variables using prepared PGQL queries. The PreparedPgqlQuery#preparedPgqlQuery method adds a prepared query to a list of queries that are executed to load the subgraph. The PreparedPgViewPgqlQuery API sets the bindings for the variables and continues with the loading of the subgraph.

For example:

- JShell
- Java
- Python

JShell



```
>(v2:Person) WHERE id(v2)=?")
preparedPgqlQuery ==>
oracle.pgx.api.subgraph.PreparedPgViewPgqlQuery@2e6b379c
opg4j> preparedPgqlQuery = preparedPgqlQuery.withStringArg(1, "PERSONS(3)")
preparedPgqlQuery ==>
oracle.pgx.api.subgraph.PreparedPgViewPgqlQuery@2e6b379c
opg4j> var graph = preparedPgqlQuery.load()
graph ==> PgxGraph[name=FRIENDS_2,N=3,E=2,created=1648566047855]
```

Java

```
import oracle.pgx.api.subgraph.*;
...
...
PgViewSubgraphReader pgViewSubgraphReader=
session.readSubgraph().fromPgView("FRIENDS");
PreparedPgViewPgqlQuery preparedPgqlQuery =
pgViewSubgraphReader.preparedPgqlQuery("MATCH (v1:Person)-[e:FriendOf]-
>(v2:Person) WHERE id(v2)=?");
preparedPgqlQuery = preparedPgqlQuery.withStringArg(1, "PERSONS(3)");
PgxGraph graph = preparedPgqlQuery.load();
```

Python

```
>>> from pypgx.api import PreparedPgqlQuery
>>> from pypgx.api import PreparedPgqlQueryStringArgument
>>> graph = session.read_subgraph_from_pg_view("FRIENDS",
... [PreparedPgqlQuery("MATCH (v1:Person)-[e:FriendOf]->(v2:Person) WHERE
id(v2)=?", [PreparedPgqlQueryStringArgument("PERSONS(3)")])])
>>> graph
PgxGraph(name: FRIENDS, v: 3, e: 2, directed: True, memory(Mb): 0)
```

8.4.3 Providing Database Connection Credentials

You can specify the database connection credentials with the PgViewSubgraphReader#fromPgView API instead of using the default credentials of the current user.

The following example shows loading of a subgraph for non-default database connection settings:

- JShell
- Java



JShell

```
opg4j> var graph = session.readSubgraph().
                          fromPqView("FRIENDS").
...>
                         username("graphuser").
                         password("<password for graphuser>").
                         keystoreAlias("database1").
                          schema("graphuser").
...>
                          jdbcUrl("jdbc:oracle:thin:@localhost:1521/
orclpdb").
...>
                          connections (12).
...>
                          queryPqql("MATCH (a:Person)").
...>
                          load()
graph ==> PgxGraph[name=FRIENDS, N=4, E=0, created=1648541234520]
```

Java

8.4.4 Dynamically Expanding a Subgraph

You can expand an in-memory subgraph by loading another subgraph into memory and merging it with the current in-memory subgraph.

The PgxGraph.expandGraph() method can be used to expand a subgraph. The following applies when merging two graphs:

- Both the graphs can have separate sets of providers.
- A graph can have some providers same as the other graph. In this case:
 - The providers with the same names must have the same labels.
 - The graph being merged must have the same or a common subset of properties as the base graph. However, it is possible that either of the graphs may have more number of properties.

The following example shows the expansion of the subgraph created in PGQL Based Subgraph Loading:

JShell



- Java
- Python

JShell

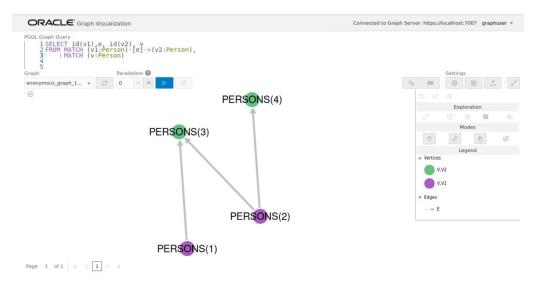
```
opg4j> graph = graph.expandGraph().
...> withPgql().
...> fromPgView("FRIENDS").
...> queryPgql("MATCH (v1:PERSON) -[e:FRIENDOF]-> (v2:PERSON) WHERE
id(v1) = 'PERSONS(2)'").
...> preparedPgqlQuery("MATCH (v:PERSON) WHERE id(v)
in ?").withStringArg(1, "PERSONS(4)").
...> expand()
graph ==> PgxGraph[name=anonymous_graph_152,N=4,E=3,created=1647347092964]
```

Java

Python

The following displays the output for the preceding PGQL query using the graph visualization tool. The subgraph is now expanded to include the friendOf relationship for PERSONS (2) in addition to PERSONS (1) which was already existing in the subgraph.

Figure 8-2 Expanding a Subgraph



Expanding a Subgraph by Explicitly Specifying the Schema Name

When expanding a graph, you can load another subgraph by reading a PG View from a different schema. For this, you must provide the schema name as an argument to the PgqlViewGraphExpander#fromPgView API. You must also ensure that you have READ permission on all the underlying metadata and data tables for the PG View.

For example:

- JShell
- Java
- Python

JShell

```
opg4j> graph = graph.expandGraph().
...> withPgql().
...> fromPgView("GRAPHUSER", "FRIENDS").
...> queryPgql("MATCH (v:Person) WHERE id(v) =
'PERSONS(1)'").
...> expand()
graph ==>
PgxGraph[name=anonymous_graph_18,N=1,E=0,created=1672848726308]
```

Java

```
graph = graph.expandGraph()
          .withPgql()
          .fromPgView("GRAPHUSER", "FRIENDS")
```



```
.queryPgql("MATCH (v:Person) WHERE id(v) = 'PERSONS(1)'")
.expand();
```

Python

```
>>> graph = graph.expand_with_pgql("MATCH (v:Person) WHERE id(v) =
'PERSONS(1)'",
... pg_view_name="FRIENDS", schema="GRAPHUSER")
>>> graph
PgxGraph(name: anonymous_graph_6, v: 2, e: 0, directed: True, memory(Mb): 0)
```

Using Merging Strategy

When expanding a graph, some vertices and edges that are in the new graph data may have already been loaded in the base graph. In such cases, if the vertex and edge property values differ for all vertices and edges that are both in the base graph and in the new graph to be merged, then the following applies:

- If the merging strategy is KEEP_CURRENT_VALUES, then the vertex and edge property values coming from the new graph are ignored.
- If the merging strategy is <code>UPDATE_WITH_NEW_VALUES</code>, then the vertex and edge property values are updated with the ones found in the new graph.

For example:

- JShell
- Java

JShell

```
opg4j> import oracle.pgx.api.expansion.PropertyMergeStrategy
opg4j> graph = graph.expandGraph().
...> withPgql().
...> fromPgView("FRIENDS").
...> queryPgql("MATCH (v1:PERSON) -[e:FRIENDOF]-> (v2:PERSON) WHERE
id(v1) = 'PERSONS(2)'").
...> preparedPgqlQuery("MATCH (v:PERSON) WHERE id(v)
in ?").withStringArg(1, "PERSONS(4)").
...>
vertexPropertiesMergingStrategy(PropertyMergeStrategy.UPDATE_WITH_NEW_VALUES)
...> expand()
```

Java

```
import oracle.pgx.api.expansion.PropertyMergeStrategy;
graph = graph.expandGraph()
```



```
.withPgql()
.fromPgView("FRIENDS")
.queryPgql("MATCH (v1:PERSON) -[e:FRIENDOF]-> (v2:PERSON)
WHERE id(v1) = 'PERSONS(2)'")
.preparedPgqlQuery("MATCH (v:PERSON) WHERE id(v)
in ?").withStringArg(1, "PERSONS(4)")
.vertexPropertiesMergingStrategy(PropertyMergeStrategy.UPD
ATE_WITH_NEW_VALUES)
.expand();
```



9

Quick Starts for Using Property Graph Views

This chapter contains quick start tutorials and other resources to help you get started on working with property graph views.

- Using Sample Data for Graph Analysis
- Quick Start: Working with Property Graph Views
 This tutorial helps you get started on creating, querying and executing graph algorithms on property graph views.
- Quick Start: Using the Python Client as a Module
 This section describes how to use the Python client as a module in Python applications.
- Oracle LiveLabs Workshops for Graphs
 You can also explore Oracle Property Graph features using the graph workshops in
 Oracle LiveLabs.

9.1 Using Sample Data for Graph Analysis

The rpm installation of the graph server provides you with sample graph data in /opt/oracle/graph/data directory.

The <code>bank_graph</code> folder contains data that represent the vertices and edges of a graph in <code>bank_nodes.csv</code> and <code>bank_edges_amt.csv</code> files respectively. You can import the graph data from these <code>.csv</code> files into the database. You can then create a graph for querying and analyses.

· Importing Data from CSV Files

9.1.1 Importing Data from CSV Files

You can import data from CSV files into the database through Oracle SQL Developer or by using Oracle Database utilities (such as SQL*Loader or External Tables).

- See Data Import Wizard in Oracle SQL Developer User's Guide on how to import data from files into tables.
- See Oracle Database Utilities for more information on data transfer utilities.

The following instructions enable you to load data into the database tables using Oracle SQL Loader.

As a prerequisite requirement, you must execute the following SQL statements to create the vertex (bank_accounts) and edge (bank_txns) tables in the database:

```
CREATE TABLE bank_accounts(id NUMBER, name VARCHAR2(10));

CREATE TABLE bank_txns(from_acct_id NUMBER, to_acct_id NUMBER, description VARCHAR2(10), amount NUMBER);
```

You can then perform the following steps to load the data:

 Create a SQL*Loader control file to load the vertices from bank_nodes.csv as shown:

```
load data
infile '<path_to_bank_nodes.csv>'
into table bank_accounts
fields terminated by "," optionally enclosed by '"'
( id, name )
```

2. Invoke SQL*Loader from the command line to load the vertices in bank_accounts table, using the preceding configuration file as shown:

```
sqlldr <dbuser>/<password> CONTROL=<path to vertex loader.ctl>
```

The bank accounts table gets successfully loaded with 1000 rows.

3. Create a SQL*Loader control file to load the edge from bank_edges_amt.csv as shown:

```
load data
infile '<path_to_bank_edges_amt.csv>'
into table bank_txns
fields terminated by "," optionally enclosed by '"'
(from acct id, to acct id, description, amount)
```

4. Invoke SQL*Loader from the command line to load the edges in bank_txns table, using the preceding configuration file as shown:

```
sqlldr <dbuser>/<password> CONTROL=<path to edge loader.ctl>
```

The bank txns table gets successfully loaded with 4996 rows.

5. Execute the following SQL statement to add the primary key constraint in the bank_accounts table:

```
ALTER TABLE bank accounts ADD PRIMARY KEY (id);
```

6. Execute the following SQL statements to add a primary key column to the bank_txns table, populate it with ROWNUM values and then define the primary key constraint:

```
ALTER TABLE bank_txns ADD txn_id NUMBER;

UPDATE bank_txns SET txn_id = ROWNUM;

COMMIT;

ALTER TABLE bank txns ADD PRIMARY KEY (txn id);
```

7. Execute the following SQL statements to add the foreign key constraints to the bank_txns table:

```
ALTER TABLE bank_txns MODIFY from_acct_id REFERENCES bank accounts(id);
```



```
ALTER TABLE bank txns MODIFY to acct id REFERENCES bank accounts(id);
```

The sample bank graph data is now available in the database tables.

9.2 Quick Start: Working with Property Graph Views

This tutorial helps you get started on creating, querying and executing graph algorithms on property graph views.

The instructions assume that you have loaded the sample bank graph data provided with the graph server installation in the database tables. See Using Sample Data for Graph Analysis for more information.

The following instructions are supported with examples that can be executed either with the OPG4J Java shell or OPG4PY Python shell or through a Java program using the PGX API.

- 1. Start the interactive graph shell CLI:
 - JShell
 - Python

JShell

```
cd /opt/oracle/graph
./bin/opg4j --no_connect
Oracle Graph Server Shell 23.1.0
```

Python

```
cd /opt/oracle/graph
./bin/opg4py --no_connect
Oracle Graph Server Shell 23.1.0
```

- 2. Obtain a JDBC database connection, if using OPG4J shell or a Java program.
 - JShell
 - Java

JShell

```
opg4j> var jdbcUrl="jdbc:oracle:thin:@<host>:<port>/<sid>"
jdbcUrl ==> "jdbc:oracle:thin:@localhost:1521/orclpdb"
opg4j> var conn =
DriverManager.getConnection(jdbcUrl,"<username>","<password>")
```



```
conn ==> oracle.jdbc.driver.T4CConnection@7d463c9f
opg4j> conn.setAutoCommit(false);
```

Java

```
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import oracle.pg.rdbms.pgql.PgqlConnection;
import oracle.pg.rdbms.pgql.PgqlStatement;
import oracle.pg.rdbms.pgql.PgqlResultSet;
import oracle.pgx.api.*;
import oracle.pg.rdbms.GraphServer;
import oracle.pg.rdbms.pgql.jdbc.PgqlJdbcRdbmsDriver;
...
...
// Get a jdbc connection
DriverManager.registerDriver(new PgqlJdbcRdbmsDriver());
String jdbcUrl="jdbc:oracle:thin:@"+<host>+":"+<port>+"/"+<sid>;
conn = DriverManager.getConnection(jdbcUrl, <username>, <password>);
conn.setAutoCommit(false);
```

3. Create a PGQL connection.

- JShell
- Java
- Python

JShell

```
opg4j> var pgqlConn = PgqlConnection.getConnection(conn)
pgqlConn ==> oracle.pg.rdbms.pgql.PgqlConnection@5c5c784c
```

Java

PgqlConnection pgqlConn = PgqlConnection.getConnection(conn);

Python

```
>>> pgql_conn =
opg4py.pgql.get_connection("<username>","<password>",
"jdbc:oracle:thin:@<host>:<port>/<sid>")
```

4. Create a PGQL statement to execute PGQL queries.

- JShell
- Java
- Python

JShell

```
opg4j> var pgqlStmt = pgqlConn.createStatement()
pgqlStmt ==> oracle.pg.rdbms.pgql.PgqlExecution@29e3c28
```

Java

```
PgqlStatement pgqlStmt = pgqlConn.createStatement();
```

Python

```
>>> pgql_statement = pgql_conn.create_statement()
```

- 5. Create a property graph view using the CREATE PROPERTY GRAPH statement:
 - JShell
 - Java
 - Python

JShell

```
opg4j> String pgql =
...> "CREATE PROPERTY GRAPH bank_graph_view "
...> + "VERTEX TABLES ( BANK_ACCOUNTS AS ACCOUNTS "
...> + "KEY (ID) "
...> + "LABEL ACCOUNTS "
...> + "PROPERTIES (ID, NAME) "
...> + "PROPERTIES (ID, NAME) "
...> + "EDGE TABLES ( BANK_TXNS AS TRANSFERS "
...> + "KEY (FROM_ACCT_ID, TO_ACCT_ID, AMOUNT) "
...> + "SOURCE KEY (FROM_ACCT_ID) REFERENCES ACCOUNTS (ID) "
...> + "DESTINATION KEY (TO_ACCT_ID) REFERENCES ACCOUNTS (ID) "
...> + "LABEL TRANSFERS "
...> + "PROPERTIES (FROM_ACCT_ID, TO_ACCT_ID, AMOUNT, DESCRIPTION) "
...> + ") OPTIONS (PG_VIEW) "
opg4j> pgqlStmt.execute(pgql)
```



Java

```
String pgql =
        "CREATE PROPERTY GRAPH " + graph + " " +
        "VERTEX TABLES ( BANK ACCOUNTS AS ACCOUNTS " +
        "KEY (ID) " +
        "LABEL ACCOUNTS " +
        "PROPERTIES (ID, NAME)" +
        ") " +
        "EDGE TABLES ( BANK TXNS AS TRANSFERS " +
        "KEY (FROM ACCT ID, TO ACCT ID, AMOUNT) " +
        "SOURCE KEY (FROM ACCT ID) REFERENCES ACCOUNTS (ID) " +
        "DESTINATION KEY (TO ACCT ID) REFERENCES ACCOUNTS (ID) " +
        "LABEL TRANSFERS " +
        "PROPERTIES (FROM_ACCT_ID, TO_ACCT_ID, AMOUNT,
DESCRIPTION) " +
       ") OPTIONS(PG_VIEW)";
     pgqlStmt.execute(pgql);
```

Python

```
>>> pggl = """
... CREATE PROPERTY GRAPH bank graph view
         VERTEX TABLES (
           BANK ACCOUNTS
            LABEL ACCOUNTS
           PROPERTIES (ID, NAME)
          )
         EDGE TABLES (
           BANK TXNS
              SOURCE KEY (FROM ACCT ID) REFERENCES BANK ACCOUNTS
. . .
(ID)
               DESTINATION KEY (TO ACCT ID) REFERENCES
. . .
BANK ACCOUNTS (ID)
             LABEL TRANSFERS
              PROPERTIES (FROM ACCT ID, TO ACCT ID, AMOUNT,
DESCRIPTION)
          ) OPTIONS (PG VIEW)
>>> pgql statement.execute(pgql)
False
```

The property graph view bank_graph_view gets created successfully.

- **6.** Execute the following query to retrieve the first 10 elements of the graph as shown:
 - JShell



- Java
- Python

JShell

```
opq4j> String pgqlQuery =
...> "SELECT e.from acct id, e.to acct id, e.amount FROM "
...> + "MATCH (n:ACCOUNTS) -[e:TRANSFERS]-> (m:ACCOUNTS) ON
BANK GRAPH VIEW "
...> + "LIMIT 10"
opg4j> var rs = pgqlStmt.executeQuery(pgqlQuery)
rs ==> oracle.pg.rdbms.pgql.pgview.PgViewResultSet@1e368085
opg4j> rs.print()
+----+
| FROM ACCT ID | TO ACCT ID | AMOUNT |
| 27
| 122
       | 606
| 495
| 640
| 146
                      | 1000
| 122
           | 606
                      | 1000
| 122
                      | 1000
                   | 1000
| 1000
| 122
| 122
           | 140
                       | 1000
| 123
           | 95
| 123 | 130
                  | 1000
$16 ==> oracle.pg.rdbms.pgql.pgview.PgViewResultSet@1e368085
```

Java

```
String pgqlQuery =
          "SELECT e.from_acct_id, e.to_acct_id, e.amount FROM " +
          "MATCH (n:ACCOUNTS) -[e:TRANSFERS]-> (m:ACCOUNTS) ON
BANK_GRAPH_VIEW " +
          "LIMIT 10";
PgqlResultSet rs = pgqlStmt.executeQuery(pgqlQuery);
rs.print();
```



	121	221	1000	
	122	27	1000	
	122	606	1000	
	122	495	1000	
	122	640	1000	
	122	140	1000	
	123	95	1000	
	123	130	1000	
+-		 	 	+

7. Load the graph into the graph server (PGX). This will enable you to run a variety of different built-in algorithms on the graph and will also improve query performance for larger graphs.

- JShell
- Java
- Python

JShell

```
opg4j> var instance = GraphServer.getInstance("https://
localhost:7007", "<username>", "<password>".toCharArray())
instance ==> ServerInstance[embedded=false,baseUrl=https://
localhost:7007]
opg4j> var session = instance.createSession("mySession")
session ==>
PgxSession[ID=43653128-59cd-4e69-992c-la2beac05857,source=mySession]
opg4j> var graph =
session.readGraphByName("BANK_GRAPH_VIEW",GraphSource.PG_VIEW)
graph ==>
PgxGraph[name=BANK_GRAPH_VIEW,N=1000,E=4996,created=1643308582055]
```

Java

```
ServerInstance instance = GraphServer.getInstance("https://
localhost:7007", "<username>", "<password>".toCharArray());
PgxSession session = instance.createSession("my-session");
PgxGraph graph =
session.readGraphByName("BANK GRAPH VIEW",GraphSource.PG VIEW);
```

```
>>> instance = graph_server.get_instance("https://
localhost:7007","<username>","<password>")
>>> session = instance.create_session("my_session")
>>> graph = session.read_graph_by_name('BANK_GRAPH_VIEW', 'pg_view')
>>> graph
```



```
PgxGraph(name: BG_PY_VIEW, v: 1000, e: 4996, directed: True, memory(Mb):
0)
```

- 8. Execute the PageRank algorithm as shown:
 - JShell
 - Java
 - Python

JShell

```
opg4j> var analyst = session.createAnalyst()
analyst ==> NamedArgumentAnalyst[session=3f0a9a71-f349-4aac-b75f-a7c4ae50851b]
opg4j> analyst.pagerank(graph)
$10 ==> VertexProperty[name=pagerank,type=double,graph=BANK GRAPH VIEW]
```

Java

```
Analyst analyst = session.createAnalyst();
analyst.pagerank(graphView);
```

Python

```
>>> analyst = session.create_analyst()
>>> analyst.pagerank(graph)
VertexProperty(name: pagerank, type: double, graph: BANK GRAPH VIEW)
```

- **9.** Query the graph to list the top 10 accounts by pagerank:
 - JShell
 - Java
 - Python

JShell



Java

```
String pgQuery = "SELECT a.id, a.pagerank FROM MATCH (a) ON BANK_GRAPH_VIEW ORDER BY a.pagerank DESC LIMIT 10"; session.queryPgql(pgQuery).print();
```

Python

9.3 Quick Start: Using the Python Client as a Module

This section describes how to use the Python client as a module in Python applications.

Remote Server

For this mode, all you need is the Python client to be installed. In your Python program, you must authenticate with the remote server before you can create a

session as illustrated in the following example. Note that you must replace the values for base url, jdbc url, username, and password with values to match your environment details.

```
import pypgx
import opg4py
import opg4py.graph server as graph server
pgql conn = opg4py.pgql.get connection("<username>","<password>",
"<jdbc url>")
pgql statement = pgql_conn.create_statement()
pgql = """
        CREATE PROPERTY GRAPH bank graph
        VERTEX TABLES (
          bank accounts
            LABEL ACCOUNTS
            PROPERTIES (ID, NAME)
        EDGE TABLES (
          bank txns
            SOURCE KEY (from acct id) REFERENCES bank accounts
            DESTINATION KEY (to_acct_id) REFERENCES bank_accounts
            LABEL TRANSFERS
            PROPERTIES (FROM ACCT ID, TO ACCT ID, AMOUNT, DESCRIPTION)
        ) OPTIONS (PG VIEW)
11 11 11
pgql statement.execute(pgql)
instance = graph server.get instance("<base url>", "<username>",
"<password>")
session = instance.create session("my session")
graph = session.read graph by name('BANK GRAPH', 'pg view')
analyst = session.create analyst()
analyst.pagerank(graph)
rs = graph.query pgql("SELECT id(x), x.pagerank FROM MATCH (x) LIMIT 5")
rs.print()
```

To execute, save the above program into a file named program.py and run the following command:

```
python3 program.py
```

You will see the following output:

+	 	-+
id(x)	pagerank	
+	 	+
BANK ACCOUNTS(2)	9.749447313256548E-4	
BANK ACCOUNTS(4)	0.004584001759076056	
BANK ACCOUNTS(6)	5.358461393401424E-4	
BANK ACCOUNTS(8)	0.0013051552434930175	
BANK ACCOUNTS (10)	0.0015040122009364232	
+	 	-+

Converting PGQL result set into pandas dataframe



Additionally, you can also convert the PGQL result set to a pandas. DataFrame object using the to_pandas() method. This makes it easier to perform various data filtering operations on the result set and it can also be used in Lambda functions. For example,

```
example_query = (
    "SELECT n.name AS name, n.age AS age "
    "WHERE (n)"
)
result_set = sample_graph.query_pgql(example_query)
result_df = result_set.to_pandas()

result_df['age_bin'] = result_df['age'].apply(lambda x: int(x)/20) #
create age bins based on age ranges
```



To view the complete set of available Python APIs, see OPG4PY Python API Reference.

Embedded Server

For this mode, the Python client and the Graph Server RPM package must be installed on the same machine.

```
import os
os.environ["PGX_CLASSPATH"] = "/opt/oracle/graph/lib/*"
instance = graph_server.get_embedded_instance()
session = instance.create_session("python_pgx_client")
print(session)
```

To execute, save the above program into a file named program.py and run the following command.

```
python3 program.py
```

After successful login, you must see a similar message indicating a PGX session was created:

```
PgxSession(id: 32fc7037-18f1-4381-ba94-107e5f63aec2, name: python pgx client)
```



To view the complete set of available Python APIs, see OPG4PY Python API Reference.



9.4 Oracle LiveLabs Workshops for Graphs

You can also explore Oracle Property Graph features using the graph workshops in Oracle LiveLabs.

See the Oracle LiveLabs Workshop for a complete example on querying, analyzing and visualizing graphs using data stored in a free tier Autonomous Database instance. You will provision a new free tier Autonomous Database instance, load data into it, create a graph, and then query, analyze and visualize the graph.



10

Getting Started with the Client Tools

You can use multiple client tools to interact with the graph server (PGX) or directly with the graph data in the database.

The following sections explain how to use the various client tools:

• Interactive Graph Shell CLIs

Both the Oracle Graph server and client packages contain interactive command-line applications for interacting with the Java APIs and the Python APIs of the product, locally or on remote computers.

Using Autonomous Database Graph Client

Using the AdbGraphClient API, you can access Graph Studio features in Autonomous Database programmatically using the Oracle Graph Client or through your Java or Python application.

Using the Graph Visualization Web Client

You can use the Graph Visualization application to visualize graphs that are either loaded into the graph server (PGX) or stored in the database.

Using the Jupyter Notebook Interface

You can use the Jupyter notebook interface to create, load, and query property graphs through Python.

Additional Client Tools for Querying PG Views

When working with property graph views (PG Views) in the database, you can use other supported client tools.

Related Topics

Oracle Graph Client Installation

You can interact with the various graph features using the client CLIs and the graph visualization web client.

10.1 Interactive Graph Shell CLIs

Both the Oracle Graph server and client packages contain interactive command-line applications for interacting with the Java APIs and the Python APIs of the product, locally or on remote computers.

The interactive graph shells dynamically interpret command-line inputs from the user, execute them by invoking the underlying functionality, and can print results or process them further. The graph shells provide a lightweight and interactive way of exercising graph functionality without creating a Java or Python application.

The graph shells are especially helpful if you want to do any of the following:

- Quickly run a "one-off" graph analysis on a specific data set, rather than creating a large application
- Run getting started examples and create demos on a sample data set
- Explore the data set, trying different graph analyses on the data set interactively

- Learn how to use the product and develop a sense of what the built-in algorithms are good for
- Develop and test custom graph analytics algorithms

The graph shell for the Java API (OPG4J) is implemented on top of the Java Shell tool (JShell). As such, it inherits all features provided by JShell such as tab-completion, history, reverse search, semicolon inference, script files, and internal variables. The graph shell for the Python API (OPG4Py) uses IPython in case it is installed.

The following sections explain in detail on how to start the graph shell CLIs:

- Starting the OPG4J Shell
- Starting the OPG4Py Shell

See Also:

- Java API Reference for information on the Java APIs
- Python API Reference for information on the Python APIs

10.1.1 Starting the OPG4J Shell

Launching the OPG4J Shell

The Java shell executables are found in <code>/opt/oracle/graph/bin</code> after the graph server (PGX) installation, and in <code><CLIENT_INSTALL_DIR>/bin</code> after the Java client installation.

The OPG4J shell uses JShell, which means the shell needs to run on Java 11 or later. See Installing the Java Client From the Graph Server and Client Downloads for more details on the prerequisites. You can then launch the OPG4J shell by entering the following in your terminal:

```
cd /opt/oracle/graph
./bin/opg4j
```

When the shell has started, the following command line prompt appears:

```
For an introduction type: /help intro
Oracle Graph Server Shell 23.1.0
Variables instance, session, and analyst ready to use.
opg4j>
```

By default, the OPG4J shell creates a local PGX instance, to run graph functions in the same JVM as the shell as described in Developing Applications Using Graph Server Functionality as a Library.



Command-line Options

To view the list of available command-line options, add --help to the opg4j command:

```
./bin/opg4j --help
```

To start the opg4j shell without connecting to the graph server (PGX), use the --no_connect option as shown:

```
./bin/opg4j --no connect
```

Starting the OPG4J Shell on Remote Mode

The OPG4J shell can connect to a graph server (PGX) instance that is running on another JVM (possibly on a different machine). In order to launch the OPG4J shell in remote mode, you must specify the --base_url parameter as shown:

```
./bin/opg4j --base url https://<host>:7007 --username <graphuser>
```

where:

- <host>: is the server host
- <graphuser>: is the database user
 You will be prompted for the database password.

Note:

The graph server (PGX), listens on port 7007 by default. If needed, you can configure the graph server to listen on a different port by changing the port value in the server configuration file (server.conf). See Configuring the Graph Server (PGX) for details.

When the shell has started, the following command line prompt appears:

```
Oracle Graph Server Shell 23.1.0 Variables instance, session, and analyst ready to use. opg4j>
```

If you have multiple versions of Java installed, you can easily switch between installations by setting the JAVA HOME variable before starting the shell. For example:

```
export JAVA HOME=/usr/lib/jvm/java-11-oracle
```

Batch Execution of Scripts

The OPG4J shell can execute a script by passing the path(s) to the script(s) to the opg4j command. For example:

```
./bin/opg4j /path/to/script.jsh
```



Predefined Functions

The OPG4J shell provides the following utility functions:

- println(String): A shorthand for System.out.println(String).
- loglevel (String loggerName, String levelName): A convenient function to set the loglevel.

The loglevel function allows you to set the log level for a logger. For example, loglevel ("ROOT", "INFO") sets the level of the root logger to INFO. This causes all logs of INFO and higher (WARN, ERROR, FATAL) to be printed to the console.

Script Arguments

You can also provide parameters to the script executed by the graph server (PGX). For example:

```
./bin/opg4j /path/to/script.jsh script-arg-1 script-arg-2
```

The script /path/to/script.jsh can then access the arguments through the arguments.scriptArgs variable. The arguments are provided as an array of strings (String[]). For example:

```
Arrays.stream(arguments.scriptArgs).forEach((a) ->
    System.out.println(a));
```

The preceding example prints the output as shown:

```
script-arg-1
script-arg-2
```

Staying in Interactive Mode

By default, the OPG4J shell exits after it finishes execution. To stay in interactive mode after the script finishes *successfully*, pass the --keep_running flag to the shell. For example:

```
./bin/opg4j -b https://myserver.com:7007/ /path/to/script.jsh --
keep_running
```

10.1.2 Starting the OPG4Py Shell

Launching the OPG4Py Shell

The OPG4Py shell executables are found in $\protect\operatorname{\mathsf{OPT}}\protect\operatorname{\mathsf{OPG4Py}}\protect\operatorname{\mathsf{Shell}}\protect\operatorname{\mathsf{executables}}\protect\operatorname{\mathsf{argenthoin}}\protect\operatorname{\mathsf{after}}\protect\operatorname{\mathsf{the}}\protect\operatorname{\mathsf{Python}}\protect\operatorname{\mathsf{client}}\protect\operatorname{\mathsf{INSTALL}}\protect\operatorname{\mathsf{DIR}}\protect\operatorname{\mathsf{After}}\protect\operatorname{\mathsf{the}}\protect\operatorname{\mathsf{Python}}\protect\operatorname{\mathsf{client}}\protect\operatorname{\mathsf{installation}}.$



Before launching the OPG4Py shell, verify that your system meets the prerequisites explained in Prerequisites for Installing the Python Client. You can then launch the OPG4Py shell by entering the following in your terminal:

```
cd /opt/oracle/graph
./bin/opg4py
```

When the shell has started, the following command line prompt appears:

```
Oracle Graph Server Shell 23.1.0 >>>
```

If IPython is installed the following prompt will appear:

```
In [1]:
```

By default, the OPG4Py shell creates a local PGX instance, to run graph functions in the same JVM as the shell as described in Developing Applications Using Graph Server Functionality as a Library.

Command-line Options

To view the list of available command-line options, add --help to the opg4py command:

```
./bin/opg4py --help
```

To start the PyPGX shell without connecting to the graph server (PGX), use the -- no connect option as shown:

```
./bin/opg4py --no connect
```

Starting the OPG4Py Shell on Remote Mode

The OPG4Py shell can connect to a graph server (PGX) instance that is running on another JVM (possibly on a different machine). In order to launch the OPG4Py shell in remote mode, you must specify the --base url parameter as shown:

```
./bin/opg4py --base_url https://<host>:7007 --username <graphuser>
```

where:

- <host>: is the server host
- <graphuser>: is the database user
 You will be prompted for the database password.



Note:

The graph server (PGX), listens on port 7007 by default. If needed, you can configure the graph server to listen on a different port by changing the port value in the server configuration file (server.conf). See Configuring the Graph Server (PGX) for details.

When the OPG4Py shell has started, the following command line prompt appears:

```
Oracle Graph Server Shell 23.1.0 >>>
```

10.2 Using Autonomous Database Graph Client

Using the AdbGraphClient API, you can access Graph Studio features in Autonomous Database programmatically using the Oracle Graph Client or through your Java or Python application.

This API provides the following capabilities:

- Authenticate with Autonomous Database
- Manage the Graph Studio environment
- Execute graph queries and algorithms against the graph server (PGX)
- Execute graph queries directly against Oracle Database

To use the AdbGraphClient API, you must have access to Oracle Graph Client installation. The API is provided by the Oracle Graph Client library which is a part of the Oracle Graph Server and Client distribution. See Installing Oracle Graph Client on how to install and get started with the graph client shell CLIs for Java or Python.

Also, prior to using the Autonomous Database Graph Client, ensure you meet all the prerequisite requirements explained in Prerequisites for Using Autonomous Database Graph Client.

The following example shows using the AdbGraphClient API to establish a connection to Graph Studio, start an environment with allocated memory, load a PG View graph into memory, execute PGQL queries and run algorithms against the graph.



See the Javadoc and Python API Reference for more information on AdbGraphClient API.

- Start the interactive graph shell CLI and connect to your Autonomous Database instance as shown:
 - JShell



- Java
- Python

JShell

```
cd /opt/oracle/graph
./bin/opg4j --no connect
For an introduction type: /help intro
Oracle Graph Server Shell 23.1.0
opg4j> import oracle.pg.rdbms.*
opg4j> var config = AdbGraphClientConfiguration.builder()
opg4j> config.database("<DB name>")
opg4j> config.tenancyOcid("<tenancy OCID>")
opg4j> config.databaseOcid("<database OCID>")
opg4j> config.username("ADBDEV")
opg4j> config.password("<password for ADBDEV>")
opg4j> config.endpoint("https://<hostname-
prefix>.adb.<region>.oraclecloudapps.com/")
opg4j> var client = new AdbGraphClient(config.build())
client ==> oracle.pg.rdbms.AdbGraphClient@7b8d1537
Java
import oracle.pg.rdbms.*;
var config = AdbGraphClientConfiguration.builder();
config.tenancyOcid("<tenancy OCID>");
```

```
config.tenancyOcid("<tenancy_OCID>");
config.databaseOcid("<database_OCID>");
config.database("<DB_name>");
config.username("ADBDEV");
config.password("<password_for_ADBDEV>");
config.endpoint("https://<hostname-
prefix>.adb.
```

var client = new AdbGraphClient(config.build());



2. Start the PGX server environment with the desired memory as shown in the following code.

This submits a job in Graph Studio for environment creation. job.get() waits for the environment to get started. You can always verify if the environment has started successfully with client.isAttached(). The method returns a boolean true if the environment is running.

However, you can skip the step of creating an environment, if client.isAttached() returns true in the first step of the code.

- JShell
- Java
- Python

JShell

```
opg4j> client.isAttached()
$9 ==> false
opg4j> var job=client.startEnvironment(10)
job ==> oracle.pg.rdbms.Job@117e9a56[Not completed]
opg4j> job.get()
$11 ==> null
opg4j> job.getName()
$11 ==> "Environment Creation - 16 GBs"
opg4j> job.getType()
$12 ==> ENVIRONMENT_CREATION
opg4j> job.getCreatedBy()
$13 ==> "ADBDEV"
opg4j> client.isAttached()
$11 ==> true
```

Java

```
if (!client.isAttached()) {
        var job = client.startEnvironment(10);
        job.get();
        System.out.println("job details: name=" + job.getName() +
"type= " + job.getType() +"created_by= " + job.getCreatedBy());
    }
job details: name=Environment Creation - 16 GBstype=
ENVIRONMENT CREATIONcreated by= ADBDEV
```

```
>>> client.is_attached()
False
>>> job = client.start_environment(10)
>>> job.get()
```



```
>>> job.get_name()
'Environment Creation - 16 GBs'
>>> job.get_created_by()
'ADBDEV'
>>> client.is_attached()
True
```

- 3. Create an instance and a session object as shown:
 - JShell
 - Java
 - Python

JShell

```
opg4j> var instance = client.getPgxInstance()
instance ==> ServerInstance[embedded=false,baseUrl=https://<hostname-
prefix>.adb.<region>.oraclecloudapps.com/graph/pgx]
opg4j> var session = instance.createSession("AdbGraphSession")
session ==> PgxSession[ID=c403be26-
ad0c-45cf-87b7-1da2a48bda54,source=AdbGraphSession]
```

Java

```
ServerInstance instance = client.getPgxInstance();
PgxSession session = instance.createSession("AdbGraphSession");
```

```
>>> instance = client.get_pgx_instance()
>>> session = instance.create_session("adb-session")
```

- 4. Load a PGView graph from your Autonomous Database instance into memory.
 - JShell
 - Java
 - Python



JShell

```
opg4j> var graph = session.readGraphByName("BANK_GRAPH",
GraphSource.PG_VIEW)
graph ==>
PgxGraph[name=BANK GRAPH, N=1000, E=5001, created=1647800790654]
```

Java

```
PgxGraph graph = session.readGraphByName("BANK_GRAPH",
GraphSource.PG VIEW);
```

Python

```
>>> graph = session.read_graph_by_name("BANK_GRAPH", "pg_view")
```

- **5.** Create an Analyst and execute a Pagerank algorithm on the graph as shown:
 - JShell
 - Java
 - Python

JShell

```
opg4j> session.createAnalyst().pagerank(graph)
$16 ==> VertexProperty[name=pagerank,type=double,graph=BANK GRAPH]
```

Java

```
session.createAnalyst().pagerank(graph);
```

```
>>> session.create_analyst().pagerank(graph)
VertexProperty(name: pagerank, type: double, graph: BANK GRAPH)
```

- **6.** Execute a PGQL query on the graph and print the result set as shown:
 - JShell
 - Java



Python

JShell

opg4j> graph.queryPgql("SELECT a.acct_id AS source, a.pagerank, t.amount, b.acct_id AS destination FROM MATCH (a)-[t]->(b) ORDER BY a.pagerank DESC LIMIT 3").print()

Java

```
PgqlResultSet rs = graph.queryPgql("SELECT a.acct_id AS source,
a.pagerank, t.amount, b.acct_id AS destination FROM MATCH (a)-[t]->(b)
ORDER BY a.pagerank DESC LIMIT 3");
rs.print();
```

Python

```
>>> rs = graph.query_pgql("SELECT a.acct_id AS source, a.pagerank,
t.amount, b.acct_id AS destination FROM MATCH (a)-[t]->(b) ORDER BY
a.pagerank DESC LIMIT 3").print()
```

On execution, the query produces the following output:

387	+	source		pagerank	 	amount		destination	+ +
	İ	387	İ	0.007302836252205922	İ	1000.0	İ	374	

7. Optionally, you can execute a PGQL query directly against the graph in the database as shown in the following code.

In order to establish a JDBC connection to the database, you must download the wallet and save it in a secure location. See JDBC Thin Connections with a Wallet on how to determine the JDBC URL connection string.

- JShell
- Java
- Python

JShell

```
opg4j> String jdbcUrl="jdbc:oracle:thin:@<tns_alias>?
TNS ADMIN=<path to wallet>"
```



```
opg4j> var conn =
DriverManager.getConnection(jdbcUrl,"ADBDEV","<password_for_ADBDEV>")
conn ==> oracle.jdbc.driver.T4CConnection@36ee8c7b
opg4j> var pgqlConn = PgqlConnection.getConnection(conn)
pgqlConn ==> oracle.pg.rdbms.pgql.PgqlConnection@5f27d271
opg4j> var pgqlStmt = pgqlConn.createStatement()
pgqlStmt ==> oracle.pg.rdbms.pgql.PgqlExecution@4349f52c
opg4j> pgqlStmt.executeQuery("SELECT a.acct_id AS source, t.amount, b.acct_id AS destination FROM MATCH (a)-[t]->(b) ON BANK_GRAPH
LIMIT 3").print()
```

Java

```
import oracle.pg.rdbms.pgql.PgqlConnection;
import oracle.pg.rdbms.pgql.PgqlStatement;
import oracle.pg.rdbms.pgql.PgqlResultSet;
import oracle.pgx.api.*;
import oracle.pq.rdbms.GraphServer;
import oracle.pg.rdbms.pgql.jdbc.PgqlJdbcRdbmsDriver;
DriverManager.registerDriver(new PgqlJdbcRdbmsDriver());
String jdbcUrl="jdbc:oracle:thin:@<tns alias>?
TNS ADMIN=<path to wallet>";
Connection conn =
DriverManager.getConnection(jdbcUrl,"ADBDEV","<password for ADBDEV>"
PgqlConnection pgqlConn = PgqlConnection.getConnection(conn);
PgqlStatement pgqlStmt = pgqlConn.createStatement();
PgqlResultSet rs = pgqlStmt.executeQuery("SELECT a.acct id AS
source, t.amount, b.acct id AS destination FROM MATCH (a)-[t]->(b)
ON BANK GRAPH LIMIT 3");
rs.print();
```

Python

```
>>> jdbcUrl = "jdbc:oracle:thin:@<tns_alias>?
TNS_ADMIN=<path_to_wallet>"
>>> pgql_conn =
opg4py.pgql.get_connection("ADBDEV","<password_for_ADBDEV>",
jdbcUrl)
>>> pgql_statement = pgql_conn.create_statement()
>>> pgql_statement.execute_query("SELECT a.acct_id AS source,
t.amount, b.acct_id AS destination FROM MATCH (a)-[t]->(b) ON
BANK GRAPH LIMIT 3").print()
```

On execution, the query produces the following output:

```
+----+
| SOURCE | AMOUNT | DESTINATION |
```



```
| 1000 | 1000 | 921 | |
| 1000 | 1000 | 662 | |
| 1000 | 1000 | 506 |
```

8. Close the session after executing all graph queries as shown:

- JShell
- Java
- Python

JShell

```
opg4j> session.close()
```

Java

opg4j> session.close();

Python

>>> session.close()

• Prerequisites for Using Autonomous Database Graph Client

10.2.1 Prerequisites for Using Autonomous Database Graph Client

As a prerequisite requirement to get started with the AdbGraphClient API, you must:

- Provision an Autonomous Database instance in Oracle Autonomous Database.
- Obtain the following tenancy details to connect:

Key	Description	More Information
tenancy OCID	The Oracle Cloud ID (OCID) of your tenancy	To determine the OCID for your tenancy, see "Where to Find your Tenancy's OCID" in: Oracle Cloud Infrastructure
		Documentation.



Key	Description	Мо	re Information	
databas e	Database name of your Autonomous Database instance		Open the OCI console and click Oracle Database in the left navigation menu.	
		2.	Click Autonomous Database and navigate to the Autonomous Databases page.	
		3.	Select the required Autonomous Database under the Display Name column and navigate to the Autonomous Database Details page.	
		4.	Note the Database Name under "General Information" in the Autonomous Database Information tab.	
databas e OCID	The Oracle Cloud ID (OCID) of your Autonomous Database	1.	Open the OCI console and click Oracle Database in the left navigation menu.	
		2.	Click Autonomous Database and navigate to the Autonomous Databases page.	
		3.	Select the required Autonomous Database under the Display Name column and navigate to the Autonomous Database Details page.	
		4.	Note the Database OCID under "General Information" in the Autonomous Database Information tab.	
userna me	Graph enabled Autonomous Database username, used for logging into Graph Studio	See Create a Graph User for more information.		
passwor d	Database password for the graph user	If the password for a graph user is forgotten, then you can always reset password for the graph user by logging into Database Actions as the ADMIN user. See Edit User for more information.		
endpoint	Graph Studio endpoint URL	1.	Select your Autonomous Database instance and navigate to the the Autonomous Database Details page.	
		2.	Click the Tools tab.	
		3.	Click on Graph Studio .	
		4.	Copy the URL of the new tab that opens the Graph Studio login screen.	
		5.	Edit the URL to remove the part after oraclecloudapps.com to obtain the endpoint URL For example, the following shows the format of a sample endpoint URL:	
			https:// <hostname_prefix>.adb.<region_identifier: .oraclecloudapps.com<="" td=""></region_identifier:></hostname_prefix>	

- Access Graph Studio and create a PG View graph.
- Download, install and start the Oracle Graph Java or Python client.



10.3 Using the Graph Visualization Web Client

You can use the Graph Visualization application to visualize graphs that are either loaded into the graph server (PGX) or stored in the database.

To run the graph visualization application for your installation, see Graph Visualization Web Client.

Related Topics

Graph Visualization Application
 The Graph Visualization application enables interactive exploration and visualization of property graphs. It can also visualize graphs stored in the database.

10.4 Using the Jupyter Notebook Interface

You can use the Jupyter notebook interface to create, load, and query property graphs through Python.

Perform the following steps to perform graph analysis using Jupyter Notebook:

1. Install the Jupyter Notebook application following the Jupyter documentation. The following example installs Jupyter with pip:

```
pip3 install --user jupyter
```

- 2. Ensure that your Jupyter installation is added to the PATH environment variable.
- 3. Run the notebook server using the jupyter notebook command.
- 4. Launch the web application using the generated URL and open a new notebook.
- 5. Create and analyse a property graph.
 - The following example shows creating a property graph view (PG View) and running graph queries:



Figure 10-1 Creating a PG View in Jupyter Notebook

 The following example shows loading the PG View into the graph server (PGX) and running graph algorithms for analysis:

Figure 10-2 Running Graph Algorithms in Jupyter Notebook

10.5 Additional Client Tools for Querying PG Views

When working with property graph views (PG Views) in the database, you can use other supported client tools.

- Using Oracle SQLcl
 You can access the graph in the database using SQLcl.
- Using Oracle SQL Developer
 You can use Oracle SQL Developer to execute PGQL statements and queries directly on graphs in Oracle Database.

10.5.1 Using Oracle SQLcl

You can access the graph in the database using SQLcl.

You can run PGQL queries on the graph in SQLcl with a plug-in that is available with Oracle Graph Server and Client. See PGQL Plug-in for SQLcl in *Oracle SQLcl User's Guide* for more information.

The example in this section helps you get started on executing PGQL queries on a graph in SQLcl. As a prerequisite, to perform the steps in the example, you must set up the bank graph data in your database schema using the sample data provided with the graph server installation. See Using Sample Data for Graph Analysis for more information.

The following example creates a property graph view using the PGQL CREATE PROPERTY GRAPH statement, executes PGQL queries against the graph and finally drops the graph using SQLcl.

1. Start SQLcl with your database schema credentials. In the following command, *graphuser* is the database user used to connect to SQLcl.

```
sql graphuser/<password_for_graphuser>@<tns_alias>

SQLcl: Release 21.2 Production on Sun Jan 30 04:30:09 2022
Copyright (c) 1982, 2022, Oracle. All rights reserved.
Connected to:
Oracle Database 21c Enterprise Edition Release 21.0.0.0.0 - Production
Version 21.3.0.0.0
```

2. Enable PGQL mode as shown:

```
SQL> pgql auto on;
PGQL Auto enabled for schema=[null], graph=[null], execute=[true],
translate=[false]
```

Note that no arguments are used in the preceding PGQL command.

3. Create a property graph view on the bank graph data tables.

```
PGQL> CREATE PROPERTY GRAPH bank graph
 2
           VERTEX TABLES (
 3
              bank accounts
                 LABEL ACCOUNTS
                 PROPERTIES (ID, NAME)
  6
             )
  7
             EDGE TABLES (
 8
               bank txns
 9
                 SOURCE KEY (from acct id) REFERENCES bank accounts (id)
                 DESTINATION KEY (to acct_id) REFERENCES bank_accounts
 10
(id)
11
                 LABEL TRANSFERS
12
                 PROPERTIES (FROM ACCT ID, TO ACCT ID, AMOUNT,
DESCRIPTION)
13*
            ) OPTIONS (PG VIEW);
```



Graph created

Set bank_graph as the default graph using the graph argument when enabling PGQL mode.

```
PGQL> pgql auto on graph bank_graph;

PGQL Auto enabled for schema=[null], graph=[BANK_GRAPH],
execute=[true], translate=[false]
```

5. Execute PGQL queries against the default graph. For example, the following PGQL query retrieves the total number of vertices as shown:

```
PGQL> SELECT COUNT(*) AS num_vertices FROM MATCH(n);

NUM_VERTICES

1000
```

Note that in the preceding query, the graph name is not specified using the \mbox{ON} clause as part of the MATCH clause.

6. Reconnect to SQLcl as another schema user.

```
PGQL> conn system/<password_for_system>@<tns_alias>;
Connected.
```

7. Enable PGQL mode using the schema argument to set the default schema used for creating the graph. Also, set bank_graph as the default graph using the graph argument:

```
PGQL> pgql auto on schema graphuser graph bank_graph;

PGQL Auto enabled for schema=[graphuser], graph=[BANK_GRAPH], execute=[true], translate=[false]
```

8. Execute a PGQL query to retrieve all the edge properties on the graph as shown:

PGQL> SELECT e.* FROM MATCH (n:accounts) -[e:transfers]->
(m:accounts) LIMIT 10;

AMOUNT	DESCRIPTION	FROM_ACCT_ID	TO_ACCT_ID
1000	transfer	178	921
1000	transfer	178	462
1000	transfer	179	688
1000	transfer	179	166
1000	transfer	179	397
1000	transfer	179	384
1000	transfer	179	900
1000	transfer	180	855
1000	transfer	180	984
1000	transfer	180	352



10 rows selected.

Therefore, you can set a default schema and execute PGQL queries against a default graph in SQLcl.

9. Finally, drop the graph after executing the required graph queries.

```
PGQL> DROP PROPERTY GRAPH bank_graph;
Graph dropped
```

Also, see Execute PGQL Queries in SQLcl for more information.

10.5.2 Using Oracle SQL Developer

You can use Oracle SQL Developer to execute PGQL statements and queries directly on graphs in Oracle Database.

- About Property Graph Support in SQL Developer
 Starting from SQL Developer Release 22.2 onwards, property graph support is available in SQL Developer. Therefore, you can run PGQL queries on property graphs using SQL Developer.
- Working with Property Graph Views
 You can view all the property graph views existing in your database schema by
 expanding PG Views under the Property Graph node in the Connections navigator.

10.5.2.1 About Property Graph Support in SQL Developer

Starting from SQL Developer Release 22.2 onwards, property graph support is available in SQL Developer. Therefore, you can run PGQL queries on property graphs using SQL Developer.

You can access the **Property Graph** node in the Connections navigator to work with property graphs.



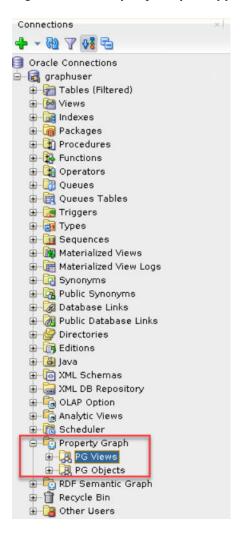


Figure 10-3 Property Graph Support in SQL Developer

Using SQL Developer, you can perform the following PGQL operations in the PGQL Worksheet:

- Create a property graph view or a property graph object against the graph data in the database.
- Apply a filter on the PG Views or PG Objects node to list graphs based on the specified filter criteria.
- Execute a PGQL INSERT, UPDATE, SELECT, or DELETE guery against the graph.
- Drop the property graph view or the property graph object.

10.5.2.2 Working with Property Graph Views

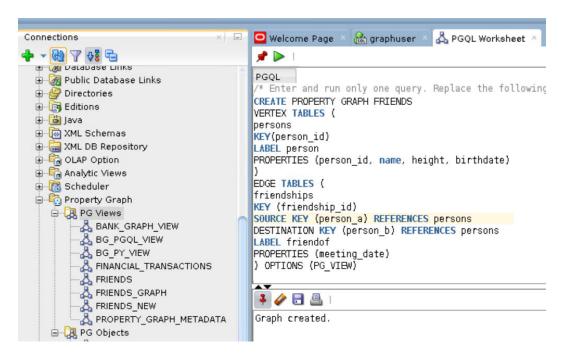
You can view all the property graph views existing in your database schema by expanding **PG Views** under the **Property Graph** node in the **Connections** navigator.

You can run PGQL queries for a property graph view in a **PGQL Worksheet**. Note that you can execute only one PGQL query at a time in the worksheet.

The following steps show a few examples for creating, updating and dropping a property graph view using SQL Developer.

- 1. Right-click the **Property Graph** node and select **Open PGQL Worksheet**.
 - **PGQL Worksheet** opens in a new tab and it contains the **Run Query** icon for executing PGQL queries.
- 2. Create a property graph view by running a CREATE PROPERTY GRAPH statement in the PGQL Worksheet. For example:

Figure 10-4 Create a Property Graph View



The result of the query execution is displayed in the bottom pane of the Editor. On successful query execution, you can right click and refresh the **PG Views** object to view the newly created graph under **PG Views**.

3. Click on the newly created graph.

This opens a **PGQL Worksheet** in a new tab with the following default query:

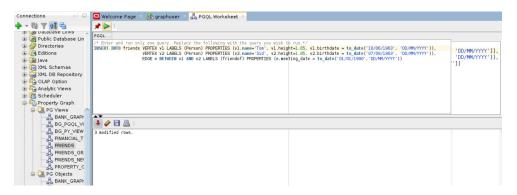
```
SELECT e, v, n FROM MATCH (v)-[e]-(n) ON \langle graph \ name \rangle LIMIT 100
```

4. Run any PGQL update query like performing an INSERT or an UPDATE operation against a property graph view.

For example, the following shows the execution of a PGQL INSERT query:



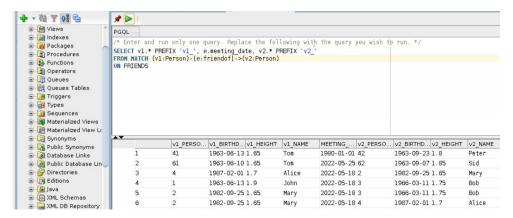
Figure 10-5 Running a PGQL INSERT Query



The vertices and edges are added to the graph.

5. Run a PGQL SELECT query to view the newly inserted vertices and edges as shown:

Figure 10-6 Running a PGQL SELECT Query



6. Delete the Property Graph View as shown:

Connections × Welcome Page A PGQL Worksheet A 💠 🕶 🚯 🔻 🚭 📌 ⊳ | ▼ Actions... 🗓 📺 Queues Tables PGQL 🗓 🎆 Triggers /* Enter and run only one query 🗓 📷 Types DROP PROPERTY GRAPH friends 🗓 🔯 Synonyms ⊕ 🍘 Database Links ⊕ ❷ Directories 🗓 🥞 Editions 🗓 🛅 Java 🕀 🔚 XML DB Repository ⊕ □ OLAP Option 🖨 🔓 Analytic Views 3 🤣 🖯 🔠 I 🗓 🔞 Scheduler 🖨 👣 Property Graph Graph dropped. PG Views 🙏 BANK_GRAPH_VIEW 🙏 BG_PGQL_VIEW BG_PY_VIEW 🚣 FINANCIAL_TRANSACTIONS 🙏 FRIENDS FRIENDS_GRAPH 🙏 FRIENDS_NEW PROPERTY_GRAPH_METADATA □ PG Objects BANK_GRAPH
BANK_GRAPH_NEW
BANKGRAPH

Figure 10-7 Dropping a Property Graph View

The graph is dropped.

11

Property Graph Query Language (PGQL)

PGQL is a SQL-like query language for property graph data structures that consist of *vertices* that are connected to other vertices by *edges*, each of which can have key-value pairs (properties) associated with them.

The language is based on the concept of *graph pattern matching*, which allows you to specify patterns that are matched against vertices and edges in a data graph.



The graph server (PGX) 23.1.0 supports PGQL 1.5 and earlier versions.

The property graph support provides two ways to execute Property Graph Query Language (PGQL) queries through Java APIs:

- Use the oracle.pgx.api Java package to query an in-memory snapshot of a graph that has been loaded into the graph server (PGX), as described in Executing PGQL Queries Against the Graph Server (PGX).
- Use the oracle.pg.rdbms.pgql Java package to directly query graph data stored in Oracle Database. See Executing PGQL Queries Against Property Graph Views and Executing PGQL Queries Against SQL Property Graphs for more information.

For more information about PGQL, see the PGQL Specification.

- Creating a Property Graph Using PGQL
- Pattern Matching with PGQL
- Edge Patterns Have a Direction with PGQL
- Vertex and Edge Labels with PGQL
- Variable-Length Paths with PGQL
- Aggregation and Sorting with PGQL
- Executing PGQL Queries Against Property Graph Views
 This topic explains how you can execute PGQL queries directly against the property graph views on Oracle Database tables.

11.1 Creating a Property Graph Using PGQL

CREATE PROPERTY GRAPH is a PGQL DDL statement to create a property graph view (PG View) from the database tables.

The CREATE PROPERTY GRAPH statement starts with the name you give the graph, followed by a set of vertex tables and edge tables. The graph can have no vertex tables or edge tables (an empty graph), or vertex tables and no edge tables (a graph with only vertices and no edges), or both vertex tables and edge tables (a graph with vertices and edges). However, a graph cannot be specified with only edge tables and no vertex tables.

Consider the <code>bank_accounts</code> and <code>bank_txns</code> database tables created using the sample graph data in <code>opt/oracle/graph/data</code> directory. See Importing Data from CSV Files for more information.

- **BANK_ACCOUNTS** is a table with columns id, name. A row is added into this table for every new account.
- BANK_TXNS is a table with columns txn_id, from_acct_id, to_acct_id, description, and amount. A row is added into this table for every new transaction from from acct id to to acct id.

You can create a PG View using the database tables as shown:

```
CREATE PROPERTY GRAPH bank_graph

VERTEX TABLES(

bank_accounts AS accounts

KEY(id)

LABEL accounts

PROPERTIES (id, name)
)

EDGE TABLES(

bank_txns AS transfers

KEY (txn_id)

SOURCE KEY (from_acct_id) REFERENCES accounts (id)

DESTINATION KEY (to_acct_id) REFERENCES accounts (id)

PROPERTIES (description, amount)
) OPTIONS (PG VIEW)
```

The following graph concepts are explained by mapping the database tables to the graph and using the preceding PGQL DDL statement:

- Vertex tables: A table that contains data entities is a vertex table (for example, bank accounts).
 - Each row in the vertex table is a vertex.
 - The columns in the vertex table are properties of the vertex.
 - The name of the vertex table is the default label for this set of vertices.
 Alternatively, you can specify a label name as part of the CREATE PROPERTY GRAPH statement.
- Edge tables: An edge table can be any table that links two vertex tables, or a table that has data that indicates an action from a source entity to a target entity. For example, transfer of money from FROM_ACCOUNT_ID to TO_ACCOUNT_ID is a natural edge.
 - Foreign key relationships can give guidance on what links are relevant in your data. CREATE PROPERTY GRAPH will default to using foreign key relationships to identify edges.
 - Some of the properties of an edge table can be the properties of the edge. For example, an edge from from_acct_id to to_acct_id can have properties description and amount.
 - The name of an edge table is the default label for the set of edges.
 Alternatively, you can specify a label name as part of the CREATE PROPERTY GRAPH statement.



Keys:

- Keys in a vertex table: The key of a vertex table identifies a unique vertex in the graph. The key can be specified in the CREATE PROPERTY GRAPH statement; otherwise, it defaults to the primary key of the table. If there are duplicate rows in the table, the CREATE PROPERTY GRAPH statement will return an error.
- Key in an edge table: The key of an edge table uniquely identifies an edge in the graph. The KEY clause when specifying source and destination vertices uniquely identifies the source and destination vertex keys.
- Table aliases: Vertex and edge tables must have unique names. If you need to identify multiple vertex tables from the same relational table, or multiple edge tables from the same relational table, you must use aliases. For example, you can create two vertex tables bank accounts and accounts from one table bank accounts, as shown:

```
CREATE PROPERTY GRAPH bank_transfers

VERTEX TABLES (bank_accounts KEY(id)

bank_accounts AS accounts KEY(id))
```

In case any of your vertex and edge table share the same name, then you must again use a table alias. In the following example, table alias is used for the edge table, DEPARTMENTS, as there is a vertex table referenced with the same name:

```
CREATE PROPERTY GRAPH hr

VERTEX TABLES (
    employees KEY(employee_id)
        PROPERTIES ARE ALL COLUMNS,
    departments KEY(department_id)
        PROPERTIES ARE ALL COLUMNS
)

EDGE TABLES (
    departments AS managed_by
        SOURCE KEY ( department_id ) REFERENCES departments ( department_id )
        DESTINATION employees
        PROPERTIES ARE ALL COLUMNS
) OPTIONS (PG VIEW)
```

Properties: The vertex and edge properties of a graph are derived from the columns of
the vertex and edge tables respectively and by default have the same name as the
underlying table columns. However, you can choose a different property name for each
column. This helps to avoid conflicts when two tables have the same column name but
with different data types.

In the following example, the vertex properties id and name are renamed to acct_no and acct_name respectively:

```
CREATE PROPERTY GRAPH bank_transfers

VERTEX TABLES (
    bank_accounts AS accounts
    LABEL accounts
    PROPERTIES (id AS acct_no, name AS acct_name)
)
```

• **REFERENCES clause**: This connects the source and destination vertices of an edge to the corresponding vertex tables.

For more details on the CREATE PROPERTY GRAPH statement, see the PGQL Specification.

Refer to the following table for creating a property graph:

Table 11-1 CREATE PROPERTY GRAPH Statement Support

Method	More Information
Metriou	More information
Create a property graph in the graph server (PGX) using the oracle.pgx.api Java package	Java APIs for Executing CREATE PROPERTY GRAPH Statements
Create a property graph in the graph server (PGX) using the pypgx.api Python package	Python APIs for Executing CREATE PROPERTY GRAPH Statements
Create a property graph view on Oracle Database tables	Creating a Property Graph View

11.2 Pattern Matching with PGQL

Pattern matching is done by specifying one or more path patterns in the MATCH clause. A single path pattern matches a linear path of vertices and edges, while more complex patterns can be matched by combining multiple path patterns, separated by comma. Value expressions (similar to their SQL equivalents) are specified in the WHERE clause and let you filter out matches, typically by specifying constraints on the properties of the vertices and edges

For example, assume a graph of TCP/IP connections on a computer network, and you want to detect cases where someone logged into one machine, from there into another, and from there into yet another. You would query for that pattern like this:

```
SELECT id(host1) AS id1, id(host2) AS id2, id(host3) AS id3
choose what to return */
FROM MATCH
    (host1) -[connection1]-> (host2) -[connection2]-> (host3)
single linear path pattern to match */
    connection1.toPort = 22 AND connection1.opened = true AND
    connection2.toPort = 22 AND connection2.opened = true AND
    connection1.bytes > 300 AND
meaningful amount of data was exchanged */
    connection2.bytes > 300 AND
    connection1.start < connection2.start AND</pre>
second connection within time-frame of first */
    connection2.start + connection2.duration < connection1.start +</pre>
connection1.duration
GROUP BY id1, id2, id3
aggregate multiple matching connections */
```

For more examples of pattern matching, see the Writing simple queries section in the PGQL specification.

11.3 Edge Patterns Have a Direction with PGQL

An edge pattern has a direction, as edges in graphs do. Thus, (a) <-[]- (b) specifies a case where *b* has an edge pointing at *a*, whereas (a) -[]-> (b) looks for an edge in the opposite direction.

The following example finds common friends of April and Chris who are older than both of them.

For more examples of edge patterns, see the Edge Patterns section in the PGQL specification.

11.4 Vertex and Edge Labels with PGQL

Labels are a way of attaching type information to edges and nodes in a graph, and can be used in constraints in graphs where not all nodes represent the same thing. For example:

The example queries a graph which contains a set of vertices with the label person, a set of vertices with the label movie, and a set of edges with the label likes. A label predicate can start with either a colon (:) or the keyword IS followed by one or more labels. If more than one label is used, then the labels are separated by a vertical bar (|).

The following query shows the preceding example query with the keyword IS for the label predicate:



See Also:

- Label Expressions section in the PGQL specification
- Label Predicates section in the PGQL specification

11.5 Variable-Length Paths with PGQL

Variable-length path patterns have a quantifier like * to match a variable number of vertices and edges. Using a PATH macro, you can specify a named path pattern at the start of a query that can be embedded into the MATCH clause any number of times, by referencing its name. The following example finds all of the common ancestors of Mario and Luigi.

```
PATH has_parent AS () -[:has_father|has_mother]-> ()
SELECT ancestor.name
FROM MATCH (p1:Person) -/:has_parent*/-> (ancestor:Person)
   , MATCH (p2:Person) -/:has_parent*/-> (ancestor)
WHERE
   p1.name = 'Mario' AND
   p2.name = 'Luigi'
```

The preceding path specification also shows the use of anonymous constraints, because there is no need to define names for intermediate edges or nodes that will not be used in additional constraints or query results. Anonymous elements can have constraints, such as <code>[:has_father|has_mother]</code> -- the edge does not get a variable name (because it will not be referenced elsewhere), but it is constrained.

For more examples of variable-length path pattern matching, see the Variable-Length Paths section in the PGQL specification.

11.6 Aggregation and Sorting with PGQL

Like SQL, PGQL has support for the following:

- GROUP BY to create groups of solutions
- MIN, MAX, SUM, and AVG aggregations
- · ORDER BY to sort results

And for many other familiar SQL constructs.

See Also:

- See Grouping and Aggregation for more information on GROUP BY
- See Sorting and Row Limiting for more information on ORDER BY

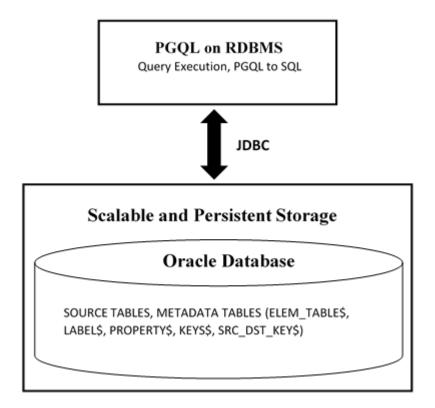


11.7 Executing PGQL Queries Against Property Graph Views

This topic explains how you can execute PGQL queries directly against the property graph views on Oracle Database tables.

The PGQL query execution flow is shown in the following figure.

Figure 11-1 PGQL on Property Graph Views in Oracle Database



The basic execution flow is:

- The PGQL query is submitted to PGQL on RDBMS through a Java API.
- The PGQL query is translated into SQL statements using the internal metadata tables for property graph views.
- 3. The translated SQL is submitted to Oracle Database by JDBC.
- 4. The SQL result set is wrapped as a PGQL result set and returned to the caller.
- Supported PGQL Features and Limitations for PG Views
 Learn about the supported PGQL features and limitations for property graph views (PG Views).
- Performance Considerations for PGQL Queries
- Using the Java and Python APIs to Run PGQL Queries



11.7.1 Supported PGQL Features and Limitations for PG Views

Learn about the supported PGQL features and limitations for property graph views (PG Views).

The following table describes the complete list of supported and unsupported PGQL features for PG Views:

Table 11-2 Supported PGQL Functionalities and Limitations for PG Views

Feature	PGQL on PG Views
CREATE PROPERTY GRAPH	Supported
DROP PROPERTY GRAPH	Supported
Fixed-length pattern matching	Supported
Variable-length pattern matching goals Variable-length pattern matching quantifiers	Supported: Reachability ANY ANY SHORTEST TOP k SHORTEST ALL Limitations: ALL SHORTEST ANY CHEAPEST TOP k CHEAPEST Supported:
	<pre>+ ? { n } { n, } { n, m } { n, m } </pre>
Variable-length path unnesting	Not supported
GROUP BY	Supported
Aggregations	Supported: COUNT MIN, MAX, AVG, SUM LISTAGG Limitations: ARRAY_AGG
DISTINCT • SELECT DISTINCT • Aggregation with DISTINCT (such as, COUNT (DISTINCT e.prop))	Supported
SELECT v.*	Supported
ORDER BY (+ASC/DESC), LIMIT, OFFSET	Supported
Data Types	All available Oracle RDBMS data types supported



Table 11-2 (Cont.) Supported PGQL Functionalities and Limitations for PG Views

Feature	PGQL on PG Views
JSON	Supported: JSON storage: JSON strings (VARCHAR2) JSON objects JSON functions: Any JSON function call that follows the syntax, json_function_name (arg1, arg2,). For example:
	<pre>json_value(department_data, '\$.department')</pre>
	Limitations: Simple Dot Notation Any optional clause in a JSON function call (such as RETURNING, ERROR, and so on) is not supported. For example: json_value(department_data, '\$.employees[1].hireDate' RETURNING DATE)
Operators	Supported:
Functions and predicates	Supported are all available functions in the Oracle RDBMS that take the form function_name(arg1, arg2,) with optional schema and package qualifiers.
	Supported PGQL functions/predicates:
	IS NULL, IS NOT NULL
	 JAVA_REGEXP_LIKE (based on CONTAINS)
	• LOWER, UPPER
	• SUBSTRING
	 ABS, CEIL/CEILING, FLOOR, ROUND
	• EXTRACT
	• ID
	• LABEL, HAS_LABEL
	• ALL_DIFFERENT
	• CAST
	• CASE
	• IN and NOT IN
	Limitations:
	• LABELS
	• IN DEGREE, OUT DEGREE



Table 11-2 (Cont.) Supported PGQL Functionalities and Limitations for PG Views

Feature	PGQL on PG Views		
User-defined functions	Supported: PL/SQL functions		
	 Functions created via the Oracle Database Multilingual Engine (MLE) 		
Subqueries: Scalar subqueries EXISTS and NOT EXISTS subqueries	Supported		
INSERT/UPDATE/DELETE	Supported		
INTERVAL literals and operations	Not supported		

Additional Information on Supported PGQL Features with Examples

11.7.1.1 Additional Information on Supported PGQL Features with Examples

The following PGQL features are supported in property graph views (PG Views):

- Recursive queries are supported for the following variable-length path finding goals:
 - Reachability
 - ANY
 - ANY SHORTEST
 - TOP k SHORTEST
- Recursive queries are supported for the following horizontal aggregations:
 - LISTAGG

```
SELECT LISTAGG(src.first_name || ' ' || src.last_name, ',')
FROM MATCH TOP 2 SHORTEST ( (n:Person) ((src)-[e:knows]->)*
(m:Person) )
WHERE n.id = 1234
```

SUM

```
SELECT SUM(e.weight + 3) FROM MATCH TOP 2 SHORTEST ( (n:Person) -[e:knows]->* (m:Person) ) WHERE n.id = 1234
```

COUNT

```
SELECT COUNT(e)
FROM MATCH TOP 2 SHORTEST ( (n:Person) -[e:knows]->* (m:Person) )
WHERE n.id = 1234
```



AVG

```
SELECT AVG(dst.age)
FROM MATCH TOP 2 SHORTEST ( (n:Person) (-[e:knows]->(dst))*
(m:Person) )
WHERE n.id = 1234
```

MIN (Only for property value or CAST expressions)

```
SELECT MIN(CAST(dst.age + 5 AS INTEGER))
FROM MATCH TOP 2 SHORTEST ( (n:Person) (-[e:knows]->(dst))*
(m:Person) )
WHERE n.id = 1234
```

MAX (Only for property value or CAST expressions)

```
SELECT MAX(dst.birthday)
FROM MATCH TOP 2 SHORTEST ( (n:Person) (-[e:knows]->(dst))*
(m:Person) )
WHERE n.id = 1234
```

The following quantifiers are supported in recursive queries:

Table 11-3 Supported Quantifiers in PGQL SELECT Queries

Syntax	Description
*	zero or more
+	one or more
?	zero or one
{n}	exactly n
{n,}	n or more
{n,m}	between n and m (inclusive)
{,m}	between zero and <i>m</i> (inclusive)

Data type casting with precision and scale is supported:

```
SELECT CAST(v.id AS VARCHAR2(10)) || '→' || CAST(w.id AS VARCHAR2(10)) AS friendOf

FROM MATCH (v) -[:friendOf]->(w)

SELECT CAST(e.mval AS NUMBER(5,2)) AS mval

FROM MATCH () -[e:knows]->()

WHERE e.mval = '342.5'
```

Both built-in Oracle Database functions and user defined functions (UDFs) are supported.
 For example:

Assuming a table has a JSON column with values such as, {"name":"John", "age": 43}:

```
SELECT JSON_VALUE(p.attributes, '$.name') AS name
FROM MATCH (p:Person)
WHERE JSON VALUE(p.attributes, '$.age') > 35
```

Assuming an Oracle Text index exists on a text column in a table:

```
SELECT n.text
FROM MATCH (n)
WHERE CONTAINS(n.text, 'cat', 1) > 0
```

Assuming a UDF updated id is registered with the graph server (PGX):

```
SELECT my.updated id(n.ID) FROM MATCH(n) LIMIT 10
```

Selecting all properties of vertices or edges is supported through SELECT v.*
clause, where v is the variable whose properties are selected. The following
example retrieves all the edge properties of a graph:

```
SELECT label(e), e.* FROM MATCH (n)-[e]->(m) ON bank_graph_view LIMIT 3
```

On execution, the preceding query retrieves all the properties that are bound to the variable ${\tt e}$ as shown:

		AMOUNT		DESCRIPTION		FROM_ACCT_ID	I	TO_ACCT_ID	İ
'	1	1000 1000	 	transfer transfer		178 178 179	 	921 462 688	

A PREFIX can be specified to avoid duplicate column names in cases where you select all properties using multiple variables. For example:

```
SELECT n.* PREFIX 'n_', e.* PREFIX 'e_', m.* PREFIX 'm_'
FROM MATCH (n:Accounts) -[e:transfers]-> (m:Accounts)
ON bank_graph_view LIMIT 3
```

The query output is as follows:

```
+-----+
| n_ID | n_NAME | e_AMOUNT | e_DESCRIPTION | e_FROM_ACCT_ID |
e_TO_ACCT_ID | m_ID | m_NAME |
+-------+
| 178 | Account | 1000 | transfer | 178 |
921 | 921 | Account |
```



Label expressions can be used such that only properties that belong to the specified vertex or edge labels are selected:

```
SELECT LABEL(n), n.* FROM MATCH (n:Accounts) ON bank graph view LIMIT 3
```

The preceding query output is as shown:

```
+-----+
| LABEL(n) | ID | NAME |
+------+
| ACCOUNTS | 1 | User1 |
| ACCOUNTS | 2 | User2 |
| ACCOUNTS | 3 | User3 |
```

- Support for ALL path finding goal to return all the paths between a pair of vertices. However, to avoid endless cycling, only the following quantifiers are supported:
 - _ ?
 - $\{n\}$
 - {n.m}
 - {,m}

For example, the following PGQL query finds all the transaction paths from account 284 to account 616:

```
SELECT LISTAGG(e.amount, ' + ') || ' = ', SUM(e.amount) AS total_amount
FROM MATCH ALL (a:Accounts) -[e:Transfers]->{1,4}(b:Accounts)
WHERE a.id = 284 AND b.id = 616
ORDER BY total_amount
```

On execution, the guery produces the following result:

Scalar subqueries which return exactly one column and one row is supported.



For example:

```
SELECT p.name AS name , (
 SELECT SUM(t.amount)
 FROM MATCH (a) <-[t:transaction] - (:Account)</pre>
) AS sum incoming , (
 SELECT SUM(t.amount)
 FROM MATCH (a) -[t:transaction]-> (:Account)
) AS sum outgoing , (
 SELECT COUNT (DISTINCT p2)
 FROM MATCH (a) -[t:transaction] - (:Account) -[:owner] ->
(p2:Person)
 WHERE p2 <> p
) AS num persons transacted with , (
 SELECT COUNT (DISTINCT c)
 FROM MATCH (a) -[t:transaction]- (:Account) -[:owner]->
(c:Company)
) AS num companies transacted with
FROM MATCH (p:Person) <-[:owner] - (a:Account)</pre>
ORDER BY sum outgoing + sum incoming DESC
```

 EXISTS and NOT EXISTS subqueries are supported. Such queries yield TRUE or FALSE depending on whether the query produces at least one results given the bindings of the outer query.

For example:

```
SELECT fof.name, COUNT(friend) AS num_common_friends
FROM MATCH (p:Person) -[:knows]-> (friend:Person) -[:knows]->
(fof:Person)
WHERE NOT EXISTS (
    SELECT * FROM MATCH (p) -[:knows]-> (fof)
)
```

The following are a few PGQL features which are not supported:

- The following PGQL SELECT features are not supported:
 - Use of bind variables in path expressions.
 If you attempt to use a bind variable, it will result in an error as shown:

in_degree and out_degree functions.



Note:

- See Supported PGQL Features and Limitations for PG Views for a complete list of supported and unsupported PGQL features for PG Views.
- See Performance Considerations for PGQL Queries for details on recommended practices to enhance query performance for recursive queries.

11.7.2 Performance Considerations for PGQL Queries

The following are some recommended practices for query performance.

- Recursive Queries
- Using Query Optimizer Hints
- Speed Up Query Translation Using Graph Metadata Cache and Translation Cache

Recursive Queries

The following indexes are recommended in order to speed up execution of recursive queries:

- For underlying VERTEX tables of the recursive pattern, an index on the key column
- For underlying EDGE tables of the recursive pattern, an index on the source key column

Note:

You can also create index on (source key, destination key).

For example, consider the following CREATE PROPERTY GRAPH statement:

```
CREATE PROPERTY GRAPH people

VERTEX TABLES(

person

KEY ( id )

LABEL person

PROPERTIES( name, age )
)

EDGE TABLES(

knows

key (person1, person2)

SOURCE KEY ( person1 ) REFERENCES person (id)

DESTINATION KEY ( person2 ) REFERENCES person (id)

NO PROPERTIES
)

OPTIONS ( PG VIEW )
```



And also consider the following query:

```
SELECT COUNT(*)
FROM MATCH ANY SHORTEST ( (n:Person) -[e:knows]->* (m:Person) )
WHERE n.id = 1234
```

In order to improve performance of the recursive part of the preceding query, the following indexes must exist:

```
• CREATE INDEX <INDEX_NAME> ON PERSON(ID)
```

```
• CREATE INDEX <INDEX_NAME> ON KNOWS(PERSON1) Or CREATE INDEX <INDEX NAME> ON KNOWS(PERSON1, PERSON2)
```

Composite Vertex Keys

For composite vertex keys, query execution can be optimized with the creation of function-base indexes on the key columns:

- For underlying VERTEX tables of the recursive pattern, a function-based index on the comma-separated concatenation of key columns
- For underlying EDGE tables of the recursive pattern, a function-based index on the comma-separated concatenation of source key columns



You can also create index on (source key columns, destination key columns).

For example, consider the following CREATE PROPERTY GRAPH statement:

```
CREATE PROPERTY GRAPH people
  VERTEX TABLES(
    person
        KEY ( id1, id2 )
        LABEL person
        PROPERTIES( name, age )
)
EDGE TABLES(
    knows
        key (id)
        SOURCE KEY ( id1person1, id2person1 ) REFERENCES person (id1,id2)
        DESTINATION KEY ( id1person2, id2person2 ) REFERENCES person
(id1,id2)
        NO PROPERTIES
)
OPTIONS ( PG_VIEW )
```

And also consider the following query:

```
SELECT COUNT(*)
FROM MATCH ANY SHORTEST ( (n:Person) -[e:knows]->* (m:Person) )
WHERE n.id = 1234
```



In order to improve performance of the recursive part of the preceding query, the following indexes must exist:

```
• CREATE INDEX <INDEX NAME> ON PERSON (ID1 || ',' || ID2)
```

```
• CREATE INDEX <INDEX_NAME> ON KNOWS (ID1PERSON1 || ',' || ID2PERSON1) or CREATE INDEX <INDEX_NAME> ON KNOWS (ID1PERSON1 || ',' || ID2PERSON1, ID1PERSON2 || ',' || ID2PERSON2)
```

If some of the columns in a composite vertex key is a string column, the column needs to be comma-escaped in the function-base index creation.

For example, if column ID1 in table PERSON of the preceding example is of type VARCHAR2 (10), you need to escape the comma for the column as follows:

```
replace(ID1, ',', '\,')
```

So, the indexes to improve performance will result as shown:

```
    CREATE INDEX <INDEX NAME> ON PERSON (replace(ID1, ',', '\,') || ',' || ID2)
```

```
• CREATE INDEX <INDEX_NAME> ON KNOWS (replace(ID1PERSON1, ',', '\,') || ',' || ID2PERSON1)
```

Using Query Optimizer Hints

The following hints can be used to influence translation of PGQL variable-length path patterns to SQL:

- REVERSE_PATH: Switches on or off the reverse path optimization (ON by default). If ON, it automatically determines if the pattern can best be evaluated from source to destination or from destination to source, based on specified filter predicates.
- PUSH_SRC_HOPS: Switches on or off pushing source filter optimization (ON by default). If ON, then filter predicates are used to limit the number of source vertices (or destination vertices if path evaluation is reversed) and thereby the search space of variable-length path pattern evaluations.
- PUSH_DST_HOPS: Switches on or off pushing destination filter optimization (OFF by default). If ON, then filter predicates are used to limit the number of destination vertices (or source vertices if path evaluation is reversed) and thereby the search space of variable-length path pattern evaluations.

The preceding hints can be configured as options parameter in the following Java API methods:

```
    executeQuery(String pgql, String options)
```

- translateQuery(String pgql, String options)
- execute(String pgql, String matchOptions, String options)

For example, consider the following PGQL query:

```
SELECT v1.name AS v1, v2.name AS v2, v3.name As v3 FROM MATCH (v1:Person)-[e1:friendOf]->(v2:Person), MATCH ANY (v2:Person)-[e2:friendOf]->*(v3:Person) WHERE v1.name= 'Bob'
```



When the preceding query is executed using the default option for PUSH_SRC_HOPS, the output for start_nodes_translation displays the filter expression as shown:

```
System.out.println(pgqlStatement.translateQuery(pgql).getSqlTranslatio
n())
...
start_nodes_translation => (to_clob('SELECT ''PERSONS'' AS
"src_table", el.person_b AS "src_key"
FROM "GRAPHUSER"."PERSONS" "V1", "GRAPHUSER"."FRIENDSHIPS" "E1"
WHERE (((el.person_a = vl.person_id) AND NOT(el.person_b IS NULL)) AND
(vl.name = ''Bob''))')),
    end_nodes_translation => (to_clob('SELECT ''PERSONS'' AS
"dst_table", v3.person_id AS "dst_key"
FROM "GRAPHUSER"."PERSONS" "V3"')),
...
...
```

If the preceding query is executed with the hint <code>PUSH_SRC_HOPS=F</code>, then the query is translated into SQL as shown:

Speed Up Query Translation Using Graph Metadata Cache and Translation Cache

The following global caches help to speed up PGQL guery translation:

- **Graph Metadata Cache:** Stores graph metadata such as tables, labels, properties, and so on.
- Translation Cache: Stores PGQL to SQL translation.

You can configure the caches using the following Java APIs:

- clearTranslationCache()
- disableTranslationCache()
- enableTranslationCache()
- setTranslationCacheMaxCapacity(int maxCapacity)
- clearGraphMetadataCache()
- disableGraphMetadataCache()



- enableGraphMetadataCache()
- setGraphMetadataCacheMaxCapacity(int maxCapacity)

These preceding methods are part of the PgqlConnection class. Separate caches are maintained for each database user such that cached objects are shared between different PgqlConnection objects if they have the same connection URL and user underneath.

11.7.3 Using the Java and Python APIs to Run PGQL Queries

You can run PGQL queries using the Java API in the <code>oracle.pg.rdbms.pgql</code> package. Also, you can use the Python OPG4Py package for executing PGQL queries against the graph data in the Oracle Database. This package contains a sub-package <code>Pgql</code> with one or more modules that wraps around the Java API in the <code>oracle.pg.rdbms.pgql</code> package.

- Creating a Property Graph View
- Executing PGQL SELECT Queries
- Executing PGQL Queries Using JDBC Driver
- Executing PGQL Queries to Modify Property Graph Views
- Dropping A Property Graph View

11.7.3.1 Creating a Property Graph View

You can create a property graph view (PG View) using the CREATE PROPERTY GRAPH statement.

Example 11-1 Creating a Property Graph View

The following example describes the creation of a PG View.

- JShell
- Java
- Python

JShell

```
opg4j> var jdbcUrl="jdbc:oracle:thin:@<host_name>:<port>/<db_service>"
opg4j> var conn =
DriverManager.getConnection(jdbcUrl,"<username>","<password>");
opg4j> var pgqlConn = PgqlConnection.getConnection(conn)
opg4j> var pgqlStmt = pgqlConn.createStatement() //create a PGQL Statement
opg4j> conn.setAutoCommit(false)
opg4j> var pgql =
...> "CREATE PROPERTY GRAPH bank_graph "
...> + "VERTEX TABLES ( bank_accounts AS Accounts "
...> + "KEY (id) "
...> + "LABEL Accounts "
...> + "PROPERTIES (id, name) "
...> + "PROPERTIES (id, name) "
...> + "DGE TABLES ( bank txns AS Transfers "
```



```
...> + "KEY (txn id) "
...> + "SOURCE KEY (from_acct_id) REFERENCES Accounts (id) "
...> + "DESTINATION KEY (to acct id) REFERENCES Accounts (id) "
...> + "LABEL Transfers "
...> + "PROPERTIES (from acct_id, to_acct_id, amount, description) "
...> + ") OPTIONS (PG VIEW) "
opg4j> pgqlStmt.execute(pgql)
Java
import java.sql.Connection;
import java.sql.Statement;
import java.sql.DriverManager;
import oracle.pg.rdbms.pgql.jdbc.PgqlJdbcRdbmsDriver;
import oracle.pg.rdbms.pgql.PgqlConnection;
import oracle.pg.rdbms.pgql.PgqlStatement;
 * This example shows how to create a property graph view.
public class PgqlCreate
 public static void main(String[] args) throws Exception
   int idx=0;
                          = args[idx++];
   String jdbcUrl
                            = args[idx++];
   String username
   String password
                            = args[idx++];
    String graph
                            = args[idx++];
    Connection conn = null;
    PgqlStatement pgqlStmt = null;
    try {
     //Get a jdbc connection
     DriverManager.registerDriver(new PgqlJdbcRdbmsDriver());
      conn = DriverManager.getConnection(jdbcUrl, username, password);
      conn.setAutoCommit(false);
      // Get a PGQL connection
      PgqlConnection pgqlConn = PgqlConnection.getConnection(conn);
      // Create a PGOL Statement
      pgqlStmt = pgqlConn.createStatement();
      // Execute PGQL Query
      String pgql =
        "CREATE PROPERTY GRAPH " + graph + " " +
        "VERTEX TABLES ( bank accounts as Accounts " +
        "KEY (id) " +
```

"LABEL \"Accounts\"" +

```
"PROPERTIES (id, name)" +
        ") " +
        "EDGE TABLES ( bank txns as Transfers " +
        "KEY (txn id) " +
        "SOURCE KEY (from acct_id) REFERENCES Accounts (id) " +
        "DESTINATION KEY (to acct id) REFERENCES Accounts (id) " +
        "LABEL \"Transfers\"" +
       "PROPERTIES (from acct id, to acct id, amount, description)" +
        ") OPTIONS (PG VIEW) ";
      // Print the results
     pgqlStmt.execute(pgql);
   finally {
     // close the statement
     if (pgqlStmt != null) {
        pgqlStmt.close();
     // close the connection
     if (conn != null) {
        conn.close();
     }
}
```

```
>>> pgql_conn = opg4py.pgql.get connection("<username>","<password>",
"jdbc:oracle:thin:@localhost:1521/orclpdb")
>>> pgql statement = pgql conn.create statement()
>>> pagl = """
           CREATE PROPERTY GRAPH bank graph
           VERTEX TABLES (
            bank accounts
                LABEL Accounts
                PROPERTIES (id, name)
            )
           EDGE TABLES (
             bank txns
                KEY (txn id)
                SOURCE KEY (from acct id) REFERENCES bank accounts
                DESTINATION KEY (to_acct id) REFERENCES bank accounts
                LABEL TRANSFERS
. . .
                PROPERTIES (from_acct_id, to_acct_id, amount, description)
            ) OPTIONS (PG VIEW)
>>> pgql statement.execute(pgql)
False
```



You can verify the property graph view creation by checking the metadata tables that get created in the database.

11.7.3.2 Executing PGQL SELECT Queries

You can run PGQL SELECT queries as described in the following examples.

Example 11-2 Running a Simple SELECT Query Using PgqlStatement and PgqlResultSet

In the following example, PgqlConnection is used to obtain a PgqlStatement. Then, it calls the <code>executeQuery</code> method of PgqlStatement, which returns a PgqlResultSet object. PgqlResultSet provides a <code>print()</code> method, which displays results in a tabular mode.

- JShell
- Java
- Python

JShell

```
opg4j> var jdbcUrl="jdbc:oracle:thin:@<host name>:<port>/<db service>"
opg4j> var conn =
DriverManager.getConnection(jdbcUrl, "<username>", "<password>");
opg4j> var pgglConn = PgglConnection.getConnection(conn)
opg4j> pgqlConn.setGraph("BANK GRAPH")
opg4j> var pgqlStmt = pgqlConn.createStatement() //create a PGQL
Statement
opg4j> String s = "SELECT n.* FROM MATCH (n:Accounts) LIMIT 3"
opg4j> var resultSet = pgqlStmt.executeQuery(s)
opg4j> resultSet.print() //Prints the query result set
+----+
| ID | NAME
+----+
| 2 | Account2 |
| 3 | Account3 |
+----+
```

Java

```
import java.sql.Connection;
import java.sql.Statement;
import java.sql.DriverManager;
import oracle.pg.rdbms.pgql.jdbc.PgqlJdbcRdbmsDriver;
import oracle.pg.rdbms.pgql.PgqlConnection;
import oracle.pg.rdbms.pgql.PgqlResultSet;
import oracle.pg.rdbms.pgql.PgqlStatement;
/*
```



```
* This example shows how to execute a SELECT query on a property graph view.
public class PgqlExample1
  public static void main(String[] args) throws Exception
    int idx=0;
    String jdbcUrl
                              = args[idx++];
    String username
                            = args[idx++];
    String password
                              = args[idx++];
    String graph
                              = args[idx++];
    Connection conn = null;
    PgqlStatement pgqlStmt = null;
    PgqlResultSet rs = null;
    try {
      //Get a jdbc connection
      DriverManager.registerDriver(new PgqlJdbcRdbmsDriver());
      conn = DriverManager.getConnection(jdbcUrl, username, password);
      conn.setAutoCommit(false);
      // Get a PGQL connection
      PgqlConnection pgqlConn = PgqlConnection.getConnection(conn);
      pgqlConn.setGraph(graph);
      // Create a PGQL Statement
      pgqlStmt = pgqlConn.createStatement();
      // Execute PGQL Query
      String query = "SELECT n.* FROM MATCH (n:Accounts) LIMIT 5";
      rs = pgqlStmt.executeQuery(query);
      // Print the results
      rs.print();
    finally {
      // close the result set
      if (rs != null) {
         rs.close();
      // close the statement
      if (pgqlStmt != null) {
        pgqlStmt.close();
      // close the connection
      if (conn != null) {
         conn.close();
      }
  }
}
```



```
>>> pgql_conn = opg4py.pgql.get connection("<username>","<password>",
"<jdbcUrl>")
>>> pgql statement = pgql conn.create statement()
>>> pgql conn.set graph("BANK GRAPH")
>>> s = "SELECT n.* FROM MATCH (n:Accounts) LIMIT 3"
>>> pgql statement.execute query(s)
>>> pgql_result_set = pgql_statement.execute_query(s)
>>> pgql result set.print()
| ID | NAME
+----+
| 1 | Account1 |
| 2 | Account2 |
| 3 | Account3 |
>>> pgql result set
PgqlResultSet(java pgql result set:
oracle.pg.rdbms.pggl.PgglResultSet, # of results: 3)
```

Also, you can convert the PGQL result set obtained in the preceding code to a Pandas dataframe using the to pandas() method.



The pandas package must be installed in your system to successfully execute the call to to_pandas(). This package is automatically installed at the time of the Python client installation for versions Python 3.8 and Python 3.9. However, if your call to to_pandas() fails, verify if the pandas module is installed in your system. In case the module is found missing or your Python version differs from the earlier mentioned versions, then install the pandas package manually.

Example 11-3 Running a SELECT Query Using PgqlPreparedStatement

- JShell
- Java
- Python

JShell

```
opg4j> var jdbcUrl="jdbc:oracle:thin:@<host_name>:<port>/<db_service>"
opg4j> var conn =
DriverManager.getConnection(jdbcUrl,"<username>","<password>");
opg4j> var pgqlConn = PgqlConnection.getConnection(conn)
```



```
opg4j> pgqlConn.setGraph("BANK GRAPH");
opg4j> String s = "SELECT n.* FROM MATCH (n:Accounts) LIMIT ?"
opg4j> var ps = pgqlConn.prepareStatement(s, 0 /* timeout */, 4 /* parallel
*/, 2 /* dynamic sampling */, -1 /* max results */, null /* match options
*/, null /* options */)
opg4j> ps.setInt(1, 3)
opg4j> var rs = ps.executeQuery()
opg4j> rs.print() //Prints the guery result set
+----+
| ID | NAME
+----+
| 2 | Account2 |
| 3 | Account3 |
+----+
Java
import java.sql.Statement;
import java.sql.DriverManager;
import oracle.pg.rdbms.pgql.jdbc.PgqlJdbcRdbmsDriver;
import oracle.pg.rdbms.pgql.*;
```

```
public class PgqlExample2
 public static void main(String[] args) throws Exception
   int idx=0;
                           = args[idx++];
   String jdbcUrl
   String username
                            = args[idx++];
    String password
                            = args[idx++];
    String graph
                             = args[idx++];
   Connection conn = null;
    PgqlStatement pgqlStmt = null;
    PgqlResultSet rs = null;
    try {
      //Get a jdbc connection
     DriverManager.registerDriver(new PgqlJdbcRdbmsDriver());
      conn = DriverManager.getConnection(jdbcUrl, username, password);
      conn.setAutoCommit(false);
      // Get a PGQL connection
```

PgqlConnection pgqlConn = PgqlConnection.getConnection(conn);

PgqlPreparedStatement pStmt = pgqlConn.prepareStatement(s, 0, 4 , 2 ,

String s = "SELECT n.* FROM MATCH (n:Accounts) LIMIT ?";

pgqlConn.setGraph(graph);

// Execute PGQL Query

-1 , null , null);

```
pStmt.setInt(1,3);
      rs = pStmt.executeQuery();
      // Print the results
      rs.print();
   finally {
      // close the result set
      if (rs != null) {
         rs.close();
      // close the statement
      if (pgqlStmt != null) {
         pgqlStmt.close();
      // close the connection
      if (conn != null) {
        conn.close();
      }
}
```

```
>>> pgql conn = opg4py.pgql.get connection("<username>","<password>",
"<jdbcUrl>")
>>> pgql_statement = pgql_conn.create_statement()
>>> pgql conn.set graph("BANK GRAPH")
>>> s = "SELECT n.* FROM MATCH (n:Accounts) LIMIT ?"
>>> ps = pgql conn.prepare statement(s, timeout=0, parallel=4,
dynamicSampling=2, maxResults=-1, matchOptions=None, options=None)
>>> ps.set int(1,3)
>>> ps.execute_query().print()
+----+
| ID | NAME
+----+
| 1 | Account1 |
| 2 | Account2 |
| 3 | Account3 |
+----+
```

Example 11-4 Running a SELECT Query with Grouping and Aggregation

- JShell
- Java
- Python



JShell

```
opg4j> var jdbcUrl="jdbc:oracle:thin:@<host name>:<port>/<db service>"
opg4j> var conn =
DriverManager.getConnection(jdbcUrl, "<username>", "<password>");
opg4j> var pgglConn = PgglConnection.getConnection(conn)
opg4j> pgqlConn.setGraph("BANK GRAPH")
opg4j> var pgqlStmt = pgqlConn.createStatement() //create a PGQL Statement
opg4j> String query = "SELECT v1.id, COUNT(v2) AS numTxns "+
           "FROM MATCH (v1)-[e IS Transfers]->(v2) "+
           "GROUP BY v1 "+
...>
          "ORDER BY numTxns DESC "+
...>
...> "LIMIT 3"
opg4j> var resultSet = pgqlStmt.executeQuery(query)
opg4j> resultSet.print() //Prints the query result set
+----+
| ID | NUMTXNS |
+----+
| 687 | 6 |
| 195 | 5
| 192 | 5
+----+
```

Java

```
import java.sql.Connection;
import java.sql.Statement;
import java.sql.DriverManager;
import oracle.pq.rdbms.pqql.jdbc.PqqlJdbcRdbmsDriver;
import oracle.pg.rdbms.pgql.PgqlConnection;
import oracle.pg.rdbms.pggl.PgglResultSet;
import oracle.pg.rdbms.pgql.PgqlStatement;
 * This example shows how to execute a SELECT query with aggregation .*/
public class PgqlExample3
{
  public static void main(String[] args) throws Exception
    int idx=0;
   String jdbcUrl
                           = args[idx++];
    String username
                            = args[idx++];
    String password
                            = args[idx++];
    String graph
                             = args[idx++];
    Connection conn = null;
    PgqlStatement pgqlStmt = null;
    PgglResultSet rs = null;
    try {
      //Get a jdbc connection
```



```
DriverManager.registerDriver(new PgqlJdbcRdbmsDriver());
      conn = DriverManager.getConnection(jdbcUrl, username, password);
      conn.setAutoCommit(false);
      // Get a PGQL connection
      PgqlConnection pgqlConn = PgqlConnection.getConnection(conn);
      pgglConn.setGraph(graph);
      // Create a PGQL Statement
      pgqlStmt = pgqlConn.createStatement();
      // Execute PGQL Query
      String query =
        "SELECT v1.id, COUNT(v2) AS numTxns "+
        "FROM MATCH (v1)-[e IS Transfers]->(v2) "+
        "GROUP BY v1 "+
        "ORDER BY numTxns DESC";
      rs = pgqlStmt.executeQuery(query);
      // Print the results
     rs.print();
   finally {
      // close the result set
     if (rs != null) {
        rs.close();
      // close the statement
      if (pgqlStmt != null) {
        pgqlStmt.close();
      // close the connection
      if (conn != null) {
         conn.close();
}
```



+.		 	+
	ID	NUMTXNS	
+.		 	+
	687	6	
	195	5	
	192	5	
+.		 	-+

Example 11-5 Showing a PGQL Path Query

- JShell
- Java
- Python

JShell

```
opg4j> var jdbcUrl="jdbc:oracle:thin:@<host name>:<port>/<db service>"
opg4j> var conn =
DriverManager.getConnection(jdbcUrl, "<username>", "<password>");
opg4j> var pgqlConn = PgqlConnection.getConnection(conn)
opg4j> pgqlConn.setGraph("BANK GRAPH")
opg4j> var pgqlStmt = pgqlConn.createStatement() //create a PGQL Statement
opg4j> String query = "PATH onehop AS ()-[IS transfers]->() "+
             "SELECT v1.id FROM MATCH (v1)-/:onehop/->(v2) "+
...>
...>
             "WHERE v2.id = 365"
opg4j> var resultSet = pgqlStmt.executeQuery(query)
opg4j> resultSet.print() //Prints the query result set
| ID |
+----+
| 132 |
| 435 |
| 296 |
| 327 |
| 328 |
| 399 |
| 684 |
| 919 |
| 923 |
| 771 |
+----+
```

Java

```
import java.sql.Connection;
import java.sql.Statement;
import java.sql.DriverManager;
```



```
import oracle.pg.rdbms.pgql.jdbc.PgqlJdbcRdbmsDriver;
import oracle.pg.rdbms.pgql.PgqlConnection;
import oracle.pg.rdbms.pggl.PgglResultSet;
import oracle.pg.rdbms.pgql.PgqlStatement;
 * This example shows how to execute a PGQL PATH guery.*/
public class PgqlExample4
  public static void main(String[] args) throws Exception
    int idx=0;
   String jdbcUrl
                            = args[idx++];
    String username
                            = args[idx++];
    String password
                            = args[idx++];
    String graph
                             = args[idx++];
    Connection conn = null;
    PgqlStatement pgqlStmt = null;
    PgqlResultSet rs = null;
    try {
      //Get a jdbc connection
      DriverManager.registerDriver(new PgqlJdbcRdbmsDriver());
      conn = DriverManager.getConnection(jdbcUrl, username, password);
      conn.setAutoCommit(false);
      // Get a PGQL connection
      PgqlConnection pgqlConn = PgqlConnection.getConnection(conn);
      pgqlConn.setGraph(graph);
      // Create a PGQL Statement
      pgqlStmt = pgqlConn.createStatement();
     // Execute PGQL Query
      String query =
                 "PATH onehop AS ()-[IS transfers]->() "+
                 "SELECT v1.id FROM MATCH (v1)-/:onehop/->(v2) "+
                 "WHERE v2.id = 365";
      rs = pgqlStmt.executeQuery(query);
      // Print the results
      rs.print();
    finally {
      // close the result set
      if (rs != null) {
        rs.close();
      // close the statement
      if (pgqlStmt != null) {
```

```
pgqlStmt.close();
}
// close the connection
if (conn != null) {
    conn.close();
    }
}
```

```
>>> pgql conn = opg4py.pgql.get connection("<username>","<password>",
"<idbcUrl>")
>>> pgql statement = pgql conn.create statement()
>>> pgql conn.set graph("BANK GRAPH")
>>> query = """
                      PATH onehop AS ()-[IS transfers]->()
                      SELECT v1.id FROM MATCH (v1) -/:onehop/->(v2)
. . .
                      WHERE v2.id = 365
            11 11 11
. . .
>>> pgql statement.execute query(query).print()
| ID |
+----+
| 132 |
| 435 |
| 296 |
| 327 |
| 328 |
| 399 |
| 684 |
| 919 |
| 923 |
| 771 |
+----+
```

11.7.3.3 Executing PGQL Queries Using JDBC Driver

The Oracle Graph Server and Client Release 21.2.0 includes a JDBC driver which allows you to run PGQL queries directly against the Oracle Database. To use the driver, register the following class at the JDBC driver manager:

```
import java.sql.DriverManager;
import oracle.pg.rdbms.pgql.jdbc.PgqlJdbcRdbmsDriver;
...
DriverManager.registerDriver(new PgqlJdbcRdbmsDriver());
```



To make JDBC use the driver, you need to prefix the JDBC URLs with jdbc:oracle:pggl as shown in this example:

```
import java.sql.Connection;
import java.sql.DriverManager;
Connection conn = DriverManager.getConnection("jdbc:oracle:pgql:@<DB
Host>:<DB Port>/<DB SID>", "<DB Username>", "<DB Password>");
```

The part after <code>jdbc:oracle:pgql</code> follows the same syntax as the regular Oracle JDBC thin driver. In other words, you can convert any valid Oracle JDBC thin driver URL into a PGQL driver URL by replacing <code>jdbc:oracle:thin</code> with <code>jdbc:oracle:pgql</code>. Once you obtained a connection object, you can use it to query property graphs using PGQL syntax. For example:

Example 11-6 Executing a PGQL Query using the PGQL JDBC driver

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.PreparedStatement;
import oracle.pq.rdbms.pqql.jdbc.PqqlJdbcRdbmsDriver;
public class PgqlJdbcTest {
  public static void main(String[] args) throws Exception {
    DriverManager.registerDriver(new PgglJdbcRdbmsDriver());
    String jdbcUrl = "jdbc:oracle:pgql:@<DB Host>:<DB Port>/<DB SID>";
    String username = "<DB Username>";
    String password = "<DB Password>";
    try (Connection conn = DriverManager.getConnection(jdbcUrl,
username, password)) {
     String query = "SELECT n.name FROM MATCH(n) ON test graph WHERE
id(n) = ?";
      PreparedStatement pstmt = conn.prepareStatement(query);
     pstmt.setLong(1, 10L);
     pstmt.execute();
      ResultSet rs = pstmt.getResultSet();
      while(rs.next()){
        System.out.println("NAME = " + rs.getString("name"));
    }
  }
}
```

Save the preceding code in a file PgglJdbcTest.java and compile using:

```
javac -cp "<graph-client>/lib/*" PgqlJdbcTest.java
```



The driver is also included in a regular graph server (RPM) install. For example:

```
javac -cp "/opt/oracle/graph/lib/*" PgqlJdbcTest.java
```

11.7.3.4 Executing PGQL Queries to Modify Property Graph Views

You can execute PGQL INSERT, UPDATE and DELETE queries against property graph views using the OPG4J Java shell, OPG4Py Python shell or through a Java or Python application.

It is important to note that unique IDs are not auto generated when inserting vertices or edges in a graph. Therefore, you must ensure that the key column values are either present in the graph properties or they are auto generated by the database (through SEQUENCE and TRIGGERS or implemented with auto increment functionality using IDENTITY column).

The following example inserts two new vertices and also adds an edge relationship between the two vertices.

- JShell
- Java
- Python

JShell

```
opq4j> String pqql =
...> "INSERT VERTEX v1 LABELS (Person) PROPERTIES (v1.name= 'ABC',
v1.height=1.6, v1.birthdate = to date('13/06/1963', 'DD/MM/YYYY')) "+
             , VERTEX v2 LABELS (Person) PROPERTIES (v2.name= 'XYZ',
v2.height=1.75, v2.birthdate = to date('19/06/1963', 'DD/MM/YYYY')) "+
...> " , EDGE e BETWEEN v1 AND v2 LABELS (friendof) PROPERTIES
( e.meeting date = to date('19/06/2021', 'DD/MM/YYYY')) "
pgql ==> "INSERT VERTEX v1 LABELS (Person) PROPERTIES (v1.name= 'ABC',
v1.height=1.6, v1.birthdate = to date('13/06/1963', 'DD/MM/YYYY'))
VERTEX v2 LABELS (Person) PROPERTIES (v2.name= 'XYZ', v2.height=1.75,
v2.birthdate = to date('19/06/1963', 'DD/MM/YYYY'))
                                                      , EDGE e BETWEEN v1
AND v2 LABELS (friendof) PROPERTIES ( e.meeting date = to date('19/06/2021',
'DD/MM/YYYY')) "
opg4j> pgqlStmt.execute(pgql)
$14 ==> false
```

Java

```
String pgq1 = ...> "INSERT VERTEX v1 LABELS (Person) PROPERTIES (v1.name= 'ABC', v1.height=1.6, v1.birthdate = to_date('13/06/1963', 'DD/MM/YYYY')) "+ ...> " , VERTEX v2 LABELS (Person) PROPERTIES (v2.name= 'XYZ', v2.height=1.75, v2.birthdate = to_date('19/06/1963', 'DD/MM/YYYY')) "+ ...> " , EDGE e BETWEEN v1 AND v2 LABELS (friendof) PROPERTIES (e.meeting_date = to_date('19/06/2021', 'DD/MM/YYYY')) "; pgqlStmt.execute(pgq1);
```



```
>>> pgql = """
... INSERT VERTEX v1 LABELS (Person) PROPERTIES (v1.name= 'ABC',
v1.height=1.6, v1.birthdate = to_date('13/06/1963', 'DD/MM/YYYY'))
... , VERTEX v2 LABELS (Person) PROPERTIES (v2.name= 'XYZ',
v2.height=1.75, v2.birthdate = to_date('19/06/1963', 'DD/MM/YYYY'))
... , EDGE e BETWEEN v1 AND v2 LABELS (friendof) PROPERTIES
( e.meeting_date = to_date('19/06/2021', 'DD/MM/YYYY'))
... """
>>> pgql_statement.execute(pgql)
False
```

The following example executes an $\tt UPDATE$ query to modify the edge property that was inserted in the preceding example and subsequently verifies the update operation through a <code>SELECT</code> query.

- JShell
- Java
- Python

JShell

```
opg4j> String pggl = "UPDATE e SET (e.meeting date =
to date('12/02/2022', 'DD/MM/YYYY')) "+
        "FROM MATCH (v1:Person) - [e:friendof] -> (v2:Person) "+
...>
        "WHERE v1.person id = 27 AND v2.person id = 28"
pgql ==> "UPDATE e SET (e.meeting date = to date('12/02/2022', 'DD/MM/
YYYY')) FROM MATCH (v1:Person)-[e:friendof]->(v2:Person) WHERE
v1.person id = 27 AND v2.person id = 28"
opq4j> pqqlStmt.execute(pqql)
$40 ==> false
opg4j>pgqlStmt.executeQuery("SELECT e.meeting date FROM MATCH
(v1:Person) - [e:friendof] -> (v2:Person) WHERE v1.person id = 27").print()
+----+
| MEETING DATE
+----+
| 2022-02-12 00:00:00.0 |
+----+
```

Java

```
String pgql ="UPDATE e SET (e.meeting_date = to_date('12/02/2022',
'DD/MM/YYYY')) "+
"FROM MATCH (v1:Person)-[e:friendof]->(v2:Person) "+
```



```
"WHERE v1.person_id = 27 AND v2.person_id = 28"; pgqlStmt.execute(pgql);
```

A DELETE query allows deleting of vertices and edges in a graph. The following example executes a DELETE query to delete an edge in the graph.

- JShell
- Java
- Python

JShell

```
opg4j> pgqlStmt.execute("DELETE e FROM MATCH (v1:Person)-[e:friendof]-
>(v2:Person) WHERE v.person_id=27")
$14 ==> false
```

Java

```
pgqlStmt.execute("DELETE e FROM MATCH (v1:Person)-[e:friendof]->(v2:Person)
WHERE v.person id=27");
```

Python

```
>>> pgql_statement.execute("DELETE e FROM MATCH (v1:Person)-[e:friendof]-
>(v2:Person) WHERE v1.person_id=27")
False
```



11.7.3.5 Dropping A Property Graph View

You can use the PGQL DROP PROPERTY GRAPH statement to drop a property graph view (PG View). Note that all the metadata tables for the PG View are dropped.

Example 11-7 Creating a Property Graph View

The following example describes the creation of a PG View.

- JShell
- Java
- Python

JShell

```
opg4j> var jdbcUrl="jdbc:oracle:thin:@<host_name>:<port>/<db_service>"
opg4j> var conn =
DriverManager.getConnection(jdbcUrl,"<username>","<password>")
opg4j> var pgqlConn = PgqlConnection.getConnection(conn)
opg4j> var pgqlStmt = pgqlConn.createStatement() //create a PGQL
Statement
opg4j> pgqlStmt.execute("DROP PROPERTY GRAPH <pgview>")
$9 ==> false
```

Java

```
import java.sql.Connection;
import java.sql.Statement;
import java.sql.DriverManager;
import oracle.pg.rdbms.pgql.jdbc.PgqlJdbcRdbmsDriver;
import oracle.pg.rdbms.pgql.PgqlConnection;
import oracle.pg.rdbms.pgql.PgqlStatement;
 ^{\star} This example shows how to drop a property graph view.
 */
public class DropPqView
 public static void main(String[] args) throws Exception
   int idx=0;
    String jdbcUrl
                              = args[idx++];
    String username
                            = args[idx++];
    String password
                             = args[idx++];
    String graph
                              = args[idx++];
    Connection conn = null;
    PgqlStatement pgqlStmt = null;
```



```
try {
     //Get a jdbc connection
     DriverManager.registerDriver(new PgqlJdbcRdbmsDriver());
     conn = DriverManager.getConnection(jdbcUrl, username, password);
     conn.setAutoCommit(false);
     // Get a PGQL connection
     PgqlConnection pgqlConn = PgqlConnection.getConnection(conn);
     // Create PGQL Statement
     pgqlStmt = pgqlConn.createStatement();
     String query = "DROP PROPERTY GRAPH " +pgview;
     pgqlStmt.execute(query);
   finally {
     // close the statement
     if (pgqlStmt != null) {
        pgqlStmt.close();
      // close the connection
     if (conn != null) {
        conn.close();
     }
}
```

```
>>> pgql_conn = opg4py.pgql.get_connection("<username>","<password>",
"jdbc:oracle:thin:@localhost:1521/orclpdb")
>>> pgql_statement = pgql_conn.create_statement()
>>> pgql = "DROP PROPERTY GRAPH <pgview>"
>>> pgql_statement.execute(pgql)
False
```



Part IV

Installing Oracle Graph Server (PGX) and Client

Get started on the installation of the Oracle Graph Server (PGX) and the graph clients.

- Oracle Graph Server and Client Installation
 This chapter describes the steps for installing the graph server and the graph clients.
- Getting Started with the Graph Server (PGX)
 Once you have installed the graph server (PGX), you can start and connect to a graph server instance.



Oracle Graph Server and Client Installation

This chapter describes the steps for installing the graph server and the graph clients.

· Before You Begin

Before you begin to work with Oracle Property Graphs, you must understand the workflow for installing Oracle Graph Server and Client.

Oracle Graph Server Installation

You must install the Oracle Graph Server in order to run graph queries and analytics in the graph server (PGX).

Oracle Graph Client Installation

You can interact with the various graph features using the client CLIs and the graph visualization web client.

Setting Up Transport Layer Security

The graph server (PGX), by default, allows only encrypted connections using Transport Layer Security (TLS). TLS requires the server to present a server certificate to the client and the client must be configured to trust the issuer of that certificate.

12.1 Before You Begin

Before you begin to work with Oracle Property Graphs, you must understand the workflow for installing Oracle Graph Server and Client.

Table 12-1 Workflow for Installing Oracle Graph Server and Client

Sequen ce	Task	Description	More Information
1	Verify Oracle Database Requirements	Ensure that your Oracle Database version is 12.2 and higher.	Verifying Database Compatibility
2	Download Oracle Graph Server and Client	Download Oracle Graph Server and Client from Oracle Software Delivery Cloud or from Oracle Technology Network.	Downloading Oracle Graph Server and Client
3	Install the PL/SQL patch in your Oracle Database	Upgrade the PL/SQL Graph packages in your Oracle Database.	Installing PL/SQL Packages in Oracle Database
4	Install Oracle Graph Server	Install Oracle Graph server, which is available as a separate downloadable package.	Installing Oracle Graph Server
5	Install Oracle Graph Clients	Install the graph clients (such as the graph shell CLIs and graph visualization application) to work with property graphs.	Oracle Graph Client Installation

Table 12-1 (Cont.) Workflow for Installing Oracle Graph Server and Client

Sequen ce	Task	Description	More Information
6	Set up transport layer security	Configure the graph server and client to trust the self-signed certificate.	Setting Up Transport Layer Security
7	Add permissions to publish the graph	Grant permissions to publish graphs.	Adding Permissions to Publish the Graph

- Verifying Database Compatibility
- · Downloading Oracle Graph Server and Client
- Installing PL/SQL Packages in Oracle Database
 Oracle Graph Server and Client will work with Oracle Database 12.2 onward.
 However, you must install the updated PL/SQL packages that are part of the
 Oracle Graph Server and Client download.

12.1.1 Verifying Database Compatibility

Oracle Graph Server and Client works with Oracle Database 12.2 onwards on both onpremises and cloud environments. The cloud environment includes working with all versions of Oracle Autonomous Database (shared) and Oracle Autonomous Database (dedicated).

However, modifying a property graph using a PGQL INSERT, UPDATE, or DELETE query is not supported for Oracle Database 12.2.

12.1.2 Downloading Oracle Graph Server and Client

You can download **Oracle Graph Server and Client** from Oracle Software Delivery Cloud or from Oracle Technology Network.

Table 12-2 summarizes all the files contained in the Oracle Graph Server and Client deployment.

<ver>> denoted in the file name in the Table 12-2 reflects the downloaded Oracle Graph
Server and Client version.

Table 12-2 Components in the Oracle Graph Server and Client Deployment

File	Component	Description
oracle-graph- <ver>.rpm</ver>	Oracle Graph Server	An rpm file to deploy Oracle Graph Server.
oracle-graph-client- <ver>.zip</ver>	Oracle Graph Client	A zip file containing Oracle Graph Client.
oracle-graph-hdfs-connector- <ver>.zip</ver>	Oracle Graph HDFS Connector	A zip file containing libraries to connect Oracle Graph Server with the Apache Hadoop Distributed Filesystem (HDFS).



Table 12-2 (Cont.) Components in the Oracle Graph Server and Client Deployment

File	Component	Description
oracle-graph-sqlcl-plugin- <ver>.zip</ver>	Oracle Graph PGQL Plugin for SQLcl	
oracle-graph-webapps- <ver>.zip</ver>	Oracle Graph Web Applications	A zip file containing .war files for deploying graph servers in an application server.
oracle-graph-plsql- <ver>.zip</ver>	Oracle Graph PL/SQL Patch	A zip file containing PL/SQL packages. It is recommended to update the PL/SQL Graph packages in your database with these packages. Instructions are in the README file.
oracle-graph-visualization- library- <ver>.zip</ver>	Oracle Graph Visualization Library	A zip file containing a Java Script library for the Graph Visualization application.

12.1.3 Installing PL/SQL Packages in Oracle Database

Oracle Graph Server and Client will work with Oracle Database 12.2 onward. However, you must install the updated PL/SQL packages that are part of the Oracle Graph Server and Client download.



You can skip this section if you are using Graph Server and Client with Oracle Autonomous Database. You only need to create roles and assign permissions by executing step-5 and step-6 in Basic Steps for Using an Oracle Database for Authentication. You can run these steps using Database Actions in Oracle Cloud Infrastructure Console.

- 1. Download the Oracle Graph PL/SQL patch component, which is a part of the Oracle Graph Server and Client download from Oracle Software Delivery Cloud.
- 2. Unzip the file oracle-graph-plsql-<ver>.zip into a directory of your choice. <ver> denotes the version downloaded for the Oracle Graph PL/SQL Patch for PL/SQL.



3. Connect to the database as a user with DBA privileges and run the create graph roles.sql script.

```
-- Connect as SYSDBA
SQL> ALTER SESSION SET CONTAINER=<YOUR_PDB_NAME>;
SQL> @create graph roles.sql
```

Optionally, if you plan to work with property graph schema (PG Schema) graphs, then you must install the PL/SQL packages by performing the following steps:

- Choose one of the following directories in the optional pg schema folder:
 - 18c_and_below: This applies only if you are working with Oracle Database
 18c or below.
 - 19c_and_above: This applies only if you are working with Oracle Database
 19c or above.
- As a database user with DBA privileges, follow the instructions in the README.md file in the appropriate directory (that matches your database version). You must do this for every PDB where you will use the graph feature. For example:

```
-- Connect as SYSDBA

SQL> ALTER SESSION SET CONTAINER=<YOUR_PDB_NAME>;
SQL> @opgremov.sql

SQL> @catopq.sql
```

See Using the Property Graph Schema for more information on PG Schema graphs.

4. Connect as a database user with DBA privileges, create a user <graphuser>, and grant the following privileges:

```
SQL> GRANT CREATE SESSION, CREATE TABLE, CREATE VIEW TO <graphuser>
```

Optionally, if you plan to work with property graph schema (PG Schema) graphs, then grant the following privileges:

```
SQL> GRANT CREATE SESSION, ALTER SESSION, CREATE TABLE, CREATE PROCEDURE, CREATE TYPE, CREATE SEQUENCE, CREATE VIEW, CREATE TRIGGER TO <graphuser>
```

5. Grant the appropriate roles (GRAPH_DEVELOPER or GRAPH_ADMINISTRATOR), to the database user created in step 4 for working with the graphs.



Note:

- See User Authentication and Authorization for more information on authorization rules for Graph Server (PGX) and Client 23.1.
- See Upgrading From Graph Server and Client 20.4.x to 21.x for more information if you are migrating to Graph Server (PGX) and Client 21.1 from an earlier version.

```
SQL> GRANT GRAPH_DEVELOPER to <graphuser>
SQL> GRANT GRAPH ADMINISTRATOR to <adminuser>
```

12.2 Oracle Graph Server Installation

You must install the Oracle Graph Server in order to run graph queries and analytics in the graph server (PGX).

The following sections explain the steps to install the Oracle Graph Server in a standalone mode or deploy the server as a web application using Oracle WebLogic Server or Apache Tomcat.

- Using the RPM Installation
 You can run the downloaded RPM file to install the Oracle Graph Server.
- Deploying Oracle Graph Server to a Web Server
 You can deploy Oracle Graph Server to Apache Tomcat or Oracle WebLogic Server.
- User Authentication and Authorization
 The Oracle Graph server (PGX) uses an Oracle Database as identity manager. Both username and password based as well as Kerberos based authentication is supported.

Related Topics

Learn About the Graph Server (PGX)
 The in-memory graph server layer enables you to analyze property graphs using parallel in-memory execution.

12.2.1 Using the RPM Installation

You can run the downloaded RPM file to install the Oracle Graph Server.

- · Prerequisites for Installing Oracle Graph Server
- Installing Oracle Graph Server
- · Uninstalling Oracle Graph Server
- Upgrading Oracle Graph Server

12.2.1.1 Prerequisites for Installing Oracle Graph Server

Your system must adhere to certain prerequisite requirements in order to install the Oracle Graph Server.

The prerequisites for installing the Oracle Graph Server are:

- Verify that you meet the following system requirements:
 - Oracle Linux 7 or 8 x64 or a similar Linux distribution such as RedHat



To run machine learning algorithms on Oracle Linux 8 (Using the Machine Learning Library (PgxML) for Graphs), you must additionally install compat-libgfortran-48 in your system: sudo yum install compat-libgfortran-48

Oracle JDK 8, JDK 11, or JDK 17

Note:

- * Due to a bug in Open JDK, it is recommended to avoid the following Oracle JDK versions:
 - * JDK 11.0.9
 - * JDK 11.0.10
 - * JDK 11.0.11
 - * JDK 11.0.12

See this note for more details.

- Compiling custom graph algorithms using the PGX Algorithm API is not supported on Oracle JDK 17.
- Verify if you already have an installed version of the graph server, by running the following command:

```
sudo rpm -q oracle-graph
[sudo] password for oracle:
oracle-graph-23.1.0-0.x86 64
```

Graph server installation may throw an error if an installation already exists. In that case, see Upgrading Oracle Graph Server to upgrade to a newer version.

12.2.1.2 Installing Oracle Graph Server

The installation steps for installing Oracle Graph Server in standalone mode are as shown:

 As a root user or using sudo, install the RPM file using the rpm command line utility:

```
sudo rpm -i oracle-graph-<version>.rpm
```



Where version> reflects the version that you downloaded. (For example: oracle-graph-23.1.0.x86_64.rpm)

The .rpm file is the graph server.

The following post-installation steps are carried out at the time of the RPM file installation:

- Creation of a working directory in /opt/oracle/graph/pgx/tmp data
- Creation of a log directory in /var/log/oracle/graph
- Automatic generation of self-signed TLS certificates in /etc/oracle/graph

Note:

- You can also choose to configure and set up transport layer security (TLS) in graph server. See Setting Up Transport Layer Security for more details.
- For demonstration purposes, if you wish to disable transport layer security (TLS) in graph server, see Disabling Transport Layer Security (TLS) in Graph Server for more details.
- 2. As root or using sudo, add operating system users allowed to use the server installation to the operating system group oraclegraph. For example:

```
usermod -a -G oraclegraph < graphuser>
```

This adds the specified graph user to the group <code>oraclegraph</code>. Note that <code><graphuser></code> must log out and log in again for this to take effect.

- **3.** As *<graphuser>*, configure the server by modifying the files under /etc/oracle/graph by following the steps under Prepare the Graph Server for Database Authentication.
- **4.** Ensure that authentication is enabled for database users that will connect to the graph server, as explained in User Authentication and Authorization.
- 5. As a root user or using sudo, start the graph server (PGX) by executing the following command:

```
sudo systemctl start pgx
```

You can verify if the graph server has started by executing the following command:

```
\verb|systemctl| status pgx|\\
```

- If the graph server has successfully started, the response may appear as:



```
☐ 32176 java -Dlogback.configurationFile=/etc/oracle/graph/logback-server.xml -Doracle.jdbc.fanEnabled=false -cp /opt/oracle/graph/pqx/bin/../../pqx/server/lib/jackson-databind...
```

The graph server is now ready to accept requests.

 If the graph server has not started, then you must check the log files in /var/log/ oracle/graph for errors. Additionally, you can also run the following command to view any systemd errors:

```
sudo journalctl -u pgx.service
```

Additional installation operations are required for specific use cases, such as:

- Analyze property graphs using Python (see Installing the Python Client From the Graph Server and Client Downloads).
- Deploy the graph server as a web application with Oracle WebLogic Server (see Deploying to Oracle WebLogic Server).
- Deploy GraphViz in Oracle WebLogic Server (see Deploying the Graph Visualization Application in Oracle WebLogic Server).
- Deploy the graph server as a web application with Apache Tomcat (see Deploying to Apache Tomcat).

For instructions to deploy the graph server in Oracle WebLogic Server or Apache Tomcat, see:

- Deploying to Oracle WebLogic Server
- Deploying to Apache Tomcat

You can also deploy the graph server behind a load balancer. See Deploying Oracle Graph Server Behind a Load Balancer for more information.

12.2.1.3 Uninstalling Oracle Graph Server

To uninstall the graph server, ensure that you first shut down the existing graph server version.

• Run the following command as a root user or with sudo:

```
\verb"sudo" rpm -e oracle-graph"
```

12.2.1.4 Upgrading Oracle Graph Server

To upgrade the graph server, ensure that you first shut down the existing graph server version. You can then run the following command with the newer RPM file as an argument.

• Run the following command as a root user or with sudo:

```
sudo rpm -U oracle-graph-23.1.0.x86_64.rpm
```



12.2.2 Deploying Oracle Graph Server to a Web Server

You can deploy Oracle Graph Server to Apache Tomcat or Oracle WebLogic Server.

The following explains the deployment instructions to a web server:

- Deploying to Apache Tomcat
 The example in this topic shows how to deploy the graph server as a web application with Apache Tomcat.
- Deploying to Oracle WebLogic Server
 The example in this topic shows how to deploy the graph server as a web application with Oracle WebLogic Server.

12.2.2.1 Deploying to Apache Tomcat

The example in this topic shows how to deploy the graph server as a web application with Apache Tomcat.

The graph server will work with Apache Tomcat 9.0.x.

- 1. Download the Oracle Graph Webapps zip file from Oracle Software Delivery Cloud. This file contains ready-to-deploy Java web application archives (.war files). The file name will be similar to this: oracle-graph-webapps-<version>.zip.
- 2. Unzip the file into a directory of your choice.
- Locate the .war file that follows the naming pattern: graph-server-<version>pgx<version>.war.
- 4. Configure the graph server.
 - a. Modify authentication and other server settings by modifying the WEB-INF/classes/ pgx.conf file inside the web application archive. See User Authentication and Authorization section for more information.
 - **b.** Optionally, change logging settings by modifying the WEB-INF/classes/logback.xml file inside the web application archive.
 - c. Optionally, change other servlet specific deployment descriptors by modifying the WEB-INF/web.xml file inside the web application archive.
- 5. Copy the .war file into the Tomcat webapps directory. For example:

cp graph-server-<version>-pgx<version>.war \$CATALINA_HOME/webapps/pgx.war



The name you give the war file in the Tomcat webapps directory determines the context path of the graph server application. It is recommended naming the war file as pgx.war.

- 6. Configure Tomcat specific settings, like the correct use of TLS/encryption.
- 7. Ensure that port 8080 is not already in use.



8. Start Tomcat:

```
cd $CATALINA_HOME
./bin/startup.sh
```

The graph server will now listen on localhost:8080/pgx.

You can connect to the server from JShell by running the following command:

```
$ <client_install_dir>/bin/opg4j --base_url https://
localhost:8080/pgx -u <graphuser>
```

Related Topics

The Tomcat documentation (select desired version)

12.2.2.2 Deploying to Oracle WebLogic Server

The example in this topic shows how to deploy the graph server as a web application with Oracle WebLogic Server.

This example shows how to deploy the graph server with Oracle WebLogic Server. Graph server supports WebLogic Server version 12.1.x and 12.2.x.

- 1. Download the Oracle Graph Webapps zip file from Oracle Software Delivery Cloud. This file contains ready-to-deploy Java web application archives (.war files). The file name will be similar to this: oracle-graph-webapps-<version>.zip.
- 2. Unzip the file into a directory of your choice.
- 3. Locate the .war file that follows the naming pattern: graph-server-<version>pgx<version>.war.
- 4. Configure the graph server.
 - a. Modify authentication and other server settings by modifying the WEB-INF/ classes/pgx.conf file inside the web application archive.
 - **b.** Optionally, change logging settings by modifying the WEB-INF/classes/logback.xml file inside the web application archive.
 - c. Optionally, change other servlet specific deployment descriptors by modifying the WEB-INF/web.xml file inside the web application archive.
 - d. Optionally, change WebLogic Server-specific deployment descriptors by modifying the WEB-INF/weblogic.xml file inside the web application archive.
- 5. Configure WebLogic specific settings, like the correct use of TLS/encryption.
- 6. Deploy the .war file to WebLogic Server. The following example shows how to do this from the command line:

```
. $MW_HOME/user_projects/domains/mydomain/bin/setDomainEnv.sh
. $MW_HOME/wlserver/server/bin/setWLSEnv.sh
java weblogic.Deployer -adminurl http://localhost:7001 -username
<username> -password <password> -deploy -source <path-to-war-file>
```

Installing Oracle WebLogic Server



12.2.2.2.1 Installing Oracle WebLogic Server

To download and install the latest version of Oracle WebLogic Server, see

http://www.oracle.com/technetwork/middleware/weblogic/documentation/index.html

12.2.3 User Authentication and Authorization

The Oracle Graph server (PGX) uses an Oracle Database as identity manager. Both username and password based as well as Kerberos based authentication is supported.

The actions that you are allowed to do on the graph server are determined by the privileges enabled by roles that have been granted to you in the Oracle Database.

- Privileges and Roles in Oracle Database
 All database users that work with graphs require the CREATE SESSION privilege in the database.
- Basic Steps for Using an Oracle Database for Authentication
 You can follow the steps explained in this section to authenticate users to the graph server (PGX).
- Prepare the Graph Server for Database Authentication Locate the pgx.conf file of your installation.
- Store the Database Password in a Keystore
- Adding Permissions to Publish the Graph
 There are two ways by which you can view any graph in your graph server (PGX) session in the graph visualization application.
- Token Expiration
 By default, tokens are valid for 1 hour.
- Advanced Access Configuration
 You can customize the following fields inside the pgx_realm block in the pgx.conf file to customize login behavior.
- Customizing Roles and Permissions
 You can fully customize the permissions to roles mapping by adding and removing roles
 and specifying permissions for a role. You can also authorize individual users instead of
- roles.

 Revoking Access to the Graph Server

 To revoke a user's ability to access the graph server, either drap the user from the
 - To revoke a user's ability to access the graph server, either drop the user from the database or revoke the corresponding roles from the user, depending on how you defined the access rules in your pgx.conf file.
- Examples of Custom Authorization Rules
 You can define custom authorization rules for developers.
- Kerberos Enabled Authentication for the Graph Server (PGX)
 The graph server (PGX) can authenticate users using an Oracle Database with Kerberos enabled as identity provider.



12.2.3.1 Privileges and Roles in Oracle Database

All database users that work with graphs require the CREATE SESSION privilege in the database.

Roles that are created for working with graphs are in Table 12-3. These roles are created when you install the PL/SQL package of the Oracle Graph Server and Client distribution on the target database.

Table 12-3 Privileges and Roles in Oracle Database

Role	Operations enabled by this role	Used By
PGX_SESSION_CREATE	Create a new PGX session using the ServerInstance.createSession API.	Graph developers and graph users
PGX_SERVER_GET_INFO	Get status information on the PGX instance using the Admin API.	Users who administer PGX
PGX_SERVER_MANAGE (includes PGX_SERVER_GET_INFO)	Manage the PGX instance using the Admin API to stop or restart PGX.	Users who administer PGX
PGX_SESSION_NEW_GRAPH	Create a new graph in PGX by loading from the database using a config file, using the CREATE PROPERTY GRAPH statement in PGQL, creating a sub-graph from another graph, or using the GraphBuilder.	Graph developers and graph users
PGX_SESSION_GET_PUBLI SHED_GRAPH	Query and view graphs published by another user to the public namespace.	Graph developers and graph users
PGX_SESSION_ADD_PUBLI SHED_GRAPH (includes PGX_SESSION_GET_PUBLI SHED_GRAPH)	Publish a graph to the public namespace.	Graph developers
PGX_SESSION_COMPILE_A LGORITHM	Compile an algorithm using the PGX Algorithm API.	Graph developers
PGX_SESSION_READ_MODE L	Load and use an ML model using PgxML.	Graph developers
PGX_SESSION_MODIFY_MODEL	Create, train, and store an ML model using PgxML.	Graph developers

Few additional roles are also created to group multiple roles together. They provide a convenient way to grant multiple roles to database users. See Mapping Graph Server Roles to Default Privileges for more information on these additional roles.

You can create additional groups that are useful for your application, as described in Adding and Removing Roles and Defining Permissions for Individual Users.

12.2.3.2 Basic Steps for Using an Oracle Database for Authentication

You can follow the steps explained in this section to authenticate users to the graph server (PGX).

1. Use an Oracle Database version that is supported by Oracle Graph Server and Client: version 12.2 or later, including Autonomous Database.

- 2. Be sure that you have ADMIN access (or SYSDBA access for non-autonomous databases) to grant and revoke users access to the graph server (PGX).
- 3. Be sure that all existing users to which you plan to grant access to the graph server have at least the CREATE SESSION privilege granted.
- Be sure that the database is accessible via JDBC from the host where the Graph Server runs.
- 5. As ADMIN (or SYSDBA on non-autonomous databases), run the following procedure to create the roles required by the graph server:

Note:

You can skip this step if you install the PL/SQL packages as part of the Oracle Graph Server and Client installation. All the roles shown in the following code are created as part of the PL/SQL installation automatically. You need to add them separately only if you are using Oracle Graph Server and Client with Autonomous Database. You can run this code using Database Actions in Oracle Cloud Infrastructure Console.

```
DECLARE
  PRAGMA AUTONOMOUS TRANSACTION;
  role exists EXCEPTION;
  PRAGMA EXCEPTION INIT (role exists, -01921);
 TYPE graph roles table IS TABLE OF VARCHAR2(50);
  graph roles graph roles table;
BEGIN
  graph roles := graph roles table(
    'GRAPH DEVELOPER',
    'GRAPH ADMINISTRATOR',
    'GRAPH USER',
    'PGX SESSION CREATE',
    'PGX SERVER GET INFO',
    'PGX SERVER MANAGE',
    'PGX SESSION READ MODEL',
    'PGX SESSION MODIFY MODEL',
    'PGX SESSION NEW GRAPH',
    'PGX SESSION GET PUBLISHED GRAPH',
    'PGX SESSION COMPILE ALGORITHM',
    'PGX SESSION ADD PUBLISHED GRAPH');
  FOR elem IN 1 .. graph roles.count LOOP
  BEGIN
    dbms_output.put_line('create_graph_roles: ' || elem || ': CREATE ROLE
 || graph roles(elem));
   EXECUTE IMMEDIATE 'CREATE ROLE ' || graph roles (elem);
   WHEN role exists THEN
      dbms output.put line('create graph roles: role already exists.
continue');
   WHEN OTHERS THEN
     RAISE;
   END;
```

```
END LOOP;
EXCEPTION
  when others then
    dbms_output.put_line('create_graph_roles: hit error ');
    raise;
END;
/
```

6. Assign default permissions to the roles GRAPH_DEVELOPER, GRAPH_USER and GRAPH_ADMINISTRATOR to group multiple permissions together.

Note:

You can skip this step if you install the PL/SQL packages as part of the Oracle Graph Server and Client installation. All the grants shown in the following code are executed as part of the PL/SQL installation automatically. You need to execute these grants separately only if you are using Oracle Graph Server and Client with Autonomous Database. You can run this code using Database Actions in Oracle Cloud Infrastructure Console.

```
GRANT PGX_SESSION_CREATE TO GRAPH_ADMINISTRATOR;
GRANT PGX_SERVER_GET_INFO TO GRAPH_ADMINISTRATOR;
GRANT PGX_SERVER_MANAGE TO GRAPH_ADMINISTRATOR;
GRANT PGX_SESSION_CREATE TO GRAPH_DEVELOPER;
GRANT PGX_SESSION_NEW_GRAPH TO GRAPH_DEVELOPER;
GRANT PGX_SESSION_GET_PUBLISHED_GRAPH TO GRAPH_DEVELOPER;
GRANT PGX_SESSION_MODIFY_MODEL TO GRAPH_DEVELOPER;
GRANT PGX_SESSION_READ_MODEL TO GRAPH_DEVELOPER;
GRANT PGX_SESSION_CREATE TO GRAPH_USER;
GRANT PGX_SESSION_GET_PUBLISHED_GRAPH TO GRAPH_USER;
```

7. Assign roles to all the database developers who should have access to the graph server (PGX). For example:

```
GRANT graph developer TO <graphuser>
```

where <graphuser> is a user in the database. You can also assign individual permissions (roles prefixed with PGX_) to users directly.

8. Assign the administrator role to users who should have administrative access. For example:

```
GRANT graph administrator to <administratoruser>
```

where <administratoruser> is a user in the database.

12.2.3.3 Prepare the Graph Server for Database Authentication

Locate the pgx.conf file of your installation.

If you installed the graph server via RPM, the file is located at: /etc/oracle/graph/pgx.conf

If you use the webapps package to deploy into Tomcat or WebLogic Server, the pgx.conf file is located inside the web application archive file (WAR file) at: WEB-INF/classes/pgx.conf

Tip: On Linux, you can use vim to edit the file directly inside the WAR file without unzipping it first. For example:

```
vim graph-server-<version>-pgx<version>.war
```

Inside the pgx.conf file, locate the jdbc url line of the realm options:

```
"pgx_realm": {
   "implementation": "oracle.pg.identity.DatabaseRealm",
   "options": {
      "jdbc_url": "<REPLACE-WITH-DATABASE-URL-TO-USE-FOR-AUTHENTICATION>",
      "token_expiration_seconds": 3600,
```

Replace the text with the JDBC URL pointing to your database that you configured in the previous step. For example:

```
"pgx_realm": {
   "implementation": "oracle.pg.identity.DatabaseRealm",
   "options": {
      "jdbc_url": "jdbc:oracle:thin:@myhost:1521/myservice",
      "token_expiration_seconds": 3600,
...
```

Then, start the graph server by running the following command as a root user or with sudo:

```
sudo systemctl start pgx
```

Preparing the Graph Server (PGX) to Connect to Autonomous Database

You can configure your graph server(PGX) to connect to an Autonomous Database instance.

Irrespective of whether your graph server (PGX) instance is running on premises or on Oracle Cloud Infrastructure (OCI), you can perform the following steps to determine the service name to connect to your Autonomous Database instance and update the JDBC URL in /etc/oracle/graph/pgx.conf file.

As a prerequisite requirement, you must generate an SSH key pair consisting of a public key and a private key in order to securely login to the environment where the graph server (PGX) is running.

 Download and save the wallet for your Autonomous Database instance from the Oracle Cloud Infrastructure (OCI) Console. See Download Client Credentials (Wallets) for more information.



2. Upload the wallet from your local machine to the environment where your graph server instance is running with the scp command as shown:

```
scp -i <path_to_ssh_private_key> <path_to_Wallet_DBname>.zip
<username>@<public_ip>:/etc/oracle/graph/wallets
```

The preceding command securely copies the wallet to /etc/oracle/graph/wallets directory on your graph server instance using your ssh private key.

3. Connect to your graph server instance using the ssh private key as shown:

```
ssh -i <ssh private key> <username>@<public ip>
```

4. Unzip the wallet to /etc/oracle/graph/wallets directory and change the group permission as shown:

```
cd /etc/oracle/graph/wallets/
unzip <Wallet_DBname>.zip
chgrp oraclegraph *
```

5. Determine the connect identifier from the tnsnames.ora file in /etc/oracle/ graph/wallets directory. For example, the entry must be similar to:

In the preceding example, graphdb low is the connect identifier.

6. Update the JDBC URL in /etc/oracle/graph/pgx.conf file with the connect identifier determined in the preceding step along with the directory path to the unzipped wallet file. For example:

```
"pgx_realm": {
   "implementation": "oracle.pg.identity.DatabaseRealm",
   "options": {
      "jdbc_url": "jdbc:oracle:thin:@graphdb_low?TNS_ADMIN=/etc/
oracle/graph/wallets",
      "token_expiration_seconds": 3600,
...
```



7. Finally, restart the graph server as shown:

```
sudo systemctl restart pgx
```

12.2.3.4 Store the Database Password in a Keystore

PGX requires a database account to read data from the database into memory. The account should be a low-privilege account (see Security Best Practices with Graph Data).

As described in Reading Graphs from Oracle Database into the Graph Server (PGX), you can read data from the database into the graph server without specifying additional authentication as long as the token is valid for that database user. But if you want to access a graph from a different user, you can do so, as long as that user's password is stored in a Java Keystore file for protection.

You can use the keytool command that is bundled together with the JDK to generate such a keystore file on the command line. See the following script as an example:

```
# Add a password for the 'database1' connection
keytool -importpass -alias database1 -keystore keystore.p12
# 1. Enter the password for the keystore
# 2. Enter the password for the database
# Add another password (for the 'database2' connection)
keytool -importpass -alias database2 -keystore keystore.p12
# List what's in the keystore using the keytool
keytool -list -keystore keystore.p12
```

If you are using Java version 8 or lower, you should pass the additional parameter – storetype pkcs12 to the keytool commands in the preceding example.

You can store more than one password into a single keystore file. Each password can be referenced using the alias name provided.

- Write the PGX graph configuration file to load a graph directly from relational tables
- Read the data
- Secure coding tips for graph client applications

Write the PGX graph configuration file to load a graph directly from relational tables

The following example loads a subset of the HR sample data from relational tables directly into PGX as a graph. The configuration file specifies a mapping from relational to graph format by using the concept of vertex and edge providers.



Note:

Specifying the <code>vertex_providers</code> and edge_providers properties loads the data into an optimized representation of the graph.

```
"name":"hr",
"jdbc url":"jdbc:oracle:thin:@myhost:1521/orcl",
"username":"hr",
"keystore alias": "database1",
"vertex id strategy": "no ids",
"vertex_providers":[
        "name": "Employees",
        "format": "rdbms",
        "database_table_name":"EMPLOYEES",
        "key column": "EMPLOYEE ID",
        "key_type": "string",
        "props":[
             {
                 "name": "FIRST NAME",
                 "type": "string"
             },
                 "name":"LAST NAME",
                 "type": "string"
             },
                 "name": "EMAIL",
                 "type": "string"
             },
                 "name": "SALARY",
                 "type": "long"
        ]
    },
        "name":"Jobs",
        "format": "rdbms",
        "database table name": "JOBS",
        "key column":"JOB ID",
        "key type": "string",
        "props":[
             {
                 "name":"JOB_TITLE",
                 "type": "string"
    },
        "name": "Departments",
```

```
"format": "rdbms",
        "database table name": "DEPARTMENTS",
        "key column": "DEPARTMENT ID",
        "key type": "string",
        "props":[
             {
                 "name": "DEPARTMENT NAME",
                 "type": "string"
        1
],
"edge providers":[
        "name":"WorksFor",
        "format": "rdbms",
        "database table name": "EMPLOYEES",
        "key column": "EMPLOYEE ID",
        "source column": "EMPLOYEE ID",
        "destination column": "EMPLOYEE ID",
        "source vertex provider": "Employees",
        "destination vertex provider": "Employees"
    },
        "name": "WorksAs",
        "format": "rdbms",
        "database table name": "EMPLOYEES",
        "key column": "EMPLOYEE ID",
        "source column": "EMPLOYEE ID",
        "destination column": "JOB ID",
        "source vertex provider": "Employees",
        "destination vertex provider": "Jobs"
    },
        "name": "WorkedAt",
        "format": "rdbms",
        "database table name": "JOB HISTORY",
        "key column": "EMPLOYEE ID",
        "source column": "EMPLOYEE ID",
        "destination column": "DEPARTMENT ID",
        "source vertex provider": "Employees",
        "destination vertex provider": "Departments",
        "props":[
                 "name": "START DATE",
                 "type": "local date"
             },
                 "name": "END DATE",
                 "type": "local date"
        1
    }
]
```

}

Read the data

Now you can instruct PGX to connect to the database and read the data by passing in both the keystore and the configuration file to PGX, using one of the following approaches:

Interactively in the graph shell

If you are using the graph shell, start it with the <code>--secret_store</code> option. It will prompt you for the keystore password and then attach the keystore to your current session. For example:

```
cd /opt/oracle/graph
./bin/opg4j --secret_store /etc/my-secrets/keystore.p12
enter password for keystore /etc/my-secrets/keystore.p12:
```

Inside the shell, you can then use normal PGX APIs to read the graph into memory by passing the JSON file you just wrote into the

readGraphWithProperties API:

```
opg4j> var graph = session.readGraphWithProperties("config.json")
graph ==> PgxGraph[name=hr, N=215, E=415, created=1576882388130]
```

As a PGX preloaded graph

As a server administrator, you can instruct PGX to load graphs into memory upon server startup. To do so, modify the PGX configuration file at /etc/oracle/graph/pgx.conf and add the path the graph configuration file to the preload_graphs section. For example:

```
"preload_graphs": [{
    "name": "hr",
    "path": "/path/to/config.json"
}],
"authorization": [{
    "pgx_role": "GRAPH_DEVELOPER",
    "pgx_permissions": [{
        "preloaded_graph": "hr",
        "grant": "read"
    }]
},
....
]
```

As root user, edit the service file at /etc/systemd/system/pgx.service and change the <code>ExecStart</code> command to specify the location of the keystore containing the password:

ExecStart=/bin/bash start-server --secret-store /etc/keystore.p12





Please note that /etc/keystore.p12 must not be password protected for this to work. Instead protect the file via file system permission that is only readable by oraclegraph user.

After the file is edited, reload the changes using:

```
sudo systemctl daemon-reload
```

Finally start the server:

```
sudo systemctl start pgx
```

In a Java application

To register a keystore in a Java application, use the registerKeystore() API on the PgxSession object. For example:

You can compile and run the preceding sample program using the Oracle Graph Client package. For example:

```
cd $GRAPH_CLIENT
// create Main.java with above contents
javac -cp 'lib/*' Main.java
java -cp '.:conf:lib/*' Main http://myhost:7007 MyKeystorePassword
path/to/config.json
```

Secure coding tips for graph client applications

When writing graph client applications, make sure to never store any passwords or other secrets in clear text in any files or in any of your code.

Do not accept passwords or other secrets through command line arguments either. Instead, use <code>Console.html#readPassword()</code> from the JDK.

12.2.3.5 Adding Permissions to Publish the Graph

There are two ways by which you can view any graph in your graph server (PGX) session in the graph visualization application.

When you log into the graph visualization tool in your browser, that will be a different session from your JShell session or application session. To visualize the graph you are working on in your JShell session or application session in your graph visualization session, you can perform one of the following two steps:

1. Get the session id of your working session using the PgxSession API, and use that session id when you log into the graph visualization application. This is the recommended option.

```
opg4j> session.getId();
$2 ==> "898bdbc3-af80-49b7-9a5e-10ace6c9071c" //session id
```

or

- 2. Grant PGX_SESSION_ADD_PUBLISHED_GRAPH permission and then publish the graph as shown:
 - a. Grant PGX_SESSION_ADD_PUBLISHED_GRAPH role in the database to the user visualizing the graph as shown in the following statement:

```
GRANT PGX SESSION ADD PUBLISHED GRAPH TO <graphuser>
```

 Publish the graph when you are ready to visualize the graph using the publish API.

Note:

- See User Authentication and Authorization for more information on authorization rules for Graph Server (PGX) and Client 21.1.
- See Upgrading From Graph Server and Client 20.4.x to 21.x for more information if you are migrating to Graph Server (PGX) and Client 23.1 from an earlier version.

12.2.3.6 Token Expiration

By default, tokens are valid for 1 hour.

Internally, the graph client automatically renews tokens which are about to expire in less than 30 minutes. This is also configurable by re-authenticating your credentials with the database. By default, tokens can only be automatically renewed for up to 24 times, then you need to login again.

If the maximum amount of auto-renewals is reached, you can log in again without losing any of your session data by using the GraphServer#reauthenticate (instance, "<user>", "<password>") API.



If a session time out occurs before you re-authenticate, then you may lose your session data.

For example:

```
opg4j> var graph = session.readGraphWithProperties(config) // fails because
token cannot be renewed anymore
opg4j> GraphServer.reauthenticate(instance, "<user>",
"<password>".toCharArray()) // log in again
opg4j> var graph = session.readGraphWithProperties(config) // works now
```

12.2.3.7 Advanced Access Configuration

You can customize the following fields inside the pgx_realm block in the pgx.conf file to customize login behavior.

Table 12-4 Advanced Access Configuration Options

Field Name	Description	Default
token_expiration_seconds	After how many seconds the generated bearer token will expire.	3600 (1 hour)
<pre>refresh_time_before_token_expiry_second s</pre>	After how many seconds a token is automatically refreshed before it expires. Note that this value must always be less than the token_expiration_second s value.	1800
connect_timeout_milliseconds	After how many milliseconds an connection attempt to the specified JDBC URL will time out, resulting in the login attempt being rejected.	10000
max_pool_size	Maximum number of JDBC connections allowed per user. If the number is reached, attempts to read from the database will fail for the current user.	64
max_num_users	Maximum number of active, signed in users to allow. If this number is reached, the graph server will reject login attempts.	512
max_num_token_refresh	Maximum amount of times a token can be automatically refreshed before requiring a login again.	24

Note:

The preceding options work only if the realm implementation is configured to be oracle.pg.identity.DatabaseRealm.

12.2.3.8 Customizing Roles and Permissions

You can fully customize the permissions to roles mapping by adding and removing roles and specifying permissions for a role. You can also authorize individual users instead of roles.

This topic includes examples of how to customize the permission mapping.

- Checking Graph Permissions Using API
- Adding and Removing Roles
 You can add new role permission mappings or remove existing mappings by modifying the authorization list.
- Defining Permissions for Individual Users
 In addition to defining permissions for roles, you can define permissions for individual users.
- Defining Permissions to Use Custom Graph Algorithms
 You can define permissions to allow developers to compile custom graph
 algorithms.

12.2.3.8.1 Checking Graph Permissions Using API

You can view your roles and graph permissions using the following PGX API methods:

Table 12-5 API for Checking Graph Permissions

Class	Method	Description
ServerInstance	getPgxUsername()	Name of the current user
ServerInstance	<pre>getPgxUserRoles()</pre>	Role names of the current user
ServerInstance	<pre>getPgxGenericPermission s()</pre>	Non-graph (system) permissions of the current user: Pgx system permissions File-location permissions
PgxGraph	getPermission()	Permission on the graph instance for a current user

You can get all permission-related information using the API in JShell as shown:

- JShell
- Java



JShell

```
/bin/opg4j -b "https://<host>:<port>" -u "<graphuser>"
opg4j> instance
instance ==> ServerInstance[embedded=false,baseUrl=https://
<host>:<port>, serverVersion=null]
opg4j>instance.getPgxUsername()
$2 ==> "ORACLE"
opg4j>instance.getPgxUserRoles()
$3 ==> [GRAPH DEVELOPER]
opg4j>instance.getPgxGenericPermissions()
$4 ==> [PGX SESSION CREATE, PGX SESSION READ MODEL,
PGX SESSION ADD PUBLISHED GRAPH, PGX SESSION NEW GRAPH,
PGX SESSION GET PUBLISHED GRAPH, PGX SESSION MODIFY MODEL]
opg4jvar g = session.readGraphWithProperties("bank graph analytics.json")
g ==> PgxGraph[name=bank graph analytics,N=1000,E=5001,created=1625697341555]
opg4j>g.getPermission() // To get graph permissions
$9 ==> MANAGE
```

Java

```
import oracle.pg.rdbms.*;
import java.sql.Connection;
import java.sql.Statement;
import oracle.pg.rdbms.pgql.PgqlConnection;
import oracle.pg.rdbms.pgql.PgqlStatement;
import oracle.pgx.api.*;
import oracle.ucp.jdbc.PoolDataSourceFactory;
import oracle.ucp.jdbc.PoolDataSource;
import java.nio.file.Files;
import java.nio.file.Path;
 ^{\star} This example shows how to get all permissions.
public class GetPermissions
  public static void main(String[] args) throws Exception
  {
    int idx=0;
    String host
                             = args[idx++];
    String port
                            = args[idx++];
    String sid
                            = args[idx++];
                            = args[idx++];
    String user
    String password
                            = args[idx++];
    String graph
                             = args[idx++];
    Connection conn = null;
    PgxPreparedStatement stmt = null;
    try {
      // Get a jdbc connection
```

```
PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
      pds.setURL("jdbc:oracle:thin:@"+host+":"+port +"/"+sid);
     pds.setUser(user);
     pds.setPassword(password);
      conn = pds.getConnection();
      conn.setAutoCommit(false);
      ServerInstance instance = GraphServer.getInstance("http://
localhost:7007", user, password.toCharArray());
      PgxSession session = instance.createSession("my-session");
      var statement = Files.readString(Path.of("/media/sf Linux/Java/
create-pg.pgql"));
      stmt = session.preparePgql(statement);
      stmt.execute();
      PgxGraph g = session.getGraph(graph);
      System.out.println("Graph: "+ g);
      String userName = instance.getPgxUsername();
      var userRoles = instance.getPgxUserRoles();
      var genericPermissions = instance.getPgxGenericPermissions();
      String graphPermission = g.getPermission().toString();
      System.out.println("Username is " + userName);
      System.out.println("User Roles are " + userRoles);
      System.out.println("Generic permissions are " +
genericPermissions);
      System.out.println("Graph permission is " + graphPermission);
    }
    finally {
      // close the sql statment
     if (stmt != null) {
        stmt.close();
      // close the connection
      if (conn != null) {
        conn.close();
    }
```

On execution, the code gives the following output:

```
Graph: PgxGraph[name=BANK_GRAPH_PG,N=1000,E=5001,created=1625731370402]
Username is ORACLE
User Roles are [GRAPH_DEVELOPER]
Generic permissions are [PGX_SESSION_MODIFY_MODEL, PGX_SESSION_CREATE,
PGX_SESSION_NEW_GRAPH, PGX_SESSION_READ_MODEL,
```

```
PGX_SESSION_ADD_PUBLISHED_GRAPH, PGX_SESSION_GET_PUBLISHED_GRAPH]
Graph permission is MANAGE
```

12.2.3.8.2 Adding and Removing Roles

You can add new role permission mappings or remove existing mappings by modifying the authorization list.

For example:

```
CREATE ROLE MY_CUSTOM_ROLE_1
GRANT PGX_SESSION_CREATE TO MY_CUSTOM_ROLE1
GRANT PGX_SERVER_GET_INFO TO MY_CUSTOM_ROLE1
GRANT MY_CUSTOM_ROLE1 TO SCOTT
```

12.2.3.8.3 Defining Permissions for Individual Users

In addition to defining permissions for roles, you can define permissions for individual users.

For example:

```
GRANT PGX_SESSION_CREATE TO SCOTT GRANT PGX SERVER GET INFO TO SCOTT
```

12.2.3.8.4 Defining Permissions to Use Custom Graph Algorithms

You can define permissions to allow developers to compile custom graph algorithms.

For example,

Add the following static permission to the list of permissions:

```
GRANT PGX SESSION COMPILE ALGORITHM TO GRAPH DEVELOPER
```

12.2.3.9 Revoking Access to the Graph Server

To revoke a user's ability to access the graph server, either drop the user from the database or revoke the corresponding roles from the user, depending on how you defined the access rules in your pgx.conf file.

For example:

```
REVOKE graph developer FROM scott
```

Revoking Graph Permissions

If you have the MANAGE permission on a graph, you can revoke graph access from users or roles using the PgxGraph#revokePermission API. For example:

```
PgxGraph g = ...
g.revokePermission(new PgxRole("GRAPH_DEVELOPER")) // revokes
previously granted role access
g.revokePermission(new PgxUser("SCOTT")) // revokes previously granted
user access
```

12.2.3.10 Examples of Custom Authorization Rules

You can define custom authorization rules for developers.

- Example 12-1
- Example 12-2
- Example 12-3
- Example 12-4

Example 12-1 Allowing Developers to Publish Graphs

Sharing of graphs with other users should be done in Oracle Database where possible. Use GRANT statements on the database tables so that other users can create graphs from the tables.

In the graph server (PGX) you can use the following permissions to share a graph that is already in memory, with other users connected to the graph server.

Table 12-6 Allowed Permissions

Permission	Actions Enabled by this Permission
READ	 READ the graph via the PGX API or in PGQL queries in PGX, create a subgraph, or clone the graph
MANAGE	 Publish the graph or snapshot Includes READ and EXPORT Grant or revoke READ and EXPORT permissions on the graph
EXPORT	Export the graph to a file.Includes READ permission.

The creator of the graph automatically gets the MANAGE permission granted on the graph. If you have the MANAGE permission, you can grant other roles or users READ or EXPORT permission on the graph. You **cannot** grant MANAGE on a graph. The following example of a user named userA shows how:

```
import oracle.pgx.api.*
import oracle.pgx.common.auth.*
```



```
PgxSession session = GraphServer.getInstance("<base-url>", "<userA>",
"<password-of-userA").createSession("userA")
PgxGraph g = session.readGraphWithProperties("examples/sample-graph.json",
"sample-graph")
g.grantPermission(new PgxRole("GRAPH_DEVELOPER"), PgxResourcePermission.READ)
g.publish()</pre>
```

Now other users with the GRAPH_DEVELOPER role can access this graph and have READ access on it, as shown in the following example of userB:

```
PgxSession session = GraphServer.getInstance("<base-url>", "<userB>",
"<password-of-userB").createSession("userB")
PgxGraph g = session.getGraph("sample-graph")
q.queryPgql("select count(*) from match (v)").print().close()</pre>
```

Similarly, graphs can be shared with individual users instead of roles, as shown in the following example:

```
g.grantPermission(new PgxUser("OTHER USER"), PgxResourcePermission.EXPORT)
```

where OTHER_USER is the user name of the user that will receive the EXPORT permission on graph ${\tt q}$.

Example 12-2 Allowing Developers to Access Preloaded Graphs

To allow developers to access preloaded graphs (graphs loaded during graph server startup), grant the read permission on the preloaded graph in the pgx.conf file. For example:

```
"preload_graphs": [{
    "path": "/data/my-graph.json",
    "name": "global_graph"
}],
"authorization": [{
    "pgx_role": "GRAPH_DEVELOPER",
    "pgx_permissions": [{
        "preloaded_graph": "global_graph"
        "grant": "read"
    },
...
```

You can grant READ, EXPORT, or MANAGE permission.

Example 12-3 Allowing Developers Access to the Hadoop Distributed Filesystem (HDFS) or the Local File System

To allow developers to read files from HDFS, you must first declare the HDFS directory and then map it to a read or write permission. For example:

```
CREATE OR REPLACE DIRECTORY pgx_file_location AS 'hdfs:/data/graphs' GRANT READ ON DIRECTORY pgx file location TO GRAPH DEVELOPER
```



Similarly, you can add another permission with GRANT WRITE to allow write access. Such a write access is required in order to export graphs.

Access to the local file system (where the graph server runs) can be granted the same way. The only difference is that location would be an absolute file path without the hdfs: prefix. For example:

CREATE OR REPLACE DIRECTORY pgx_file_location AS '/opt/oracle/graph/data'

Note that in addition to the preceding configuration, the operating system user that runs the graph server process must have the corresponding directory privileges to actually read or write into those directories.

Example 12-4 Allowing Access to Directories on Autonomous Database

To allow developers to read and write from files in Oracle Autonomous Database, you must perform the following steps:

 Connect to your Autonomous Database instance as an ADMIN user using any of the SQL based Oracle Database tools or using Database Actions, the built-in webbased interface.

See Also:

- Connect to Autonomous Database Using Oracle Database Tools
- Connect with Built-in Oracle Database Actions
- 2. Create the directory by specifying the path to the directory using the graph: prefix as shown:

CREATE OR REPLACE DIRECTORY pgx_file_location AS 'graph:/opt/oracle/graph/data'

3. Grant read or write permissions to the directory for the desired role. For example:

GRANT READ ON DIRECTORY pgx file location TO GRAPH DEVELOPER

12.2.3.11 Kerberos Enabled Authentication for the Graph Server (PGX)

The graph server (PGX) can authenticate users using an Oracle Database with Kerberos enabled as identity provider.

You can log into the graph server using a Kerberos ticket and the actions which you are allowed to do on the graph server are determined by the roles that have been granted to you in the Oracle Database.

- Prerequisite Requirements
- Prepare the Graph Server for Kerberos Authentication
- Login to the Graph Server Using Kerberos Ticket





Kerberos Enabled Authentication for the Graph Visualization Application

12.2.3.11.1 Prerequisite Requirements

In order to enable Kerberos authentication on the graph server (PGX), the following system requirements must be met:

- The database needs to have Kerberos authentication enabled. See Configuring Kerberos Authentication for more information.
- Both the database and the Kerberos Authentication Server need to be reachable from the host where the graph server runs.
- The database is prepared for graph server authentication. That is, relevant graph roles have been granted to users who will log into the graph server.

12.2.3.11.2 Prepare the Graph Server for Kerberos Authentication

The following are the steps to enable Kerberos authentication on the graph server (PGX):

1. Locate the pgx.conf file of your installation.



If you installed the graph server via RPM, the file is located at: /etc/oracle/graph/pgx.conf

2. Locate the krb5 conf file line of the realm options, inside the pgx.conf file:

3. Replace the text with the krb5.conf file that you are using for the database and user authentication. For example:

```
"pgx_realm": {
    "implementation": "oracle.pg.identity.DatabaseRealm",
    "options": {
        ...
        "krb5_conf_file": "/etc/krb5.conf",
        "krb5_ticket_cache_dir": "/dev/shm",
        "krb5_max_cache_size": 1024
```



```
}
},
```

Note:

The file provided for the krb5_conf_file option needs to be valid and readable by the graph server. In case you don't replace the krb5_conf_file value or the value is empty, then the graph server will not use Kerberos authentication.

Also, you can set the cache directory that will be used for the graph server to temporarily store Kerberos tickets given by clients as well as the maximum cache size after which new login attempts will be rejected. The cache size represents the maximum amount of concurrent Kerberos sessions active on the graph server.

12.2.3.11.3 Login to the Graph Server Using Kerberos Ticket

The following are the steps to login to the graph server (PGX) using Kerberos ticket:

1. Create a new Kerberos ticket using the okinit command:

```
$ okinit <username>
```

This will prompt for your password and then create a new Kerberos ticket.

2. Connect to a remote graph server with only the base URL parameter using JShell:

```
$ opg4j -b https://localhost:7007
```

Or using Python client:

```
$ opg4py -b https://localhost:7007
```

On Linux, JShell and Python interactive client shells automatically detect the Kerberos ticket on your local file system and use that to authenticate with the graph server.

3. In case the auto-detection is not working, you can also explicitly pass in the ticket to the shell. Run the oklist command, to find the location of the ticket on the local file system.

```
$ oklist
Kerberos Utilities for Linux: Version 19.0.0.0.0 - Production on 31-
MAR-2021 15:26:46
Copyright (c) 1996, 2019 Oracle. All rights reserved.
Configuration file : /etc/krb5.conf.
Ticket cache: FILE:/tmp/krb5cc_54321
Default principal: oracle@realm
```



4. Specify your Kerberos ticket path using the --kerberos_ticket parameter. For example, using JShell:

```
$ opg4j -b https://localhost:7007 --kerberos ticket /tmp/krb5cc 54321
```

Or using Python Client:

```
$ opg4py -b https://localhost:7007 --kerberos ticket /tmp/krb5cc 54321
```

If you are using a Java client program (or JShell on embedded mode), you can get a server instance using the following API:

```
...
ServerInstance instance = GraphServer.getInstance("https://
localhost:7007", "/tmp/krb5cc_54321");
PgxSession session = instance.createSession("my-session");
...
```

If you are using a Python Client program (or opg4py on embedded mode), you can get a server instance using the following API

```
instance = graph_server.get_instance("https://localhost:7007", "/tmp/
krb5cc_54321")
session = instance.create_session("my-session")
...
```

If you are connecting to a remote graph server, all you need is the Oracle Graph Client to be installed. For example:

```
import sys
import pypgx as pgx

sys.path.append("/path/to/graph/client/oracle-graph-client-21.2.0/python/
pypgx/pg/rdbms")

import graph_server

base_url = "https://localhost:7007"
kerberos_ticket = "/tmp/krb5cc_54321"

instance = graph_server.get_instance(base_url, kerberos_ticket)
print(instance)
```

12.3 Oracle Graph Client Installation

You can interact with the various graph features using the client CLIs and the graph visualization web client.

The following sections explain the steps to install the various clients:

Graph Clients

The Oracle Graph client installation supports a Java and a Python client.

Graph Visualization Web Client

You can run the Graph Visualization web application in a standalone mode or it can be deployed to a web container.

Related Topics

Getting Started with the Client Tools

You can use multiple client tools to interact with the graph server (PGX) or directly with the graph data in the database.

12.3.1 Graph Clients

The Oracle Graph client installation supports a Java and a Python client.

The following sections explain the steps to install the clients:

Oracle Graph Java Client

You can install the Java client from the <code>oracle-graph-client-23.1.0.zip</code> file that is shipped with Oracle Graph Server and Client or you can use the Java client on Mayen Central.

Oracle Graph Python Client

You can install the Python client by downloading the <code>oracle-graph-client-23.1.0.zip</code> file that is shipped with Oracle Graph Server and Client or from PyPI.

12.3.1.1 Oracle Graph Java Client

You can install the Java client from the <code>oracle-graph-client-23.1.0.zip</code> file that is shipped with Oracle Graph Server and Client or you can use the Java client on Maven Central.

- Installing the Java Client From the Graph Server and Client Downloads
 You can download the zip file for Oracle Graph Client 23.1.0 and install the Java
 client.
- Using Oracle Graph Java Client on Maven Central You can obtain the property graph Java client from Maven Central.

12.3.1.1.1 Installing the Java Client From the Graph Server and Client Downloads

You can download the zip file for Oracle Graph Client 23.1.0 and install the Java client.

The prerequisites for installing the Java client are:

- A Unix-based operation system (such as Linux) or macOS or Microsoft Windows
- Oracle JDK 11 or JDK 17



Note:

Due to a bug in Open JDK, which causes a deadlock when you attempt to copy and paste into a JShell session, it is recommended that you avoid the following Oracle JDK versions:

- JDK 11.0.9
- JDK 11.0.10
- JDK 11.0.11
- JDK 11.0.12
- 1. Download the Oracle Graph Client from Oracle Software Cloud.

```
For example, oracle-graph-client-23.1.0.zip.
```

- 2. Unzip the file into a directory of your choice.
- 3. Configure your client to trust the self-signed keystore. See Configuring a Client to Trust the Self-Signed Keystore for more information.
- 4. Start the OPG4J shell to connect to the graph server (PGX) as shown:

```
cd <CLIENT_INSTALL_DIR>
./bin/opg4j --base url https://<host>:7007 --username <graphuser>
```

In the preceding code:

- **<CLIENT_INSTALL_DIR>:** Directory where the shell executables are located. The shell executables are generally found in <code>/opt/oracle/graph/bin</code> after server installation, and <code><CLIENT_INSTALL_DIR>/bin</code> after the client installation.
- <host>: Server host

Note:

The graph server (PGX), listens on port 7007 by default. If needed, you can configure the graph server to listen on a different port by changing the port value in the server configuration file (server.conf). See Configuring the Graph Server (PGX) for details.

<graphuser>: Database user

You will be prompted for the database password.

See Starting the OPG4J Shell for more information on the different ways you can start the OPG4J shell.

The OPG4J shell starts and the following command line prompt appears as shown:

```
For an introduction type: /help intro
Oracle Graph Server Shell 23.1.0
Variables instance, session, and analyst ready to use.
opg4j>
```



See Also:

Java API Reference for more information on the Java APIs

12.3.1.1.2 Using Oracle Graph Java Client on Maven Central

You can obtain the property graph Java client from Maven Central.

The Maven artifact for the graph Java client is described as follows:

Group Name: com.oracle.database.graph

· Artifact Name: opg-client

Version: 23.1.0

You can perform the following steps to use the graph Java client from Maven Central:

1. Download and Install Apache Maven on your system.

See Apache Maven Project for more information.

- 2. Add the bin folder with the mvn command to the PATH variable.
- 3. Build your Maven project and navigate to the project directory.
- 4. Edit the pom.xml file on the following:
 - a. Add the graph Java client dependency as shown:

Note:

If you use Gradle as a build tool, then the equivalent dependency declaration for the Java client is:

```
implementation group: 'com.oracle.database.graph', name:
'opg-client', version: '23.1.0'
```

b. Add the following repository as the Java client depends on the Spoofax Language Workbench Library to compile PGQL queries:



5. Optionally, you can skip step 4 and copy the following minimal POM configuration in dir>/pom.xml file:

```
www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/maven-v4 0 0.xsd">
 <modelVersion>4.0.0</modelVersion>
 <groupId>com.mycompany.app</groupId>
 <artifactId>my-app</artifactId>
 <packaging>jar</packaging>
 <version>1.0-SNAPSHOT
 <name>my-app</name>
 <repositories>
   <repository>
     <id>spoofax</id>
     <url>https://artifacts.metaborg.org/content/repositories/releases/
url>
   </repository>
 </repositories>
 <dependencies>
   <dependency>
     <groupId>com.oracle.database.graph
     <artifactId>opg-client</artifactId>
     <version>23.1.0</version>
   </dependency>
 </dependencies>
</project>
```

6. Build your Java code in project_dir>/src/main/java/com/mycompany/app and compile with Maven.

For example, the following code is stored in a file cr_dir>/src/main/java/com/
mycompany/app/App1.java:

```
package com.mycompany.app;
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import oracle.pg.rdbms.pgql.PgqlConnection;
import oracle.pg.rdbms.pgql.PgqlStatement;
import oracle.pg.rdbms.pgql.PgqlResultSet;
import oracle.pgx.api.*;
import oracle.pg.rdbms.GraphServer;
import oracle.pq.rdbms.pqql.jdbc.PqqlJdbcRdbmsDriver;
public class App1 {
 public static void main(String[] args) throws Exception {
    String dbConnectString = args[0];
    String username = args[1];
    String password = args[2];
    // Obtain a JDBC database connection
```

```
DriverManager.registerDriver(new PgglJdbcRdbmsDriver());
    String jdbcUrl = "jdbc:oracle:pgql:@" + dbConnectString;
    System.out.println("connecting to " + jdbcUrl);
    try (Connection conn = DriverManager.getConnection(jdbcUrl,
username, password)) {
      conn.setAutoCommit(false);
      // Create PGQL connection
      PgglConnection pgglConn = PgglConnection.getConnection(conn);
      // Create a PGQL statement to execute PGQL queries
      PgqlStatement pgqlStmt = pgqlConn.createStatement();
      // Create a property graph view using the CREATE PROPERTY
GRAPH statement
     String pgViewName = "BANK GRAPH VIEW";
      String createPgViewQuery =
          "CREATE PROPERTY GRAPH " + pqViewName + " " +
          "VERTEX TABLES ( BANK ACCOUNTS AS ACCOUNTS " +
          "KEY (ID) " +
          "LABEL ACCOUNTS " +
          "PROPERTIES (ID, NAME)" +
          ") " +
          "EDGE TABLES ( BANK TXNS AS TRANSFERS " +
          "KEY (FROM ACCT ID, TO ACCT ID, AMOUNT) " +
          "SOURCE KEY (FROM ACCT ID) REFERENCES ACCOUNTS (ID) " +
          "DESTINATION KEY (TO ACCT ID) REFERENCES ACCOUNTS (ID) " +
          "LABEL TRANSFERS " +
          "PROPERTIES (FROM ACCT ID, TO ACCT ID, AMOUNT,
DESCRIPTION)" +
          ") OPTIONS (PG VIEW)";
      pgqlStmt.execute(createPgViewQuery);
      // Execute a guery to retrieve the first 10 elements of the
graph
      String pgqlQuery =
          "SELECT e.from acct id, e.to acct id, e.amount FROM " +
          "MATCH (n:ACCOUNTS) -[e:TRANSFERS]-> (m:ACCOUNTS) ON " +
          pgViewName + " LIMIT 10";
      PgqlResultSet rs = pgqlStmt.executeQuery(pgqlQuery);
      rs.print();
     // Drop the property graph view using the DROP PROPERTY GRAPH
statement
      String dropPgViewQuery = "DROP PROPERTY GRAPH " + pgViewName;
     pgqlStmt.execute(dropPgViewQuery);
   System.exit(0);
  }
```

You can then compile and run the preceding code by navigating to your project directory and running the following command:

```
mvn compile exec:java -Dexec.mainClass="com.mycompany.app.App1"-
Dexec.arguments='<db-connect-string>,<username>,<password>'
```

On successful processing, the code may produce an output similar to the following. Note, your output may be different depending on your *<db-connect-string>*.

```
[INFO] --- exec-maven-plugin:3.1.0:java (default-cli) @ my-app --- connecting to jdbc:oracle:pgql:@myhost:1521/oradb name = Baz
```

12.3.1.2 Oracle Graph Python Client

You can install the Python client by downloading the oracle-graph-client-23.1.0.zip file that is shipped with Oracle Graph Server and Client or from PyPI.

Alternatively, you can also install the python client in embedded mode.

- Prerequisites for Installing the Python Client
- Installing the Python Client From the Graph Server and Client Downloads
 You can download the zip file for oracle-graph-client-23.1.0 from the Graph Server
 and Client downloads and install the Python client.
- Installing the Python Client from PyPI
 Starting from Oracle Graph Server and Client Release 23.1, you can install the Python client from PyPI.
- Installing the Python Client in Embedded Mode
 You can install and work with the Python client in embedded mode.
- Uninstalling the Python Client
 This section describes how to uninstall the Python client.

12.3.1.2.1 Prerequisites for Installing the Python Client

You must ensure that the following prerequisites are met before you install the Python client:

- 1. Make sure that the following software is installed in your system:
 - Oracle JDK 8 or later
 - Python 3.6 or later

However, if you want to use Pandas related functionality, you must install Python 3.8 or Python 3.9. This is because the pandas package is only shipped for these Python versions.

To verify you are using the right version of the Python client, run the following command:

```
$> python3 --version
Python 3.6.1
```



Note:

Python 2.x is not supported.

For more information on installing Python 3 on Oracle Linux, see Python for Oracle Linux.

2. Ensure that python3-devel is installed in your system.

sudo yum install python3-devel

12.3.1.2.2 Installing the Python Client From the Graph Server and Client Downloads

You can download the zip file for oracle-graph-client-23.1.0 from the Graph Server and Client downloads and install the Python client.

Prior to installing the Python client, ensure that your system meets all the required prerequisites.

You can perform the following steps to install and connect using the Python client:

- 1. Download the Oracle Graph Client from Oracle Software Cloud.
 - For example, oracle-graph-client-23.1.0.zip.
- 2. Install the client through pip.

For example,

```
pip3 install --user oracle-graph-client-23.1.0.zip
```

- 3. Configure your client to trust the self-signed keystore. See Configuring a Client to Trust the Self-Signed Keystore for more information.
- 4. Start the OPG4Py shell to connect to the graph server(PGX) by running the following command:

```
cd <CLIENT_INSTALL_DIR>
./bin/opg4py --base url https://<host>:7007
```

In the preceding code:

- <CLIENT_INSTALL_DIR>: Directory where the shell executables are located.
 The shell executables are found in <CLIENT_INSTALL_DIR>/bin after the client installation.
- <host>: Server host



The graph server (PGX), listens on port 7007 by default. If needed, you can configure the graph server to listen on a different port by changing the port value in the server configuration file (server.conf). See Configuring the Graph Server (PGX) for details.

You are prompted to enter your username and password.

See Starting the OPG4Py Shell for more information on the different ways you can start the OPG4Py shell.

The OPG4Py shell starts and the following command line prompt appears as shown:

```
Oracle Graph Server Shell 23.1.0 >>>
```



You can also install the python client library in Jupyter Notebook. Using the Python API, you can then connect to the graph server (PGX) to run PGQL queries and graph algorithms in a Jupyter Notebook environment. See Using the Jupyter Notebook Interface for more details.

See Also:

Python API Reference for more information on the Python APIs

12.3.1.2.3 Installing the Python Client from PyPI

Starting from Oracle Graph Server and Client Release 23.1, you can install the Python client from PyPI.

You can install the oracle-graph-client-23.1.0.zip package from the PyPI repository using pip.

Before installing the Python client from PyPI, your system must meet the prerequsites mentioned in Prerequisites for Installing the Python Client. In addition:

- Ensure that you set the JAVA HOME environment variable.
- If you are behind a proxy, then set the https_proxy environment variable to the proxy server.

You can install and verify the Python client installation as shown:

1. Install the client through pip.

For example,

```
pip install --user oracle-graph-client
```

This installs the Python client along with all the required dependencies.

2. Verify that your installation is successful.

```
$ python3
Python 3.8.12 (default, Apr 5 2022, 08:07:47)
[GCC 8.5.0 20210514 (Red Hat 8.5.0-10.0.1)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import opg4py
>>> import pypgx
```





Python API Reference for more information on the Python APIs

12.3.1.2.4 Installing the Python Client in Embedded Mode

You can install and work with the Python client in embedded mode.

To install the embedded Python client:

1. Run the following pip command:

```
pip3 install --user /opt/oracle/graph/client/
```

2. Start the OPG4Py shell in embedded mode as shown:

```
cd /opt/oracle/graph
./bin/opg4py
```

Note that the shell executables are found in /opt/oracle/graph/bin after the server installation.

The OPG4Py shell starts and the following command line prompt appears as shown:

```
Oracle Graph Server Shell 23.1.0
>>> instance
ServerInstance(embedded: True, version: 23.1.1)
>>>
```

12.3.1.2.5 Uninstalling the Python Client

This section describes how to uninstall the Python client.

To uninstall the Python client, run the following command:

```
pip3 uninstall pypgx
```

12.3.2 Graph Visualization Web Client

You can run the Graph Visualization web application in a standalone mode or it can be deployed to a web container.

- Running the Graph Visualization Application in Standalone Mode
 If you install the graph server rpm file, the Graph Visualization application starts up by default when you start the PGX server.
- Deploying the Graph Visualization Application
 You must download the oracle-graph-webapps-<version>.zip package and
 deploy the web application archive (WAR) file into your Oracle Weblogic 12.2 (or
 later) or Apache Tomcat (9.x or later) web containers.



Configuring Advanced Options for PGQL Driver Selection
 The Graph Visualization application can be configured to communicate either with the graph server (PGX) or to the Oracle Database.

12.3.2.1 Running the Graph Visualization Application in Standalone Mode

If you install the graph server ${\tt rpm}$ file, the Graph Visualization application starts up by default when you start the PGX server.

The Graph Visualization application requires Oracle Graph Server to be installed as a prerequisite component.

See Installing Oracle Graph Server for more information.

To start the Graph Visualization application in standalone mode:

1. Start the graph server (PGX) as shown:

```
sudo systemctl start pgx
```

The Graph Visualization application starts up by default.

- Configure your Graph Visualization application to trust the self-signed keystore. See Configuring a Client to Trust the Self-Signed Keystore for more information.
- 3. Connect to your browser for running the Graph Visualization application as shown

```
https://localhost:7007/ui
```

One of the following messages may appear:

- Your connection is not private
- Your connection is not secure

Click the Continue or Accept button to proceed.

The Graph Visualization Login screen opens as shown:



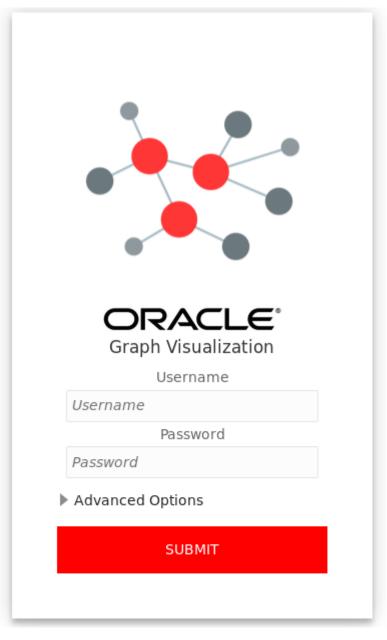


Figure 12-1 Graph Visualization Login

- 4. Enter your database Username and Password.
- 5. Select and configure the required PGQL Driver.

See Configuring Advanced Options for PGQL Driver Selection for more information.

6. Click Submit.

You are now signed into the Graph Visualization application.

The title bar on the query visualization page displays the connection mode along with the relevant URL.



12.3.2.2 Deploying the Graph Visualization Application

You must download the <code>oracle-graph-webapps-<version>.zip</code> package and deploy the web application archive (WAR) file into your Oracle Weblogic 12.2 (or later) or Apache Tomcat (9.x or later) web containers.

- Deploying the Graph Visualization Application to Apache Tomcat
- Deploying the Graph Visualization Application in Oracle WebLogic Server
 The following instructions are for deploying the Graph Visualization application in Oracle
 WebLogic Server 12.2.1.3. You might need to make slight modifications, as appropriate,
 for different versions of the Weblogic Server.

12.3.2.2.1 Deploying the Graph Visualization Application to Apache Tomcat

The following are the steps to deploy the Graph Visualization application to Apache Tomcat.

- 1. Download the Oracle Graph Webapps zip file from Oracle Software Delivery Cloud. This file contains ready-to-deploy Java web application archives (.war files). The file name will be similar to this: oracle-graph-webapps-<version>.zip.
- 2. Configure Tomcat specific settings, like the correct use of TLS/encryption.
- 3. Ensure that port 8080 is not already in use.
- 4. Start Tomcat:

```
cd $CATALINA_HOME
./bin/startup.sh
```

The Graph Visualization application is now listening on localhost:8080/ui

5. Navigate to the Graph Visualization Application using the URL, localhost:8080/ui in your browser.

The Graph Visualization login page appears as shown in Figure 12-1.

- Enter your database credentials and configure the required PGQL driver.
 See Configuring Advanced Options for PGQL Driver Selection for more information.
- 7. Click Submit.

You are now signed into the Graph Visualization application. The title bar on the query visualization page displays the connection mode along with the relevant URL.

12.3.2.2.2 Deploying the Graph Visualization Application in Oracle WebLogic Server

The following instructions are for deploying the Graph Visualization application in Oracle WebLogic Server 12.2.1.3. You might need to make slight modifications, as appropriate, for different versions of the Weblogic Server.

Download the Oracle Graph Webapps zip file from Oracle Software Delivery Cloud. This
file contains ready-to-deploy Java web application archives (.war files). The file name will
be similar to this: oracle-graph-webapps-<version>.zip

2. Start WebLogic Server.

```
# Start Server
cd $MW_HOME/user_projects/domains/base_domain
./bin/startWebLogic.sh
```

3. Enable tunneling.

In order to be able to deploy the Graph Visualization application WAR file over HTTP, you must enable tunneling first. Go to the WebLogic admin console (by default on http://localhost:7001/console). Select **Environment** (left panel) > **Servers** (left panel). Click the server that will run Graph Visualization (main panel). Select (top tab bar), check **Enable Tunneling**, and click **Save**.

4. Deploy the graphviz-<version>-pgviz<graphviz-version>-wls.war file.
To deploy the WAR file to WebLogic Server, use the following command, replacing the <<...>> markers with values matching your installation:

```
cd $MW_HOME/user_projects/domains/base_domain
source bin/setDomainEnv.sh
java weblogic.Deployer -adminurl <<admin-console-url>> -username
<<admin-user>> -password <<admin-password>> -deploy -upload <<path/
to>>/graphviz-<<version>>-pgviz<<graphviz-version>>.war
```

To undeploy, you can use the following command:

```
java weblogic.Deployer -adminurl <<admin-console-url>> -username
<<admin-user>> -password <<admin-password>> -name <<path/to>>/
graphviz-<<version>>-pgviz<<graphviz-version>>.war -undeploy
```

To test the deployment, navigate using your browser to: https://<<fqdn-ip>>:<<port>>/ui.

The Graph Visualization Login screen appears as shown in Figure 12-1.

- Enter your database credentials and configure the required PGQL driver.
 See Configuring Advanced Options for PGQL Driver Selection for more information.
- 6. Click Submit.

You are now logged in and the Graph Visualization query user interface (UI) appears and the graphs from PGX are retrieved.

The title bar on the query visualization page displays the connection mode along with the relevant URL.

12.3.2.3 Configuring Advanced Options for PGQL Driver Selection

The Graph Visualization application can be configured to communicate either with the graph server (PGX) or to the Oracle Database.

You can apply the required configuration at the time of login through the **Advanced Options** settings in the Graph Visualization login page.

You can dynamically change and configure the PGQL driver by following the instructions as appropriate for your preference:

- Configuring the Graph Visualization Application for PGQL on Graph Server (PGX)
- Configuring the Graph Visualization Application for PGQL on Database

12.3.2.3.1 Configuring the Graph Visualization Application for PGQL on Graph Server (PGX)

To configure Graph Visualization application to communicate with a PGX deployment (PGQL on Graph Server):

- Click Advanced Options in the Graph Visualization login page.
- 2. Select Graph Server as shown:

Figure 12-2 PGQL on Graph Server (PGX)

Advanced Options

Graph Server Database

https://localhost:7007

3. Optionally, modify your PGX Base URL.

Note:

- By default, the Graph Visualization application connects to the graph server (PGX) using the PGX base URL defined in the web.xml file for your installation.
- If you wish to disable transport layer security (TLS) in graph server, see
 Disabling Transport Layer Security (TLS) in Graph Server for more details.
- 4. Optionally, enter **Session Id**.

When the Graph Visualization application is using PGQL on Graph Server (PGX), the application will use your Oracle Database as identity manager by default. This means that you log into the application using existing Oracle Database credentials (username and password), and the actions which you are allowed to do on the graph server are determined by the roles that have been granted to you in the Oracle Database.



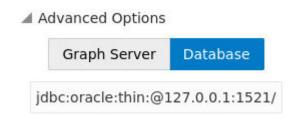
If you wish to enable Kerberos Authentication for the Graph Visualization Application, see Kerberos Enabled Authentication for the Graph Visualization Application for more information.

12.3.2.3.2 Configuring the Graph Visualization Application for PGQL on Database

To configure the Graph Visualization application to communicate with Oracle Database (PGQL on Database):

- 1. Click **Advanced Options** in the Graph Visualization login page.
- Select **Database** as shown:

Figure 12-3 PGQL on Database



3. Optionally, modify the JDBC URL for your Oracle database.



- By default, the Graph Visualization application connects to the database using the JDBC URL defined in the web.xml file for your installation.
- If you wish to enable Kerberos Authentication for the Graph Visualization Application, see Kerberos Enabled Authentication for the Graph Visualization Application for more information.
- If you wish to disable transport layer security (TLS) in graph server, see Disabling Transport Layer Security (TLS) in Graph Server for more details.

12.4 Setting Up Transport Layer Security

The graph server (PGX), by default, allows only encrypted connections using Transport Layer Security (TLS). TLS requires the server to present a server certificate to the client and the client must be configured to trust the issuer of that certificate.

In this release of Graph Server and Client, the RPM file installation, will generate a self-signed server keystore file by default. This $server_keystore.jks$ file contains the server certificate and server private key and is generated into /etc/oracle/graph, for the server to enable TLS. Note that the default password for the generated keystore is changeit and this is configured using an environment variable

PGX SERVER KEYSTORE PASSWORD in /etc/systemd/system/pgx.service file as shown:

[Service]

Environment="PGX_SERVER_KEYSTORE_PASSWORD=changeit"



If this default keystore configuration is sufficient for you to get started and if your connections are only to localhost, you can skip to Configuring a Client to Trust the Self-Signed Keystore.

If you prefer to use a self-signed server certificate, then refer to Using a Self-Signed Server Certificate for more information. However, it is important to note that the server configuration fields, <code>server_cert</code> and <code>server_private_key</code> are deprecated and will be desupported in a future release. After that, you will be required to use the server keystore to store the server certificate and the server private key.

- Using a Self-Signed Server Keystore
 - This section describes the steps to generate a self-signed keystore into /etc/oracle/graph and configure the graph server (PGX) and client to use the keystore.
- Using a Self-Signed Server Certificate
 This section describes the steps to generate a self-signed certificate into /etc/oracle/graph and configure the graph server (PGX) to use this certificate.

12.4.1 Using a Self-Signed Server Keystore

This section describes the steps to generate a self-signed keystore into /etc/oracle/graph and configure the graph server (PGX) and client to use the keystore.

- Generating a Self-Signed Server Keystore
 You can create a server key store using the keytool command.
- Configuring the Graph Server (PGX) When Using a Server Keystore
 You must specify the path to the server keystore in the graph server (PGX) configuration file.
- Configuring a Client to Trust the Self-Signed Keystore
 You must configure your client application to accept the self-signed keystore.

12.4.1.1 Generating a Self-Signed Server Keystore

You can create a server key store using the keytool command.

The following steps show how to create a server keystore with a self-signed certificate:

1. Go to the following directory:

```
cd /etc/oracle/graph
```

2. Run the following command:

```
keytool -genkey -alias pgx -keyalg RSA -keystore server keystore.jks
```

3. Provide the requested details. For example:

```
Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]: localhost
What is the name of your organizational unit?
  [Unknown]: OU
What is the name of your organization?
  [Unknown]: MyOrganization
What is the name of your City or Locality?
```



```
[Unknown]: MyTown
What is the name of your State or Province?
  [Unknown]: MyState
What is the two-letter country code for this unit?
  [Unknown]: US
Is CN=localhost, OU=OU, O=MyOrganization, L=MyTown, ST=MyState,
C=US correct?
  [no]: yes
```

The server keystore.jks is created successfully in cd /etc/oracle/graph.

12.4.1.2 Configuring the Graph Server (PGX) When Using a Server Keystore

You must specify the path to the server keystore in the graph server (PGX) configuration file.



If you deploy the graph server into your web server using the web applications download package, then this section does not apply. Please refer to the manual of your web server for instructions on how to configure TLS.

1. Edit the file at /etc/oracle/graph/server.conf to specify server keystore alias, server keystore provider, server keystore type and the path to the server keystore as shown:

```
{
  "port": 7007,
  "enable_tls": true,
  "enable_client_authentication": false,
  "server_keystore": "/etc/oracle/graph/server_keystore.jks",
  "server_keystore_alias": "pgx",
  "server_keystore_type": "PKCS12",
  "server_keystore_provider": "SUN",
  "ca_certs": [],
  "working_dir": "/opt/oracle/graph/pgx/tmp_data"
}
```

 Set the keystore password using an OS environment variable called PGX_SERVER_KEYSTORE_PASSWORD or with a java property called pgx.SERVER_KEYSTORE PASSWORD.

For example, to set the keystore password in PGX_SERVER_KEYSTORE_PASSWORD, edit the file at /etc/systemd/system/pgx.service as shown:

```
[Service]
Environment="PGX_SERVER_KEYSTORE_PASSWORD=<keystore_password>"
```



3. Reload the systemd configuration by running the following command:

```
sudo systemctl daemon-reload
```

4. Restart the graph server.

Note:

- You should use a certificate issued by a certificate authority (CA) which is trusted by your organization. If you do not have a CA certificate, you can temporarily create a self-signed certificate and get started.
- Always use a valid certificate trusted by your organization. We do not recommend the usage of self-signed certificates for production environments.

12.4.1.3 Configuring a Client to Trust the Self-Signed Keystore

You must configure your client application to accept the self-signed keystore.

To configure a client to trust the self-signed keystore, the root certificate must be imported to your Java installation local trust store.

• For a Java or a Python client, you must import the root certificate to all the Java installations used by all the clients.

Note:

The JShell client requires Java 11 or later.

- For the Graph Visualization application, you must import the root certificate to the system Java installation of the environment running the graph server (PGX) or the web server serving the graph visualization application. That is, the JDK installation which is used by the OS user running the server that serves the Graph Visualization application.
- For the Graph Zeppelin interpreter client, you must import the root certificate to the Java installation used by the Zeppelin server.

You can import the root certificate as shown in the following step:

- Run the following command as a root user or with sudo:
 - 1. For Java 8 (make sure JAVA HOME is set):

```
sudo keytool -importkeystore -srckeystore /etc/oracle/graph/
server_keystore.jks -destkeystore $JAVA_HOME/jre/lib/security/cacerts
-deststorepass changeit -srcstorepass changeit -noprompt
```

2. For Java 11 or later (make sure JAVA HOME is set):

```
sudo keytool -importkeystore -srckeystore /etc/oracle/graph/
server_keystore.jks -destkeystore $JAVA_HOME/lib/security/cacerts -
deststorepass changeit -srcstorepass changeit -noprompt
```



where <code>changeit</code> is the sample keystore password. You can change this password to a password of your choice. Be sure to remember this password as you will need it to modify the certificate.

1. If you are upgrading the graph server from a previous release, you must first delete the certificate by running the following command appropriate to your Java version. You must run the command using sudo or as a root user:

For Java 8:

```
sudo keytool -delete -alias pgx -keystore $JAVA_HOME/jre/lib/
security/cacerts -storepass changeit
```

For Java 11 or later:

```
sudo keytool -delete -alias pgx -keystore $JAVA_HOME/lib/
security/cacerts -storepass changeit
```

2. Import the new certificate as shown in the preceding step.

12.4.2 Using a Self-Signed Server Certificate

This section describes the steps to generate a self-signed certificate into /etc/oracle/graph and configure the graph server (PGX) to use this certificate.

- Generating a Self-Signed Server Certificate
 You can create a self-signed server certificate using the openssl command.
- Configuring the Graph Server (PGX)
 You must specify the path to the server certificate and the server's private key in PEM format in the graph server (PGX) configuration file.
- Configuring a Client to Trust the Self-Signed Certificate
 You must configure your client application to accept the self-signed graph server
 (PGX) certificate.

12.4.2.1 Generating a Self-Signed Server Certificate

You can create a self-signed server certificate using the openss1 command.

The following steps show how to generate a self-signed server certificate.

1. Go to the following directory:

```
cd /etc/oracle/graph
```

2. Execute the following commands:

```
openssl req -new -newkey rsa:2048 -days 365 -nodes -x509 -subj "/
C=US/ST=MyState/L=MyTown/O=MyOrganization/CN=ROOT" -keyout
ca_key.pem -out ca_certificate.pem
openssl genrsa -out server_key_traditional.pem 2048
openssl pkcs8 -topk8 -in server_key_traditional.pem -inform pem -
out server_key.pem -outform pem -nocrypt
openssl req -new -subj "/C=US/ST=MyState/L=MyTown/O=MyOrganization/
CN=localhost" -key server key.pem -out server.csr
```



```
chmod 600 server_key.pem
openssl x509 -req -CA ca_certificate.pem -CAkey ca_key.pem -in server.csr
-out server_certificate.pem -days 365 -CAcreateserial
chown oraclegraph:oraclegraph server_key.pem
```

Note:

- The certificate mentioned in the above example will only work for the host localhost. If you have a different domain, you must replace localhost with your domain name.
- The above self-signed certificate is valid only for 365 days.

12.4.2.2 Configuring the Graph Server (PGX)

You must specify the path to the server certificate and the server's private key in PEM format in the graph server (PGX) configuration file.

Note:

If you deploy the graph server into your web server using the web applications download package, then this section does not apply. Please refer to the manual of your web server for instructions on how to configure TLS.

1. Edit the file at /etc/oracle/graph/server.conf, and specify the paths to the server certificate and the server's private key in PEM format, as shown:

```
"port": 7007,
"enable_tls": true,
"server_private_key": "/etc/oracle/graph/server_key.pem",
"server_cert": "/etc/oracle/graph/server_certificate.pem",
"enable_client_authentication": false,
"working_dir": "/opt/oracle/graph/pgx/tmp_data"
```

2. Restart the graph server.

Note:

- You should use a certificate issued by a certificate authority (CA) which is trusted by your organization. If you do not have a CA certificate, you can temporarily create a self-signed certificate and get started.
- Always use a valid certificate trusted by your organization. We do not recommend the usage of self-signed certificates for production environments.



12.4.2.3 Configuring a Client to Trust the Self-Signed Certificate

You must configure your client application to accept the self-signed graph server (PGX) certificate.

To configure a client to trust the self-signed certificate, the root certificate must be imported to your Java installation local trust store.

 For a Java or a Python client, you must import the root certificate to all the Java installations used by all the clients.



The JShell client requires Java 11 or later.

- For the Graph Visualization application, you must import the root certificate to the system Java installation of the environment running the graph server (PGX) or the web server serving the graph visualization application. That is, the JDK installation which is used by the OS user running the server that serves the Graph Visualization application.
- For the Graph Zeppelin interpreter client, you must import the root certificate to the Java installation used by the Zeppelin server.

You can import the root certificate as shown in the following step:

- Run the following command as a root user or with sudo:
 - 1. For Java 8 (make sure JAVA HOME is set):

```
sudo keytool -import -trustcacerts -keystore $JAVA_HOME/jre/lib/
security/cacerts -storepass changeit -alias pgx -file /etc/
oracle/graph/ca_certificate.pem -noprompt
```

2. For Java 11 or later (make sure JAVA HOME is set):

```
sudo keytool -import -trustcacerts -keystore $JAVA_HOME/lib/
security/cacerts -storepass changeit -alias pgx -file /etc/
oracle/graph/ca certificate.pem -noprompt
```

where changeit is the sample keystore password. You can change this password to a password of your choice. Be sure to remember this password as you will need it to modify the certificate.

1. If you are upgrading the graph server from a previous release, you must first delete the certificate by running the following command appropriate to your Java version. You must run the command using sudo or as a root user:

For Java 8:

```
sudo keytool -delete -alias pgx -keystore $JAVA_HOME/jre/lib/
security/cacerts -storepass changeit
```



For Java 11 or later:

sudo keytool -delete -alias pgx -keystore $\AVA_HOME/lib/security/cacerts -storepass changeit$

2. Import the new certificate as shown in the preceding step.



13

Getting Started with the Graph Server (PGX)

Once you have installed the graph server (PGX), you can start and connect to a graph server instance.

- Starting the Graph Server (PGX)
 This section describes the commands to start and stop the graph server (PGX).
- Connecting to the Graph Server (PGX)
 This section explains how to connect to the graph server (PGX) running in remote mode or when deployed as a web application on Apache Tomcat or Oracle WebLogic Server.

13.1 Starting the Graph Server (PGX)

This section describes the commands to start and stop the graph server (PGX).

A preconfigured version of Apache Tomcat is bundled, which allows you to start the graph server (PGX) by running a script.

As a prerequisite to start the graph server in remote mode, you must ensure that Oracle graph server is installed in your system. See Installing Oracle Graph Server for instructions to install the graph server (PGX).



See Usage Modes of the Graph Server (PGX) for more information on the different graph server execution modes.

- · Starting and Stopping the Graph Server (PGX) Using the Command Line
- Configuring the Graph Server (PGX)

13.1.1 Starting and Stopping the Graph Server (PGX) Using the Command Line

PGX is integrated with systemd to run it as a Linux service in the background.

If you need to configure the server before starting it, see Configuring the Graph Server (PGX) and Configuration Parameters for the Graph Server (PGX) Engine for more information on the configuration options.

The commands to start and stop the graph server (PGX) and the PGX engine are as follows:



You can run the following commands without sudo if you are the root user.

To start the PGX server as a daemon process, run the following command:

sudo systemctl start pgx

To stop the server, run the following command:

sudo systemctl stop pgx

If the server does not start up, you can see if there are any errors by running:

sudo journalctl -u pgx.service

For more information about how to interact with systemd on Oracle Linux, see the Oracle Linux administrator's documentation.

13.1.2 Configuring the Graph Server (PGX)

You can configure the graph server (PGX) by modifying the /etc/oracle/graph/server.conf file. The following table shows the valid configuration options, which can be specified in JSON format.

Table 13-1 Configuration Parameters for the Graph Server (PGX)

•	Parameter	Туре	Description	Default
	ca_certs	of	List of files storing trusted certificates (PEM format). If enable_tls is set to false, this field has no effect.	[]



Table 13-1 (Cont.) Configuration Parameters for the Graph Server (PGX)

Parameter	Туре	Description	Default
ciphers	array	List of cipher suites to be	["TLS_ECDHE_ECDSA_WITH_AES
	of	used by the server. For	28 GCM SHA256",
	string	example, [cipher1, cipher2.]	"TLS ECDHE ECDSA WITH AES 2
	_		6 GCM SHA384",
			"TLS ECDHE RSA WITH AES 128
			GCM SHA256",
			"TLS ECDHE RSA WITH AES 256
			GCM SHA384",
			"TLS ECDHE ECDSA WITH AES 1
			8 CBC SHA256",
			"TLS ECDHE RSA WITH AES 128
			CBC SHA256",
			"TLS ECDHE ECDSA WITH AES 2
			6 CBC SHA384",
			"TLS ECDHE RSA WITH AES 256
			CBC SHA384",
			"TLS DHE RSA WITH AES 128 (
			M SHA256",
			"TLS DHE DSS WITH AES 128 (
			M SHA256",
			"TLS DHE RSA WITH AES 128 (
			C SHA256",
			"TLS DHE DSS WITH AES 128 (
			C SHA256",
			"TLS_DHE_DSS_WITH_AES_256_(
			M_SHA384",
			"TLS_DHE_RSA_WITH_AES_256_(
			C_SHA256",
			"TLS_DHE_DSS_WITH_AES_256_0
			C_SHA256",
			"TLS_ECDHE_ECDSA_WITH_AES_1
			8_CBC_SHA",
			"TLS_ECDHE_RSA_WITH_AES_256
			CBC_SHA",
			"TLS_ECDHE_ECDSA_WITH_AES_2
			6_CBC_SHA",
			"TLS_DHE_DSS_WITH_AES_128_(
			C_SHA",
			"TLS_DHE_RSA_WITH_AES_128_(
			C_SHA",
			"TLS_DHE_DSS_WITH_AES_256_0
			C_SHA",
			"TLS_DHE_RSA_WITH_AES_256_0
			C_SHA",
			"TLS_RSA_WITH_AES_128_GCM_S
			A256",
			"TLS_DH_DSS_WITH_AES_128_GC
			_SHA256",
			"TLS_ECDH_ECDSA_WITH_AES_12
			_GCM_SHA256",
			"TLS RSA WITH AES 256 GCM S



Table 13-1 (Cont.) Configuration Parameters for the Graph Server (PGX)

Parameter	Туре	Description	Default
			A384",
			"TLS DH DSS WITH AES 256 GCM
			SHA384",
			"TLS ECDH ECDSA WITH AES 256
			_GCM_SHA384",
			"TLS_RSA_WITH_AES_128_CBC_SH
			A256",
			"TLS_DH_DSS_WITH_AES_128_CBC SHA256",
			"TLS ECDH ECDSA WITH AES 128
			CBC SHA256",
			"TLS_RSA_WITH_AES_256_CBC_SH
			A256",
			"TLS_DH_DSS_WITH_AES_256_CBC SHA256",
			"TLS_ECDH_ECDSA_WITH_AES_256
			_CBC_SHA384",
			"TLS_RSA_WITH_AES_128_CBC_SH
			A", "TLS DH DSS WITH AES 128 CBC
			SHA",
			"TLS_ECDH_ECDSA_WITH_AES_128
			CBC SHA",
			"TLS RSA WITH AES 256 CBC SH
			A",
			"TLS_DH_DSS_WITH_AES_256_CBC SHA",
			"TLS ECDH ECDSA WITH AES 256
			_CBC_SHA"]
context_path	string	This can be used to	/
		change the context path.	
		For example, if you specify	
		port as 7007 and	
		context path as /pgx, the server will listen on	
		https://	
		localhost:7007/pgx	
onablo +la	booles		+ 2010
enable_tls		If true, the server enables transport layer security	true
	n	(TLS).	
nort	intor		7007
port	intege	Port the graph server (PGX) server should listen	1001
	r	On.	
			



Table 13-1 (Cont.) Configuration Parameters for the Graph Server (PGX)

Parameter	Туре	Description	Default
server_cert	string	The path to the server certificate to be presented to TLS clients (PEM format). This file must only contain one certificate. If your certificate is a chain and contains a root certificate, add it to ca_certs instead. If enable_tls is set to false, this field has no effect Note: Starting from Graph Server and Client Release 22.3 onwards, this field is deprecated. Use server_keystore instead.	NULL
server_keysto re	string	The path to the keystore to be used for server connections. If this field is present along with server_cert or server_private_key, then an error will be raised. If enable_tls is set to false, then this field has no effect.	NULL
server_keysto re_alias	string	This is the server keystore alias of server_keystore.	NULL
server_keysto re_provider	string	This is the server keystore provider of server_keystore.	SunJSSE
server_keysto re_type	string	This is the server keystore type of server_keystore.	JKS



Table 13-1 (Cont.) Configuration Parameters for the Graph Server (PGX)

Parameter	Туре	Description	Default
server_privat e_key	string	This is the path to the file storing the private key of the server (PEM format). For security reasons, the file must have only Read and Write permissions only for the owner (600 permissions in a POSIX filesystem), otherwise an error will be thrown. If enable_tls is set to false, this field has no effect. Note: Starting from Graph Server and Client Release 22.3 onwards, this field is deprecated. Use server_keystore instead.	NULL
tls_version	string	TLS version to be used by the server. For example, ${\tt TLSv1.2}$	TLSv1.2
working_dir	string	The working directory used by the server to store temporary files. Needs to be writable by the process which started the server and should not be touched by any other process while the server is running.	

The graph server (PGX) enables two-way SSL/TLS (Transport Layer Security) by default. The server enforces TLS 1.2 and disables certain cipher suites known to be vulnerable to attacks. Upon a TLS handshake, both the server and the client present certificates to each other, which are used to validate the authenticity of the other party. Client certificates are also used to authorize client applications.

Example Configuration of server.conf File

```
{
  "port": 7007,
  "enable_tls": true,
  "server_cert": "server_cert.pem",
  "server_private_key": "server_key.pem",
  "ca_certs": [
      "server_cert.pem"
]
}
```



Example Configuration of server.conf File Using a Keystore

```
"port": 7007,
"enable_tls": true,
"enable_client_authentication": true,
"server_keystore": "/pgx/cert/server_keystore.rsa",
"server_keystore_alias": "pgx",
"server_keystore_provider": "JsafeJCE",
"server_keystore_type": "PKCS12"
```

13.2 Connecting to the Graph Server (PGX)

This section explains how to connect to the graph server (PGX) running in remote mode or when deployed as a web application on Apache Tomcat or Oracle WebLogic Server.

The prerequisite requirement to connect to the graph server is to have the graph server (PGX) up and running. See Starting and Stopping the Graph Server (PGX) Using the Command Line for more information on the commands to start the graph server.



If you are using the graph server (PGX) as a library, see Using the Graph Server (PGX) as a Library for more information.

- Connecting with the Graph Client CLIs
- Connecting with Java
- · Connecting with Python

13.2.1 Connecting with the Graph Client CLIs

The simplest way to connect to a remote graph server (PGX) instance is to specify the base URL of the server along with the database user name required for the graph server (PGX) authentication as shown:

- JShell
- Python

JShell

```
cd /opt/oracle/graph
./bin/opg4j --base url https://<host>:<port> --username <graphuser>
```



Python

```
cd /opt/oracle/graph
./bin/opg4py --base_url https://<host>:<port> --username <graphuser>
```

where:

- <host>: is the server host name
- <port>: is the server port
- <graphuser>: is the database user
 You will be prompted for the database password.

See Also:

- User Authentication and Authorization
- Java API Reference for information on the Java APIs
- Python API Reference for information on the Python APIs

About Logging HTTP Requests

The graph shell suppresses all debugging messages by default. To see which HTTP requests are executed, set the log level for oracle.pgx to DEBUG, as shown in this example:



Enabling these logs can lead to sensitive information like passwords getting printed on the screen.

- JShell
- Python

JShell

```
opg4j> loglevel("oracle.pgx","DEBUG")
===> Log level of oracle.pgx logger set to DEBUG
opg4j> session.readGraphWithProperties("bank_graph_analytics.json",
"bank_graph");
06:29:03,702 DEBUG CommonsVfsProvider - resolve
bank_graph_analytics.json
06:29:03,702 DEBUG AbstractConfigFactory - parse graph config from
bank graph analytics.json (parent: file:///opt/oracle/graph)
```



```
06:29:03,709 DEBUG RemoteUtils - create session cookie (session ID =
f5d029d7-2924-4cd4-86a9-6999c1ce5e3f)
06:29:03,713 DEBUG RemoteUtils - no value for the sticky cookie given
06:29:03,713 DEBUG RemoteUtils - create csrf token cookie (token =
36acbee2-6b78-4c13-b114-41040809833a)
06:29:03,713 DEBUG HttpRequestExecutor - Requesting POST https://
localhost:7007/core/v1/loadGraph HTTP/1.1 with payload
{"graphConfig":"HRcBVFVc00dfXU9bUEhGEEYOdkMUElZYRFpcZqBeDxBYCxFcRGY2c21wIBUBE
ElaW11HQV5vVhpYAFRIFAMbeC5+NithRx9EEkwUVRADR1FBXVhGF1BpT0ZfSEFpAlZBQ1RXG1kTKi
EXSREVCUAWU1dmElJfRlxKa11GDBJdSV1EQwMPd1paVhZfFxYXSREIB1qBEqqbMEVMXEpUUV9HQUq
WSV1FFVBDV01QVq1uAApZEF4IRA9GdHdqMGhkdhseFk1REBBdQ11CCFZDaU9cSxdUGzpFF1wQD1EB
QhADRnZOUVZHWllHQUqWQVdXBVBDURsDQkFSEQBUEVY5DVAdb19YFEdEXF4QDktVDxdRUBQUBVhZV
1tYSqZuFwRXCVY5CFQJVRADRnVsfHJtcWlzJjdrbHViQxUPXVxAZhdIEwAXXxEKCVsDEh4bAlhfX1
hGFhcWEQBWQEsUHGQBFE9cSxdUGzpFF1wQD1EBQkEbXmxWEFJXTXJXDAhBQFYUWxtkchsVGw1QDqA
XXxEnBVYLRVxNFxUBEFVdVUldDQMWF0MUAktIV01cZqhUGjpYBEMWD1sDEqhNFkJITxUQUExAAqZV
X11pFVhPWlxmVwJcBkcPR3EnKH47fn19IWQPHhtZUVRrFx1ESBoMQ1BDQ1xeXBETT0dTCkELB0FGC
me":"bank graph"," csrf token":"36acbee2-6b78-4c13-b114-41040809833a"}
06:29:03,788 DEBUG HttpRequestExecutor - received HTTP status 202
06:29:03,789 DEBUG HttpRequestExecutor -
{"futureId": "7f7a2206-8881-4c1e-909f-6e8778be617c"}
06:29:03,789 DEBUG PgxRemoteFuture - Requesting GET https://localhost:7007/
core/v1/futures/x-future-id/status HTTP/1.1
06:29:03,801 DEBUG PgxRemoteFuture - Requesting GET https://localhost:7007/
core/v1/futures/x-future-id/value HTTP/1.1
06:29:03,831 DEBUG RemoteUtils - received HTTP status 201
06:29:03,831 DEBUG RemoteUtils - {"id":"8B473228-0751-49A9-
A945-9A0E4011AB69", "links": [{"href": "https://localhost:7007/core/v1/graphs/x-
graph-id", "rel": "self", "method": "GET", "interaction": ["async-polling"] },
{"href": "https://localhost:7007/core/v1/graphs/x-graph-
id", "rel": "canonical", "method": "GET", "interaction": ["async-
polling"]}], "graphName": "bank graph", "vertexTables": { "Accounts":
{"name":"Accounts", "metaData": {"name":"Accounts", "idType":"integer", "labels":
["Accounts"], "properties": [], "edgeProviderNamesWhereSource":
["Transfers"], "edgeProviderNamesWhereDestination":
["Transfers"], "id":null, "links":null}, "providerLabels":
["Accounts"], "entityKeyType": "integer", "isIdentityKeyMapping": false, "vertexPr
operties":{}, "vertexLabels":{"id":"04156FFE-
A3C1-4A6D-87E5-879A0895BBD4", "links": [{"href": "https://localhost:7007/
core/v1/graphs/x-graph-id/properties/x-property-
name", "rel": "self", "method": "GET", "interaction": ["async-polling"] },
{"href": "https://localhost:7007/core/v1/graphs/x-graph-id/properties/x-
property-name", "rel": "canonical", "method": "GET", "interaction": ["async-
polling"] }], "dimension":-1, "propertyId": "04156FFE-
A3C1-4A6D-87E5-879A0895BBD4", "name": vertex labels ", "entityType": "vertex"
","type":"ro string set","namespace":"2C17C639-3771-3E30-88AE-34D6B380C5EC","t
ransient":false}, "transient":false}}, "edgeTables":{"Transfers":
{"name": "Transfers", "metaData":
{"name":"Transfers", "idType":"long", "directed":true, "labels":
["Transfers"], "properties":
[{"name":"AMOUNT","id":null,"propertyType":"float","dimension":0,"transient":
true, "links": null, "propertyId": "AF2A2D0A-9C8C-478F-
BD74-3444A7DD7339"}], "sourceVertexProviderName": "Accounts", "destinationVertex
ProviderName": "Accounts", "id":null, "links":null}, "providerLabels":
["Transfers"], "entityKeyType": "long", "isIdentityKeyMapping": true, "sourceVerte
```

```
xTableName": "Accounts", "destinationVertexTableName": "Accounts", "edgePro
perties":{"4046D845-D0C6-4231-A69B-F69D4963CD91":{"id":"4046D845-
D0C6-4231-A69B-F69D4963CD91", "links": [{"href": "https://localhost:7007/
core/v1/graphs/x-graph-id/properties/x-property-
name","rel":"self","method":"GET","interaction":["async-polling"]},
{"href": "https://localhost:7007/core/v1/graphs/x-graph-id/properties/x-
property-name", "rel": "canonical", "method": "GET", "interaction": ["async-
polling"]}], "dimension": 0, "propertyId": "4046D845-D0C6-4231-A69B-
F69D4963CD91", "name": "AMOUNT", "entityType": "edge", "type": "float", "names
pace":"2C17C639-3771-3E30-88AE-34D6B380C5EC","transient":false}},"edgeL
abel":{"id":"9763546A-1860-49A4-9292-77D2AA04F4BB","links
06:29:03,836 DEBUG PgxSession - engine reports latest snapshot is
621849 milli-seconds old. Max age is 0 milli-seconds
06:29:03,836 DEBUG PgxSession - ==> try to check out newer snapshot
06:29:03,836 DEBUG RemoteUtils - create session cookie (session ID =
f5d029d7-2924-4cd4-86a9-6999c1ce5e3f)
06:29:03,836 DEBUG RemoteUtils - no value for the sticky cookie given
06:29:03,836 DEBUG RemoteUtils - create csrf token cookie (token =
36acbee2-6b78-4c13-b114-41040809833a)
06:29:03,836 DEBUG HttpRequestExecutor - Requesting POST https://
localhost:7007/core/v1/graphs/x-graph-id/refresh HTTP/1.1 with payload
{"blockIfFull":false," csrf token":"36acbee2-6b78-4c13-
b114-41040809833a"}
06:29:03,878 DEBUG HttpRequestExecutor - received HTTP status 202
06:29:03,878 DEBUG HttpRequestExecutor -
{"futureId": "898d546e-583f-4d37-9ca9-d1e10134037f"}
06:29:04,135 DEBUG PgxRemoteFuture - Requesting GET https://
localhost:7007/core/v1/futures/x-future-id/status HTTP/1.1
06:29:04,828 DEBUG PgxRemoteFuture - Requesting GET https://
localhost:7007/core/v1/futures/x-future-id/value HTTP/1.1
06:29:04,858 DEBUG RemoteUtils - received HTTP status 201
06:29:04,859 DEBUG RemoteUtils - {"id":"BE960B34-E135-4CF8-AB2F-
E1A6E2D7DB60", "links": [{"href": "https://localhost:7007/core/v1/
graphs/x-graph-id", "rel": "self", "method": "GET", "interaction": ["async-
polling"]},{"href":"https://localhost:7007/core/v1/graphs/x-graph-
id", "rel": "canonical", "method": "GET", "interaction": ["async-
polling"]}], "graphName": "bank graph", "vertexTables": { "Accounts":
{"name": "Accounts", "metaData":
{"name":"Accounts","idType":"integer","labels":
["Accounts"], "properties":[], "edgeProviderNamesWhereSource":
["Transfers"], "edgeProviderNamesWhereDestination":
["Transfers"], "id":null, "links":null}, "providerLabels":
["Accounts"], "entityKeyType": "integer", "isIdentityKeyMapping": false, "ve
rtexProperties":{},"vertexLabels":{"id":"19D95502-40D5-47F2-9F45-
B1CD09ECB989", "links": [{"href": "https://localhost:7007/core/v1/
graphs/x-graph-id/properties/x-property-
name","rel":"self","method":"GET","interaction":["async-polling"]},
{"href": "https://localhost:7007/core/v1/graphs/x-graph-id/properties/x-
property-name", "rel": "canonical", "method": "GET", "interaction": ["async-
polling"]}],"dimension":-1,"propertyId":"19D95502-40D5-47F2-9F45-
B1CD09ECB989", "name": " vertex labels ", "entityType": "vertex", "type": "
ro string set", "namespace": "2C17C639-3771-3E30-88AE-34D6B380C5EC", "tran
sient":false}, "transient":false}}, "edgeTables":{"Transfers":
{"name": "Transfers", "metaData":
{"name":"Transfers", "idType":"long", "directed":true, "labels":
```

```
["Transfers"], "properties":
[{"name":"AMOUNT","id":null,"propertyType":"float","dimension":0,"transient":
true, "links":null, "propertyId": "9A49BC0C-F8AA-465A-B8D6-
CA5A92BAE2C9"}], "sourceVertexProviderName": "Accounts", "destinationVertexProvi
derName":"Accounts", "id":null, "links":null}, "providerLabels":
["Transfers"], "entityKeyType": "long", "isIdentityKeyMapping": true, "sourceVerte
xTableName": "Accounts", "destinationVertexTableName": "Accounts", "edgePropertie
s":{"FED6FE43-D311-46B6-9A5A-E8DC0D7B56C6":{"id":"FED6FE43-D311-46B6-9A5A-
E8DC0D7B56C6", "links": [{"href": "https://localhost:7007/core/v1/graphs/x-
graph-id/properties/x-property-
name","rel":"self","method":"GET","interaction":["async-polling"]},
{"href": "https://localhost:7007/core/v1/graphs/x-graph-id/properties/x-
property-name", "rel": "canonical", "method": "GET", "interaction": ["async-
polling"|}|,"dimension":0,"propertyId":"FED6FE43-D311-46B6-9A5A-
E8DC0D7B56C6", "name": "AMOUNT", "entityType": "edge", "type": "float", "namespace":
"2C17C639-3771-3E30-88AE-34D6B380C5EC", "transient": false}}, "edgeLabel":
{"id":"371D2AC6-4EC5-45AD-8885-B3590F56D944","links
$5 ==> PgxGraph[name=bank graph, N=1000, E=5001, created=1621160944599]
```

Python

```
>>>setloglevel("oracle.pgx","DEBUG")
>>>session.read graph with properties("/scratch/PG/data/
bank graph analytics.json")
10:37:46.308 [main] DEBUG oracle.pqx.vfs.CommonsVfsProvider - resolve /
scratch/PG/data/bank graph analytics.json
10:37:46.314 [main] DEBUG oracle.pgx.config.AbstractConfigFactory - parse
graph config from /scratch/PG/data/bank graph analytics.json (parent:
file:///scratch/PG/data)
10:37:46.464 [main] DEBUG oracle.pgx.client.RemoteUtils - create session
cookie (session ID = 786242ef-a430-4964-8379-662079514d2e)
10:37:46.465 [main] DEBUG oracle.pgx.client.RemoteUtils - no value for the
sticky cookie given
10:37:46.466 [main] DEBUG oracle.pqx.client.RemoteUtils - create csrf token
cookie (token = f02ccd16-56e1-4e61-8d17-d6122f5ea48f)
10:37:46.565 [main] DEBUG oracle.pgx.client.HttpRequestExecutor - Requesting
POST https://localhost:7007/core/v1/loadGraph HTTP/1.1 with payload
{"graphConfig": "TBpAV0ZGAB5yEUZcRkRQXERHDwJoTBtGU09tU1hVQFxaRqhHfnwUHhZcBAtIQ
w4RcU5XVkNaWUsRGxtGU09tRE5JUBMORlsLQ11RV0YQSURBDlVXWUNTGwxPD1tBU1hZU21ZVU5mWF
{\tt BEFFsLUBoMRkZHABsBQ1BSRExWWEVRckxSVVVIaVhTXVIbDxN2JXwuaHl1cXtnKzJ+QxgRVkJGVFd}
ADWIRRV1PW0UQHBVJR15EFxBfbEMURk1CAEQXQ11dREhTXEQWARpdVlRIFAwQeXMbSB1PRkYcR10U
CBZBERRED1MRHA9aWFtRDwIReXhqcxRPbUpkGRNRAFUAaEhEXUJbAQNfEhYJa1YWU1NNcltcW0xAW
BOIE15dFx0WC1MIUhoMEGBABAheB1FBOw8YG11RVGdHTklIFAwOXFhXUhMYR14KVlxfXFMOXx0PAk
ZWUV1RZ11RVGdeVkldX1hVEq1fVF1HAU9JFVxTQUBbCwdZCFtdb1tRS0JRVWdDRVZbX1JXQhUDF3B
XB10QWUxFEBqQAQNeFV1dUVldVlhrTldfQlRDFAwQZHhmdHJ3MG0scxoaEFBTEQdPAEdWb11VW1pR
clzSWlwPDBRwcXlyamVsKmFHGxpFXUFABgNyAltfRUBaGwwWa2p8emZsdXVmb359Fx0WAl0XWllCE
A40FwJPDEcRHA9ES11EXhoJbEIPOk9CVRUDF1dYC1MRFROUXFVfAEOXO3V+f3h6bRRJAUMRO0BdUx
QIEkRNR1haAxBJFVZXX1EQX0RpJGdwYmRkbX97YxpOahUPRV1HQ1RcakdRFkYAT2dGQFtEDAJIExY
JEmxXWllBQ0xAFURwGhRcUVpcFwsWBlMLXGdRQFVCDTlMD1VfSVldWkUWUDgzNzmCqX+4", "graph
Name":null, csrf token":"f02ccd16-56e1-4e61-8d17-d6122f5ea48f"}
10:37:47.082 [main] DEBUG oracle.pgx.client.HttpRequestExecutor - received
HTTP status 202
10:37:47.082 [main] DEBUG oracle.pgx.client.HttpRequestExecutor -
{"futureId": "aeae66b7-e1d6-46f3-b78f-1c0d9642d308"}
```

```
10:37:47.086 [pgx-client-thread-2] DEBUG
oracle.pgx.client.PgxRemoteFuture - Requesting GET https://
localhost:7007/core/v1/futures/x-future-id/status HTTP/1.1
10:37:50.434 [pgx-client-thread-2] DEBUG
oracle.pgx.client.PgxRemoteFuture - Requesting GET https://
localhost:7007/core/v1/futures/x-future-id/value HTTP/1.1
10:37:50.539 [pgx-client-thread-2] DEBUG oracle.pgx.client.RemoteUtils
- received HTTP status 201
10:37:50.539 [pqx-client-thread-2] DEBUG oracle.pqx.client.RemoteUtils
- {"id":"2EE6F933-5679-4387-B79B-A4AAD0814DC6","links":
[{"href":"https://localhost:7007/core/v1/graphs/x-graph-
id","rel":"self","method":"GET","interaction":["async-polling"]},
{"href": "https://localhost:7007/core/v1/graphs/x-graph-
id", "rel": "canonical", "method": "GET", "interaction": ["async-
polling"]}], "graphName": "bank graph analytics", "vertexTables":
{"Accounts": {"name": "Accounts", "metaData":
{"name":"Accounts","idType":"integer","labels":
["Accounts"], "properties":
[{"name":"ID", "id":null, "propertyType":"integer", "dimension":0, "transie
nt":true, "links":null, "propertyId": "C933AB01-358B-4553-974D-0C54845719F
{"name":"NAME","id":null,"propertyType":"string","dimension":0,"transie
nt":true, "links":null, "propertyId": "E0EDF68C-5AFE-4886-
B3A0-385CE56F1814"}], "edgeProviderNamesWhereSource":
["Transfers"], "edgeProviderNamesWhereDestination":
["Transfers"], "id":null, "links":null}, "providerLabels":
["Accounts"], "keyPropertyName":null, "entityKeyType": "integer", "isIdenti
tyKeyMapping":false, "vertexProperties": { "F5023C5A-5294-4383-
BA4F-0109C67A020F":{"id":"F5023C5A-5294-4383-
BA4F-0109C67A020F", "links": [{"href": "https://localhost:7007/core/v1/
graphs/x-graph-id/properties/x-property-
name","rel":"self","method":"GET","interaction":["async-polling"]},
{"href": "https://localhost:7007/core/v1/graphs/x-graph-id/properties/x-
property-name", "rel": "canonical", "method": "GET", "interaction": ["async-
polling"]}], "dimension":0, "propertyId": "F5023C5A-5294-4383-
BA4F-0109C67A020F", "name": "ID", "entityType": "vertex", "type": "integer", "
namespace":"2C17C639-3771-3E30-88AE-34D6B380C5EC","transient":false},"C
588F89E-53EB-46DD-A83D-1078138C42C7":{"id":"C588F89E-53EB-46DD-
A83D-1078138C42C7", "links": [{"href": "https://localhost:7007/core/v1/
graphs/x-graph-id/properties/x-property-
name","rel":"self","method":"GET","interaction":["async-polling"]},
{"href": "https://localhost:7007/core/v1/graphs/x-graph-id/properties/x-
property-name", "rel": "canonical", "method": "GET", "interaction": ["async-
polling"]}], "dimension":0, "propertyId": "C588F89E-53EB-46DD-
A83D-1078138C42C7", "name": "NAME", "entityType": "vertex", "type": "string",
"namespace":"2C17C639-3771-3E30-88AE-34D6B380C5EC","transient":false}},
"vertexLabels":{"id":"89FB1A38-BCDF-4FB0-9F8C-59471872BEA3","links":
[{"href": "https://localhost:7007/core/v1/graphs/x-graph-id/
properties/x-property-name","rel":"self","method":"GET","interaction":
["async-polling"]}, {"href": "https://localhost:7007/core/v1/graphs/x-
graph-id/properties/x-property-name", "rel": "canonical", "method": "GET
10:37:50.655 [pqx-client-thread-2] DEBUG oracle.pqx.api.PqxSession -
engine reports latest snapshot is 0 milli-seconds old. Max age is 0
milli-seconds
10:37:50.655 [pgx-client-thread-2] DEBUG oracle.pgx.api.PgxSession -
```

```
==> within range. Return snapshot
PgxGraph(name: bank_graph_analytics, v: 1000, e: 5001, directed: True,
memory(Mb): 0)
```

13.2.2 Connecting with Java

You can obtain a connection to a remote graph server (PGX) instance by simply passing the base URL of the remote PGX instance to the <code>getInstance()</code> method. By doing this, your application automatically uses the PGX client libraries to connect to a remotely-located graph server (PGX).

You can specify the base URL when you initialize the graph server (PGX) instance using Java. An example is as follows. A URL to an graph server (PGX) is provided to the getInMemAnalyst API call.

```
import oracle.pgx.api.*;
import oracle.pg.rdbms.*;
ServerInstance instance = GraphServer.getInstance("https://
<hostname>:<port>","<username>","<password>".toCharArray());
PgxSession session = instance.createSession("my-session");
```



See Java API Reference for more information on the Java APIs.

Starting and Stopping the PGX Engine

13.2.2.1 Starting and Stopping the PGX Engine

You can start the graph server (PGX) from the application by making a call to instance.startEngine() which takes a JSON object as an argument for PGX configuration.



- See Connecting with Java for more information about connecting to a graph server (PGX) instance and obtaining a ServerInstance object.
- See Configuration Parameters for the Graph Server (PGX) Engine for the various configuration options for the graph server (PGX).

Stopping the PGX Engine

You can stop the PGX engine using one of the following APIs:

```
instance.shutdownEngineNow(); // cancels pending tasks, throws exception if engine is not running instance.shutdownEngineNowIfRunning(); // cancels pending tasks, only tries
```

```
to shut down if engine is running
if (instance.shutdownEngine(30, TimeUnit.SECONDS) == false) {
   // doesn't accept new tasks but finishes up remaining tasks
   // pending tasks didn't finish after 30 seconds
}
```

Note:

Shutting down the PGX engine keeps the Apache Tomcat server alive, but new sessions cannot be created. Also, all the current sessions and tasks will be cancelled and terminated.

13.2.3 Connecting with Python

You can connect to a remote graph server (PGX) instance in your Python program. You must first authenticate with the remote server before you can create a session as illustrated in the following example:

```
import pypgx
import opg4py
import opg4py.graph server as graph server
pgql conn = opg4py.pgql.get connection("<username>","<password>",
"<jdbc url>")
pgql statement = pgql conn.create statement()
pgql = """
        CREATE PROPERTY GRAPH bank graph
        VERTEX TABLES (
          bank accounts
            LABEL ACCOUNTS
            PROPERTIES (ID, NAME)
        )
        EDGE TABLES (
          bank txns
            SOURCE KEY (from acct id) REFERENCES bank accounts
            DESTINATION KEY (to acct id) REFERENCES bank accounts
            LABEL TRANSFERS
            PROPERTIES (FROM_ACCT_ID, TO_ACCT_ID, AMOUNT, DESCRIPTION)
        ) OPTIONS (PG VIEW)
pgql statement.execute(pgql)
instance = graph server.get instance("<base url>", "<username>",
"<password>")
session = instance.create session("my session")
graph = session.read graph by name('BANK GRAPH', 'pg view')
analyst = session.create analyst()
analyst.pagerank(graph)
rs = graph.query pgql("SELECT id(x), x.pagerank FROM MATCH (x) LIMIT
5")
rs.print()
```



To execute, save the above program into a file named program.py and run the following command:

```
python3 program.py
```

You will see the following output:

Converting PGQL result set into pandas dataframe

Additionally, you can also convert the PGQL result set to a pandas. DataFrame object using the to_pandas() method. This makes it easier to perform various data filtering operations on the result set and it can also be used in Lambda functions. For example,

```
example_query = (
    "SELECT n.name AS name, n.age AS age "
    "WHERE (n)"
)
result_set = sample_graph.query_pgql(example_query)
result_df = result_set.to_pandas()

result_df['age_bin'] = result_df['age'].apply(lambda x: int(x)/20) # create
age bins based on age ranges
```

Note:

To view the complete set of available Python APIs, see OPG4PY Python API Reference.

Part V

Using the Graph Server (PGX)

The graph server (PGX) of Oracle Graph supports a set of analytical functions.



You can only load property graph views into the graph server (PGX). Loading SQL property graphs into the graph server (PGX) is not supported.

This part describes the following:

- Developing Applications with Graph Analytics
 In order to run graph algorithms, the graph application connects to the graph server (PGX) in the middle tier, which in turn connects to the Oracle Database.
- Using the Machine Learning Library (PgxML) for Graphs
 The graph server (PGX) provides a machine learning library oracle.pgx.api.mllib,
 which supports graph-empowered machine learning algorithms.
- Executing PGQL Queries Against the Graph Server (PGX)
 This section describes the Java APIs that are used to execute PGQL queries in the graph server (PGX).
- REST Endpoints for the Graph Server



Developing Applications with Graph Analytics

In order to run graph algorithms, the graph application connects to the graph server (PGX) in the middle tier, which in turn connects to the Oracle Database.

Note:

You can only load property graph views into the graph server (PGX). Loading a SQL property graph into the graph server (PGX) is not supported.

About Vertex and Edge IDs

The graph server (PGX) enforces by default the existence of a unique identifier for each vertex and edge in a graph.

Graph Management in the Graph Server (PGX)

You can load a graph into the graph server (PGX) and perform different actions such as publish, store, or delete a graph.

Keeping the Graph in Oracle Database Synchronized with the Graph Server

You can use the FlashbackSynchronizer API to automatically apply changes made to graph in the database to the corresponding PgxGraph object in memory, thus keeping both synchronized.

Optimizing Graphs for Read Versus Updates in the Graph Server (PGX)

The graph server (PGX) can store an optimized graph for other reads or updates. This is only relevant when the updates are made directly to a graph instance in the graph server.

Executing Built-in Algorithms

The graph server (PGX) contains a set of built-in algorithms that are available as Java APIs.

Using Custom PGX Graph Algorithms

A custom PGX graph algorithm allows you to write a graph algorithm in Java syntax and have it automatically compiled to an efficient parallel implementation.

Creating Subgraphs

You can create subgraphs based on a graph that has been loaded into memory. You can use filter expressions or create bipartite subgraphs based on a vertex (node) collection that specifies the left set of the bipartite graph.

Using Automatic Delta Refresh to Handle Database Changes

You can automatically refresh (auto-refresh) graphs periodically to keep the in-memory graph synchronized with changes to the property graph stored in the property graph tables in Oracle Database (VT\$ and GE\$ tables).

User-Defined Functions (UDFs) in PGX

User-defined functions (UDFs) allow users of PGX to add custom logic to their PGQL queries or custom graph algorithms, to complement built-in functions with custom requirements.

Using the Graph Server (PGX) as a Library

When you utilize PGX as a library in your application, the graph server (PGX) instance runs in the same JVM as the Java application and all requests are translated into direct function calls instead of remote procedure invocations.

14.1 About Vertex and Edge IDs

The graph server (PGX) enforces by default the existence of a unique identifier for each vertex and edge in a graph.

When loading a graph into the graph server(PGX), you can retrieve these unique vertex and edge IDs using PgxGraph.getVertex(ID id) and PgxGraph.getEdge(ID id), or by PGQL queries using the built-in id() method.

The following supported ID generation strategies can be selected through the configuration parameters <code>vertex id strategy</code> and <code>edge id strategy</code>:

- keys as ids: This is the default strategy to generate vertex IDs.
- partitioned_ids: This is the recommended strategy for partitioned graphs.
- unstable generated ids: This results in system generated vertex or edge IDs.
- no_ids: This strategy disables vertex or edge IDs and therefore prevents you from calling APIs using vertex or edge IDs.

Using keys to generate IDs

The default strategy to generate the vertex IDs is to use the keys provided during loading of the graph (keys_as_ids). In that case, each vertex should have a vertex key that is unique across all providers.

For edges, by default no keys are required in the edge data, and edge IDs will be automatically generated by PGX (unstable_generated_ids). This automatic ID generation can be applied for vertex IDs also. Note that the generation of vertex or edge IDs is not guaranteed to be deterministic. If required, it is also possible to load edge keys as IDs.

The partitioned_ids strategy requires keys to be unique only within a vertex or edge provider (data source). The keys do not have to be globally unique. Globally unique IDs are derived from a combination of the provider name and the key inside the provider, as provider_name>(<unique_key_within_provider>). For example, Account (1).

The partititioned_ids strategy can be set through the configuration fields vertex_id_strategy and edge_id_strategy. For example,



```
"key type": "integer",
      "props": [
          "name": "ID",
          "type": "integer"
        },
          "name": "NAME",
          "type": "string"
      ],
      "loading": {
        "create key mapping" : true
    }
  ],
  "edge providers": [
    {
      "name": "Transfers",
      "format": "rdbms",
      "database table name": "BANK TXNS",
      "key column": "ID",
      "source column": "FROM ACCT ID",
      "destination column": "TO ACCT ID",
      "source vertex provider": "Accounts",
      "destination vertex provider": "Accounts",
      "props": [
          "name": "ID",
          "type": "integer"
          "name": "AMOUNT",
          "type": "double"
      ],
      "loading": {
        "create key mapping" : true
    }
  1
}
```

Note:

All available key types are supported in combination with partitioned IDs.

After the graph is loaded, PGX maintains information about which property of a provider corresponds to the key of the provider. In the preceding example, the vertex property ${\tt ID}$ happens to correspond to the vertex key and also the edge property ${\tt ID}$ happens to correspond to the edge key. Each provider can have at most one such "key property" and the property can have any name.

Key properties are used for internal optimizations as well as for providing keys for the vertex or edge or both when inserting new entities. Key properties are currently non-updatable. Trying to update a key property will result in an error. For example,

vertex key property ID cannot be updated

Using an auto-incrementer to generate partitioned IDs

It is recommended to always set <code>create_key_mapping</code> to <code>true</code> to benefit from performance optimizations. But if there are no single-column keys for edges, <code>create_key_mapping</code> can be set to false. Similarly, <code>create_key_mapping</code> can be set to false for vertex providers also. IDs will be generated via an auto-incrementer, for <code>example Accounts(1)</code>, <code>Accounts(2)</code>, <code>Accounts(3)</code>.

See PGQL Queries with Partitioned IDs for more information on executing PGQL queries with partitioned IDs.

14.2 Graph Management in the Graph Server (PGX)

You can load a graph into the graph server (PGX) and perform different actions such as publish, store, or delete a graph.

- Reading Graphs from Oracle Database into the Graph Server (PGX)
 Once logged into the graph server (PGX), you can read graphs from the database into the graph server.
- Storing a Graph Snapshot on Disk
 After reading a graph into memory, you can make any changes to the graph (such as running the PageRank algorithm and storing the values as vertex properties), and then store this snapshot of the graph on disk.
- Publishing a Graph
 You can publish a graph that can be referenced by other sessions.
- Deleting a Graph
 In order to reduce the memory usage of the graph server (PGX), the session must drop the unused graph objects created through the getGraph() method, by invoking the destroy() method.

14.2.1 Reading Graphs from Oracle Database into the Graph Server (PGX)

Once logged into the graph server (PGX), you can read graphs from the database into the graph server.

Your database user must exist and have read access on the graph data in the database.

There are several ways to load a property graph view into the graph server (PGX) from Oracle Database:

- Using the readGraphByName API see Loading a PG View Using the readGraphByName API for more details.
- Using the PGQL CREATE PROPERTY GRAPH statement see Creating a Property Graph Using PGQL for more details.



- Using the PgViewSubgraphReader#fromPgView API to create and load a subgraph see Loading a Subgraph from Property Graph Views for more details.
- Using a PGX graph configuration file in JSON format see Loading a Graph Using a JSON Configuration File for more details.
- Using the GraphConfigBuilder class to create Oracle RDBMS graph configurations programmatically through Java methods - see Loading a Graph by Defining a Graph Configuration Object for more details.
- Reading Entity Providers at the Same SCN
 If you have a graph which consists of multiple vertex or edge tables or both, then you can read all the vertices and edges at the same System Change Number (SCN).
- Progress Reporting and Estimation for Graph Loading
 Loading a large graph into the graph server(PGX) can be a long running operation.
 However, if you load the graph using an asynchronous action, then you can monitor the progress of the graph loading operation.
- API for Loading Graphs into Memory
 Learn about the APIs used for loading a graph using a JSON configuration file or graph
 configuration object.
- Graph Configuration Options
 Learn about the graph configuration options.
- Data Loading Security Best Practices
 Loading a graph from the database requires authentication and it is therefore important to adhere to certain security guidelines when configuring access to this kind of data source.
- Data Format Support Matrix
 Learn about the different data formats supported in the graph server (PGX).
- Immutability of Loaded Graphs
 Once the graph is loaded into the graph server (PGX), the graph and its properties are automatically marked as immutable.

14.2.1.1 Reading Entity Providers at the Same SCN

If you have a graph which consists of multiple vertex or edge tables or both, then you can read all the vertices and edges at the same System Change Number (SCN).

This helps to overcome issues such as reading edge providers at a later SCN than the SCN at which the vertices were read, as some edges may reference missing vertices.

Note that reading a graph from the database is still possible even if Flashback is not enabled on Oracle Database. In case of multiple databases, SCN can be used to maintain consistency for entity providers belonging to the same database only.

You can use the as_of flag in the graph configuration to specify at what SCN an entity provider must be read. The valid values for the as of flag are as follows:

Table 14-1 Valid values for "as_of" Key in Graph Configuration

Value	Description
A positive long value	This is a parseable SCN value.
" <current-scn>"</current-scn>	The current SCN is determined at the beginning of the graph loading.



Table 14-1 (Cont.) Valid values for "as_of" Key in Graph Configuration

Value	Description
" <no-scn>"</no-scn>	This is to disable SCN at the time of graph loading.
null	This defaults to " <current-scn>" behavior.</current-scn>

If "as_of" is omitted for a vertex or an edge provider in the graph configuration file, then this follows the same behavior as "as_of": null.

Example 14-1 Graph Configuration Using "as_of" for Vertex and Edge Providers in the Same Database

The following example configuration has three vertex providers and one edge provider pointing to the same database.

```
"name": "employee graph",
"vertex providers": [
 {
    "name": "Department",
   "as of": "<current-scn>",
    "format": "rdbms",
    "database table name": "DEPARTMENTS",
    "key column": "DEPARTMENT ID",
    "props": [
      {
        "name": "DEPARTMENT NAME",
        "type": "string"
    ]
 },
   "name": "Location",
    "as of": "28924323",
    "format": "rdbms",
    "database table name": "LOCATIONS",
    "key column": "LOCATION ID",
    "props": [
        "name": "CITY",
        "type": "string"
    ]
 },
   "name": "Region",
   "as of": "<no-scn>",
    "format": "rdbms",
    "database_table_name": "REGIONS",
    "key column": "REGION ID",
    "props": [
      {
```



```
"name": "REGION NAME",
          "type": "string"
      ]
    }
  ],
  "edge providers": [
    {
      "name": "LocatedAt",
      "format": "rdbms",
      "database table name": "DEPARTMENTS",
      "key column": "DEPARTMENT ID",
      "source column": "DEPARTMENT ID",
      "destination column": "LOCATION_ID",
      "source vertex provider": "Department",
      "destination vertex provider": "Location"
  ]
}
```

When reading the <code>employee_graph</code> using the preceding configuration file, the graph is read at the same SCN for the <code>Department</code> and <code>LocatedAt</code> entity providers. This is explained in the following table:

Table 14-2 Example Scenario Using "as_of"

Entity Provider	"as_of"	SCN Value
Department	" <current-scn>"</current-scn>	SCN determined automatically
Location	"28924323"	"28924323" used as SCN
Region	" <no-scn>"</no-scn>	No SCN used
LocatedAt	"as_of" flag is omitted	SCN determined automatically

The current SCN value of the database can be determined using one of the following options:

Querying V\$DATABASE view:

```
SELECT CURRENT SCN FROM V$DATABASE;
```

Using DBMS FLASHBACK package:

```
SELECT DBMS FLASHBACK.GET SYSTEM CHANGE NUMBER FROM DUAL;
```

If you do not have the required privileges to perform either of the preceding operations, then you can use:

```
SELECT TIMESTAMP TO SCN(SYSDATE) FROM DUAL;
```

However, note that this option is less precise than the earlier two options.

You can then read the graph into the graph server using the JSON configuration file as shown:



- JShell
- Java
- Python

JShell

opg4j> var g = session.readGraphWithProperties("employee graph.json")

Java

PgxGraph g = session.readGraphWithProperties("employee graph.json");

Python

g = session.read graph with properties("employee graph.json")

14.2.1.2 Progress Reporting and Estimation for Graph Loading

Loading a large graph into the graph server(PGX) can be a long running operation. However, if you load the graph using an asynchronous action, then you can monitor the progress of the graph loading operation.

The following table shows the asynchronous graph loading APIs supported for the following formats:

Table 14-3 Asynchronous Graph Loading APIs

Data Format	API
PG VIEWS	session.readGraphByNameAsync()
PG SCHEMA	session.readGraphWithPropertiesAsync()
CSV	session.readGraphFileAsync()

These supported APIs return a PgxFuture object.

You can then use the PgxFuture.getProgress() method to collect the following statistics:

- Report on the progress of the graph loading operation
- Estimate of the remaining vertices and edges that need to be loaded into memory

For example, the following code shows the steps to load a PG view graph asynchronously and subsequently obtain the FutureProgress object to report and estimate the loading progress. However, note that the graph loading estimate (for example, the number of loaded entities and providers or the number of total entities and providers) can be obtained only until the graph loading operation is in progress.



Also, the system internally computes the graph loading progress for every 10000 entries of entities that are loaded into the graph server (PGX).

- JShell
- Java

JShell

```
opg4j> var graphLoadingFuture =
session.readGraphByNameAsync("BANK_GRAPH_VIEW", GraphSource.PG_VIEW)
readGraphFuture ==> oracle.pgx.api.PgxFuture@6106dfb6[Not completed]

opg4j> while (!graphLoadingFuture.isDone()) {
    ...> var progress = graphLoadingFuture.getProgress();
    ...> if (graphLoadingProgress = progress.asGraphLoadingProgress();
    ...> if (graphLoadingProgress.isPresent()) {
    ...> var numLoadedVertices =
    graphLoadingProgress.get().getNumLoadedVertices();
    ...> }
    ...> Thread.sleep(1000);
    ...> }

opg4j> var graph = graphLoadingFuture.get();
graph ==> PgxGraph[name=BANK_GRAPH_VIEW_3,N=999,E=4993,created=1664289985985]
```

Java

```
PgxFuture<PgxGraph> graphLoadingFuture =
session.readGraphByNameAsync("BANK_GRAPH_VIEW", GraphSource.PG_VIEW);
while (!graphLoadingFuture.isDone()) {
  FutureProgress progress = graphLoadingFuture.getProgress();
  Optional < GraphLoadingProgress > graphLoadingProgress =
progress.asGraphLoadingProgress();
  if (graphLoadingProgress.isPresent()) {
    long numLoadedVertices =
graphLoadingProgress.get().getNumLoadedVertices();
  }
  Thread.sleep(1000);
}
PgxGraph graph = graphLoadingFuture.get();
```

It is recommended that you do not use the FutureProgress object in a chain of asynchronous operations.

14.2.1.3 API for Loading Graphs into Memory

Learn about the APIs used for loading a graph using a JSON configuration file or graph configuration object.

The following methods in PgxSession can be used to load graphs into the graph server (PGX).

- Java
- Python

Java

```
PgxGraph readGraphWithProperties(String path)
PgxGraph readGraphWithProperties(String path, String newGraphName)
PgxGraph readGraphWithProperties(GraphConfig config)
PgxGraph readGraphWithProperties(GraphConfig config, String newGraphName)
PgxGraph readGraphWithProperties(GraphConfig config, boolean forceUpdateIfNotFresh)
PgxGraph readGraphWithProperties(GraphConfig config, boolean forceUpdateIfNotFresh, String newGraphName)
PgxGraph readGraphWithProperties(GraphConfig config, long maxAge, TimeUnit maxAgeTimeUnit)
PgxGraph readGraphWithProperties(GraphConfig config, long maxAge, TimeUnit maxAgeTimeUnit, boolean blockIfFull, String newGraphName)
```

Python

The first argument (path to a graph configuration file or a parsed config object) is the meta-data of the graph to be read. The meta-data includes the following information:

- Location of the graph data such as file location and name, DB location and connection information, and so on
- Format of the graph data such as plain text formats, XML-based formats, binary formats, and so on
- Types and Names of the properties to be loaded

The <code>forceUpdateIfNotFresh</code> and <code>maxAge</code> arguments can be used to fine-control the age of the snapshot to be read. The graph server (PGX) will return an existing graph snapshot if the given graph specification was already loaded into memory by a



different session. So, the maxAge argument becomes important if reading from a database in which the data might change frequently. If no forceUpdateIfNotFresh or maxAge is specified, PGX will favor cached data over reading new snapshots into memory.

14.2.1.4 Graph Configuration Options

Learn about the graph configuration options.

The following table lists the JSON fields that are common to all graph configurations:

Table 14-4 Graph Config JSON Fields

Field	Туре	Description	Def aul t
name	string	Name of the graph.	Re quir ed
array_compaction_th reshold	number	[only relevant if the graph is optimized for updates] Threshold used to determined when to compact the delta-logs into a new array. If lower than the engine min_array_compaction_threshold value, min_array_compaction_threshold will be used instead	0.2
attributes	object	Additional attributes needed to read and write the graph data.	nul 1
data_source_id	string	Data source id to use to connect to an RDBMS instance.	nul 1
edge_id_strategy	<pre>enum[no_ids, keys_as_ids, unstable_gen erated_ids]</pre>	Indicates what ID strategy should be used for the edges of this graph. If not specified (or set to null), the strategy will be determined during loading or using a default value.	nul 1
edge_id_type	enum[long]	Type of the edge ID. Setting it to long requires the IDs in the edge providers to be unique across the graphs; those IDs will be used as global IDs. Setting it to null (or omitting it) will allow repeated IDs across different edge providers and PGX will automatically generate globally-unique IDs for the edges.	nul 1
edge_providers	array of object	List of edge providers in this graph.	[]
error_handling	object	Error handling configuration.	nul l
external_stores	array of object	Specification of the external stores where external string properties reside.	[]
jdbc_url	string	JDBC URL pointing to an RDBMS instance	nu 11
keystore_alias	string	Alias to the keystore to use when connecting to database.	nul 1



Table 14-4 (Cont.) Graph Config JSON Fields

Field	Туре	Description	Def aul t
loading	object	Loading-specific configuration to use.	nul 1
local_date_format	array of string	array of local_date formats to use when loading and storing local_date properties. See DateTimeFormatter for more details of the format string	[]
max_prefetched_rows	integer	Maximum number of rows prefetched during each round trip resultset-database.	10 00 0
num_connections	integer	Number of connections to read and write data from or to the RDBMS table.	<n -="" cp="" o-="" of="" us=""></n>
optimized_for	<pre>enum[read, updates]</pre>	Indicates if the graph should use data- structures optimized for read-intensive scenarios or for fast updates.	rea d
password	string	Password to use when connecting to database.	nul 1
point2d	string	Longitude and latitude as floating point values separated by a space.	0.0
prepared_queries	array of object	An additional list of prepared queries with arguments, working in the same way as 'queries'. Data matching at least one those queries will also be loaded.	[]
queries	array of string	A list of queries used to determine which data to load from the database. Data matching at least one of the queries will be loaded. Not setting any query will load the entire graph.	
redaction_rules	array of object	Array of redaction rules.	[]
rules_mapping	array of object	Mapping for redaction rules to users and roles.	[]
schema	string	Schema to use when reading or writing RDBMS objects	nu 11
source_name	string	Name of the database graph, if the graph is loaded from a database.	nul 1
source_type	enum[pg_view]	Source type for database graphs.	nul 1
time_format	array of string	The time format to use when loading and storing time properties. See DateTimeFormatter for a documentation of the format string.	[]



Table 14-4 (Cont.) Graph Config JSON Fields

Field	Туре	Description	Def aul t
time_with_timezone_ format	array of string	The time with timezone format to use when loading and storing time with timezone properties. Please see DateTimeFormatter for more information of the format string.	[]
timestamp_format	array of string	The timestamp format to use when loading and storing timestamp properties. See DateTimeFormatter for more information of the format string.	[]
timestamp_with_time zone_format	array of string	The timestamp with timezone format to use when loading and storing timestamp with timezone properties. See DateTimeFormatter for more information of the format string.	[]
username	string	Username to use when connecting to an RDBMS instance.	nu 11
vector_component_de limiter	character	Delimiter for the different components of vector properties.	;
vertex_id_strategy	<pre>enum[no_ids, keys_as_ids, unstable_gen erated_ids]</pre>	Indicates what ID strategy should be used for the vertices of this graph. If not specified (or set to null), the strategy will be automatically detected.	nul 1
vertex_id_type	<pre>enum[int, integer, long, string]</pre>	Type of the vertex ID. For homogeneous graphs, if not specified (or set to null), it will default to a specific value (depending on the origin of the data).	nul 1
vertex_providers	array of object	List of vertex providers in this graph.	[]

Note:

Database connection fields specified in the graph configuration will be used as default in case underlying data provider configuration does not specify them.

Provider Configuration JSON file Options

You can specify the meta-information about each provider's data using provider configurations. Provider configurations include the following information about the provider data:

- Location of the data: a file, multiple files or database providers
- Information about the properties: name and type of the property



Table 14-5 Provider Configuration JSON file Options

Field	Туре	Description	Default
format	enum[pgb, csv, rdbms]	Provider format.	Require d
name	string	Entity provider name.	Require d
attributes	object	Additional attributes needed to read and write the graph data.	null
<pre>destination_ver tex_provider</pre>	string	Name of the destination vertex provider to be used for this edge provider.	null
error_handling	object	Error handling configuration.	null
has_keys	boolean	Indicates if the provided entities data have keys.	true
key_type	<pre>enum[int, integer, long, string]</pre>	Type of the keys.	long
keystore_alias	string	Alias to the keystore to use when connecting to database.	null
label	string	label for the entities loaded from this provider.	null
loading	object	Loading-specific configuration.	null
<pre>local_date_form at</pre>	array of string	Array of local_date formats to use when loading and storing local_date properties. See DateTimeFormatter for a documentation of the format string.	[]
password	string	Password to use when connecting to database.	null
point2d	string	Longitude and latitude as floating point values separated by a space.	0.0
props	array of object	Specification of the properties associated with this entity provider.	[]
<pre>source_vertex_p rovider</pre>	string	Name of the source vertex provider to be used for this edge provider.	null
time_format	array of string	The time format to use when loading and storing time properties. See DateTimeFormatter for a documentation of the format string.	[]
time_with_timez one_format	array of string	The time with timezone format to use when loading and storing time with timezone properties. See DateTimeFormatter for a documentation of the format string.	[]
timestamp_forma t	array of string	The timestamp format to use when loading and storing timestamp properties. See DateTimeFormatter for a documentation of the format string.	[]



Table 14-5 (Cont.) Provider Configuration JSON file Options

Field	Туре	Description	Default
timestamp_with_ timezone_format		The timestamp with timezone format to use when loading and storing timestamp with timezone properties. See DateTimeFormatter for a documentation of the format string.	[]
<pre>vector_componen t_delimiter</pre>	character	Delimiter for the different components of vector properties.	;

Provider Labels

The label field in the provider configuration can be used to set a label for the entities loaded from the provider. If no label is specified, all entities from the provider are labeled with the name of the provider. It is only possible to set the same label for two different providers if they have exactly the same properties (same names and same types).

Property Configuration

The props entry in the Provider configuration is an object with the following JSON fields:

Table 14-6 Property Configuration

Field	Туре	Description	Default
name	string	Name of the property.	Require d
type	<pre>enum[boolean , integer, vertex, edge, float, long, double, string, date, local_date, time, timestamp, time_with_ti mezone, timestamp_wi th_timezone, point2d]</pre>	Type of the property . Note: date is deprecated, use one of local_date /time/ timestamp/ time_with_ timezone/ timestamp_ with_timez one instead).	Require

vertex/edge are place-holders for the
type specified in vertex_id_type/
edge_id_type fields.



Table 14-6 (Cont.) Property Configuration

Field	Туре	Description	Default
aggregate	<pre>enum[identit y, group_key, min, max, avg, sum, concat, count]</pre>	[currently unsupported] which aggregation function to use, aggregation always happens by vertex key.	null
column	value	Name or index (starting from 0) of the column holding the property data. If it is not specified, the loader will try to use the property name as column name (for CSV format only).	null
default	value	Default value to be assigned to this property if datasource does not provide it. In case of date type: string is expected to be formatted with yyyy-MM-dd HH:mm:ss. If no default is present (null), non-existent properties will contain default Java types (primitives) or empty string (string) or 01.01.1970 00:00 (date).	null
dimension	integer	Dimension of property.	0
<pre>drop_after_loadi ng</pre>	boolean	[currently unsupported] indicating helper properties only used for aggregation, which are dropped after loading	false
field	value	Name of the JSON field holding the property data. Nesting is denoted by dot - separation. Field names containing dots are possible, in this case the dots need to be escaped using backslashes to resolve ambiguities. Only the exactly specified object are loaded, if they are non existent, the default value is used.	null
format	array of string	Array of formats of property.	[]
group_key	string	[currently unsupported] can only be used if the property / key is part of the grouping expression.	null
<pre>max_distinct_str ings_per_pool</pre>	integer	[only relevant if string_pooling_strategy is indexed] Amount of distinct strings per property after which to stop pooling. If the limit is reached an exception is thrown. If set to null, the default value from the global PGX configuration will be used.	null
stores	array of object	A list of storage identifiers that indicate where this property resides.	[]
string_pooling_s trategy	<pre>enum[indexed , on_heap, none]</pre>	Indicates which string pooling strategy to use. If set to null, the default value from the global PGX configuration will be used.	null



Loading Configuration

The loading entry is a JSON object with the following fields:

Table 14-7 Loading Configuration

Field	Туре	Description	Default
<pre>create_key_mappi ng</pre>	boolean	If true, a mapping between entity keys and internal IDs is prepared during loading.	true
filter	string	[currently unsupported] the filter expression	null
grouping_by	array of string	[currently unsupported] array of edge properties used for aggregator. For Vertices, only the ID can be used (default)	[]
load_labels	boolean	Whether or not to load the entity label if it is available.	false
strict_mode	boolean	If true, exceptions are thrown and logged with ERROR level whenever loader encounters problems with input file, such as invalid format, repeated keys, missing fields, mismatches and other potential errors. If false, loader may use less memory during loading phase, but behave unexpectedly with erratic input files.	true

Error Handling Configuration

The error_handling entry is a JSON object with the following fields:

Table 14-8 Error Handling Configuration

Field	Туре	Description	Default
on_missed_prop_k ey	<pre>enum[silent, log_warn, log_warn_once, error]</pre>	Error handling for a missing property key.	log_warn_ once
<pre>on_missing_verte x</pre>	<pre>enum[ignore_edge , ignore_edge_log, ignore_edge_log_ once, create_vertex, create_vertex_lo g, create_vertex_lo g_once, error]</pre>	Error handling for a missing source or destination vertex of an edge in a vertex data source.	error
on_parsing_issue	<pre>enum[silent, log_warn, log_warn_once, error]</pre>	Error handling for incorrect data parsing. If set to silent, log_warn or log_warn_once, will attempt to continue loading. Some parsing issues may not be recoverable and provoke the end of loading.	error



Table 14-8 (Cont.) Error Handling Configuration

Field	Туре	Description	Default
on_prop_conversi	<pre>enum[silent, log_warn, log_warn_once, error]</pre>	Error handling when encountering a different property type other than the one specified, but coercion is possible.	log_warn_ once
on_type_mismatch	<pre>enum[silent, log_warn, log_warn_once, error]</pre>	Error handling when encountering a different property type other than the one specified, but coercion is <i>not</i> possible.	error
<pre>on_vector_length _mismatch</pre>	<pre>enum[silent, log_warn, log_warn_once, error]</pre>	Error handling for a vector property that does not have the correct dimension.	error



The only supported setting for the <code>on_missing_vertex</code> error handling configuration is <code>ignore_edge</code>.

14.2.1.5 Data Loading Security Best Practices

Loading a graph from the database requires authentication and it is therefore important to adhere to certain security guidelines when configuring access to this kind of data source.

The following guidelines are recommended:

- The user or role used to access the data should be a read-only account that only has access to the required graph data.
- The graph data should be marked as read-only, for example, with non-updateable views in the case of the database.

14.2.1.6 Data Format Support Matrix

Learn about the different data formats supported in the graph server (PGX).

The following table illustrates how the different data formats differ in the way IDs, labels and vector properties are handled.



The table refers to limitations of the PGX implementation of the format and not necessarily to limitations of the format itself.



Format	Vertex IDs	Edge IDs	Vertex Labels	Edge Labels	Vector properties
PGB	int, long, string	long	multiple	single	<pre>supported (vectors can be of type integer, long, float or double)</pre>
CSV	int, long, string	long	multiple	single	supported (vectors can be of type integer, long, float or double)
ADJ_LIST	int, long, string	Not supported	Not supported	Not supported	supported (vectors can be of type integer, long, float or double)
EDGE_LIST	int, long, string	Not supported	multiple	single	<pre>supported (vectors can be of type integer, long, float or double)</pre>
GRAPHML	int, long, string	Not supported	Not supported	Not supported	Not supported

Table 14-9 Data Format Support Matrix

14.2.1.7 Immutability of Loaded Graphs

Once the graph is loaded into the graph server (PGX), the graph and its properties are automatically marked as immutable.

The immutability of loaded graphs is due to the following design choices:

- Typical graph analyses happen on a snapshot of a graph instance, and therefore they do not require mutations of the graph instance.
- Immutability allows PGX to use an internal graph representation optimized for fast analysis.
- In remote mode, the graph instance might be shared among multiple clients.

However, the graph server (PGX) also provides methods to customize and mutate graph instances for the purpose of analysis. See Graph Mutation and Subgraphs for more information.

14.2.2 Storing a Graph Snapshot on Disk

After reading a graph into memory, you can make any changes to the graph (such as running the PageRank algorithm and storing the values as vertex properties), and then store this snapshot of the graph on disk.

If you want to save the state of the graph in memory, then a snapshot of a graph can be saved as a file in binary format (PGB file).

In general, if you must shut down the graph server, then it is recommended that you store all the graph queries and analytics APIs that have been run on the graph. Once the graph server (PGX) is restarted, you can reload the graph and rerun the APIs.

However, if you must save the state of the graph, then the following example explains how to store the graph snapshot on disk.



As a prerequisite for storing the graph snapshot, you need to explicitly authorize access to the corresponding directories by defining a directory object pointing to the directory (on the graph server) that contains the files to read or write.

```
CREATE OR REPLACE DIRECTORY pgx_file_location AS '<path_to_dir>';
GRANT READ, WRITE ON directory pgx_file_location to GRAPH DEVELOPER;
```

Also, note the following:

- The directory in the CREATE DIRECTORY statement must exist on the graph server (PGX).
- The directory must be readable (and/or writable) at the OS level by the graph server (PGX).

The preceding code grants the privileges on the directory to the <code>GRAPH_DEVELOPER</code> role. However, you can also grant permissions to an individual user:

```
GRANT READ, WRITE ON DIRECTORY pgx file location TO <graph user>;
```

You can then run the following code to load a property graph view into the graph server (PGX) and save the graph snapshot as a file. Note that multiple PGB files will be generated, one for each vertex and edge provider in the graph.

```
opg4j> var g = session.readGraphByName("BANK_GRAPH",
GraphSource.PG_VIEW)
g ==> PgxGraph[name=BANK_GRAPH_NEW,N=999,E=4993,created=1676021791568]
opg4j> analyst.pagerank(graph)
$8 ==> VertexProperty[name=pagerank,type=double,graph=BANK_GRAPH]
// Now save the state of this graph
opg4j> var storedPgbConfig = graph.store(ProviderFormat.PGB,
"<path to dir>")
```

In a three-tier deployment, the file is written on the server-side file system. You must also ensure that the file location to write is specified in the graph server (PGX). (As explained in Three-Tier Deployments of Oracle Graph with Autonomous Database, in a three-tier deployment, access to the PGX server file system requires a list of allowed locations to be specified.)

14.2.3 Publishing a Graph

You can publish a graph that can be referenced by other sessions.

Publishing a Single Graph Snapshot

The PgxGraph#publish() method can be used to publish the current selected snapshot of the graph. The publish operation will move the graph name from the session-private namespace to the public namespace. If a graph with the same name has been already published, then the publish() method will fail with an exception. Graphs published with snapshots and single published snapshots share the same namespace.



Table 12-6 describes the grants required to publish a graph.

Note that calling the <code>publish()</code> method without arguments publishes the snapshot with its persistent properties only. However, if you want to publish specific transient properties, then you must list them within the <code>publish()</code> call as shown:

- JShell
- Java
- Python

JShell

```
opg4j> var prop1 = graph.createVertexProperty(PropertyType.INTEGER, "prop1")
opg4j> prop1.fill(0)
opg4j> var cost = graph.createEdgeProperty(PropertyType.DOUBLE, "cost")
opg4j> cost.fill(0d)
opg4j> graph.publish(List.of(prop1), List.of(cost))
```

Java

```
VertexProperty<Integer, Integer> prop1 =
graph.createVertexProperty(PropertyType.INTEGER, "prop1");
prop1.fill(0);
EdgeProperty<Double> cost = graph.createEdgeProperty(PropertyType.DOUBLE,
"cost");
cost.fill(0d);
List<VertexProperty<Integer, Integer> vertexProps = Arrays.asList(prop);
List<EdgeProperty<Double>> edgeProps = Arrays.asList(cost);
graph.publish(vertexProps, edgeProps);
```

Python

```
prop = graph.create_vertex_property("integer", "prop1")
prop1.fill(0)
cost = graph.create_edge_property("double", "cost")
cost.fill(0d)
vertex_props = [prop]
edge_props = [cost]
graph.publish(vertex props, edge props)
```

Publishing a Graph with Snapshots

If you want to make all snapshots of the graph visible to other sessions, then use the publishWithSnapshots() method. When a graph is published with snapshots, the GraphMetaData information of each snapshot is also made available to the other sessions, with the exception of the graph configuration, which is null.



When calling the <code>publishWithSnapshots()</code> method, all the persistent properties of all the snapshots are published and made visible to the other sessions. Transient properties are session-private and therefore they must be published explicitly. Once published, all properties become read-only.

Similar to publishing a single graph snapshot, the publishWithSnapshots() method will move the graph name from the session-private namespace to the public namespace. If a graph with the same name has been already published, then the publishWithSnapshots() method will fail with an exception.

Also, note that the published properties, like the original transient properties, are associated to the specific snapshot on which they were created. Therefore, they are not visible on other snapshots.

If you want to publish specific transient properties, you should list them within the publishWithSnapshots() call, as in the following example:

- JShell
- Java
- Python

JShell

```
opg4j> var prop1 = graph.createVertexProperty(PropertyType.INTEGER,
"prop1")
opg4j> prop1.fill(0)
opg4j> var cost = graph.createEdgeProperty(PropertyType.DOUBLE, "cost")
opg4j> cost.fill(0d)
opg4j> graph.publishWithSnapshots(List.of(prop1), List.of(cost))
```

Java

```
VertexProperty<Integer, Integer> prop1 =
graph.createVertexProperty(PropertyType.INTEGER, "prop1");
prop1.fill(0);
EdgeProperty<Double> cost =
graph.createEdgeProperty(PropertyType.DOUBLE, "cost");
cost.fill(0d);
List<VertexProperty<Integer, Integer> vertexProps =
Arrays.asList(prop);
List<EdgeProperty<Double>> edgeProps = Arrays.asList(cost);
graph.publishWithSnapshots(vertexProps,edgeProps);
```

Python

```
VertexProperty<Integer, Integer> prop1 =
graph.createVertexProperty(PropertyType.INTEGER, "prop1")
prop1.fill(0)
EdgeProperty<Double> cost =
graph.createEdgeProperty(PropertyType.DOUBLE, "cost")
```



```
cost.fill(0d)
List<VertexProperty<Integer, Integer> vertexProps = Arrays.asList(prop)
List<EdgeProperty<Double>> edgeProps = Arrays.asList(cost)
graph.publishWithSnapshots(vertexProps,edgeProps)
```

Referencing a Published Graph from Another Session

You can reference a published graph by its name in another session, using the PgxSession#getGraph() method.

The following example references a published graph myGraph in a new session (session2):

- JShell
- Java
- Python

JShell

```
opg4j> var session2 = instance.createSession("session2")
opg4j> var graph2 = session2.getGraph(Namespace.PUBLIC, "myGraph")
```

Java

```
PgxSession session2 = instance.createSession("session2");
PgxGraph graph2 = session2.getGraph(Namespace.PUBLIC, "myGraph");
```

Python

```
session2 = pypgx.get_session("session2");
PgxGraph graph2 = session2.get graph("myGraph")
```

session2 can access only the published snapshot. If the graph has been published without snapshots, calling the getAvailableSnapshots() method will return an empty queue.

In case if the graph snapshots have been published, then the call to getGraph() returns the most recent available snapshot. session2 can see all the available snapshots through the getAvailableSnapshots() method. You can then set a specific snapshot using the PgxSession#setSnapshot() method.





If a referenced graph is not required anymore, then it is important that you release the graph. See Deleting a Graph for more information.

Publishing a Property

After publishing (a single snapshot or all of them), you can still publish transient properties individually. Published properties are associated to a specific snapshot on which they are created, and hence visible only on that snapshot.

- JShell
- Java
- Python

JShell

```
opg4j> graph.getVertexProperty("prop1").publish()
opg4j> graph.getEdgeProperty("cost").publish()
```

Java

```
graph.getVertexProperty("prop1").publish();
graph.getEdgeProperty("cost").publish();
```

Python

```
graph.get_vertex_property("prop1").publish()
graph.get edge property("cost").publish()
```

Getting a Published Property in Another Session

Sessions referencing a published graph (with or without snapshots) can reference a published property through the PgxGraph#getVertexProperty and PgxGraph#getEdgeProperty.

- JShell
- Java
- Python



JShell

```
opg4j> var session2 = instance.createSession("session2")
opg4j> var graph2 = session2.getGraph(Namespace.PUBLIC, "myGraph")
opg4j> var vertexProperty = graph2.getVertexProperty("prop1")
opg4j> var edgeProperty = graph2.getEdgeProperty("cost")
```

Java

```
PgxSession session2 = instance.createSession("session2");
PgxGraph graph2 = session2.getGraph(Namespace.PUBLIC, "myGraph");
VertexProperty<Integer, Integer> vertexProperty =
graph2.getVertexProperty("prop1");
EdgeProperty<Double> edgeProperty = graph2.getEdgeProperty("cost");
```

Python

```
session2 = pypgx.get_session(session_name ="session2")
graph2 = session2.get_graph("myGraph")
vertex_property = graph2.get_vertex_property("prop1")
edge property = graph2.get edge property("cost")
```

Pinning a Published Graph

You can pin a published graph so that it remains published even if no session uses it.

- JShell
- Java
- Python

JShell

```
opg4j> graph.pin()
```

Java

```
graph.pin();
```

Python

```
>>> graph.pin()
```



Unpinning a Published Graph

You can unpin a published graph that was earlier pinned. By doing this, you can remove the graph and all its snapshots, if no other session is using a snapshot of the graph.

- JShell
- Java
- Python

JShell

```
opg4j> var graph = session.getGraph("bank_graph_analytics")
graph ==>
PgxGraph[name=bank_graph_analytics, N=999, E=4993, created=1660217577201]
opg4j> graph.unpin()
```

Java

```
PgxGraph graph = session.getGraph("bank_graph_analytics");
graph.unpin();
```

Python

```
>>> graph = session.get_graph("bank_graph_analytics")
>>> graph.unpin()
```

Related Topics

Namespaces and Sharing

The graph server (PGX) supports separate namespaces that help you to organize vour entities.

14.2.4 Deleting a Graph

In order to reduce the memory usage of the graph server (PGX), the session must drop the unused graph objects created through the getGraph() method, by invoking the destroy() method.

Calling the $\mathtt{destroy}()$ method not only destroys the specified graph, but all of its associated properties, including transient properties as well. In addition, all of the collections related to the graph instance (for example, a $\mathtt{VertexSet}$) are also destroyed automatically. If a session holds multiple $\mathtt{PgxGraph}$ objects referencing the same graph, invoking $\mathtt{destroy}()$ on any of them will invalidate all the $\mathtt{PgxGraph}$ objects referencing that graph, making any operation on those objects fail.



For example:

- JShell
- Java
- Python

JShell

Java

```
PgxGraph graph1 = session.getGraph("myGraphName");

// graph2 references the same graph of graph1
PgxGraph graph2 = session.getGraph("myGraphName");

// Delete graph2
graph2.destroy();

// Both the following calls throw an exception, as both references are not valid anymore
Set<VertexProperty<?, ?>> properties = graph1.getVertexProperties();
properties = graph2.getVertexProperties();
```

Python

```
graph1 = session.get_graph("myGraphName")

# graph2 references the same graph of graph1
    graph2 = session.get_graph("myGraphName")

# Delete graph2
graph2.destroy()

# Both the following calls throw an exception, as both references are not valid anymore
properties = graph1.get_vertex_properties()
properties = graph2.get vertex properties()
```



The same behavior occurs when multiple <code>PgxGraph</code> objects reference the same snapshot. Since a snapshot is effectively a graph, destroying a <code>PgxGraph</code> object referencing a certain snapshot invalidates all <code>PgxGraph</code> objects referencing the same snapshot, but does not invalidate those referencing other snapshots:

```
// Get a snapshot of "myGraphName"
PgxGraph graph1 = session.getGraph("myGraphName");

// graph2 and graph3 reference the same snapshot as graph1
PgxGraph graph2 = session.getGraph("myGraphName");
PgxGraph graph3 = session.getGraph("myGraphName");

// Assume another snapshot is created ...

// Make graph3 references the latest snapshot available session.setSnapshot(graph3, PgxSession.LATEST_SNAPSHOT);
graph2.destroy();

// Both the following calls throw an exception, as both references are not valid anymore Set<VertexProperty<?, ?>> properties = graph1.getVertexProperties();

properties = graph2.getVertexProperties();

// graph3 is still valid, so the call succeeds properties = graph3.getVertexProperties();
```

Note:

Even if a graph is destroyed by a session, the graph data may still remain in the server memory, if the graph is currently shared by other sessions. In such a case, the graph may still be visible among the available graphs through the PgxSession.getGraphs() method.

As a safe alternative to the manual removal of each graph, the PGX API supports some implicit resource management features which allow developers to safely omit the destroy() call. See Resource Management Considerations for more information.

14.3 Keeping the Graph in Oracle Database Synchronized with the Graph Server

You can use the FlashbackSynchronizer API to automatically apply changes made to graph in the database to the corresponding PgxGraph object in memory, thus keeping both synchronized.

This API uses Oracle's Flashback Technology to fetch the changes in the database since the last fetch and then push those changes into the graph server using the ChangeSet API. After the changes are applied, the usual snapshot semantics of the graph server apply: each delta fetch application creates a new in-memory snapshot. Any queries or algorithms that are executing concurrently to snapshot creation are unaffected by the changes until the corresponding session refreshes its PgxGraph

object to the latest state by calling the session.setSnapshot(graph,
PgxSession.LATEST SNAPSHOT) procedure.

Also, if the changes from the previous fetch operation no longer exist, then the synchronizer will throw an exception. This occurs if the previous fetch duration is longer than the UNDO_RETENTION parameter setting in the database. To avoid this exception, ensure to fetch the changes at intervals less than the UNDO_RETENTION parameter value. The default setting for the UNDO_RETENTION parameter is 900 seconds. See *Oracle Database Reference* for more information.

Prerequisites for Synchronizing

The Oracle database must have Flashback enabled and the database user that you use to perform synchronization must have:

- Read access to all tables which need to be kept synchronized.
- Permission to use flashback APIs. For example:

```
GRANT EXECUTE ON DBMS FLASHBACK TO <user>
```

The database must also be configured to retain changes for the amount of time needed by your use case.

Types of graphs that can be synchronized

Not all PgxGraph objects in PGX can be synchronized. The following limitations apply:

- Only the original creator of the graph can synchronize it. That is, the current user must have the MANAGE permission of the graph.
- Only graphs loaded from database tables (PG View graphs) can be synchronized.
 Graphs created from other formats or graphs created via the graph builder API or PG View graphs created from database views cannot be synchronized.
- Only the latest snapshot of a graph can be synchronized.

Types of changes that can be synchronized

The synchronizer supports keeping the in-memory graph snapshot in sync with the following database-side modifications:

- insertion of new vertices and edges
- removal of existing vertices and edges
- update of property values of any vertex or edge

The synchronizer does not support schema-level changes to the input graph, such as:

- alteration of the list of input vertex or edge tables
- alteration of any columns of any input tables (vertex or edge tables)

Furthermore, the synchronizer does not support updates to vertex and edge keys.

For a detailed example, see the following topic:

Synchronizing a PG View Graph
You can synchronize a graph loaded into the graph server (PGX) from a property graph
view (PG View) with the changes made to the graph in the database.



Synchronizing a Published Graph

You can synchronize a published graph by configuring the Flashback Synchronizer with a PartitionedGraphConfig object containing the graph schema along with the database connection details.

14.3.1 Synchronizing a PG View Graph

You can synchronize a graph loaded into the graph server (PGX) from a property graph view (PG View) with the changes made to the graph in the database.

The following example shows the steps for synchronizing a PG View using the FlashbackSynchronizer API:

- 1. Load the PG View graph into the graph server (PGX) using the readGraphByName () API as shown:
 - JShell
 - Java
 - Python

JShell

```
opg4j> var graph =
session.readGraphByName("BANK_GRAPH_VIEW", GraphSource.PG_VIEW,

ReadGraphOption.optimizeFor(GraphOptimizedFor.UPDATES), ReadGraphOpti
on.synchronizable())
graph ==>
PgxGraph[name=BANK_GRAPH_VIEW, N=999, E=4993, created=1660275936010]
```

Java

```
PgxGraph graph =
session.readGraphByName("BANK_GRAPH_VIEW", GraphSource.PG_VIEW,
ReadGraphOption.optimizeFor(GraphOptimizedFor.UPDATES), ReadGraphOpti
on.synchronizable());
```

Python

```
>>> graph = session.read graph by name('BANK GRAPH VIEW', 'pg view')
```

2. Open a new JDBC connection to the database and change the data in the underlying database tables for the PG View graph. For example, the following code updates the database value for one of the edge properties:



- JShell
- Java
- Python

JShell

```
opg4j> var conn =
DriverManager.getConnection(<jdbcUrl>, <username>, <password>)
conn ==> oracle.jdbc.driver.T4CConnection@60f7261f
opg4j> var stmt = conn.createStatement()
stmt ==> oracle.jdbc.driver.OracleStatementWrapper@1a914a00
opg4j> stmt.executeQuery("UPDATE bank_txns SET amount=4000 WHERE
txn_id=3")
$5 ==> oracle.jdbc.driver.ForwardOnlyResultSet@627d5f99
opg4j> conn.setAutoCommit(false)
opg4j> conn.commit()
```

Java

```
Connection conn =
DriverManager.getConnection(<jdbcUrl>, <username>, <password>);
Statement stmt = conn.createStatement();
stmt.executeQuery("UPDATE bank_txns SET amount=4000 WHERE txn_id=3");
conn.setAutoCommit(false);
conn.commit();
```

Python

Committing the changes to the database causes the graph in the memory to go out of sync with the database source tables.

3. Synchronize the in-memory graph with the database by creating a new synchronizer object as shown in the following code:

```
Synchronizer synchronizer = new
Synchronizer.Builder<FlashbackSynchronizer>()
    .setType(FlashbackSynchronizer.class)
    .setGraph(graph)
    .setConnection(conn)
    .build();
```



Internally, the graph server keeps track of the Oracle system change number (SCN) the current graph snapshot belongs to. The synchronizer is a client-side component which connects to the database, detects changes by comparing state of the original input tables using the current SCN via the flashback mechanism and then sends any changes to the graph server using the changeset API. In order to do so, the synchronizer needs to know how to connect to the database (conn parameter) as well as which graph to keep in sync (graph parameter).

Alternatively, you can use this equivalent shortcut as shown:

- JShell
- Java
- Python

JShell

```
opg4j> var synchronizer =
graph.createSynchronizer(FlashbackSynchronizer.class, conn)
synchronizer ==> oracle.pgx.api.FlashbackSynchronizer@4ac2b4c6
```

Java

```
Synchronizer synchronizer =
graph.createSynchronizer(FlashbackSynchronizer.class, conn);
```

Python

```
>>> synchronizer =
graph.create_synchronizer(synchronizer_class='oracle.pgx.api.Flashba
ckSynchronizer', connection=conn)
```

- **4.** Fetch and apply the database changes by calling the sync() function and create a new in-memory graph snapshot:
 - JShell
 - Java
 - Python



JShell

```
opg4j> graph=synchronizer.sync()
g ==> PgxGraph[name=BANK_GRAPH_VIEW, N=999, E=4993, created=1660308128037]
```

Java

```
graph=synchronizer.sync();
```

Python

```
>>> graph = synchronizer.sync()
```

Splitting the Fetching and Applying of Changes

The synchronizer.sync() invocation in the preceding code, fetches the changes and applies them in one call. However, you can encode a more complex update logic by splitting this process into separate fetch() and apply() invocations. For example:

```
synchronizer.fetch(); // fetches changes from the database
if (synchronizer.getGraphDelta().getTotalNumberOfChanges() > 100) { //
only create snapshot if there have been more than 100 changes
   synchronizer.apply();
}
```

- **5.** Query the graph to verify the updates to the edge property.
 - JShell
 - Java
 - Python

JShell

```
opg4j> graph.queryPgql("SELECT e.amount FROM MATCH (v1:Accounts)-
[e:Transfers]->(v2:Accounts) WHERE e.from_acct_id = 179 AND
e.to_acct_id=688").print()
```

Java

```
graph.queryPgql("SELECT e.amount FROM MATCH (v1:Accounts)-[e:Transfers]-
>(v2:Accounts) WHERE e.from acct id = 179 AND e.to acct id=688").print();
```



Python

```
>>> graph.query_pgql("SELECT e.amount FROM MATCH (v1:Accounts) -
[e:Transfers] -> (v2:Accounts) WHERE e.from_acct_id = 179 AND
e.to_acct_id=688").print()
```

On execution, the preceding example produces the following output:

```
+----+
| amount |
+----+
| 4000.0 |
```

14.3.2 Synchronizing a Published Graph

You can synchronize a published graph by configuring the Flashback Synchronizer with a PartitionedGraphConfig object containing the graph schema along with the database connection details.

The PartitionedGraphConfig object can be created either through the PartitionedGraphConfigBuilder API or by reading the graph configuration from a JSON file.

Though synchronization of graphs created via graph configuration objects is supported in general, the following few limitations apply:

- Only partitioned graph configurations with all providers being database tables are supported.
- Each edge or vertex provider or both must specify the owner of the table by setting the username field. For example, if user SCOTT owns the table, then set the user name accordingly for the providers.
- Snapshot source must be set to CHANGE SET.
- It is highly recommended to optimize the graph for update operations in order to avoid memory exhaustion when creating many snapshots.

The following example shows the sample configuration for creating the PartitionedGraphConfig object:

- JSON Configuration
- GraphConfigBuilder API

JSON Configuration

```
{
    ...
    "optimized_for": "updates",
    "vertex providers": [
```



```
"username":"<username>",
...
],
"edge_providers": [
...
"username":"<username>",
...
],
"loading": {
    "snapshots_source": "change_set"
}
```

GraphConfigBuilder API

As a prerequisite requirement, you must have a graph that is published in an earlier session. For example:

- JShell
- Java
- Python

JShell

```
opg4j> var graph =
session.readGraphWithProperties("<path_to_json_config_file>")
graph ==>
PgxGraph[name=bank_graph_analytics_fb,N=999,E=4993,created=1664310157103]
opg4j> graph.publishWithSnapshots()
```

Java

```
PgxGraph graph =
session.readGraphWithProperties("<path_to_json_config_file>");
graph.publishWithSnapshots();
```



Python

```
>>> graph =
session.read_graph_with_properties("<path_to_json_config_file>")
>>> graph.publish_with_snapshots()
```

You can now perform the following steps to synchronize the published graph using a graph configuration object which is built from a JSON file.

- 1. Get the published graph as shown:
 - JShell
 - Java
 - Python

JShell

```
opg4j> var graph = session.getGraph("bank_graph")
graph ==>
PgxGraph[name=bank_graph_analytics_fb,N=999,E=4993,created=166431015
7103]
```

Java

```
PgxGraph graph = session.getGraph("bank graph");
```

Python

```
>>> graph = session.get graph("bank graph")
```

- 2. Build the graph configuration object using a JSON file path as shown:
 - JShell
 - Java
 - Python

JShell

```
opg4j> var cfg =
GraphConfigFactory.forPartitioned().fromFilePath("path to json confi
```



Java

```
PartitionedGraphConfig cfg =
GraphConfigFactory.forPartitioned().fromFilePath("path_to_json_config_file");
```

Python

```
>>> from pypgx.api import GraphConfigFactory
>>> cfg =
GraphConfigFactory.for_partitioned().from_file_path("path_to_json_config_f
ile")
```

Alternatively, you can also build the graph configuration object using the <code>GraphConfigBuilder</code> API as shown in Loading a Graph by Defining a Graph Configuration Object.

- 3. Change the data in the database table using the JDBC connection:
 - JShell
 - Java
 - Python

JShell

```
opg4j> var conn =
DriverManager.getConnection(<jdbcUrl>, <username>, <password>)
conn ==> oracle.jdbc.driver.T4CConnection@60f7261f
opg4j> var stmt = conn.createStatement()
stmt ==> oracle.jdbc.driver.OracleStatementWrapper@1a914a00
opg4j> stmt.executeQuery("UPDATE bank txns SET amount=9000 WHERE
```



```
txn_id=3")
$5 ==> oracle.jdbc.driver.ForwardOnlyResultSet@627d5f99
opg4j> conn.setAutoCommit(false)
opg4j> conn.commit()
```

Java

```
Connection conn =
DriverManager.getConnection(<jdbcUrl>, <username>, <password>);
Statement stmt = conn.createStatement();
stmt.executeQuery("UPDATE bank_txns SET amount=9000 WHERE
txn_id=3");
conn.setAutoCommit(false);
conn.commit();
```

Python

```
>>> conn = opg4py.pgql.get_connection("graphuser", "graphuser",
"jdbc:oracle:thin:@localhost:1521/orclpdb").get_jdbc_connection()
>>> conn.prepareStatement("UPDATE bank_txns SET amount=9000 WHERE
txn_id=3").execute()
False
>>> conn.commit()
```

- 4. Configure the Flashback synchronizer using the graph configuration object and the connection details:
 - JShell
 - Java
 - Python

JShell

Java

```
Synchronizer synchronizer = new
Synchronizer.Builder<FlashbackSynchronizer>()
```



```
.setType(FlashbackSynchronizer.class)
.setGraph(graph)
.setConnection(conn)
.setGraphConfiguration(cfg)
.build();
```

Python

- 5. Synchronize the published graph as shown:
 - JShell
 - Java
 - Python

JShell

```
opg4j> graph=synchronizer.sync()
graph ==> PgxGraph[name=bank graph, N=999, E=4993, created=1664454171605]
```

Java

```
graph=synchronizer.sync();
```

Python

```
>>> graph = synchronizer.sync()
```

- **6.** Query the graph to verify the updates to the edge property.
 - JShell
 - Java
 - Python



JShell

```
opg4j> graph.queryPgql("SELECT e.amount FROM MATCH (v1:Accounts) -
[e:Transfers]->(v2:Accounts) WHERE v1.ID=179 and v2.ID=688").print()
```

Java

```
graph.queryPgql("SELECT e.amount FROM MATCH (v1:Accounts) -
[e:Transfers] -> (v2:Accounts) WHERE v1.ID=179 and
v2.ID=688").print();
```

Python

```
graph.query_pgql("SELECT e.amount FROM MATCH (v1:Accounts) -
[e:Transfers] -> (v2:Accounts) WHERE v1.ID=179 and
v2.ID=688").print();
```

On execution, the preceding example produces the following output:

```
| amount |
|-----+
| 9000.0 |
```

14.4 Optimizing Graphs for Read Versus Updates in the Graph Server (PGX)

The graph server (PGX) can store an optimized graph for other reads or updates. This is only relevant when the updates are made directly to a graph instance in the graph server.

Graph Optimized for Reads

Graphs optimized for reads will provide the best performance for graph analytics and PGQL queries. In this case there could be potentially higher latencies to update the graph (adding or removing vertex and edges or updating the property values of previously existing vertex or edges through <code>GraphChangeSet</code> API). There could also be higher memory consumption. When using graphs optimized for reads, each updated graph or graph snapshot consumes memory proportional to the size of the graph in terms of vertices and edges.

The <code>optimized_for</code> configuration property can be set to <code>reads</code> when loading the graph into the graph server (PGX) to create a graph instance that is optimized for reads.

Graph Optimized for Updates

Graphs optimized for updates use a representation enabling low-latency update of graphs. With this representation, the graph server can reach millisecond-scale

latencies when updating graphs with millions of vertices and edges (this is indicative and will vary depending on the hardware configuration).

To achieve faster update operations, graph server avoids as much as possible doing a full duplication of the previous graph (snapshot) to create a new graph (snapshot). This also improves the memory consumption (in typical scenarios). New snapshots (or new graphs) will only consume additional memory proportional to the memory required for the changes applied.

In this representation, there could be lower performance of graph gueries and analytics.

The <code>optimized_for</code> configuration property can be set to <code>updates</code> when loading the graph into the graph server (PGX) to create a graph instance that is optimized for reads.

14.5 Executing Built-in Algorithms

The graph server (PGX) contains a set of built-in algorithms that are available as Java APIs.

The following table provides an overview of the available algorithms, grouped by category. Note that these algorithms can be invoked through the Analyst Class.



See the supported Built-In Algorithms on GitHub for more details.

Table 14-10 Overview of Built-In Algorithms

Category	Algorithms	
Classic graph algorithms	Prim's Algorithm	
Community detection	Conductance Minimization (Soman and Narang Algorithm), Infomap, Label Propagation, Louvain	
Connected components	Strongly Connected Components, Weakly Connected Components (WCC)	
Link predition	WTF (Whom To Follow) Algorithm	
Matrix factorization	Matrix Factorization	
Other	Graph Traversal Algorithms	
Path finding	All Vertices and Edges on Filtered Path, Bellman-Ford Algorithms, Bidirectional Dijkstra Algorithms, Compute Distance Index, Compute High-Degree Vertices, Dijkstra Algorithms, Enumerate Simple Paths, Fast Path Finding, Fattest Path, Filtered Fast Path Finding, Hop Distance Algorithms	
Ranking and walking	Closeness Centrality Algorithms, Degree Centrality Algorithms, Eigenvector Centrality, Hyperlink-Induced Topic Search (HITS), PageRank Algorithms, Random Walk with Restart, Stochastic Approach for Link-Structure Analysis (SALSA) Algorithms, Vertex Betweenness Centrality Algorithms	
Structure evaluation	Adamic-Adar index, Bipartite Check, Conductance, Cycle Detection Algorithms, Degree Distribution Algorithms, Eccentricity Algorithms, K-Core, Local Clustering Coefficient (LCC), Modularity, Partition Conductance, Reachability Algorithms, Topological Ordering Algorithms, Triangle Counting Algorithms	



This following topics describe the use of the graph server (PGX) using Triangle Counting and PageRank analytics as examples.

- About Built-In Algorithms in the Graph Server (PGX)
- Running the Triangle Counting Algorithm
- · Running the PageRank Algorithm

14.5.1 About Built-In Algorithms in the Graph Server (PGX)

The graph server (PGX) contains a set of built-in algorithms that are available as Java APIs. The details of the APIs are documented in the Javadoc that is included in the product documentation library. Specifically, see the BuiltinAlgorithms interface Method Summary for a list of the supported in-memory analyst methods.

For example, this is the PageRank procedure signature:

```
/**
  * Classic pagerank algorithm. Time complexity: O(E * K) with E =
number of edges, K is a given constant (max
  * iterations)
  *
  * @param graph
  * graph
  * graph
  * @param e
  * maximum error for terminating the iteration
  * @param d
  * damping factor
  * @param max
  * maximum number of iterations
  * @return Vertex Property holding the result as a double
  */
  public <ID extends Comparable<ID>> VertexProperty<ID, Double>
pagerank(PgxGraph graph, double e, double d, int max);
```

14.5.2 Running the Triangle Counting Algorithm

For triangle counting, the <code>sortByDegree</code> boolean parameter of <code>countTriangles()</code> allows you to control whether the graph should first be sorted by degree (<code>true</code>) or not (<code>false</code>). If <code>true</code>, more memory will be used, but the algorithm will run faster; however, if your graph is very large, you might want to turn this optimization off to avoid running out of memory.

- JShell
- Java

JShell

```
opg4j> analyst.countTriangles(graph, true)
==> 1
```

Java

```
import oracle.pgx.api.*;
Analyst analyst = session.createAnalyst();
long triangles = analyst.countTriangles(graph, true);
```

The algorithm finds one triangle in the sample graph.



Tip:

When using the graph shell, you can increase the amount of log output during execution by changing the logging level. See information about the :loglevel command with :h :loglevel.

14.5.3 Running the PageRank Algorithm

PageRank computes a rank value between 0 and 1 for each vertex (node) in the graph and stores the values in a double property. The algorithm therefore creates a *vertex property* of type double for the output.

In the graph server (PGX), there are two types of vertex and edge properties:

- Persistent Properties: Properties that are loaded with the graph from a data source are fixed, in-memory copies of the data on disk, and are therefore persistent. Persistent properties are read-only, immutable and shared between sessions.
- Transient Properties: Values can only be written to transient properties, which are
 private to a session. You can create transient properties by calling
 createVertexProperty and createEdgeProperty on PgxGraph objects, or by copying
 existing properties using clone() on Property objects.

Transient properties hold the results of computation by algorithms. For example, the PageRank algorithm computes a rank value between 0 and 1 for each vertex in the graph and stores these values in a transient property named pg_rank. Transient properties are destroyed when the Analyst object is destroyed.

This example obtains the top three vertices with the highest PageRank values. It uses a transient vertex property of type double to hold the computed PageRank values. The PageRank algorithm uses the following default values for the input parameters: error (tolerance = 0.001), damping factor = 0.85, and maximum number of iterations = 100.

- JShell
- Java

JShell

```
opg4j> rank = analyst.pagerank(graph, 0.001, 0.85, 100);
==> ...
```



```
opg4j> rank.getTopKValues(3)
==> 128=0.1402019732468347
==> 333=0.12002296283541904
==> 99=0.09708583862990475
```

Java

```
import java.util.Map.Entry;
import oracle.pgx.api.*;

Analyst analyst = session.createAnalyst();
VertexProperty<Integer, Double> rank = analyst.pagerank(graph, 0.001, 0.85, 100);
for (Entry<Integer, Double> entry : rank.getTopKValues(3)) {
   System.out.println(entry.getKey() + "=" + entry.getValue());
}
```

14.6 Using Custom PGX Graph Algorithms

A custom PGX graph algorithm allows you to write a graph algorithm in Java syntax and have it automatically compiled to an efficient parallel implementation.

- Writing a Custom PGX Algorithm
- · Compiling and Running a Custom PGX Algorithm
- Example Custom PGX Algorithm: PageRank

14.6.1 Writing a Custom PGX Algorithm

A PGX algorithm is a regular .java file with a single class definition that is annotated with @GraphAlgorithm. For example:

```
import oracle.pgx.algorithm.annotations.GraphAlgorithm;
@GraphAlgorithm
public class MyAlgorithm {
    ...
}
```

A PGX algorithm class must contain exactly one public method which will be used as entry point. The class may contain any number of private methods.

For example:

```
import oracle.pgx.algorithm.PgxGraph;
import oracle.pgx.algorithm.VertexProperty;
import oracle.pgx.algorithm.annotations.GraphAlgorithm;
import oracle.pgx.algorithm.annotations.Out;

@GraphAlgorithm
```



```
public class MyAlgorithm {
    public int myAlgorithm(PgxGraph g, @Out VertexProperty<Integer>
distance) {
        System.out.println("My first PGX Algorithm program!");
        return 42;
    }
}
```

As with normal Java methods, a PGX algorithm method only supports primitive data types as return values (an integer in this example). More interesting is the <code>@Out</code> annotation, which marks the vertex property <code>distance</code> as output parameter. The caller passes output parameters by reference. This way, the caller has a reference to the modified property after the algorithm terminates.

- Collections
- Iteration
- Reductions

14.6.1.1 Collections

To create a collection you call the .create() function. For example, a VertexProperty<Integer> is created as follows:

```
VertexProperty<Integer> distance = VertexProperty.create();
```

To get the value of a property at a certain vertex v:

```
distance.get(v);
```

Similarly, to set the property of a certain vertex v to a value e:

```
distance.set(v, e);
```

You can even create properties of collections:

```
VertexProperty<VertexSequence> path = VertexProperty.create();
```

However, PGX Algorithm assignments are always *by value* (as opposed to *by reference*). To make this explicit, you *must* call .clone() when assigning a collection:

```
VertexSequence sequence = path.get(v).clone();
```

Another consequence of values being passed by value is that you can check for equality using the == operator instead of the Java method .equals(). For example:

```
PgxVertex v1 = G.getRandomVertex();
PgxVertex v2 = G.getRandomVertex();
System.out.println(v1 == v2);
```



14.6.1.2 Iteration

The most common operations in PGX algorithms are iterations (such as looping over all vertices, and looping over a vertex's neighbors) and graph traversal (such as breath-first/depth-first). All collections expose a forEach and forSequential method by which you can iterate over the collection in parallel and in sequence, respectively.

For example:

To iterate over a graph's vertices in parallel:

```
G.getVertices().forEach(v -> {
    ...
});
```

To iterate over a graph's vertices in sequence:

```
G.getVertices().forSequential(v -> {
    ...
});
```

• To traverse a graph's vertices from r in breadth-first order:

Inside the forward (or backward) lambda you can access the current level of the BFS (or DFS) traversal by calling currentLevel().

14.6.1.3 Reductions

Within these parallel blocks it is common to atomically update, or reduce to, a variable defined outside the lambda. These atomic reductions are available as methods on Scalar<T>: reduceAdd, reduceAnd, and so on. For example, to count the number of vertices in a graph:

```
public int countVertices() {
    Scalar<Integer> count = Scalar.create(0);

G.getVertices().forEach(n -> {
        count.reduceAdd(1);
    });

return count.get();
}
```

Sometimes you want to update multiple values atomically. For example, you might want to find the smallest property value as well as the vertex whose property value

attains this smallest value. Due to the parallel execution, two separate reduction statements might get you in an inconsistent state.

To solve this problem the Reductions class provides argMin and argMax functions. The first argument to argMin is the current value and the second argument is the potential new minimum. Additionally, you can chain andUpdate calls on the ArgMinMax object to indicate other variables and the values that they should be updated to (atomically). For example:

14.6.2 Compiling and Running a Custom PGX Algorithm

To be able to compile and run a custom PGX algorithm, you must perform the following actions:



Compiling a custom PGX Algorithm using the PGX Algorithm API is not supported on Oracle JDK 17.

- 1. Set the following two configuration parameters in the conf/pgx.conf file:
 - Set the graph algorithm language option to JAVA.
 - Set the <code>java_home_dir</code> option to the path to your Java home (use <system-java-home-dir> to have PGX infer Java home from the system properties).

```
(
  "graph_algorithm_language": "JAVA",
  "java_home_dir": "<system-java-home-dir>"
```

- 2. Create a session.
 - JShell
 - Java
 - Python

JShell

```
cd /opt/oracle/graph
./bin/opg4j
```

Java

```
import oracle.pgx.algorithm.*;
PgxSession session = Pgx.createSession("my-session");
```

Python

```
session = instance.create session("my-session")
```

- **3.** Compile a PGX Algorithm. For example:
 - JShell
 - Java
 - Python

JShell

```
opg4j> var myAlgorithm = session.compileProgram("/path/to/
MyAlgorithm.java")
myAlgorithm ==> CompiledProgram[name=MyAlgorithm]
```

Java

```
import oracle.pgx.algorithm.CompiledProgram;
CompiledProgram myAlgorithm = session.compileProgram("/path/to/MyAlgorithm.java");
```

Python

```
my algorithm = session.compile program("/path/to/MyAlgorithm.java")
```

- **4.** Run the algorithm. For example:
 - JShell
 - Java
 - Python



JShell

```
opg4j> var graph =session.readGraphWithProperties("/path/to/
bank_graph_analytics.json")
graph ==>
PgxGraph[name=bank_graph_analytics,N=1000,E=5001,created=1633504705054]
opg4j> var property = graph.createVertexProperty(PropertyType.INTEGER)
property ==>
VertexProperty[name=vertex_prop_integer_9,type=integer,graph=bank_graph_an
alytics]
opg4j> myAlgorithm.run(graph, property)
$6 ==> {
    "success" : true,
    "canceled" : false,
    "exception" : null,
    "returnValue" : 42,
    "executionTimeMs" : 0
}
```

Java

```
import oracle.pgx.algorithm.VertexProperty;
PgxGraph graph = session.readGraphWithProperties("/path/to/bank_graph_analytics.json");
VertexProperty property = graph.createVertexProperty(PropertyType.INTEGER);
myAlgorithm.run(graph, property);
```

Python

```
graph = session.read_graph_with_properties("/path/to/
bank_graph_analytics.json")
property = graph.create_vertex_property("integer")
my_algorithm.run(graph, property)
{'success': True, 'canceled': False, 'exception': None, 'return_value':
42, 'execution_time(ms)': 1}
```

14.6.3 Example Custom PGX Algorithm: PageRank

The following is an implementation of pagerank as a PGX algorithm:

```
import oracle.pgx.algorithm.PgxGraph;
import oracle.pgx.algorithm.Scalar;
import oracle.pgx.algorithm.VertexProperty;
import oracle.pgx.algorithm.annotations.GraphAlgorithm;
import oracle.pgx.algorithm.annotations.Out;

@GraphAlgorithm
public class Pagerank {
   public void pagerank(PgxGraph G, double tol, double damp, int max iter,
```

```
boolean norm, @Out VertexProperty<Double> rank) {
    Scalar<Double> diff = Scalar.create();
    int cnt = 0;
    double N = G.getNumVertices();
    rank.setAll(1 / N);
    do {
      diff.set(0.0);
      Scalar<Double> dangling factor = Scalar.create(0d);
      if (norm) {
        dangling factor.set(damp / N * G.getVertices().filter(v ->
v.getOutDegree() == 0).sum(rank::get));
      G.getVertices().forEach(t -> {
        double in sum = t.getInNeighbors().sum(w -> rank.get(w) /
w.getOutDegree());
        double val = (1 - damp) / N + damp * in sum +
dangling factor.get();
        diff.reduceAdd(Math.abs(val - rank.get(t)));
        rank.setDeferred(t, val);
      });
      cnt++;
    } while (diff.get() > tol && cnt < max iter);</pre>
}
```

14.7 Creating Subgraphs

You can create subgraphs based on a graph that has been loaded into memory. You can use filter expressions or create bipartite subgraphs based on a vertex (node) collection that specifies the left set of the bipartite graph.



Starting from Graph Server and Client Release 22.3, creating subgraphs using filter expressions is deprecated. It is recommended that you load a subgraph from property graph views. See Loading a Subgraph from Property Graph Views for more information.

For information about reading a graph into memory, see Reading Graphs from Oracle Database into the Graph Server (PGX) for the various methods to load a graph into the graph server (PGX).

- About Filter Expressions
- Using a Simple Filter to Create a Subgraph
- · Using a Complex Filter to Create a Subgraph
- Using a Vertex Set to Create a Bipartite Subgraph



14.7.1 About Filter Expressions

Filter expressions are expressions that are evaluated for each vertex or edge. The expression can define predicates that a vertex or an edge must fulfil in order to be contained in the result, in this case a subgraph.

Consider an example graph that consists of four vertices (nodes) and four edges. For an edge to match the filter expression src.prop == 10, the source vertex prop property must equal 10. Two edges match that filter expression, as shown in the following figure.

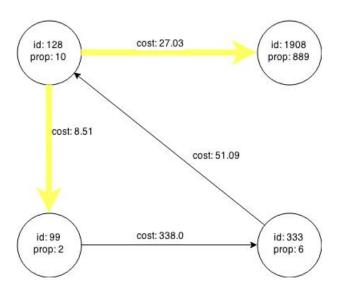


Figure 14-1 Edges Matching src.prop == 10

The following figure shows the graph that results when the filter is applied.

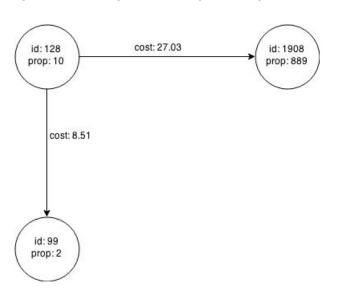


Figure 14-2 Graph Created by the Simple Filter



The vertex filter src.prop == 10 filters out the edges associated with vertex 333 and the vertex itself.

14.7.2 Using a Simple Filter to Create a Subgraph

The following examples create the subgraph described in About Filter Expressions.

- JShell
- Java

JShell

```
var subgraph = graph.filter(new VertexFilter("vertex.prop == 10"))
```

Java

```
import oracle.pgx.api.*;
import oracle.pgx.api.filter.*;

PgxGraph graph = session.readGraphWithProperties(...);

PgxGraph subgraph = graph.filter(new VertexFilter("vertex.prop == 10"));
```

14.7.3 Using a Complex Filter to Create a Subgraph

This example uses a slightly more complex filter. It uses the <code>outDegree</code> function, which calculates the number of outgoing edges for an identifier (source <code>src</code> or destination <code>dst</code>). The following filter expression matches all edges with a <code>cost</code> property value greater than 50 and a destination vertex (node) with an <code>outDegree</code> greater than 1.

```
dst.outDegree() > 1 && edge.cost > 50
```

One edge in the sample graph matches this filter expression, as shown in the following figure.



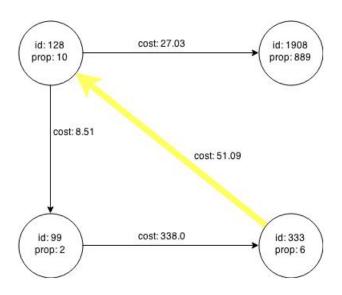


Figure 14-3 Edges Matching the outDegree Filter

The following figure shows the graph that results when the filter is applied. The filter excludes the edges associated with the vertices 99 and 1908, and so excludes those vertices also.

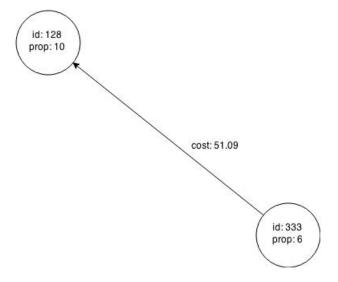


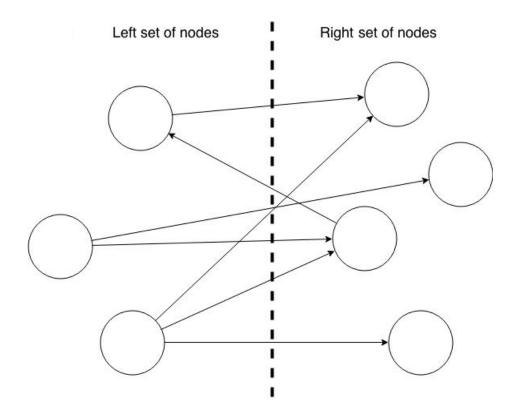
Figure 14-4 Graph Created by the outDegree Filter

14.7.4 Using a Vertex Set to Create a Bipartite Subgraph

You can create a bipartite subgraph by specifying a set of vertices (nodes), which are used as the left side. A bipartite subgraph has edges only between the left set of vertices and the right set of vertices. There are no edges within those sets, such as between two nodes on the left side. In the graph server (PGX), vertices that are isolated because all incoming and outgoing edges were deleted are not part of the bipartite subgraph.

The following figure shows a bipartite subgraph. No properties are shown.





The following examples create a bipartite subgraph from a simple graph consisting of four vertices and four edges. The vertex ID values for the four vertices are 99, 128, 1908 and 333 respectively. See Figure 14-1 in About Filter Expressions for more information on the vertex and edge property values including the edge direction between the vertices.

You must first create a vertex collection and fill it with the vertices for the left side. In the example shown, vertices with vertex ID values 333 and 99 are added to the left side of the vertex collection.

Using the Shell to Create a Bipartite Subgraph

```
opg4j> s = graph.createVertexSet()
==> ...
opg4j> s.addAll([graph.getVertex(333), graph.getVertex(99)])
==> ...
opg4j> s.size()
==> 2
opg4j> bGraph = graph.bipartiteSubGraphFromLeftSet(s)
==> PGX Bipartite Graph named sample-sub-graph-4
```

Using Java to Create a Bipartite Subgraph

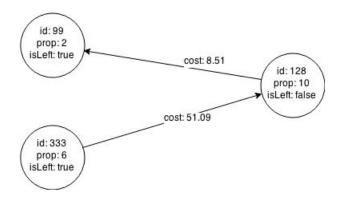
```
import oracle.pgx.api.*;

VertexSet<Integer> s = graph.createVertexSet();
s.addAll(graph.getVertex(333), graph.getVertex(99));
BipartiteGraph bGraph = graph.bipartiteSubGraphFromLeftSet(s);
```



When you create a subgraph, the graph server (PGX) automatically creates a Boolean vertex (node) property that indicates whether the vertex is on the left side. You can specify a unique name for the property.

The resulting bipartite subgraph looks like this:



Vertex with ID 1908 is excluded from the bipartite subgraph. The only edge that connected that vertex extended from 128 to 1908. The edge was removed, because it violated the bipartite properties of the subgraph. Vertex 1908 had no other edges, and so was removed as well. Moreover, the edge from the vertex with the ID 128 to the vertex with ID 99 is not present in the bipartite subgraph, because edges are only allowed to go from left to right (and not from right to left).

14.8 Using Automatic Delta Refresh to Handle Database Changes

You can automatically refresh (auto-refresh) graphs periodically to keep the in-memory graph synchronized with changes to the property graph stored in the property graph tables in Oracle Database (VT\$ and GE\$ tables).

Note that the auto-refresh feature is not supported when loading a graph into PGX in memory directly from relational tables.

- Configuring the Graph Server (PGX) for Auto-Refresh
- · Configuring Basic Auto-Refresh
- Reading the Graph Using the Graph Server (PGX) or a Java Application
- Checking Out a Specific Snapshot of the Graph
- Advanced Auto-Refresh Configuration
- · Special Considerations When Using Auto-Refresh

14.8.1 Configuring the Graph Server (PGX) for Auto-Refresh

Because auto-refresh can create many snapshots and therefore may lead to a high memory usage, by default the option to enable auto-refresh for graphs is available only to administrators.



To allow all users to auto-refresh graphs, you must include the following line into the graph server (PGX) configuration file (located in /etc/oracle/graph/pgx.conf):

```
{
   "allow_user_auto_refresh": true
}
```

14.8.2 Configuring Basic Auto-Refresh

Auto-refresh is configured in the loading section of the graph configuration. The example in this topic sets up auto-refresh to check for updates every minute, and to create a new snapshot when the data source has changed.

The following block (JSON format) enables the auto-refresh feature in the configuration file of the sample graph:

```
"format": "pg",
 "jdbc url": "jdbc:oracle:thin:@mydatabaseserver:1521/dbName",
 "username": "scott",
 "password": "<password>",
 "name": "my graph",
 "vertex props": [{
   "name": "prop",
   "type": "integer"
 } ],
 "edge props": [{
    "name": "cost",
   "type": "double"
 "separator": " ",
 "loading": {
   "auto refresh": true,
    "update interval sec": 60
 },
}
```

Notice the additional loading section containing the auto-refresh settings. You can also use the Java APIs to construct the same graph configuration programmatically:

```
GraphConfig config = GraphConfigBuilder.forPropertyGraphRdbms()
    .setJdbcUrl("jdbc:oracle:thin:@mydatabaseserver:1521/dbName")
    .setUsername("scott")
    .setPassword("<password>")
    .setName("my_graph")
    .addVertexProperty("prop", PropertyType.INTEGER)
    .addEdgeProperty("cost", PropertyType.DOUBLE)
    .setAutoRefresh(true)
    .setUpdateIntervalSec(60)
    .build();
```



14.8.3 Reading the Graph Using the Graph Server (PGX) or a Java Application

After creating the graph configuration, you can load the graph into the graph server (PGX) using the regular APIs.

```
opg4j> G = session.readGraphWithProperties("graphs/my-config.pg.json")
```

After the graph is loaded, a background task is started automatically, and it periodically checks the data source for updates.

14.8.4 Checking Out a Specific Snapshot of the Graph

The database is queried every minute for updates. If the graph has changed in the database after the time interval passed, the graph is reloaded and a new snapshot is created inmemory automatically.

You can "check out" (move a pointer to a different version of) the available in-memory snapshots of the graph using the <code>getAvailableSnapshots()</code> method of <code>PgxSession</code>. Example output is as follows:

```
opg4j> session.getAvailableSnapshots(G)
==> GraphMetaData [getNumVertices()=4, getNumEdges()=4, memoryMb=0,
dataSourceVersion=1453315103000, creationRequestTimestamp=1453315122669
(2016-01-20 10:38:42.669), creationTimestamp=1453315122685 (2016-01-20
10:38:42.685), vertexIdType=integer, edgeIdType=long]
==> GraphMetaData [getNumVertices()=5, getNumEdges()=5, memoryMb=3,
dataSourceVersion=1452083654000, creationRequestTimestamp=1453314938744
(2016-01-20 10:35:38.744), creationTimestamp=1453314938833 (2016-01-20
10:35:38.833), vertexIdType=integer, edgeIdType=long]
```

The preceding example output contains two entries, one for the originally loaded graph with 4 vertices and 4 edges, and one for the graph created by auto-refresh with 5 vertices and 5 edges.

To check out out a specific snapshot of the graph, use the setSnapshot() methods of PgxSession and give it the creationTimestamp of the snapshot you want to load.

For example, if G is pointing to the newer graph with 5 vertices and 5 edges, but you want to analyze the older version of the graph, you need to set the snapshot to 1453315122685. In the graph shell:

```
opg4j> G.getNumVertices()
==> 5
opg4j> G.getNumEdges()
==> 5
opg4j> session.setSnapshot( G, 1453315122685 )
==> null
opg4j> G.getNumVertices()
==> 4
```



```
opg4j> G.getNumEdges()
==> 4
```

You can also load a specific snapshot of a graph directly using the readGraphAsOf() method of PgxSession. This is a shortcut for loading a graph with readGraphWithProperty() followed by a setSnapshot(). For example:

```
opg4j> G = session.readGraphAsOf( config, 1453315122685 )
```

If you do not know or care about what snapshots are currently available in-memory, you can also specify a time span of how "old" a snapshot is acceptable by specifying a maximum allowed age. For example, to specify a maximum snapshot age of 60 minutes, you can use the following:

```
opg4j>G = session.readGraphWithProperties(config, 601, TimeUnit.MINUTES)
```

If there are one or more snapshots in memory younger (newer) than the specified maximum age, the youngest (newest) of those snapshots will be returned. If all the available snapshots are older than the specified maximum age, or if there is no snapshot available at all, then a new snapshot will be created automatically.

14.8.5 Advanced Auto-Refresh Configuration

You can specify advanced options for auto-refresh configuration.

Internally, the graph server (PGX) fetches the changes since the last check from the database and creates a new snapshot by applying the delta (changes) to the previous snapshot. There are two timers: one for fetching and caching the deltas from the database, the other for actually applying the deltas and creating a new snapshot.

Additionally, you can specify a threshold for the number of cached deltas. If the number of cached changes grows above this threshold, a new snapshot is created automatically. The number of cached changes is a simple sum of the number of vertex changes plus the number of edge changes.

The deltas are fetched periodically and cached on the graph server (PGX) for two reasons:

- To speed up the actual snapshot creation process
- To account for the case that the database can "forget" changes after a while

You can specify both a threshold and an update timer, which means that both conditions will be checked before new snapshot is created. At least one of these parameters (threshold or update timer) must be specified to prevent the delta cache from becoming too large. The interval at which the source is queried for changes must not be omitted.

The following parameters show a configuration where the data source is queried for new deltas every 5 minutes. New snapshots are created every 20 minutes or if the cached deltas reach a size of 1000 changes.

```
{
    "format": "pg",
```



```
"jdbc_url": "jdbc:oracle:thin:@mydatabaseserver:1521/dbName",
"username": "scott",
"password": "<your_password>",
"name": "my_graph",

"loading": {
    "auto_refresh": true,
    "fetch_interval_sec": 300,
    "update_interval_sec": 1200,
    "update_threshold": 1000,
    "create_edge_id_index": true,
    "create_edge_id_mapping": true
}
```

14.8.6 Special Considerations When Using Auto-Refresh

This section explains a few special considerations when you enable auto-refresh for graphs in the graph server (PGX):

If you call graph.destroy(), auto-refresh does not immediately stop.
 It only stops once the graph is actually freed from the server memory.

This happens when all the following conditions are true:

- 1. No other session is referencing that graph.
- 2. PGX consumes more than release_memory_threshold memory.

 release_memory_threshold is a pgx.conf option that defaults to 85% of available system memory.
- The PGX "garbage collector" has been run.
 memory_cleanup_interval is a pgx.conf option which defaults to once every 10
- If you configure the graph to be loaded with auto-refresh, you cannot omit the jdbc url, username and keystore parameters from the graph configuration file since auto-refreshed graphs are not "user bound". You cannot obtain the connection settings from the user who initiated it.

14.9 User-Defined Functions (UDFs) in PGX

User-defined functions (UDFs) allow users of PGX to add custom logic to their PGQL queries or custom graph algorithms, to complement built-in functions with custom requirements.

Caution:

UDFs enable running arbitrary code in the PGX server, possibly accessing sensitive data. Additionally, any PGX session can invoke any of the UDFs that are enabled on the PGX server. The application administrator who enables UDFs is responsible for checking the following:

- All the UDF code can be trusted.
- The UDFs are stored in a secure location that cannot be tampered with.

Furthermore, PGX assumes UDFs to be state-less and side-effect free.

PGX supports two types of UDFs:

- Java UDFs
- JavaScript UDFs

How to Use Java UDFs

The following simple example shows how to register a Java UDF at the PGX server and invoke it.

1. Create a class with a public static method. For example:

```
package my.udfs;
public class MyUdfs {
  public static String concat(String a, String b) {
    return a + b;
  }
}
```

2. Compile the class and compress into a JAR file. For example:

```
mkdir ./target
javac -d ./target *.java
cd target
jar cvf MyUdfs.jar *
```

- 3. Copy the JAR file into /opt/oracle/graph/pgx/server/lib.
- 4. Create a UDF JSON configuration file. For example, assume that /path/to/my/ udfs/dir/my udfs.json contains the following:

```
"user defined functions": [
   "namespace": "my",
   "language": "java",
   "implementation_reference": "my.udfs.MyUdfs",
   "function name": "concat",
   "return type": "string",
   "arguments": [
```



```
{
    "name": "a",
    "type": "string"
},
{
    "name": "b",
    "type": "string"
}

}
]
}
```

5. Point to the directory containing the UDF configuration file in /etc/oracle/graph/pgx.conf. For example:

```
"udf config directory": "/path/to/my/udfs/dir/"
```

6. Restart the PGX server. For example:

```
sudo systemctl restart pgx
```

7. Try to invoke the UDF from within a PGQL query. For example:

```
graph.queryPgql("SELECT my.concat(my.concat(n.firstName, ' '),
n.lastName) FROM MATCH (n:Person)")
```

8. Try to invoke the UDF from within a PGX algorithm. For example:

Note:

For each UDF you want to use, you need to create an abstract method with the same schema that gets annotated with the <code>@Udf</code> annotation.

```
import oracle.pgx.algorithm.annotations.Udf;
....

@GraphAlgorithm
public class MyAlogrithm {
   public void bomAlgorithm(PgxGraph g, VertexProperty<String> firstName,
VertexProperty<String> lastName, @Out VertexProperty<String> fullName) {
    ... fullName.set(v, concat(firstName.get(v), lastName.get(v))); ...
}

@Udf(namespace = "my")
   abstract String concat(String a, String b);
}
```



JavaScript UDFs

The requirements for a JavaScript UDF is as follows:

- The JavaScript source must contain all dependencies.
- The source must contain at least one valid export.
- The language parameter must be set to javascript in the UDF configuration file.

For example, consider a JavaScript source file format.js as shown:

```
//format.js
const fun = function(name, country) {
  if (country == null) return name;
  else return name + " (" + country + ")";
}
module.exports = {stringFormat: fun};
```

In order to load the UDF from format.js, the UDF configuration file will appear as follows:

```
{
  "namespace": "my",
  "function_name": "format",
  "language": "javascript",
  "source_location": "format.js",
  "source_function_name": "stringFormat",
  "return_type": "string",
  "arguments": [
      {
            "name": "name",
            "type": "string"
      },
      {
            "name": "country",
            "type": "string"
      }
      ]
}
```

Note:

In this case, since the name of the UDF and the implementing method differ, you need to set the name of the UDF in the <code>source_function_name</code> field. Also, you can provide the path of the source code file in the <code>source_location</code> field.



UDF Configuration File Information

A UDF configuration file is a JSON file containing an array of user_defined_functions. (An example of such a file is in the step to "Create a UDF JSON configuration file" in the preceding How to Use Java UDFs subsection.)

Each user-defined function supports the fields shown in the following table.

Table 14-11 Fields for Each UDF

Field	Data Type	Description	Required?
function_name	string	Name of the function used as identifier in PGX	Required
language	enum[java, javascript]	Source language for he function (java or javascript)	Required
return_type	enum[boolean, integer, long, float, double, string]	Return type of the function	Required
arguments	array of object	Array of arguments. For each argument: type, argument name, required?	D
implementation_reference	string	Reference to the function name on the classpath	null
namespace	string	Namespace of the function in PGX	null
source_code	string	Source code of the function provided inline	null
source_function_name	string	Name of the function in the source language	null
source_location	string	Local file path to the function's source code	null

All configured UDFs must be unique with regard to the combination of the following fields:

- namespace
- function_name
- arguments

14.10 Using the Graph Server (PGX) as a Library

When you utilize PGX as a library in your application, the graph server (PGX) instance runs in the same JVM as the Java application and all requests are translated into direct function calls instead of remote procedure invocations.

In this case, you must install the graph server (PGX) using RPM in the same machine as the client applications. The shell executables provided by the graph server installation helps you to launch the Java or the Python shell in an embedded server mode. See Installing Oracle Graph Server for more information.



You can now start the Java shell without any parameters as shown:

```
cd /opt/oracle/graph
./bin/opg4j
```

The local PGX instance will try to load a PGX configuration file from:

```
/etc/oracle/graph/pgx.conf
```

You can change the location of the configuration file by passing the --pgx_conf command-line option followed by the path to the configuration file:

```
# start local PGX instance with custom config
./bin/opg4j --pgx conf <path to pgx.conf>
```

You can also start the Python shell without any parameters as shown:

```
cd /opt/oracle/graph/
./bin/opg4py
```

When using Java, you can obtain a reference to the local PGX instance as shown:

```
import oracle.pg.rdbms.*;
import oracle.pgx.api.*;
...
ServerInstance instance = GraphServer.getEmbeddedInstance();
```

In a Python application, you can obtain a reference to the local PGX instance as shown:

```
import os
os.environ["PGX_CLASSPATH"] = "/opt/oracle/graph/lib/*"
import opg4py.graph_server as graph_server
...
instance = graph server.get embedded instance()
```

Starting the PGX Engine

PGX provides a convenience mechanism to start the PGX Engine when using the graph server (PGX) as a library. That is, the graph server (PGX) is automatically initialized and starts up automatically when <code>ServerInstance.createSession()</code> is called the first time. This is provided that the engine is not already running at that time.

For this implicit initialization, PGX will configure itself with the PGX configuration file at the default locations. If the PGX configuration file is not found, PGX will configure itself using default parameter values as shown in Configuration Parameters for the Graph Server (PGX) Engine.

Stopping the PGX Engine

When using the graph server (PGX) as a library, the <code>shutdownEngine()</code> method will be called automatically via a JVM shutdown hook on exit. Specifically, the shutdown hook is invoked once all the non-daemon threads of the application exit.



It is recommended that you do not terminate your PGX application forcibly with kill -9, as it will not clear the temp directory. See tmp_dir in Configuration Parameters for the Graph Server (PGX) Engine.



15

Using the Machine Learning Library (PgxML) for Graphs

The graph server (PGX) provides a machine learning library <code>oracle.pgx.api.mllib</code>, which supports graph-empowered machine learning algorithms.

The following machine learning algorithms are currently supported:

- Using the DeepWalk Algorithm
 DeepWalk is a widely employed vertex representation learning algorithm used in industry.
- Using the Supervised GraphWise Algorithm (Vertex Embeddings and Classification)
 Supervised GraphWise is an inductive vertex representation learning algorithm which is able to leverage vertex feature information. It can be applied to a wide variety of tasks, including vertex classification and link prediction.
- Using the Unsupervised GraphWise Algorithm
 Unsupervised GraphWise is an unsupervised inductive vertex representation learning algorithm which is able to leverage vertex information. The learned embeddings can be used in various downstream tasks including vertex classification, vertex clustering and similar vertex search.
- Using the Supervised EdgeWise Algorithm (Edge Embeddings and Classification)
 SupervisedEdgeWise is an inductive edge representation learning algorithm which is able to leverage vertex and edge feature information. It can be applied to a wide variety of tasks, including edge classification and link prediction.
- Using the Pg2vec Algorithm
 Pg2vec learns representations of graphlets (partitions inside a graph) by employing edges as the principal learning units and thereby packing more information in each learning unit (as compared to employing vertices as learning units) for the representation learning task.
- Model Repository and Model Stores
 A model store can be used to persist the trained graph server (PGX) machine learning models along with a model name (a unique identifier of the model in a particular model store) and a description.



Model Repository and Model Stores for information on model store management and how models can be persisted in a model store.

15.1 Using the DeepWalk Algorithm

DeepWalk is a widely employed vertex representation learning algorithm used in industry.

It consists of two main steps:

- 1. First, the random walk generation step computes random walks for each vertex (with a pre-defined walk length and a pre-defined number of walks per vertex).
- 2. Second, these generated walks are fed to a **Word2vec** algorithm to generate the vector representation for each vertex (which is the word in the input provided to the Word2vec algorithm). See KDD paper for more details on DeepWalk algorithm.

DeepWalk creates vertex embeddings for a specific graph and cannot be updated to incorporate modifications on the graph. Instead, a new DeepWalk model should be trained on this modified graph. Lastly, it is important to note that the memory consumption of the DeepWalk model is O(2n*d) where n is the number of vertices in the graph and d is the embedding length.

The following describes the usage of the main functionalities of DeepWalk in PGX using DBpedia graph as an example with 8,637,721 vertices and 165,049,964 edges:

- Loading a Graph
- Building a Minimal DeepWalk Model
- Building a Customized DeepWalk Model
- Training a DeepWalk Model
- Getting the Loss Value For a DeepWalk Model
- Computing Similar Vertices for a Given Vertex
- Computing Similar Vertices for a Vertex Batch
- Getting All Trained Vertex Vectors
- Storing a Trained DeepWalk Model
- Loading a Pre-Trained DeepWalk Model
- Destroying a DeepWalk Model

15.1.1 Loading a Graph

The following describes the steps for loading a graph:

- 1. Create a Session and an Analyst.
 - JShell
 - Java
 - Python

JShell

```
cd /opt/oracle/graph/
./bin/opg4j
// starting the shell will create an implicit session and analyst
```



Java

```
import oracle.pgx.api.*;
import oracle.pgx.api.mllib.DeepWalkModel;
import oracle.pgx.api.frames.*;
...
PgxSession session = Pgx.createSession("my-session");
Analyst analyst = session.createAnalyst();
```

Python

```
session = pypgx.get_session(session_name="my-session")
analyst = session.create analyst()
```

2. Load the graph.



Though the DeepWalk algorithm implementation can be applied to directed or undirected graphs, currently only undirected random walks are considered.

- JShell
- Java
- Python

JShell

```
opg4j> var graph = session.readGraphWithProperties("<path>/<graph.json>")
```

Java

PgxGraph graph = session.readGraphWithProperties("<path>/<graph.json>");

Python

```
graph = session.read graph with properties("<path>/<graph.json>")
```

15.1.2 Building a Minimal DeepWalk Model

You can build a DeepWalk model using the minimal configuration and default hyper-parameters as described in the following code:

- JShell
- Java
- Python

JShell

Java

```
DeepWalkModel model = analyst.deepWalkModelBuilder()
    .setWindowSize(3)
    .setWalksPerVertex(6)
    .setWalkLength(4)
    .build();
```

Python

```
model =
analyst.deepwalk_builder(window_size=3, walks_per_vertex=6, walk_length=4)
```

15.1.3 Building a Customized DeepWalk Model

You can build a DeepWalk model using customized hyper-parameters as described in the following code:

- JShell
- Java
- Python

JShell

```
setLearningRate(0.05).
setMinLearningRate(0.0001).
setWindowSize(3).
setWalksPerVertex(6).
setWalkLength(4).
setSampleRate(0.00001).
setNegativeSample(2).
setValidationFraction(0.01).
build()
```

Java

```
DeepWalkModel model= analyst.deepWalkModelBuilder()
    .setMinWordFrequency(1)
    .setBatchSize(512)
    .setNumEpochs(1)
    .setLayerSize(100)
    .setLearningRate(0.05)
    .setMinLearningRate(0.0001)
    .setWindowSize(3)
    .setWalkSPerVertex(6)
    .setWalkLength(4)
    .setSampleRate(0.00001)
    .setNegativeSample(2)
    .setValidationFraction(0.01)
    .build();
```

Python

See DeepWalkModelBuilder in Javadoc for more explanation for each builder operation along with the default values.

15.1.4 Training a DeepWalk Model

You can train a DeepWalk model with the specified default or customized settings as described in the following code:

- JShell
- Java
- Python

JShell

```
opg4j> model.fit(graph)
```

Java

```
model.fit(graph);
```

Python

model.fit(graph)

15.1.5 Getting the Loss Value For a DeepWalk Model

You can fetch the loss value on a specified fraction of training data, that is set in builder using <code>setValidationFraction</code> as described in the following code:

- JShell
- Java
- Python

JShell

```
opg4j> var loss = model.getLoss()
```

Java

```
double loss = model.getLoss();
```

Python

loss = model.loss



15.1.6 Computing Similar Vertices for a Given Vertex

You can fetch the k most similar vertices for a given vertex as described in the following code:

- JShell
- Java
- Python

JShell

```
opg4j> var similars = model.computeSimilars("Albert_Einstein", 10)
opg4j> similars.print()
```

Java

```
PgxFrame similars = model.computeSimilars("Albert_Einstein", 10);
similars.print();
```

Python

```
similars = model.compute_similars("Albert_Einstein",10)
similars.print()
```

Searching for similar vertices for Albert_Einstein using the trained model, will result in the following output:



15.1.7 Computing Similar Vertices for a Vertex Batch

You can fetch the ${\tt k}$ most similar vertices for a list of input vertices as described in the following code:

- JShell
- Java
- Python

JShell

```
opg4j> var vertices = new ArrayList()
opg4j> vertices.add("Machine_learning")
opg4j> vertices.add("Albert_Einstein")
opg4j> batchedSimilars = model.computeSimilars(vertices, 10)
opg4j> batchedSimilars.print()
```

Java

```
List vertices = Arrays.asList("Machine_learning", "Albert_Einstein");
PgxFrame batchedSimilars = model.computeSimilars(vertices, 10);
batchedSimilars.print();
```

Python

```
vertices = ["Machine_learning","Albert_Einstein"]
batched_similars = model.compute_similars(vertices,10)
batched_similars.print()
```

The following describes the output result:



15.1.8 Getting All Trained Vertex Vectors

You can retrieve the trained vertex vectors for the current DeepWalk model and store it in the database as described in the following code:

- JShell
- Java
- Python

JShell

```
opg4j> var vertexVectors = model.getTrainedVertexVectors().flattenAll()
opg4j> vertexVectors.write().db().name("vertex
vectors").tablename("vertexVectors").overwrite(true).store()
```

Java

```
PgxFrame vertexVectors = model.getTrainedVertexVectors().flattenAll();
vertexVectors.write()
   .db()
   .name("vertex vectors")
   .tablename("vertexVectors")
   .overwrite(true)
   .store();
```

Python

```
vertex_vectors = model.trained_vectors.flatten_all()
vertex_vectors.write().db().table_name("table_name").name("vertex_vectors").o
verwrite(True).store()
```



15.1.9 Storing a Trained DeepWalk Model

You can store models in database. The models get stored as a row inside a model store table.

The following code shows how to store a trained DeepWalk model in database in a specific model store table:

- JShell
- Java
- Python

JShell

Java

Python



Note:

All the preceding examples assume that you are storing the model in the current logged in database. If you must store the model in a different database then refer to the examples in Storing a Trained Model in Another Database.

Storing a Trained Model in Another Database

15.1.9.1 Storing a Trained Model in Another Database

You can store models in a different database other than the one used for login.

The following code shows how to store a trained model in a different database:

- JShell
- Java
- Python

JShell

```
opg4j> model.export().db().
              username("user").
                                                // DB user to use for
storing the model
             password("password").
                                                 // password of the DB user
              jdbcUrl("jdbcUrl").
                                                 // jdbc url to the DB
             modelstore("modelstoretablename"). // name of the model store
table
             modelname("model").
                                                 // model name (primary key
of model store table)
             description("a model description"). // description to store
alongside the model
             store()
```



```
model.export().db(username="user",
                                                            # DB user
to use for storing the model
                  password="password",
                                                            # password
of the DB user
                  jdbc url="jdbc url",
                                                            # jdbc url
to the DB
                  model store="modelstoretablename",
                                                            # name of
the model store table
                  model name="model",
                                                            # model
name (primary key of model store table)
                  model description="a model description") #
description to store alongside the model
```

15.1.10 Loading a Pre-Trained DeepWalk Model

You can load models from a database.

You can load a pre-trained DeepWalk model from a model store table in database as described in the following code:

Loading a Pre-Trained DeepWalk Model Using JShell

Loading a Pre-Trained DeepWalk Model Using Java

```
DeepWalkModelmodel = analyst.loadDeepWalkModel().db()
    .modelstore("modeltablename") // name of the model store table
    .modelname("model") // model name (primary key of model store table)
    .load();
```

Loading a Pre-Trained DeepWalk Model Using Python



Note:

All the preceding examples assume that you are loading the model from the current logged in database. If you must load the model from a different database then refer to the examples in Loading a Pre-Trained Model From Another Database.

Loading a Pre-Trained Model From Another Database

15.1.10.1 Loading a Pre-Trained Model From Another Database

You can load models from a different database other than the one used for login.

You can load a pre-trained model from a model store table in database as described in the following code:

- JShell
- Java
- Python

JShell

where <modelLoader> applies as follows:

- loadDeepWalkModel(): Loads a Deepwalk model
- loadSupervisedGraphWiseModel(): Loads a Supervised GraphWise model
- loadUnsupervisedGraphWiseModel(): Loads an Unsupervised GraphWise model
- loadSupervisedEdgeWiseModel(): Loads a Supervised EdgeWise model
- loadPg2vecModel(): Loads a Pg2vec model



```
.modelstore("modeltablename") // name of the model store table
.modelname("model") // model name (primary key of model
store table)
.load();
```

where <*modeltype*> can have the following values based on the model to be loaded:

- DeepWalkModel: represents a Deepwalk model
- SupervisedGraphWiseModel: represents a Supervised GraphWise model
- UnsupervisedGraphWiseModel: represents an Unsupervised GraphWise model
- SupervisedEdgeWiseModel: represents a Supervised EdgeWise model
- Pg2vecModel: represents a Pg2vec model

where <modelLoader> applies as follows:

- loadDeepWalkModel(): Loads a Deepwalk model
- loadSupervisedGraphWiseModel(): Loads a Supervised GraphWise model
- loadUnsupervisedGraphWiseModel(): Loads an Unsupervised GraphWise model
- loadSupervisedEdgeWiseModel(): Loads a Supervised EdgeWise model
- loadPg2vecModel(): Loads a Pg2vec model

Python

where <modelLoader> applies as follows:

- get deepwalk model loader(): Loads a Deepwalk model
- get_supervised_graphwise_model_loader(): Loads a Supervised GraphWise model
- get_unsupervised_graphwise_model_loader(): Loads an Unsupervised GraphWise model
- get_supervised_edgewise_model_loader(): Loads a Supervised EdgeWise model
- get pg2vec model loader(): Loads a Pg2vec model



15.1.11 Destroying a DeepWalk Model

You can destroy a DeepWalk model as described in the following code:

- JShell
- Java
- Python

JShell

```
opg4j> model.destroy()
```

Java

```
model.destroy();
```

Python

model.destroy()

15.2 Using the Supervised GraphWise Algorithm (Vertex Embeddings and Classification)

Supervised GraphWise is an inductive vertex representation learning algorithm which is able to leverage vertex feature information. It can be applied to a wide variety of tasks, including vertex classification and link prediction.

Supervised GraphWise is based on GraphSAGE by Hamilton et al.

Model Structure

A Supervised GraphWise model consists of graph convolutional layers followed by several prediction layers.

The forward pass through a convolutional layer for a vertex proceeds as follows:

- A set of neighbors of the vertex is sampled.
- 2. The previous layer representations of the neighbors are mean-aggregated, and the aggregated features are concatenated with the previous layer representation of the vertex.
- 3. This concatenated vector is multiplied with weights, and a bias vector is added.
- 4. The result is normalized to such that the layer output has unit norm.



The prediction layers are standard neural network layers.

The following describes the usage of the main functionalities of the implementation of **GraphSAGE** in PGX using the Cora graph as an example:

- Loading a Graph
- Building a Minimal GraphWise Model
- Advanced Hyperparameter Customization
- Building a GraphWise Model Using Heterogeneous Graphs
- Classification Versus Regression Models on Supervised GraphWise Models
- Setting a Custom Loss Function and Batch Generator (for Anomaly Detection)
- Training a Supervised GraphWise Model
- Getting the Loss Value For a Supervised GraphWise Model
- Inferring the Vertex Labels for a Supervised GraphWise Model
- Evaluating the Supervised GraphWise Model Performance
- Inferring Embeddings for a Supervised GraphWise Model
- Storing a Trained Supervised GraphWise Model
- · Loading a Pre-Trained Supervised GraphWise Model
- Destroying a Supervised GraphWise Model
- Explaining a Prediction of a Supervised GraphWise Model

15.2.1 Loading a Graph

The following describes the steps for loading a graph:

- 1. Create a Session and an Analyst.
 - JShell
 - Java
 - Python

```
cd /opt/oracle/graph/
./bin/opg4j

// starting the shell will create an implicit session and analyst
opg4j> import oracle.pgx.config.mllib.ActivationFunction
opg4j> import oracle.pgx.config.mllib.WeightInitScheme
opg4j> PgxSession session = Pgx.createSession("my-session")
opg4j> Analyst analyst = session.createAnalyst()
```



```
import oracle.pgx.api.*;
import oracle.pgx.api.mllib.SupervisedGraphWiseModel;
import oracle.pgx.api.filter.VertexFilter;
import oracle.pgx.api.frames.*;
import oracle.pgx.config.mllib.ActivationFunction;
import oracle.pgx.config.mllib.GraphWiseConvLayerConfig;
import oracle.pgx.config.mllib.GraphWisePredictionLayerConfig;
import oracle.pgx.config.mllib.SupervisedGraphWiseModelConfig;
import oracle.pgx.config.mllib.WeightInitScheme;
PgxSession session = Pgx.createSession("my-session");
Analyst analyst = session.createAnalyst();
```

Python

```
session = pypgx.get_session(session_name="my-session")
analyst = session.create analyst()
```

2. Load the graph.

- JShell
- Java
- Python

JShell



```
List<PgxVertex> testVertices = fullGraph.getVertices()
    .stream()
    .filter(v->!trainGraph.hasVertex(v.getId()))
    .collect(Collectors.toList());
```

```
from pypgx.api.filters import VertexFilter
full_graph = session.read_graph_with_properties("<path>/
<cora_full_graph.json>")
vertex_filter =
VertexFilter.from_pgql_result_set(session.query_pgql("SELECT v FROM cora MATCH (v) WHERE ID(v) % 4 > 0"),"v")
train_graph = full_graph.filter(vertex_filter)
test_vertices = []
train_vertices = train_graph.get_vertices()
for v in full_graph.get_vertices():
    if(not train_vertices.contains(v)):
        test vertices.append(v)
```

15.2.2 Building a Minimal GraphWise Model

You can build a GraphWise model using the minimal configuration and default hyperparameters as described in the following code. You can create a model with one of the following options:

- only vertex properties
- · only edge properties
- both vertex and edge properties
- JShell
- Java
- Python



```
SupervisedGraphWiseModel model = analyst.supervisedGraphWiseModelBuilder()
    .setVertexInputPropertyNames("features")
    .setVertexTargetPropertyName("labels")
    .setEdgeInputPropertyNames("cost")
    .build();
```

Python



Even though only one vertex and one edge property is specified in the preceding example, you can specify a list of vertex or edge properties.

15.2.3 Advanced Hyperparameter Customization

You can build a GraphWise model using rich hyperparameter customization.

This is done through the following two sub-config classes:

- 1. GraphWiseConvLayerConfig
- 2. GraphWisePredictionLayerConfig

You can create a model with one of the following options:

- only vertex properties
- only edge properties
- both vertex and edge properties

The following code describes the implementation of the configuration using the preceding classes in GraphWise model. The example also specifies a weight decay parameter of 0.001 and dropout with dropping probability 0.5 for the GraphWise model to counteract overfitting.

- JShell
- Java
- Python



JShell

```
opg4j> var weightProperty = analyst.pagerank(trainGraph).getName();
opg4j> var convLayerConfig = analyst.graphWiseConvLayerConfigBuilder().
                setNumSampledNeighbors (25).
                setActivationFunction(ActivationFunction.TANH).
                setWeightInitScheme (WeightInitScheme.XAVIER).
                setWeightedAggregationProperty(weightProperty).
                setDropoutRate(0.5).
                build()
opg4j> var predictionLayerConfig =
analyst.graphWisePredictionLayerConfigBuilder().
                setHiddenDimension(32).
                setActivationFunction(ActivationFunction.RELU).
                setWeightInitScheme(WeightInitScheme.HE).
                setDropoutRate(0.5).
                build()
opg4j> var model = analyst.supervisedGraphWiseModelBuilder().
                setVertexInputPropertyNames("vertex features").
                setEdgeInputPropertyNames("edge features").
                setVertexTargetPropertyName("labels").
                setConvLayerConfigs(convLayerConfig).
                setPredictionLayerConfigs(predictionLayerConfig).
                setWeightDecay(0.001).
                setNormalize(false).
                setEmbeddingDim(256).
                setLearningRate (0.05).
                setNumEpochs(30).
                setSeed(42).
                setShuffle(false).
                setStandardize(true).
                setBatchSize(64).
                build()
```

```
String weightProperty = analyst.pagerank(trainGraph).getName();
GraphWiseConvLayerConfig convLayerConfig =
analyst.graphWiseConvLayerConfigBuilder()
    .setNumSampledNeighbors(25)
    .setActivationFunction(ActivationFunction.TANH)
    .setWeightInitScheme(WeightInitScheme.XAVIER)
    .setWeightedAggregationProperty(weightProperty)
    .setDropoutRate(0.5)
    .build();
GraphWisePredictionLayerConfig predictionLayerConfig =
analyst.graphWisePredictionLayerConfigBuilder()
    .setHiddenDimension(32)
    .setActivationFunction(ActivationFunction.RELU)
    .setWeightInitScheme(WeightInitScheme.HE)
    .setDropoutRate(0.5)
    .build();
```



```
SupervisedGraphWiseModel model = analyst.supervisedGraphWiseModelBuilder()
    .setVertexInputPropertyNames("vertex features")
    .setEdgeInputPropertyNames("edge features")
    .setVertexTargetPropertyName("labels")
    .setConvLayerConfigs(convLayerConfig)
    .setPredictionLayerConfigs(predictionLayerConfig)
    .setWeightDecay(0.001)
    .setNormalize(false)
    .setEmbeddingDim(256)
    .setLearningRate(0.05)
    .setNumEpochs(30)
    .setSeed(42)
    .setShuffle(false)
    .setStandardize(true)
    .setBatchSize(64)
    .build();
```

```
weightProperty = analyst.pagerank(train graph).name
conv layer config = dict(num sampled neighbors=25,
                         activation fn='tanh',
                         weight init scheme='xavier',
                         neighbor weight property name=weightProperty,
                         dropout rate=0.5)
conv layer = analyst.graphwise conv layer config(**conv layer config)
pred layer config = dict(hidden dim=32,
                         activation fn='relu',
                         weight_init_scheme='he',
                         dropout rate=0.5)
pred layer = analyst.graphwise pred layer config(**pred layer config)
params = dict(vertex target property name="labels",
              conv layer config=[conv layer],
              pred layer config=[pred layer],
              vertex input property names=["vertex features"],
              edge input property names=["edge features"],
              seed=17,
              weight_decay=0.001,
              normalize=false,
              layer size=256,
              learning rate=0.05,
              num epochs=30,
              seed=42,
              standardize=true,
              batch size=64
)
model = analyst.supervised graphwise builder(**params)
```

See SupervisedGraphWiseModelBuilder, GraphWiseConvLayerConfigBuilder and GraphWisePredictionLayerConfigBuilder in Javadoc for a full description of all available hyperparameters and their default values.

15.2.4 Building a GraphWise Model Using Heterogeneous Graphs

You can build a GraphWise model using partitioned graphs which have different providers and features.

- JShell
- Java
- Python

JShell

Java

```
SupervisedGraphWiseModel model =
analyst.supervisedGraphWiseModelBuilder()
    .setVertexInputPropertyNames("vertex_provider1_features",
"vertex_provider2_features")
    .setEdgeInputPropertyNames("edge_provider_features")
    .setVertexTargetPropertyName("target_property")
    .build();
```

Python

Also, you can select the providers as shown:



- JShell
- Java
- Python

JShell

Java

```
SupervisedGraphWiseModel model = analyst.supervisedGraphWiseModelBuilder()
    .setVertexInputPropertyNames("vertex_provider1_features",
"vertex_provider2_features")
    .setEdgeInputPropertyNames("edge_provider_features")
    .setVertexTargetPropertyName("target_property")
    .setTargetVertexLabels("provider1")
    .build();
```

Python

If you wish to control the flow of the embeddings at each layer, you can enable or disable the required connections. By default, all the connections are enabled.

- JShell
- Java
- Python



JShell

Java

```
GraphWiseConvLayerConfig convLayerConfig =
analyst.graphWiseConvLayerConfigBuilder()
    .setNumSampledNeighbors(10)
    .useVertexToVertexConnection(true)
    .useEdgeToVertexConnection(true)
    .useEdgeToEdgeConnection(false)
    .useVertexToEdgeConnection(false)
    .build();
SupervisedGraphWiseModel model =
analyst.supervisedGraphWiseModelBuilder()
    .setVertexInputPropertyNames("vertex provider1 features",
"vertex provider2 features")
    .setEdgeInputPropertyNames("edge provider features")
    .setVertexTargetPropertyName("target property")
    .setTargetVertexLabels("provider1")
    .setConvLayerConfigs(convLayerConfig)
    .build();
```

Python



15.2.5 Classification Versus Regression Models on Supervised GraphWise Models

When predicting a property, the loss function defines if the model will perform classification tasks or regression tasks.

For classification tasks, the Supervised GraphWise model will infer labels. Even if the property is a number, the model will assign one label for each value found and classify on it. The possible losses for classification tasks are softmax cross entropy, sigmoid cross entropy, and DevNet loss.

For regression tasks, the Supervised GraphWise model will infer values for the property. The loss for regression tasks is the ${\tt MSE}$ loss.

- JShell
- Java
- Python

JShell

```
opg4j> import oracle.pgx.config.mllib.loss.LossFunctions
opg4j> var model = analyst.supervisedGraphWiseModelBuilder().
    setVertexInputPropertyNames("vertex_features").
    setEdgeInputPropertyNames("edge_features").
    setVertexTargetPropertyName("scores").
    setConvLayerConfigs(convLayerConfig).
    setPredictionLayerConfigs(predictionLayerConfig).
    setLossFunction(LossFunctions.MSELoss()).
    setBatchGenerator(BatchGenerators.STRATIFIED_OVERSAMPLING).
    build()
```



```
.setVertexTargetPropertyName("scores")
.setConvLayerConfigs(convLayerConfig)
.setPredictionLayerConfigs(predictionLayerConfig)
.setLossFunction(LossFunctions.MSELoss())
.setBatchGenerator(BatchGenerators.STRATIFIED_OVERSAMPLING)
.build();
```

15.2.6 Setting a Custom Loss Function and Batch Generator (for Anomaly Detection)

It is possible to select different loss functions for the supervised model by providing a LossFunction object, and different batch generators by providing a BatchGenerator object. This is useful for applications such as Anomaly Detection, which can be cast into the standard supervised framework but require different loss functions and batch generators.

SupervisedGraphWise model can use the DevNetLoss and the StratifiedOversamplingBatchGenerator. The DevNetLoss takes confidence margin and the value the anomaly takes in the target property as the two parameters.

The following example assumes that the <code>convLayerConfig</code> has already been defined:

- JShell
- Java
- Python

```
opg4j> import oracle.pgx.config.mllib.loss.LossFunctions
opg4j> import oracle.pgx.config.mllib.batchgenerator.BatchGenerators
opg4j> var predictionLayerConfig =
analyst.graphWisePredictionLayerConfigBuilder().
```



```
setHiddenDimension(32).
    setActivationFunction(ActivationFunction.LINEAR).
    build()

opg4j> var model = analyst.supervisedGraphWiseModelBuilder().
    setVertexInputPropertyNames("vertex_features").
    setEdgeInputPropertyNames("edge_features").
    setVertexTargetPropertyName("labels").
    setConvLayerConfigs(convLayerConfig).
    setPredictionLayerConfigs(predictionLayerConfig).
    setLossFunction(LossFunctions.devNetLoss(5.0, true)).
    setBatchGenerator(BatchGenerators.STRATIFIED_OVERSAMPLING).
    build()
```

```
import oracle.pgx.config.mllib.loss.LossFunctions;
import oracle.pgx.config.mllib.batchgenerator.BatchGenerators;
GraphWisePredictionLayerConfig predictionLayerConfig =
analyst.graphWisePredictionLayerConfigBuilder()
    .setHiddenDimension(32)
    .setActivationFunction(ActivationFunction.LINEAR)
    .build();
SupervisedGraphWiseModel model = analyst.supervisedGraphWiseModelBuilder()
    .setVertexInputPropertyNames("vertex features")
    .setEdgeInputPropertyNames("edge features")
    .setVertexTargetPropertyName("labels")
    .setConvLayerConfigs(convLayerConfig)
    .setPredictionLayerConfigs (predictionLayerConfig)
    .setLossFunction(LossFunctions.devNetLoss(5.0, true))
    .setBatchGenerator(BatchGenerators.STRATIFIED OVERSAMPLING)
    .build();
```

Python

15.2.7 Training a Supervised GraphWise Model

You can train a Supervised GraphWise model on a graph as described in the following code:

- JShell
- Java
- Python

JShell

```
opg4j> model.fit(trainGraph)
```

Java

```
model.fit(trainGraph);
```

Python

model.fit(train_graph)

15.2.8 Getting the Loss Value For a Supervised GraphWise Model

You can fetch the training loss value as described in the following code:

- JShell
- Java
- Python

JShell

```
opg4j> var loss = model.getTrainingLoss()
```

```
double loss = model.getTrainingLoss();
```



```
loss = model.get_training_loss()
```

15.2.9 Inferring the Vertex Labels for a Supervised GraphWise Model

You can infer the labels for vertices on any graph (including vertices or graphs that were not seen during training) as described in the following code:

- JShell
- Java
- Python

JShell

```
opg4j> var labels = model.inferLabels(fullGraph, testVertices)
opg4j> labels.head().print()
```

Java

```
PgxFrame labels = model.inferLabels(fullGraph,testVertices);
labels.head().print();
```

Python

```
labels = model.infer_labels(full_graph, test_vertices)
labels.print()
```

The output will be similar to the following example output:

+-	vertexId		label
	2		Neural Networks
	6		Theory
	7		Case Based
	22		Rule Learning
	30		Theory
	34		Neural Networks
	47		Case Based
	48		Probabalistic Methods



Similarly, you can also get the model confidence for each class by inferring the prediction logits as described in the following code:

- JShell
- Java
- Python

JShell

```
opg4j> var logits = model.inferLogits(fullGraph, testVertices)
opg4j> labels.head().print()
```

Java

```
PgxFrame logits = model.inferLogits(fullGraph,testVertices);
logits.head().print();
```

Python

```
logits = model.infer_logits(full_graph, test_vertices)
logits.print()
```

15.2.10 Evaluating the Supervised GraphWise Model Performance

You can evaluate various classification metrics for the model using the evaluateLabels method as described in the following code:

- JShell
- Java
- Python

```
opg4j> model.evaluateLabels(fullGraph, testVertices).print()
```



```
model.evaluateLabels(fullGraph, testVertices).print();
```

Python

```
model.evaluate_labels(full_graph, test_vertices).print()
```

The output will be similar to the following example output:

15.2.11 Inferring Embeddings for a Supervised GraphWise Model

You can use a trained model to infer embeddings for unseen nodes and store in the database as described in the following code:

- JShell
- Java
- Python

JShell

```
opg4j> var vertexVectors = model.inferEmbeddings(fullGraph,
testVertices).flattenAll()
opg4j> vertexVectors.write().
   db().
   name("vertex vectors").
   tablename("vertexVectors").
   overwrite(true).
   store()
```

```
PgxFrame vertexVectors =
model.inferEmbeddings(fullGraph, testVertices).flattenAll();
vertexVectors.write()
   .db()
   .name("vertex vectors")
   .tablename("vertexVectors")
```



```
.overwrite(true)
.store();
```

```
vertex_vectors = model.infer_embeddings(full_graph,
test_vertices).flatten_all()
vertex_vectors.write().db().table_name("table_name").name("vertex_vecto
rs").overwrite(True).store()
```

The schema for the vertexVectors will be as follows without flattening (flattenAll splits the vector column into separate double-valued columns):



All the preceding examples assume that you are inferring the embeddings for a model in the current logged in database. If you must infer embeddings for the model in a different database then refer to the examples in Inferring Embeddings for a Model in Another Database.

Inferring Embeddings for a Model in Another Database

15.2.11.1 Inferring Embeddings for a Model in Another Database

You can infer embeddings on a trained model and store in a different database other than the one used for login.

The following code shows how to infer embeddings and store in a different database:

- JShell
- Java
- Python



```
PgxFrame vertexVectors =
model.inferEmbeddings(fullGraph, testVertices).flattenAll();
vertexVectors.write()
    .db()
    .username("user")
                                         // DB user to use for storing the
model
    .password("password")
                                         // password of the DB user
    .jdbcUrl("jdbcUrl")
                                        // jdbc url to the DB
    .name("vertex vectors")
    .tablename("vertexVectors")
                                         // indicates the name of the table
in which the data should be stored
    .overwrite(true)
    .store();
```

Python

15.2.12 Storing a Trained Supervised GraphWise Model

You can store models in database. The models get stored as a row inside a model store table.

The following code shows how to store a trained Supervised GraphWise model in database in a specific model store table:

- JShell
- Java



JShell

Java

Python

Note:

All the preceding examples assume that you are storing the model in the current logged in database. If you must store the model in a different database then refer to the examples in Storing a Trained Model in Another Database.

15.2.13 Loading a Pre-Trained Supervised GraphWise Model

You can load models from a database.

You can load a pre-trained Supervised GraphWise model from a model store table in database as described in the following code:



- JShell
- Java
- Python

JShell

Java

```
SupervisedGraphWiseModel model = analyst.loadSupervisedGraphWiseModel().db()
    .modelstore("modeltablename") // name of the model store table
    .modelname("model") // model name (primary key of model store table)
    .load();
```

Python

Note:

All the preceding examples assume that you are loading the model from the current logged in database. If you must load the model from a different database then refer to the examples in Loading a Pre-Trained Model From Another Database.

15.2.14 Destroying a Supervised GraphWise Model

You can destroy a GraphWise model as described in the following code:

- JShell
- Java
- Python



JShell

```
opg4j> model.destroy()
```

Java

model.destroy();

Python

model.destroy()

15.2.15 Explaining a Prediction of a Supervised GraphWise Model

In order to understand which features and vertices are important for a prediction of the Supervised GraphWise model, you can generate a SupervisedGnnExplanation using a technique similar to the GNNExplainer by Ying et al.

The explanation holds information related to:

- **Graph structure**: An importance score for each vertex
- Features: An importance score for each graph property



The vertex being explained is always assigned importance 1. Further, the feature importances are scaled such that the most important feature has importance 1.

Additionally, an <code>SupervisedGnnExplanation</code> contains the inferred embeddings, logits, and label. You can get explanations for a model's predictions by using the <code>SupervisedGnnExplainer</code> object. The object can be obtained using the <code>gnnExplainer</code> method. After obtaining the <code>SupervisedGnnExplainer</code> object, you can use the <code>inferAndExplain</code> method to request an explanation for a vertex.

The parameters of the explainer can be configured while the explainer is being created or afterwards using the relevant setter functions. The configurable parameters for the <code>SupervisedGnnExplainer</code> are as follows:

- numOptimizationSteps: Number of optimization steps used by the explainer.
- learningRate: Learning rate of the explainer.
- marginalize: Determines if the explainer loss is marginalized over features. This can help in cases where there are important features that take values close to zero. Without marginalization the explainer can learn to mask such features out even if they are important. Marginalization solves this by learning a mask for the deviation from the estimated input distribution.



Note that, in order to achieve best results, the features should be centered around 0.

For example, assume a simple graph that contains a feature that correlates with the label and another feature that does not. It is therefore expected that the importance of the features to differ significantly (with the feature correlating with the label being more important), while structural importance does not play a big role. In this case, you can generate an explanation as shown:

- JShell
- Java
- Python

```
opg4j> var simpleGraph = session.createGraphBuilder().
                          addVertex(0).setProperty("label feature",
0.5).setProperty("const feature", 0.5).
                          setProperty("label", true).
                          addVertex(1).setProperty("label_feature",
-0.5).setProperty("const_feature", 0.5).
                          setProperty("label", false).
                          addEdge(0, 1).build()
// build and train a Supervised GraphWise model as explained in Advanced
Hyperparameter Customization
// obtain and configure GnnExplainer
var explainer = model.gnnExplainer().learningRate(0.05)
explainer.numOptimizationSteps(200)
// explain prediction of vertex 0
opg4j> var explanation = explainer.inferAndExplain(simpleGraph,
simpleGraph.getVertex(0))
// if you used the devNet loss, you can add the decision threshold as an
extra parameter:
// var explanation = explainer.inferAndExplain(simpleGraph,
simpleGraph.getVertex(0), 6f)
opg4j> var constProperty = simpleGraph.getVertexProperty("const feature")
opg4j> var labelProperty = simpleGraph.getVertexProperty("label feature")
// retrieve feature importances
opg4j> var featureImportances = explanation.getVertexFeatureImportance()
opg4j> var importanceConstProp = featureImportances.get(constProperty) //
small as unimportant
opg4j> var importanceLabelProp = featureImportances.get(labelProperty) //
large (1) as important
// retrieve computation graph with importances
opg4j> var importanceGraph = explanation.getImportanceGraph()
```



```
// retrieve importance of vertices
opg4j> var importanceProperty =
explanation.getVertexImportanceProperty()
opg4j> var importanceVertex0 = importanceProperty.get(0) // has
importance 1
opg4j> var importanceVertex1 = importanceProperty.get(1) // available
if vertex 1 part of computation
```

```
PgxGraph simpleGraph = session.createGraphBuilder()
    .addVertex(0).setProperty("label feature",
0.5).setProperty("const feature", 0.5)
    .setProperty("label", true)
    .addVertex(1).setProperty("label feature",
-0.5).setProperty("const feature", 0.5)
    .setProperty("label", false)
    .addEdge(0, 1).build();
// build and train a Supervised GraphWise model as explained in Advanced
Hyperparameter Customization
// obtain and configure the explainer
SupervisedGnnExplainerexplainer=model.gnnExplainer().learningRate(0.05)
explainer.numOptimizationSteps(200);
// explain prediction of vertex 0
SupervisedGnnExplanation<Integer> explanation =
explainer.inferAndExplain(simpleGraph,
    simpleGraph.getVertex(0));
// if we used the devNet loss, we can add the decision threshold as an
extra parameter:
// SupervisedGnnExplanation<Integer> explanation =
explainer.inferAndExplain(simpleGraph, simpleGraph.getVertex(0), 6f);
VertexProperty<Integer, Float> constProperty =
simpleGraph.getVertexProperty("const feature");
VertexProperty<Integer, Float> labelProperty =
simpleGraph.getVertexProperty("label feature");
// retrieve feature importances
Map<VertexProperty<Integer, ?>, Float> featureImportances =
explanation.getVertexFeatureImportance();
float importanceConstProp = featureImportances.get(constProperty); //
small as unimportant
float importanceLabelProp = featureImportances.get(labelProperty); //
large (1) as important
// retrieve computation graph with importances
PgxGraph importanceGraph = explanation.getImportanceGraph();
```



```
// retrieve importance of vertices
VertexProperty<Integer, Float> importanceProperty =
explanation.getVertexImportanceProperty();
float importanceVertex0 = importanceProperty.get(0); // has importance 1
float importanceVertex1 = importanceProperty.get(1); // available if vertex
1 part of computation
Python
simple graph = session.create graph builder()
    .add vertex(0).set property("label feature",
0.5).set property("const feature", 0.5)
    .set property("label", true)
    .add vertex(1).set property("label feature",
-0.5).set property("const feature", 0.5)
    .set property("label", false)
    .add edge(0, 1).build()
# build and train a Supervised GraphWise model as explained in Advanced
Hyperparameter Customization
# obtain the explainer
explainer = model.gnn explainer(learning rate=0.05)
explainer.num optimization steps=200
# explain prediction of vertex 0
explanation =
explainer.inferAndExplain(simple graph, simple graph.get vertex(0))
# if we used the devNet loss, we can add the decision threshold as an extra
parameter:
# explanation = explainer.inferAndExplain(simple graph,
simple graph.get vertex(0), 6)
const property = simple graph.get vertex property("const feature")
label property = simple graph.get vertex property("label feature")
# retrieve feature importances
feature importances = explanation.get vertex feature importance()
importance const prop = feature importances[const property]
importance label prop = feature importances[label property]
# retrieve computation graph with importances
importance graph = explanation.get importance graph()
# retrieve importance of vertices
importance property = explanation.get vertex importance property()
importance vertex 0 = importance property[0]
```

importance vertex 1 = importance property[1]



See Also:

- Building a Minimal GraphWise Model
- Training a Supervised GraphWise Model

15.3 Using the Unsupervised GraphWise Algorithm

Unsupervised GraphWise is an unsupervised inductive vertex representation learning algorithm which is able to leverage vertex information. The learned embeddings can be used in various downstream tasks including vertex classification, vertex clustering and similar vertex search.

Unsupervised GraphWise is based on Deep Graph Infomax (DGI) by Velickovic et al.

Model Structure

A Unsupervised GraphWise model consists of graph convolutional layers followed by a DGI Layer.

The forward pass through a convolutional layer for a vertex proceeds as follows:

- A set of neighbors of the vertex is sampled.
- 2. The previous layer representations of the neighbors are mean-aggregated, and the aggregated features are concatenated with the previous layer representation of the vertex.
- 3. This concatenated vector is multiplied with weights, and a bias vector is added.
- 4. The result is normalized to such that the layer output has unit norm.

The DGI Layer consists of three parts enabling unsupervised learning using embeddings produced by the convolution layers.

- **1. Corruption function:** Shuffles the node features while preserving the graph structure to produce negative embedding samples using the convolution layers.
- Readout function: Sigmoid activated mean of embeddings, used as summary of a graph.
- 3. **Discriminator:** Measures the similarity of positive (unshuffled) embeddings with the summary as well as the similarity of negative samples with the summary from which the loss function is computed.

Since none of these contains mutable hyperparameters, the default DGI layer is always used and cannot be adjusted.

The following describes the usage of the main functionalities of the implementation of pgi in PGX using the Cora graph as an example:

- Loading a Graph
- Building a Minimal Unsupervised GraphWise Model
- Advanced Hyperparameter Customization
- Building an Unsupervised GraphWise Model Using Heterogeneous Graphs
- Training an Unsupervised GraphWise Model



- Getting the Loss Value for an Unsupervised GraphWise Model
- Inferring Embeddings for an Unsupervised GraphWise Model
- Storing an Unsupervised GraphWise Model
- Loading a Pre-Trained Unsupervised GraphWise Model
- Destroying an Unsupervised GraphWise Model
- Explaining a Prediction for an Unsupervised GraphWise Model

15.3.1 Loading a Graph

The following describes the steps for loading a graph:

- 1. Create a Session and an Analyst.
 - JShell
 - Java
 - Python

JShell

```
cd /opt/oracle/graph/
./bin/opg4j
// starting the shell will create an implicit session and analyst
opg4j> import oracle.pgx.config.mllib.ActivationFunction
opg4j> import oracle.pgx.config.mllib.WeightInitScheme
opg4j> PgxSession session = Pgx.createSession("my-session")
opg4j> Analyst analyst = session.createAnalyst()
```

Java

```
import oracle.pgx.api.*;
import oracle.pgx.api.mllib.UnsupervisedGraphWiseModel;
import oracle.pgx.api.frames.*;
import oracle.pgx.config.mllib.ActivationFunction;
import oracle.pgx.config.mllib.GraphWiseConvLayerConfig;
import oracle.pgx.config.mllib.UnsupervisedGraphWiseModelConfig;
import oracle.pgx.config.mllib.WeightInitScheme;

PgxSession session = Pgx.createSession("my-session");
Analyst analyst = session.createAnalyst();
```

Python

```
session = pypgx.get_session()
analyst = session.analyst
```



2. Load the graph.

- JShell
- Java
- Python

JShell

```
opg4j> var graph = session.readGraphWithProperties("<path/to/
graph config.json>")
```

Java

```
PgxGraph graph = session.readGraphWithProperties("<path/to/
graph config.json>");
```

Python

```
graph = session.read_graph_with_properties("<path/to/
graph config.json>")
```

You do not need to use a test graph or test vertices, since the model is trained to be unsupervised.

15.3.2 Building a Minimal Unsupervised GraphWise Model

You can build an Unsupervised GraphWise model with only vertex properties, or only edge properties or both using the minimal configuration and default hyper-parameters.

- JShell
- Java
- Python



Python

```
model =
analyst.unsupervised_graphwise_builder(vertex_input_property_names=["features"])
```

15.3.3 Advanced Hyperparameter Customization

You can build an Unsupervised GraphWise model with only vertex properties or only edge properties or both using rich hyperparameter customization.

This is implemented using the sub-config class, GraphWiseConvLayerConfig.

The following code describes the implementation of the configuration in a Unsupervised GraphWise model. The example also specifies a weight decay parameter of 0.001 and dropout with dropping probability 0.5 for the model to counteract overfitting.

- JShell
- Java
- Python



```
setConvLayerConfigs(convLayerConfig).
setDgiLayerConfig(dgiLayerConfig).

setLossFunction(UnsupervisedGraphWiseModelConfig.LossFunction.SIGMOID_C
ROSS_ENTROPY).

setEmbeddingDim(256).
setLearningRate(0.05).
setNumEpochs(30).
setSeed(42).
setShuffle(false).
setStandardize(true).
setBatchSize(64).
build()
```

```
String weightProperty = analyst.pagerank(trainGraph).getName();
GraphWiseConvLayerConfig convLayerConfig =
analyst.graphWiseConvLayerConfigBuilder()
    .setNumSampledNeighbors(25)
    .setActivationFunction(ActivationFunction.TANH)
    .setWeightInitScheme(WeightInitScheme.XAVIER)
    .setWeightedAggregationProperty(weightProperty)
    .setDropoutRate(0.5)
    .build();
GraphWiseDgiLayerConfig dgiLayerConfig =
analyst.graphWiseDgiLayerConfigBuilder()
    .setCorruptionFunction(new PermutationCorruption())
    .setDiscriminator(GraphWiseDgiLayerConfig.Discriminator.BILINEAR)
    .setReadoutFunction(GraphWiseDqiLayerConfig.ReadoutFunction.MEAN)
    .build();
UnsupervisedGraphWiseModel model =
analyst.unsupervisedGraphWiseModelBuilder()
    .setVertexInputPropertyNames("vertex features")
    .setEdgeInputPropertyNames("edge features")
    .setDgiLayerConfig(dgiLayerConfig)
    .setLossFunction(UnsupervisedGraphWiseModelConfig.LossFunction.SIGM
OID CROSS ENTROPY)
    .setConvLayerConfigs(convLayerConfig)
    .setWeightDecay(0.001)
    .setEmbeddingDim(256)
    .setLearningRate(0.05)
    .setNumEpochs(30)
    .setSeed(42)
    .setShuffle(false)
    .setStandardize(true)
    .setBatchSize(64)
    .build();
```



```
weightProperty = analyst.pagerank(train graph).name
conv layer config = dict(num sampled neighbors=25,
                         activation fn='tanh',
                         weight init scheme='xavier',
                         neighbor weight property name=weightProperty,
                         dropout rate=0.5)
conv layer = analyst.graphwise conv layer config(**conv layer config)
dgi layer config = dict(corruption function=None,
                        readout function="mean",
                        discriminator="bilinear")
dgi_layer = analyst.graphwise_dgi_layer_config(**dgi_layer_config)
params = dict(conv layer config=[conv layer],
              dgi layer config=dgi layer,
              loss fn="sigmoid cross entropy",
              vertex input property names=["vertex features"],
              edge input property names=["edge features"],
              weight decay=0.001,
              layer size=256,
              learning rate=0.05,
              num epochs=30,
              seed=42,
              standardize=true,
              batch size=64
)
model = analyst.unsupervised graphwise builder(**params)
```

See UnsupervisedGraphWiseModelBuilder and GraphWiseConvLayerConfigBuilder in Javadoc for full description of all available hyperparameters and their default values.

15.3.4 Building an Unsupervised GraphWise Model Using Heterogeneous Graphs

You can build an Unsupervised GraphWise model using partitioned graphs which have different providers and features.

- JShell
- Java
- Python



JShell

Java

```
UnsupervisedGraphWiseModel model =
analyst.unsupervisedGraphWiseModelBuilder()
    .setVertexInputPropertyNames("vertex_provider1_features",
"vertex_provider2_features")
    .setEdgeInputPropertyNames("edge_provider_features")
    .setVertexTargetPropertyName("target_property")
    .build();
```

Python

Also, you can select specific providers as shown:

- JShell
- Java
- Python



```
UnsupervisedGraphWiseModel model =
analyst.unsupervisedGraphWiseModelBuilder()
    .setVertexInputPropertyNames("vertex_provider1_features",
"vertex_provider2_features")
    .setEdgeInputPropertyNames("edge_provider_features")
    .setTargetVertexLabels("provider1")
    .build();
```

Python

If you wish to control the flow of the embeddings at each layer, you can enable or disable the required connections. By default, all the connections are enabled.

- JShell
- Java
- Python

JShell

Java

```
GraphWiseConvLayerConfig convLayerConfig =
analyst.graphWiseConvLayerConfigBuilder()
    .setNumSampledNeighbors(10)
```



```
.useVertexToVertexConnection(true)
.useEdgeToVertexConnection(true)
.useEdgeToEdgeConnection(false)
.useVertexToEdgeConnection(false)
.build();

UnsupervisedGraphWiseModel model =
analyst.unsupervisedGraphWiseModelBuilder()
.setVertexInputPropertyNames("vertex_provider1_features",
"vertex_provider2_features")
.setEdgeInputPropertyNames("edge_provider_features")
.setTargetVertexLabels("provider1")
.setConvLayerConfigs(convLayerConfig)
.build();
```

```
conv layer config = dict(num sampled neighbors=25,
                         activation fn='tanh',
                         weight init scheme='xavier',
                         neighbor weight property name=weightProperty,
                         vertex to vertex connection=True,
                         edge_to_vertex_connection=True,
                         vertex to edge connection=False,
                         edge to edge connection=False)
conv_layer = analyst.graphwise_conv_layer_config(**conv_layer_config)
params =
dict(vertex input property names=["vertex provider1 features",
"vertex_provider2 features"],
              edge_input_property_names=["edge_provider_features"],
              target vertex labels=["provider1"],
              conv_layer_config=[conv_layer])
model = analyst.unsupervised graphwise builder(**params)
```

15.3.5 Training an Unsupervised GraphWise Model

You can train an Unsupervised GraphWise model on a graph as shown:

- JShell
- Java
- Python



```
opg4j> model.fit(trainGraph)
```

Java

```
model.fit(trainGraph);
```

Python

model.fit(train graph)

15.3.6 Getting the Loss Value for an Unsupervised GraphWise Model

You can fetch the training loss value for an Unsupervised GraphWise Model as shown in the following code:

- JShell
- Java
- Python

JShell

```
opg4j> var loss = model.getTrainingLoss()
```

Java

```
double loss = model.getTrainingLoss();
```

Python

loss = model.get_training_loss()

15.3.7 Inferring Embeddings for an Unsupervised GraphWise Model

You can use a trained model to infer embeddings for unseen nodes and store them in the database as described in the following code:



- JShell
- Java
- Python

```
opg4j> var vertexVectors = model.inferEmbeddings(fullGraph,
fullGraph.getVertices()).flattenAll()
opg4j> vertexVectors.write().
   db().
   name("vertex vectors").
   tablename("vertexVectors").
   overwrite(true).
   store()
```

Java

```
PgxFrame vertexVectors =
model.inferEmbeddings(fullGraph, fullGraph.getVertices()).flattenAll();
vertexVectors.write()
   .db()
   .name("vertex vectors")
   .tablename("vertexVectors")
   .overwrite(true)
   .store();
```

Python

```
vertex_vectors =
model.infer_embeddings(full_Graph,full_Graph.get_vertices()).flattenAl
l()
vertex_vectors.write().db().table_name("table_name").name("vertex_vecto
rs").overwrite(True).store()
```

The schema for the vertexVectors will be as follows without flattening (flattenAll splits the vector column into separate double-valued columns):





All the preceding examples assume that you are inferring the embeddings for a model in the current logged in database. If you must infer embeddings for the model in a different database then refer to the examples in Inferring Embeddings for a Model in Another Database.

15.3.8 Storing an Unsupervised GraphWise Model

You can store models in database. The models get stored as a row inside a model store table.

- JShell
- Java
- Python

JShell

Java

Python





All the preceding examples assume that you are storing the model in the current logged in database. If you must store the model in a different database then refer to the examples in Storing a Trained Model in Another Database.

15.3.9 Loading a Pre-Trained Unsupervised GraphWise Model

You can load models from a database.

- JShell
- Java
- Python

JShell

Java

```
UnsupervisedGraphWiseModel model =
analyst.loadUnsupervisedGraphWiseModel().db()
    .modelstore("modeltablename") // name of the model store table
    .modelname("model") // model name (primary key of model
store table)
    .load();
```

Python





All the preceding examples assume that you are loading the model from the current logged in database. If you must load the model from a different database then refer to the examples in Loading a Pre-Trained Model From Another Database.

15.3.10 Destroying an Unsupervised GraphWise Model

You can destroy an Unsupervised GraphWise model as described in the following code:

- JShell
- Java
- Python

JShell

```
opg4j> model.destroy()
```

Java

model.destroy();

Python

model.destroy()

15.3.11 Explaining a Prediction for an Unsupervised GraphWise Model

In order to understand which features and vertices are important for a prediction of the Unsupervised GraphWise model, you can generate an UnsupervisedGnnExplanation using a technique similar to the GNNExplainer by Ying et al.

The explanation holds information related to:

- **Graph structure**: An importance score for each vertex
- Features: An importance score for each graph property



The vertex being explained is always assigned importance 1. Further, the feature importances are scaled such that the most important feature has importance 1.



Additionally, an UnsupervisedGnnExplanation contains the inferred embedding. You can get explanations for a model's predictions by using the UnsupervisedGnnExplainer object. The object can be obtained using the gnnExplainer method. After obtaining the UnsupervisedGnnExplainer object, you can use the inferAndExplain method to request an explanation for a vertex.

The parameters of the explainer can be configured while the explainer is being created or afterwards using the relevant setter functions. The configurable parameters for the <code>UnsupervisedGnnExplainer</code> are as follows:

- numOptimizationSteps: Number of optimization steps used by the explainer.
- learningRate: Learning rate of the explainer.
- marginalize: Determines if the explainer loss is marginalized over features. This
 can help in cases where there are important features that take values close to
 zero. Without marginalization the explainer can learn to mask such features out
 even if they are important. Marginalization solves this by learning a mask for the
 deviation from the estimated input distribution.
- numClusters: Number of clusters to use in the explainer loss. The unsupervised explainer uses k-means clustering to compute the explainer loss that is optimized. If the approximate number of components in the graph is known, it is a good idea to set the number of clusters to this number.
- numSamples: Number of vertex samples to use to optimize the explainer. For the sake of performance, the explainer computes the loss on this number of randomly sampled vertices. Using more samples will be more accurate but will take longer and use more resources.

Note that, in order to achieve best results, the features should be centered around 0.

For example, assume a simple graph, componentGraph which contains k densely connect *components*, that is, there are many edges between vertices of the same component and few edges between any two components. By training an Unsupervised GraphWise model on this graph, you can expect a model that produces similar embeddings for vertices in a densely connected component.

The following example shows how to generate an explanation on an inference <code>componentGraph</code>. It is expected that vertices from the same component to have a higher importance than vertices from a different component. Note that the feature importances are not relevant in this example.

- JShell
- Java
- Python

```
opg4j> var componentGraph =
session.readGraphWithProperties("<path_to_component_graph.json>")
// explain prediction of vertex 0
opg4j> var feat1Property = componentGraph.getVertexProperty("feat1")
opg4j> var feat2Property = componentGraph.getVertexProperty("feat2")
```



```
// build and train an Unsupervised GraphWise model as explained in Advanced
Hyperparameter Customization
// obtain and configure the explainer
// setting the numClusters argument to the expected number of clusters may
// explanation results as the explainer optimization will try to cluster
samples into
// this number of clusters
opg4j> var explainer = model.gnnExplainer().numClusters(50)
// set the number of samples to compute the loss over during explainer
optimization
opg4j> explainer.numSamples(10000)
// explain prediction of vertex 0
opg4j> var explanation = explainer.inferAndExplain(componentGraph,
componentGraph.getVertex(0), 10)
// retrieve computation graph with importance
opg4j> var importanceGraph = explanation.getImportanceGraph()
// retrieve importance of vertices
// vertex 1 is in the same densely connected component as vertex 0
// vertex 2 is in a different component
opg4j> var importanceProperty = explanation.getVertexImportanceProperty()
opg4j> var importanceVertex0 = importanceProperty.get(0) // has importance 1
opg4j> var importanceVertex1 = importanceProperty.get(1) // high importance
opq4i> var importanceVertex2 = importanceProperty.get(2) // low importance
opg4j> var featureImportances = explanation.getVertexFeatureImportance()
opg4j> var importanceConstProp = featureImportances.get(constProperty) //
small as unimportant
opq4j> var importanceLabelProp = featureImportances.get(labelProperty) //
large (1) as important
// optionally retrieve feature importance
opg4j> var featureImportances = explanation.getVertexFeatureImportance()
opg4j> var importanceFeat1Prop = featureImportances.get(feat1Property)
opq4j> var importanceFeat2Prop = featureImportances.get(feat2Property)
Java
PgxGraph componentGraph =
session.readGraphWithProperties("<path to component graph.json>") // load
component graph
VertexProperty<Integer, Float> feat1Property =
componentGraph.getVertexProperty("feat1");
VertexProperty<Integer, Float> feat2Property =
componentGraph.getVertexProperty("feat2");
// build and train an Unsupervised GraphWise model as explained in Advanced
Hyperparameter Customization
```

```
// obtain and configure the explainer
// setting the numClusters argument to the expected number of clusters
may improve
// explanation results as the explainer optimization will try to
cluster samples into
// this number of clusters
UnsupervisedGnnExplainer explainer =
model.gnnExplainer().numClusters(50);
// set the number of samples to compute the loss over during explainer
optimization
explainer.numSamples(10000);
// explain prediction of vertex 0
UnsupervisedGnnExplanation<Integer> explanation =
explainer.inferAndExplain(componentGraph, componentGraph.getVertex(0));
// retrieve computation graph with importances
PgxGraph importanceGraph = explanation.getImportanceGraph();
// retrieve importance of vertices
// vertex 1 is in the same densely connected component as vertex 0
// vertex 2 is in a different component
VertexProperty<Integer, Float> importanceProperty =
explanation.getVertexImportanceProperty();
float importanceVertex0 = importanceProperty.get(0); // has
importance 1
float importanceVertex1 = importanceProperty.get(1); // high
importance
float importanceVertex2 = importanceProperty.get(2); // low importance
// retrieve feature importance (not relevant for this example)
Map<VertexProperty<Integer, ?>, Float> featureImportances =
explanation.getVertexFeatureImportance();
float importanceFeat1Prop = featureImportances.get(feat1Property);
float importanceFeat2Prop = featureImportances.get(feat2Property);
Python
# load 'component graph' with vertex features 'feat1' and 'feat2'
feat1 property = component graph.get vertex property("feat1")
feat2 property = component graph.get vertex property("feat2")
# build and train an Unsupervised GraphWise model as explained in
Advanced Hyperparameter Customization
# obtain and configure the explainer
# setting the num clusters argument to the expected number of clusters
may improve
# explanation results as the explainer optimization will try to
cluster samples into
# this number of clusters
explainer = model.gnn explainer(num clusters=50)
# set the number of samples to compute the loss over during explainer
```

```
optimization
explainer.num samples = 10000
# explain prediction of vertex 0
explanation = explainer.infer and explain(
    graph=component graph,
    vertex=component graph.get vertex(0)
)
# retrieve computation graph with importances
importance graph = explanation.get importance graph()
# retrieve importance of vertices
# vertex 1 is in the same densely connected component as vertex 0
# vertex 2 is in a different component
importance property = explanation.get vertex importance property()
importance vertex 0 = importance property[0] # has importance 1
importance vertex 1 = importance property[1] # high importance
importance vertex 2 = importance property[2] # low importance
# retrieve feature importance (not relevant for this example)
feature importances = explanation.get vertex feature importance()
importance feat1 prop = feature importances[feat1 property]
importance feat2 prop = feature importances[feat2 property]
```

See Also:

- Building a Minimal Unsupervised GraphWise Model
- · Training an Unsupervised GraphWise Model

15.4 Using the Supervised EdgeWise Algorithm (Edge Embeddings and Classification)

SupervisedEdgeWise is an inductive edge representation learning algorithm which is able to leverage vertex and edge feature information. It can be applied to a wide variety of tasks, including edge classification and link prediction.

SupervisedEdgeWise is based on top of the GraphWise model, leveraging the source vertex embedding and the destination vertex embedding generated by the GraphWise model to generate inductive edge embeddings.

Model Structure

A SupervisedEdgeWise model consists of graph convolutional layers followed by several prediction layers.



First, the source and destination vertices of the target edge are processed through the convolutional layers. The forward pass through a convolutional layer for a vertex proceeds as follows:

- A set of neighbors of the vertex is sampled.
- 2. The previous layer representations of the neighbors are mean-aggregated, and the aggregated features are concatenated with the previous layer representation of the vertex.
- 3. This concatenated vector is multiplied with weights, and a bias vector is added.
- 4. The result is normalized such that the layer output has unit norm.

The edge embedding layer concatenates the source vertex embedding, the edge features and the destination vertex embedding, and then forwards it through a linear layer to get the edge embedding.

The prediction layers are standard neural network layers.

- Loading a Graph
- Building a Minimal Supervised EdgeWise Model
- Advanced Hyperparameter Customization
- Applying EdgeWise for Partitioned Graphs
- Classification Versus Regression on Supervised EdgeWise Models
- Setting a Custom Loss Function and Batch Generator (for Anomaly Detection)
- Setting the Edge Embedding Production Method
- Training the Supervised EdgeWise Model
- Getting the Loss Value for a Supervised EdgeWise Model
- Inferring Edge Labels for a Supervised EdgeWise Model
- Evaluating Model Performance
- Inferring Embeddings for a Supervised EdgeWise Model
- Storing a Supervised EdgeWise Model
- Loading a Pre-Trained Supervised EdgeWise Model
- Destroying a Supervised EdgeWise Model
- Example: Predicting Ratings on the Movielens Dataset

15.4.1 Loading a Graph

The following describes the steps for loading a graph:

- 1. Create a Session and an Analyst.
 - JShell
 - Java
 - Python



```
cd /opt/oracle/graph/
./bin/opg4j

// starting the shell will create an implicit session and analyst
opg4j> import oracle.pgx.config.mllib.ActivationFunction
opg4j> import oracle.pgx.config.mllib.WeightInitScheme
opg4j> var session = Pgx.createSession("my-session")
opg4j> var analyst = session.createAnalyst()
```

Java

```
import oracle.pgx.api.*;
import oracle.pgx.api.mllib.SupervisedEdgeWiseModel;
import oracle.pgx.api.filter.EdgeFilter;
import oracle.pgx.api.frames.*;
import oracle.pgx.config.mllib.ActivationFunction;
import oracle.pgx.config.mllib.GraphWiseConvLayerConfig;
import oracle.pgx.config.mllib.GraphWisePredictionLayerConfig;
import oracle.pgx.config.mllib.SupervisedEdgeWiseModelConfig;
import oracle.pgx.config.mllib.WeightInitScheme;
PgxSession session = Pgx.createSession("my-session");
Analyst analyst = session.createAnalyst();
```

Python

```
session = pypgx.get_session(session_name="my-session")
analyst = session.create_analyst()
```

2. Load the graph.

- JShell
- Java
- Python



```
filter(e -> !trainGraph.hasEdge(e.getId())).
collect(Collectors.toList())
```

Python

```
from pypgx.api.filters import EdgeFilter
full_graph =
session.read_graph_with_properties("<path_to_movielens.json>")
edge_filter = EdgeFilter.from_pgql_result_set(
    session.query_pgql("SELECT e FROM movielens MATCH (v1) -[e]->
(v2) WHERE ID(e) % 4 > 0"), "e"
)
train_graph = full_graph.filter(edge_filter)
test_edges = []
train_edges = train_graph.get_edges()
for e in full_graph.get_edges():
    if(not train_edges.contains(e)):
        test vertices.append(e)
```

15.4.2 Building a Minimal Supervised EdgeWise Model

You can build an EdgeWise model using the minimal configuration and default hyper-parameters as described in the following code. Note that even though only one feature property is needed (either on vertices with setVertexInputPropertyNames or edges with setEdgeInputPropertyNames) for the model to work, you can specify as many as required.

- JShell
- Java
- Python



Java

```
SupervisedEdgeWiseModel model = analyst.supervisedEdgeWiseModelBuilder()
    .setVertexInputPropertyNames("vertex_features")
    .setEdgeInputPropertyNames("edge_features")
    .setEdgeTargetPropertyName("labels")
    .build();
```

Python

15.4.3 Advanced Hyperparameter Customization

You can build a Supervised EdgeWise model using rich hyperparameter customization. This is implemented using the sub-config classes:

- GraphWiseConvLayerConfig
- GraphWisePredictionLayerConfig

The following code describes the implementation of the configuration in a Supervised EdgeWise model. The example also specifies a weight decay parameter of 0.001 and dropout with dropping probability 0.5 to counteract overfitting.

- JShell
- Java
- Python



```
setWeightInitScheme(WeightInitScheme.XAVIER).
                setWeightedAggregationProperty(weightProperty).
                setDropoutRate(0.5).
                build()
opg4j> var predictionLayerConfig =
analyst.graphWisePredictionLayerConfigBuilder().
         setHiddenDimension(32).
         setActivationFunction(ActivationFunction.RELU).
         setWeightInitScheme(WeightInitScheme.HE).
         setDropoutRate(0.5).
        build()
opg4j> var model = analyst.supervisedEdgeWiseModelBuilder().
         setVertexInputPropertyNames("vertex features").
         setEdgeInputPropertyNames("edge features").
         setEdgeTargetPropertyName("labels").
         setConvLayerConfigs(convLayerConfig).
         setPredictionLayerConfigs(predictionLayerConfig).
         setWeightDecay(0.001).
        build()
```

```
String weightProperty = analyst.pagerank(trainGraph).getName();
GraphWiseConvLayerConfig convLayerConfig =
analyst.graphWiseConvLayerConfigBuilder()
    .setNumSampledNeighbors(25)
    .setActivationFunction(ActivationFunction.TANH)
    .setWeightInitScheme(WeightInitScheme.XAVIER)
    .setWeightedAggregationProperty(weightProperty)
    .setDropoutRate(0.5)
    .build();
GraphWisePredictionLayerConfig predictionLayerConfig =
analyst.graphWisePredictionLayerConfigBuilder()
    .setHiddenDimension(32)
    .setActivationFunction(ActivationFunction.RELU)
    .setWeightInitScheme (WeightInitScheme.HE)
    .setDropoutRate(0.5)
    .build();
SupervisedEdgeWiseModel model =
analyst.supervisedEdgeWiseModelBuilder()
    .setVertexInputPropertyNames("vertex features")
    .setEdgeInputPropertyNames("edge features")
    .setEdgeTargetPropertyName("labels")
    .setConvLayerConfigs(convLayerConfig)
    .setPredictionLayerConfigs(predictionLayerConfig)
    .setWeightDecay(0.001)
    .build();
```



```
weightProperty = analyst.pagerank(train graph).name
conv layer config = dict(num sampled neighbors=25,
                         activation fn='tanh',
                         weight init scheme='xavier',
                         neighbor weight property name=weightProperty,
                         dropout rate=0.5)
conv layer = analyst.graphwise conv layer config(**conv layer config)
pred layer config = dict(hidden dim=32,
                         activation fn='relu',
                         weight init scheme='he',
                         dropout rate=0.5)
pred layer = analyst.graphwise pred layer config(**pred layer config)
params = dict(edge_target_property_name="labels",
              conv layer config=[conv layer],
              pred layer config=[pred layer],
              vertex input property names=["vertex_features"],
              edge input property names=["edge features"],
              seed=17,
              weight decay=0.001)
model = analyst.supervised edgewise builder(**params)
```

15.4.4 Applying EdgeWise for Partitioned Graphs

You can apply EdgeWise on partitioned graphs, where you have different providers and different features.

- JShell
- Java
- Python



```
SupervisedEdgeWiseModel model =
analyst.supervisedEdgeWiseModelBuilder()
    .setVertexInputPropertyNames("vertex_provider_features")
    .setEdgeInputPropertyNames("edge_provider1_features",
"edge_provider2_features")
    .setEdgeTargetPropertyName("target_property")
    .build();
```

Python

You can select which providers you want to train or infer on:

- JShell
- Java
- Python

JShell

Java

```
SupervisedEdgeWiseModel model =
analyst.supervisedEdgeWiseModelBuilder()
    .setVertexInputPropertyNames("vertex_provider_features")
    .setEdgeInputPropertyNames("edge_provider1_features",
"edge_provider2_features")
    .setEdgeTargetPropertyName("target_property")
    .setTargetEdgeLabels("provider1")
    .build();
```



If you wish to control the flow of the embeddings at each graph convolutional layer of the underlying Graphwise model, then you can enable or disable the connections of interest. By default, all the connections are enabled.

- JShell
- Java
- Python

JShell

Java

```
GraphWiseConvLayerConfig convLayerConfig =
analyst.graphWiseConvLayerConfigBuilder()
    .setNumSampledNeighbors(10)
    .useVertexToVertexConnection(true)
    .useEdgeToVertexConnection(true)
    .useEdgeToEdgeConnection(false)
    .useVertexToEdgeConnection(false)
    .build();
SupervisedEdgeWiseModel model = analyst.supervisedEdgeWiseModelBuilder()
```



```
.setVertexInputPropertyNames("vertex_provider1_features",
"vertex_provider2_features")
    .setEdgeInputPropertyNames("edge_provider_features")
    .setEdgeTargetPropertyName("target_property")
    .setTargetEdgeLabels("provider1")
    .setConvLayerConfigs(convLayerConfig)
    .build();
```

```
conv layer config = dict(num sampled neighbors=25,
                         activation fn='tanh',
                         weight init scheme='xavier',
                         neighbor weight property name=weightProperty,
                         vertex to vertex connection=True,
                         edge to vertex connection=True,
                         vertex to edge connection=False,
                         edge to edge connection=False)
conv layer = analyst.graphwise conv layer config(**conv layer config)
params = dict(edge target property name="target property",
vertex input property names=["vertex provider1 features",
"vertex provider2 features"],
              edge_input_property_names=["edge provider features"],
              target edge labels=["provider1"],
              conv layer config=[conv layer])
model = analyst.supervised edgewise builder(**params)
```

15.4.5 Classification Versus Regression on Supervised EdgeWise Models

When predicting a property, the loss function defines if the model will perform classification tasks or regression tasks.

For classification tasks, the Supervised EdgeWise model will infer labels. Even if this property is a number, the model will assign one label for each value found and classify on it. The possible losses for classification tasks are softmax cross entropy, sigmoid cross entropy, and DevNet loss.

For regression tasks, the Supervised EdgeWise model will infer values for the property. The loss for regression tasks is the ${\tt MSE}$ loss.

It is possible to select different loss functions for the supervised model by providing a LossFunction object.



- JShell
- Java
- Python

Java

```
import oracle.pgx.config.mllib.loss.LossFunctions;

SupervisedEdgeWiseModel model = analyst.supervisedEdgeWiseModelBuilder()
    .setVertexInputPropertyNames("vertex_features")
    .setEdgeInputPropertyNames("edge_features")
    .setEdgeTargetPropertyName("labels")
    .setLossFunction(LossFunctions.MSE_LOSS)
    .build();
```

Python

15.4.6 Setting a Custom Loss Function and Batch Generator (for Anomaly Detection)

In addition to different loss functions, it is also possible to select different batch generators by providing a batch generator type. This is useful for applications such as Anomaly Detection, which can be cast into the standard supervised framework but require different loss functions and batch generators.

SupervisedEdgeWise model can use the DevNetLoss and the StratifiedOversamplingBatchGenerator. DevNetLoss takes confidence margin and the value the anomaly takes in the target property as the two parameters.



The following example assumes that the <code>convLayerConfig</code> has already been defined:

- JShell
- Java
- Python

JShell

```
opg4j> import oracle.pgx.config.mllib.loss.LossFunctions
opg4j> import oracle.pgx.config.mllib.batchgenerator.BatchGenerators
opg4j> var predictionLayerConfig =
analyst.graphWisePredictionLayerConfigBuilder().
         setHiddenDimension(32).
         setActivationFunction(ActivationFunction.LINEAR).
         build()
opg4j> var model = analyst.supervisedEdgeWiseModelBuilder().
         setVertexInputPropertyNames("vertex features").
         setEdgeInputPropertyNames("edge features").
         setEdgeTargetPropertyName("labels").
         setConvLayerConfigs(convLayerConfig).
         setPredictionLayerConfigs(predictionLayerConfig).
         setLossFunction(LossFunctions.devNetLoss(5.0, true)).
         setBatchGenerator(BatchGenerators.STRATIFIED OVERSAMPLING).
         build()
```

Java

```
import oracle.pgx.config.mllib.loss.LossFunctions;
import oracle.pgx.config.mllib.batchgenerator.BatchGenerators;
GraphWisePredictionLayerConfig predictionLayerConfig =
analyst.graphWisePredictionLayerConfigBuilder()
    .setHiddenDimension(32)
    .setActivationFunction(ActivationFunction.LINEAR)
    .build();
SupervisedEdgeWiseModel model =
analyst.supervisedEdgeWiseModelBuilder()
    .setVertexInputPropertyNames("vertex features")
    .setEdgeInputPropertyNames("edge features")
    .setEdgeTargetPropertyName("labels")
    .setConvLayerConfigs(convLayerConfig)
    .setPredictionLayerConfigs(predictionLayerConfig)
    .setLossFunction(LossFunctions.devNetLoss(5.0, true))
    .setBatchGenerator(BatchGenerators.STRATIFIED OVERSAMPLING)
    .build();
```



15.4.7 Setting the Edge Embedding Production Method

By default, the edge embedding is computed by combining the source vertex embedding, the destination vertex embedding and the edge features. You can manually set these by setting the EdgeCombinationMethod with booleans parameters:

- JShell
- Java
- Python



```
import oracle.pgx.config.mllib.edgecombination.EdgeCombinationMethod;
import oracle.pgx.config.mllib.edgecombination.EdgeCombinationMethods;

EdgeCombinationMethod method =
    EdgeCombinationMethods.concatEdgeCombinationMethod(useSourceVertex,
    useDestinationVertex, useEdge);

SupervisedEdgeWiseModel model =
    analyst.supervisedEdgeWiseModelBuilder()
        .setVertexInputPropertyNames("vertex_features")
        .setEdgeInputPropertyNames("edge_features")
        .setEdgeTargetPropertyName("labels")
        .setEdgeCombinationMethod(method)
        .build();
```

Python

15.4.8 Training the Supervised EdgeWise Model

You can train a SupervisedEdgeWiseModel on a graph as shown:

- JShell
- Java
- Python

JShell

opg4j> model.fit(trainGraph)

```
model.fit(trainGraph);
```

Python

model.fit(train_graph)

15.4.9 Getting the Loss Value for a Supervised EdgeWise Model

You can fetch the training loss value for a Supervised EdgeWise Model as shown in the following code:

- JShell
- Java
- Python

JShell

```
opg4j> var loss = model.getTrainingLoss()
```

Java

```
double loss = model.getTrainingLoss();
```

Python

```
loss = model.get_training_loss()
```

15.4.10 Inferring Edge Labels for a Supervised EdgeWise Model

You can infer the edge labels on any graph (including edges or graphs that were not seen during training):

- JShell
- Java
- Python



```
opg4j> var labels = model.infer(fullGraph, testEdges)
opg4j> labels.head().print()
```

Java

```
PgxFrame labels = model.infer(fullGraph, testEdges);
labels.head().print();
```

Python

```
labels = model.infer(full_graph,test_edges)
labels.print()
```

If the loss is <code>SigmoidCrossEntropy</code> or <code>DevNetLoss</code>, then it is also possible to set the decision threshold applied to the logits by adding it as an extra parameter, which is by default 0:

- JShell
- Java
- Python

JShell

```
opg4j> var labels = model.infer(fullGraph, testEdges, 6f)
opg4j> labels.head().print()
```

Java

```
PgxFrame labels = model.infer(fullGraph,testEdges,6f);
labels.head().print();
```

Python

```
labels = model.infer(full_graph, full_graph.get_edges(), 6)
labels.print()
```



The output will be similar to the follow	ving example output:
--	----------------------

+-		 	+
	edgeId	value	
+-		 	+
	68472	2.2346956729888916	
	53436	2.1515913009643555	
	73364	1.9499346017837524	
	12096	2.1704165935516357	
	78740	2.1174447536468506	
	27664	2.1041007041931152	
	34844	2.148571491241455	
	74224	2.089123010635376	
	33744	2.0866644382476807	
	32812	2.0604987144470215	
+-		 	+

Similarly, if the task is a classification task, you can get the model confidence for each class by inferring the prediction logits:

- JShell
- Java
- Python

JShell

```
opg4j> var logits = model.inferLogits(fullGraph, testEdges)
opg4j> logits.head().print()
```

Java

```
PgxFrame logits = model.inferLogits(fullGraph,testEdges);
logits.head().print();
```

Python

```
logits = model.infer_logits(full_graph, test_edges)
logits.print()
```

If the model is a classification model, the inferLabels method is also available and it is equivalent to the infer methoid.

15.4.11 Evaluating Model Performance

You can use the evaluate convenience method to evaluate various metrics for the model:



- JShell
- Java
- Python

```
opg4j> model.evaluate(fullGraph, testEdges).print()
```

Java

```
model.evaluate(fullGraph,testEdges).print();
```

Python

```
model.evaluate(full_graph,test_edges).print()
```

Similar to inferring labels, if the task is a classification task, you can add the decision threshold as an extra parameter:

- JShell
- Java
- Python

JShell

```
opg4j> model.evaluate(fullGraph, testEdges, 6f).print()
```

Java

```
model.evaluate(fullGraph,testEdges, 6f).print();
```

Python

```
model.evaluate(full graph,test edges, 6).print()
```



For a classification model, the output will be similar to the following:

+		 	 	 	+
	-	Precision			
+		 	 	 	-+
		0.8523			
+		 	 	 	-+

For a regression model, the output will be similar to the following:

Note that for a classification model, the <code>evaluateLabels</code> method is also available and this is equivalent to the <code>evaluate</code> method.

15.4.12 Inferring Embeddings for a Supervised EdgeWise Model

You can use a trained model to infer embeddings for unseen nodes and store them in the database as described in the following code:

- JShell
- Java
- Python

JShell

```
opg4j> var edgeVectors = model.inferEmbeddings(fullGraph,
testEdges).flattenAll()
opg4j> edgeVectors.write().
         db().
         name("edge vectors").
         tablename("edgeVectors").
         overwrite(true).
         store()
```

Java

```
PgxFrame edgeVectors = model.inferEmbeddings(fullGraph,
testEdges).flattenAll();
edgeVectors.write()
   .db()
   .name("edge vectors")
   .tablename("edgeVectors")
```



```
.overwrite(true)
.store();
```

```
edge_vectors = model.infer_embeddings(full_Graph,
test_edges).flatten_all()
edge_vectors.write().db().table_name("table_name").name("edge_vectors")
.overwrite(True).store()
```

The schema for the <code>edgeVectors</code> will be as follows without flattening (flattenAll splits the vector column into separate double-valued columns):

All the preceding examples assume that you are inferring the embeddings for a model in the current logged in database. If you must infer embeddings for the model in a different database, then you must additionally provide the database credentials such as username, password and JDBC URL to the inferEmbeddings method. Refer to Inferring Embeddings for a Model in Another Database for an example.

15.4.13 Storing a Supervised EdgeWise Model

You can store models in the database. The models get stored as a row inside a model store table.

The following shows how to store a trained SupervisedEdgeWise model in the database in a specific model store table:

- JShell
- Java
- Python



Python

Note:

All the preceding examples assume that you are storing the model in the current logged in database. If you must store the model in a different database then refer to the examples in Storing a Trained Model in Another Database.

15.4.14 Loading a Pre-Trained Supervised EdgeWise Model

You can load a pre-trained SupervisedEdgeWise model from a model store table in the database as shown:

- JShell
- Java
- Python



```
SupervisedEdgeWiseModel model =
analyst.loadSupervisedEdgeWiseModel().db()
    .modelstore("modeltablename") // name of the model store table
    .modelname("model") // model name (primary key of model
store table)
    .load();
```

Python

Note:

All the preceding examples assume that you are loading the model from the current logged in database. If you must load the model from a different database then refer to the examples in Loading a Pre-Trained Model From Another Database.

15.4.15 Destroying a Supervised EdgeWise Model

You can destroy a Supervised EdgeWise model as described in the following code:

- JShell
- Java
- Python

JShell

```
opg4j> model.destroy()
```

Java

model.destroy();



model.destroy()

15.4.16 Example: Predicting Ratings on the Movielens Dataset

This section describes the usage of SupervisedEdgeWise in the graph server (PGX) using the Movielens graph as an example.

This data set consists of 100,000 ratings (1-5) from 943 users on 1682 movies, with simple demographic information for the users (age, gender, occupation) and movies (year, aggravating, genre). Users and movies are vertices, while ratings of users to movies are edges with a rating feature.

The following example predicts the ratings using the <code>SupervisedEdgeWise</code> model. The model is first built and it is then fit on the <code>trainGraph</code>.

- JShell
- Java
- Python

```
opg4j> import oracle.pgx.config.mllib.loss.LossFunctions
opg4j> var convLayer = analyst.graphWiseConvLayerConfigBuilder().
        setNumSampledNeighbors(10).
opg4j> var predictionLayer = analyst.graphWisePredictionLayerConfigBuilder().
        setHiddenDimension(16).
        build()
opg4j> var model = analyst.supervisedEdgeWiseModelBuilder().
        setVertexInputPropertyNames("movie year", "avg rating",
"movie genres", // Movies features
            "user occupation label", "user gender", "raw user age"). //
Users features
        setEdgeTargetPropertyName("user rating").
        setConvLayerConfigs(convLayer).
        setPredictionLayerConfigs(predictionLayer).
        setNumEpochs(10).
        setEmbeddingDim(32).
        setLearningRate(0.003).
        setStandardize(true).
        setNormalize(true).
        setSeed(0).
        setLossFunction(LossFunctions.MSE LOSS).
        build()
opg4j> model.fit(trainGraph)
```



```
import oracle.pgx.config.mllib.loss.LossFunctions;
GraphWiseConvLayerConfig convLayer =
analyst.graphWiseConvLayerConfigBuilder()
        .setNumSampledNeighbors(10)
        .build();
GraphWisePredictionLayerConfig predictionLayer =
analyst.graphWisePredictionLayerConfigBuilder()
      .setHiddenDimension(16)
      .build();
SupervisedEdgeWiseModel model =
analyst.supervisedEdgeWiseModelBuilder()
        .setVertexInputPropertyNames("movie year", "avg rating",
"movie genres", // Movies features
            "user occupation label", "user gender", "raw user age") //
Users features
        .setEdgeTargetPropertyName("user rating")
        .setConvLayerConfigs(convLayer)
        .setPredictionLayerConfigs(predictionLayer)
        .setNumEpochs(10)
        .setEmbeddingDim(32)
        .setLearningRate(0.003)
        .setStandardize(true)
        .setNormalize(true)
        .setSeed(0)
        .setLossFunction(LossFunctions.MSE LOSS)
        .build();
model.fit(trainGraph);
Python
from pypgx.api.mllib import MSELoss
conv layer config = dict(num sampled neighbors=10)
conv layer = analyst.graphwise conv layer config(**conv layer config)
pred layer config = dict(hidden dim=16)
pred layer = analyst.graphwise pred layer config(**pred layer config)
params = dict(edge target property name="labels",
              conv layer config=[conv layer],
```

pred layer config=[pred layer],

num_epochs=10,
layer size=32,

vertex input property names=["movie year", "avg rating",

"user occupation label", "user gender",

edge input property names=["user rating"],



"movie genres",

"raw user age"],

Since EdgeWise is inductive, you can infer the ratings for unseen edges:

- JShell
- Java
- Python

JShell

```
opg4j> var labels = model.infer(fullGraph, testEdges)
opg4j> labels.head().print()
```

Java

```
PgxFrame labels = model.infer(fullGraph, testEdges);
labels.head().print();
```

Python

```
labels = model.infer(full_graph, test_edges)
labels.print()
```

This returns the rating prediction for any edge as:



```
| 33744 | 3.2331383228302 |
| 32812 | 3.8763082027435303 |
```

You can also evaluate the performance of the model:

- JShell
- Java
- Python

JShell

```
opg4j> model.evaluate(fullGraph, testEdges).print()
```

Java

```
model.evaluate(fullGraph, testEdges).print();
```

Python

```
model.evaluate(full graph, test edges).print()
```

This returns the following output:

15.5 Using the Pg2vec Algorithm

Pg2vec learns representations of graphlets (partitions inside a graph) by employing edges as the principal learning units and thereby packing more information in each learning unit (as compared to employing vertices as learning units) for the representation learning task.

It consists of three main steps:

- Random walks for each vertex (with pre-defined length per walk and pre-defined number of walks per vertex) are generated.
- 2. Each edge in this random walk is mapped as a property.edge-word in the created document (with the document label as the graph-id) where the property.edge-



word is defined as the concatenation of the properties of the source and destination vertices.

3. The generated documents (with their attached document labels) are fed to a doc2vec algorithm which generates the vector representation for each document, which is a graph in this case.

Pg2vec creates graphlet embeddings for a specific set of graphlets and cannot be updated to incorporate modifications on these graphlets. Instead, a new Pg2vec model should be trained on these modified graphlets.

The following represents the memory consumption of Pg2vec model.

0(2(n+m)*d)

where:

- n: is the number of vertices in the graph
- m: is the number of graphlets in the graph
- d: is the embedding length

The following describes the usage of the main functionalities of the implementation of Pg2vec in PGX using NCI109 dataset as an example with 4127 graphs in it:

- Loading a Graph
- Building a Minimal Pg2vec Model
- Building a Customized Pg2vec Model
- Training a Pg2vec Model
- Getting the Loss Value For a Pg2vec Model
- Computing Similar Graphlets for a Given Graphlet
- Computing Similars for a Graphlet Batch
- Inferring a Graphlet Vector
- Inferring Vectors for a Graphlet Batch
- Storing a Trained Pg2vec Model
- Loading a Pre-Trained Pg2vec Model
- Destroying a Pg2vec Model

15.5.1 Loading a Graph

The following describes the steps for loading a graph:

- 1. Create a Session and an Analyst.
 - JShell
 - Java
 - Python



JShell

```
cd /opt/oracle/graph/
./bin/opg4j
// starting the shell will create an implicit session and analyst
```

Java

```
import oracle.pgx.api.*;
import oracle.pgx.api.mllib.Pg2vecModel;
import oracle.pgx.api.frames.*;
...
PgxSession session = Pgx.createSession("my-session");
Analyst analyst = session.createAnalyst();
```

Python

```
session = pypgx.get_session(session_name="my-session")
analyst = session.create analyst()
```

2. Load the graph.

- JShell
- Java
- Python

JShell

```
opg4j> var graph = session.readGraphWithProperties("<path>/
<graph.json>")
```

Java

```
PgxGraph graph = session.readGraphWithProperties("<path>/
<graph.json>");
```

```
graph = session.read graph with properties("<path>/<graph.json>")
```



15.5.2 Building a Minimal Pg2vec Model

You can build a Pg2vec model using the minimal configuration and default hyper-parameters as described in the following code:

- JShell
- Java
- Python

JShell

Java

```
Pg2vecModel model = analyst.pg2vecModelBuilder()
    .setGraphLetIdPropertyName("graph_id")
    .setVertexPropertyNames(Arrays.asList("category"))
    .setWindowSize(4)
    .setWalksPerVertex(5)
    .setWalkLength(8)
    .build();
```

Python

```
model = analyst.pg2vec_builder(
    graphlet_id_property_name="graph_id",
    vertex_property_names=["category"],
    window_size=4,
    walks_per_vertex=5,
    walk_length=8)
```

You can specify the property name to determine each graphlet using the Pg2vecModelBuilder#setGraphLetIdPropertyName operation and also employ the vertex properties in Pg2vec which are specified using the

Pg2vecModelBuilder#setVertexPropertyNames operation.



You can also use the weakly connected component (WCC) functionality in PGX to determine the graphlets in a given graph.

15.5.3 Building a Customized Pg2vec Model

You can build a Pg2vec model using customized hyper-parameters as described in the following code:

- JShell
- Java
- Python

JShell

```
opg4j> var model = analyst.pg2vecModelBuilder().
                setGraphLetIdPropertyName("graph id").
                setVertexPropertyNames(Arrays.asList("category")).
                setMinWordFrequency(1).
                setBatchSize(128).
                setNumEpochs(5).
                setLayerSize(200).
                setLearningRate(0.04).
                setMinLearningRate(0.0001).
                setWindowSize(4).
                setWalksPerVertex(5).
                setWalkLength(8).
                setUseGraphletSize(true).
                setValidationFraction(0.05).
                setGraphletSizePropertyName("propertyName>").
                build()
```

```
Pg2vecModel model= analyst.pg2vecModelBuilder()
    .setGraphLetIdPropertyName("graph id")
    .setVertexPropertyNames(Arrays.asList("category"))
    .setMinWordFrequency(1)
    .setBatchSize(128)
    .setNumEpochs(5)
    .setLayerSize(200)
    .setLearningRate(0.04)
    .setMinLearningRate(0.0001)
    .setWindowSize(4)
    .setWalksPerVertex(5)
    .setWalkLength(8)
    .setUseGraphletSize(true)
    .setValidationFraction(0.05)
    .setGraphletSizePropertyName("cpropertyName>")
    .build();
```



```
model = analyst.pg2vec_builder(
    graphlet_id_property_name="graph_id",
    vertex_property_names=["category"],
    min_word_frequency=1,
    batch_size=128,
    num_epochs=5,
    layer_size=200,
    learning_rate=0.04,
    min_learning_rate=0.0001,
    window_size=4,
    walks_per_vertex=5,
    walk_length=8,
    use_graphlet_size=true,
    graphlet_size_property_name="roperty_name>",
    validation_fraction=0.05)
```

See Pg2vecModelBuilder in Javadoc for more explanation for each builder operation along with the default values.

15.5.4 Training a Pg2vec Model

You can train a Pg2vec model with the specified default or customized settings as described in the following code:

- JShell
- Java
- Python

JShell

```
opg4j> model.fit(graph)
```

Java

```
model.fit(graph);
```

Python

model.fit(graph)

15.5.5 Getting the Loss Value For a Pg2vec Model

You can fetch the training loss value on a specified fraction of training data (set in builder using setValidationFraction) as described in the following code:

- JShell
- Java
- Python

JShell

```
opg4j> var loss = model.getLoss()
```

Java

```
double loss = model.getLoss();
```

Python

loss = model.loss

15.5.6 Computing Similar Graphlets for a Given Graphlet

You can fetch the ${\bf k}$ most similar graphlets for a given graphlet as described in the following code:

- JShell
- Java
- Python

JShell

```
opg4j> var similars = model.computeSimilars(52, 10)
```

Java

PgxFrame similars = model.computeSimilars(52, 10);



```
similars = model.compute_similars(52, 10)
```

Searching for similar vertices for graphlet with ID = 52 using the trained model and printing it with similars.print(), will result in the following output:

+	 	+
dstGraphlet	similarity	I
+	 	+
52	1.0	
10	0.8748674392700195	
23	0.8551455140113831	
26	0.8493421673774719	
47	0.8411962985992432	
25	0.8281504511833191	
43	0.8202780485153198	
24	0.8179885745048523	
8	0.796689510345459	
9	0.7947834134101868	
+	 	+

The following depicts the visualization of two similar graphlets (top: ID = 52 and bottom: ID = 10):



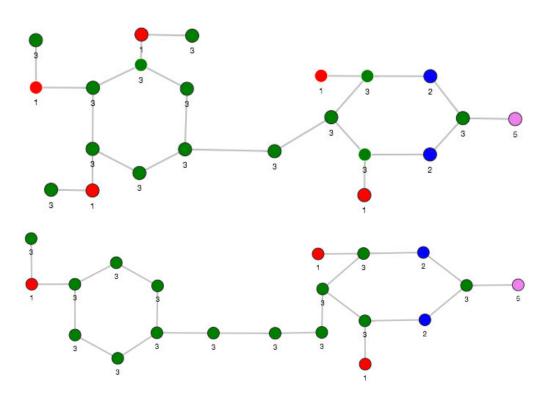


Figure 15-1 Pg2vec - Visualization of Two Similar Graphlets

15.5.7 Computing Similars for a Graphlet Batch

You can fetch the ${\bf k}$ most similar graphlets for a batch of input graphlets as described in the following code:

- JShell
- Java
- Python

JShell

```
opg4j> var graphlets = new ArrayList()
opg4j> graphlets.add(52)
opg4j> graphlets.add(41)
opg4j> var batchedSimilars = model.computeSimilars(graphlets, 10)
```

```
List graphlets = Arrays.asList(52,41);
PgxFrame batchedSimilars = model.computeSimilars(graphlets,10);
```

```
batched_similars = model.compute_similars([52,41],10)
```

Searching for similar vertices for graphlet with ID = 52 and ID = 41 using the trained model and printing it with batched similars.print(), will result in the following output:

+	srcGraphlet		dstGraphlet	 	similarity	+
	52		52		1.0	
	52		10		0.8748674392700195	
	52		23		0.8551455140113831	
	52		26		0.8493421673774719	
	52		47		0.8411962985992432	
	52		25		0.8281504511833191	
	52		43		0.8202780485153198	
	52		24		0.8179885745048523	
	52		8		0.796689510345459	
	52		9		0.7947834134101868	
	41		41		1.0	
	41		197		0.9653506875038147	
	41		84		0.9552277326583862	
	41		157		0.9465565085411072	
	41		65		0.9287481307983398	
	41		248		0.9177336096763611	
	41		315		0.9043129086494446	
	41		92		0.8998928070068359	
	41		297		0.8897411227226257	
	41		50		0.8810243010520935	
+						. +

15.5.8 Inferring a Graphlet Vector

You can infer the vector representation for a given new graphlet as described in the following code:

- JShell
- Java
- Python

JShell

opg4j> var graphlet = session.readGraphWithProperties("<path>/
<graphletConfig.json>")



```
opg4j> var inferredVector = model.inferGraphletVector(graphlet)
opg4j> inferredVector.print()
```

Java

Python

```
graphlet = session.read_graph_with_properties("<path>/
<graphletConfig.json>")
inferred_vector = model.infer_graphlet_vector(graphlet)
inferred_vector.print()
```

The schema for the inferredVector will be similar to the following output:

15.5.9 Inferring Vectors for a Graphlet Batch

You can infer the vector representations for multiple graphlets (specified with different graph-ids in a graph) as described in the following code:

- JShell
- Java
- Python

JShell

```
opg4j> var graphlet = session.readGraphWithProperties("<path>/
<graphletConfig.json>")
opg4j> var inferredVectorBatched =
model.inferGraphletVectorBatched(graphlets)
opg4j> inferredVectorBatched.print()
```

```
PgxGraph graphlet = session.readGraphWithProperties("<path>/
<graphletConfig.json>");
```

```
PgxFrame inferredVectorBatched = model.inferGraphletVectorBatched(graphlets);
inferredVector.print();
```

```
graphlets = session.read_graph_with_properties("<path>/
<graphletConfig.json>")
inferred_vector_batched = model.infer_graphlet_vector_batched(graphlets)
inferred_vector_batched.print()
```

The schema is same as for inferGraphletVector but with more rows corresponding to the input graphlets.

15.5.10 Storing a Trained Pg2vec Model

You can store models in database. The models get stored as a row inside a model store table.

The following code shows how to store a trained Pg2vec model in database in a specific model store table:

- JShell
- Java
- Python

JShell





All the preceding examples assume that you are storing the model in the current logged in database. If you must store the model in a different database then refer to the examples in Storing a Trained Model in Another Database.

15.5.11 Loading a Pre-Trained Pg2vec Model

You can load models from a database.

You can load a pre-trained Pg2vec model from a model store table in database as described in the following:

- JShell
- Java
- Python

JShell

```
Pg2vecModel model = analyst.loadPg2vecModel().db()
    .modelstore("modeltablename") // name of the model store table
    .modelname("model") // model name (primary key of model store table)
    .load();
```





All the preceding examples assume that you are loading the model from the current logged in database. If you must load the model from a different database then refer to the examples in Loading a Pre-Trained Model From Another Database.

15.5.12 Destroying a Pg2vec Model

You can destroy a Pg2vec model as described in the following code:

- JShell
- Java
- Python

JShell

```
opg4j> model.destroy()
```

Java

model.destroy();

Python

model.destroy()

15.6 Model Repository and Model Stores

A model store can be used to persist the trained graph server (PGX) machine learning models along with a model name (a unique identifier of the model in a particular model store) and a description.

The model repository API provides the following capabilities:

- Create a new model store
- · List all the available model stores in the model repository
- Store a model in the model store
- List all the models in a given model store
- Load a model from the model store
- Get the model description for a model that is stored in the given model store
- Delete a model from the given model store
- Delete the existing model stores
- Database-Backed Model Repository

15.6.1 Database-Backed Model Repository

In a database-backed model repository, each model store corresponds to a table in the database. Internally, the tables are prefixed by 'GMLS_'.

The following steps describe the usage of the model repository API with code examples.

- 1. Create a model repository object as shown:
 - JShell
 - Java
 - Python

JShell

```
opg4j> var mr = analyst.modelRepository().db().open()
mr ==> oracle.pgx.api.mllib.DbModelRepository@5aac6f9f
```

Java

DbModelRepository mr = analyst.modelRepository().db().open();

```
>>> mr = analyst.model_repository().db()
>>> mr
<pypgx.api.mllib._model_repo.ModelRepository object at
0x7f637496df60>
```



The preceding example assumes that you are creating the model repository from the current logged in database. If you must create the repository in a different database, then refer to the following example:

- JShell
- Java
- Python

JShell

Java

- 2. Create a model store as shown:
 - JShell
 - Java

JShell

```
opg4j> var modelstore = "modelstore"
modelstore ==> "modelstore"
opg4j> mr.create(modelstore)

Java
String modelstore = "modelstore";
mr.create(modelstore);
```

Python

```
>>> mr.create("modelstore")
```

- 3. List the model store as shown and verify that the model store is empty:
 - JShell
 - Java
 - Python

JShell

```
opg4j> mr.listModelStoresNames()
$4 ==> [DW, deepwalk_model, modelstore, modelstoretablename]
opg4j> mr.listModelStoresNamesMatching(modelstore)
$5 ==> [modelstore, modelstoretablename]
opg4j> mr.listModels(modelstore)
$6 ==> []
```

Java

```
mr.listModelStoresNames();
mr.listModelStoresNamesMatching(modelstore);
mr.listModels(modelstore);
```

```
>>> mr.list_model_stores_names()
>>> mr.list_model_stores_names_matching("modelstore")
>>> mr.list_models("modelstore")
```



- 4. Create and fit a DeepWalk model as shown:
 - JShell
 - Java
 - Python

JShell

```
opg4j > var walkLength = 5
opg4j> var walksPerVertex = 4
opg4j> var embeddingSize = 20
opg4j> var batchSize = 128
opg4j> var model = analyst.deepWalkModelBuilder()
                           .setLayerSize(embeddingSize)
                           .setWalkLength(walkLength)
                           .setWalksPerVertex(walksPerVertex)
                           .setValidationFraction(1)
                           .setBatchSize(batchSize).build()
model ==> oracle.pgx.api.mllib.DeepWalkModel@34be7efb
opg4j> var smallGraphDeepWalk =
session.readGraphByName("<graph name>",GraphSource.PG VIEW)
smallGraphDeepWalk ==>
PgxGraph[name=BANK GRAPH VIEW 2, N=1000, E=5001, created=1649075718843]
opg4j> model.fit(smallGraphDeepWalk)
```

Java

```
>>> model =
analyst.deepwalk builder(window size=3,walks per vertex=6,walk length=4)
```

```
graph = session.read_graph_by_name("<graph_name>", 'pg_view')
>>> model.fit(graph)
```

- **5.** Store the trained model in the model store as shown:
 - JShell
 - Java
 - Python

JShell

Java

```
>>> model.export().db(model_store = "modelstore", model_name =
"DeepWalkModel",
... model_description = "DeepWalk model description")
```

- **6.** Verify that the model is now stored in the model store as shown:
 - JShell
 - Java



JShell

```
opg4j> mr.listModels(modelstore)
$11 ==> [DeepWalkModel]
```

Java

```
mr.listModels(modelstore);
```

Python

```
>>> mr.list models("modelstore")
```

- 7. Load the model from the model store as shown:
 - JShell
 - Java
 - Python

JShell

Java



The preceding example assumes that you are loading the model from the current logged in database. If you must load the model from a different database then refer to the example in Loading a Pre-Trained Model From Another Database.

8. Get the model description from the model store as shown:

- JShell
- Java
- Python

JShell

```
opg4j> mr.getModelDescription(modelstore,modelName)
$14 ==> "DeepWalk: model desc"
```

Java

mr.getModelDescription(modelstore, modelName);

Python

```
>>> mr.get_model_description("modelstore","DeepWalkModel")
'DeepWalk model description'
```

- 9. Delete the model from the model store as shown:
 - JShell
 - Java
 - Python

JShell

opg4j> mr.deleteModel(modelstore, modelName)

Java

mr.deleteModel(modelstore, modelName);

>>> mr.delete_model("modelstore","DeepWalkModel")

10. Delete the model store as shown:

- JShell
- Java
- Python

JShell

opg4j> mr.deleteModelStore(modelstore)

Java

mr.deleteModelStore(modelstore);

Python

>>> ("modelstore")



Executing PGQL Queries Against the Graph Server (PGX)

This section describes the Java APIs that are used to execute PGQL queries in the graph server (PGX).

- Getting Started with PGQL
- Creating Property Graphs Using Options
- Supported PGQL Features and Limitations on the Graph Server (PGX)
- Java APIs for Executing CREATE PROPERTY GRAPH Statements
- Python APIs for Executing CREATE PROPERTY GRAPH Statements
- Java APIs for Executing SELECT Queries
 This section describes the APIs to execute SELECT queries in the graph server (PGX).
- Java APIs for Executing UPDATE Queries
 The UPDATE queries make changes to existing graphs using the INSERT, UPDATE, and DELETE operations as detailed in the section Graph Modification of the PGQL 1.3 specification.
- PGQL Queries with Partitioned IDs
- Security Tools for Executing PGQL Queries

 To safeguard against query injection, bind variables can be used in place of literals while printIdentifier (String identifier) can be used in place of identifiers like graph names, labels, and property names.
- Best Practices for Tuning PGQL Queries
 This section describes best practices regarding memory allocation, parallelism, and query planning.

16.1 Getting Started with PGQL

This section provides an example on how to get started with PGQL. It assumes a database realm that has been previously set up (follow the steps in Prepare the Graph Server for Database Authentication). It also assumes that the user has read access to the HR schema.

First, create a graph with employees, departments, and employee works at department, by executing a CREATE PROPERTY GRAPH statement.

Example 16-1 Creating a graph in the graph server (PGX)

The following statement creates a graph in the graph server (PGX)

```
String statement =
    "CREATE PROPERTY GRAPH hr_simplified "
+ " VERTEX TABLES ( "
+ " hr.employees LABEL employee "
+ " PROPERTIES ARE ALL COLUMNS EXCEPT ( job id, manager id,
```

```
department id ), "
   + " hr.departments LABEL department "
          PROPERTIES ( department id, department name ) "
   + " ) "
   + " EDGE TABLES ( "
   + " hr.employees AS works_at "
           SOURCE KEY ( employee id ) REFERENCES employees
(employee id) "
   + " DESTINATION departments " + " PROPERTIES ( employee_id ) "
   + " ) ";
session.executePgql(statement);
 * To get a handle to the graph, execute:
PgxGraph g = session.getGraph("HR SIMPLIFIED");
/**
 * You can use this handle to run PGQL queries on this graph.
 ^{\star} For example, to find the department that "Nandita Sarchand" works
for, execute:
*/
String query =
   "SELECT dep.department name "
 + "FROM MATCH (emp:Employee) -[:works at]-> (dep:Department) "
 + "WHERE emp.first name = 'Nandita' AND emp.last name = 'Sarchand' "
 + "ORDER BY 1";
PgqlResultSet resultSet = g.queryPgql(query);
resultSet.print();
+----+
| department name |
+----+
| Shipping
+----+
 * To get an overview of the types of vertices and their frequencies,
execute:
String guery =
     "SELECT label(n), COUNT(*) "
   + "FROM MATCH (n) "
   + "GROUP BY label(n) "
   + "ORDER BY COUNT(*) DESC";
PgqlResultSet resultSet = g.queryPgql(query);
resultSet.print();
+----+
| label(n) | COUNT(*) |
+----+
| EMPLOYEE | 107
| DEPARTMENT | 27
+----+
```

16.2 Creating Property Graphs Using Options

Using the **OPTIONS** clause in the CREATE PROPERTY GRAPH statement, you can specify any of the options explained in the following sections:

Using Graph Optimization Options

You can load a graph for querying and analytics or for performing update operations. Depending on your requirement, you can optimize the read or update performance using the **OPTIONS** clause in the CREATE PROPERTY GRAPH statement.

The following table describes the valid options that are supported in the OPTIONS clause:

Table 16-1 Graph Optimization Options

OPTIONS	Description
OPTIMIZED_FOR_READ	This can be used for read-intensive scenarios.
OPTIMIZED_FOR_UPDATES	This is the default option and can be used for fast updates.
SYNCHRONIZABLE	This assures that the graph can be synchronized via Flashback Technology. However, exceptions are thrown if one of the edge keys is either composite or non-numeric. In these cases, the graph can normally still be loaded, but PGX generates a new (numeric and noncomposite) edge key. Such edges can therefore not be synchronized with the database.

For example, the following graph is set using <code>OPTIMIZED_FOR_UPDATES</code> and <code>SYNCHRONIZABLE</code> options:

```
CREATE PROPERTY GRAPH hr

VERTEX TABLES (
employees LABEL employee, departments LABEL department)

EDGE TABLES (
departments AS managed by
```



```
SOURCE KEY ( department_id ) REFERENCES departments (department_id)
DESTINATION employees
NO PROPERTIES
) OPTIONS (OPTIMIZED_FOR_UPDATES, SYNCHRONIZABLE)
```

Note:

- SYNCHRONIZABLE option can be used in combination with
 OPTIMIZED_FOR_UPDATES and OPTIMIZED_FOR_READ. But,
 OPTIMIZED_FOR_UPDATES and OPTIMIZED_FOR_READ cannot be used
 together and in such a case an exception will be thrown.
- If you are creating a synchronizable graph, then ensure that the vertex and edge keys are numeric and non-composite.

Using Options to Handle Edges with Missing Vertices

If either the source or destination vertex or both are missing for an edge, then you can configure one of the following values in the OPTIONS clause in the CREATE PROPERTY GRAPH statement:

- IGNORE EDGE ON MISSING VERTEX: Specifies that the edge for a missing vertex must be ignored.
- IGNORE EDGE AND LOG ON MISSING VERTEX: Specifies that the edge for a missing vertex must be ignored and all ignored edges must be logged.
- IGNORE EDGE AND LOG ONCE ON MISSING VERTEX: Specifies that the edge for a
 missing vertex must be ignored and only the first ignored edge must be logged.
- ERROR ON MISSING VERTEX (default): Specifies that an error must be thrown for edges with missing vertices.

For example, the following graph is set using ERROR ON MISSING VERTEX option:

```
CREATE PROPERTY GRAPH region_graph

VERTEX TABLES (

regions KEY (region_id),

countries KEY (country_id)
)

EDGE TABLES (

countries AS countries_regions

SOURCE KEY ( country_id ) REFERENCES countries(country_id)

DESTINATION KEY (region_id) REFERENCES regions(region_id)

NO PROPERTIES
) OPTIONS ( ERROR ON MISSING VERTEX)
```

On execution, the following error response is shown:

```
unknown vertex ID received in destination 4 of edge 5
```

When using IGNORE EDGE AND LOG ON MISSING VERTEX or IGNORE EDGE AND LOG ONCE ON MISSING VERTEX option, you must update the default Logback configuration

file in /etc/oracle/graph/logback.xml and the graph server (PGX) logger configuration file in /etc/oracle/graph/logback-server.xml to log the DEBUG logs. Only then you can view the ignored edges in /var/opt/log/pgx-server.log file.

16.3 Supported PGQL Features and Limitations on the Graph Server (PGX)

The following table provides the complete list of supported and unsupported PGQL functionalities for the graph server (PGX):

Table 16-2 Supported PGQL Functionalities and Limitations on the Graph Server (PGX)

Features	PGQL on the Graph Server (PGX)
CREATE PROPERTY GRAPH	Supported
	Limitations:
	No composite keys for vertices
DROP PROPERTY GRAPH	Not Supported
Fixed-length pattern matching	Supported
Variable-length pattern matching goals	Supported:
	 Reachability
	• ANY
	• ANY SHORTEST
	• TOP k SHORTEST
	• ALL SHORTEST
	ANY CHEAPEST
	 TOP k CHEAPEST
	• ALL
Variable-length pattern matching quantifiers	Supported:
	• *
	• +
	• ?
	• {n}
	• { n, }
	• {n, m}
	• { , m }
	Limitations:
	• ? is only supported for reachability
	In case of ANY CHEAPEST and TOP k CHEAPEST anhy* is supported.
	CHEAPEST, only * is supported
Variable-length path unnesting	Supported: ONE ROW PER VERTEX
	• ONE ROW PER VERIEX • ONE ROW PER STEP
	Limitation:
CDOID DV	* quantifier is not supported Supported
GROUP BY	Supported
HAVING	Supported



Table 16-2 $\,$ (Cont.) Supported PGQL Functionalities and Limitations on the Graph Server (PGX)

Features	PGQL on the Graph Server (PGX)
Aggregations	Supported: COUNT
	• MIN, MAX, AVG, SUM
	• LISTAGG
	ARRAY AGG
	Limitations:
	ARRAY_AGG is only supported as horizontal aggregation (in combination with variable-length path) but not in combination with vertical aggregation
DISTINCT	Supported
• SELECT DISTINCT	
 Aggregation with DISTINCT (such as, COUNT (DISTINCT e.prop)) 	
SELECT v.*	Supported
ORDER BY (+ASC/DESC), LIMIT, OFFSET	Supported
Data Types	Supported:
	• INTEGER (32-bit)
	• LONG (64-bit)
	• FLOAT (32-bit)
	• DOUBLE (64-bit)
	 STRING (no maximum length)
	• BOOLEAN
	• DATE
	• TIME
	• TIME WITH TIME ZONE
	• TIMESTAMP
	TIMESTAMP WITH TIME ZONE
JSON	No built-in JSON support. However, JSON values can be stored as STRING and manipulated or queried through user-defined functions (UDFs) written in Java or JavaScript.
Operators	Supported:
	 Relational: +, -, *, /, %, - (unary minus)
	• Arithmetic: =, <>, <, >, <=, >=
	• Logical: AND, OR, NOT
	String: (concat)



Table 16-2 (Cont.) Supported PGQL Functionalities and Limitations on the Graph Server (PGX)

Features	PGQL on the Graph Server (PGX)
Functions and predicates	Supported: IS NULL, IS NOT NULL JAVA_REGEXP_LIKE (based on CONTAINS) LOWER, UPPER SUBSTRING ABS, CEIL/CEILING, FLOOR, ROUND EXTRACT ID LABEL, LABELS, HAS_LABEL ALL_DIFFERENT IN_DEGREE, OUT_DEGREE CAST CASE IN and NOT IN MATCH_NUMBER ELEMENT NUMBER
User-defined functions	Supported: Java UDFs JavaScript UDFs
Subqueries: Scalar subqueries EXISTS and NOT EXISTS subqueries	Supported
INSERT/UPDATE/DELETE	Supported
INTERVAL literals and operations	Supported literals:
	SECONDMINUTE
	• HOUR
	• DAY
	• MONTH
	YEAR Supported operations:
	Supported operations:
	 Add INTERVAL to datetime (+) Subtract INTERVAL from datetime (-)

Also, the following explains certain supported and unsupported PGQL features:

- Support for Selecting All Properties
- Unnesting of Variable-Length Path Queries
- Using INTERVAL Literals in PGQL Queries
- Limitations on Quantifiers
- Limitations on WHERE and COST Clauses in Quantified Patterns



16.3.1 Support for Selecting All Properties

You can use SELECT $\, v.* \, to \, select \, all \, properties \, of \, the \, vertices \, or \, edges \, that \, bind \, to \, the \, variable \, v. \, For example:$

```
SELECT label(n), n.* FROM MATCH (n) ORDER BY "number", "name"
```

On execution, the query output is as shown:

+	 		+
label(n)	number	1	name
+	 		+
Account	1001		<null> </null>
Account	2090		<null> </null>
Account	8021		<null> </null>
Account	10039		<null> </null>
Person	<null></null>		Camille
Person	<null></null>		Liam
Person	<null></null>		Nikita
Company	<null></null>		Oracle
+	 		+

You can use label expressions to select properties that belong to the specified vertex or edge labels. For example:

```
SELECT label(n), n.* FROM MATCH (n:Person) ORDER BY "name"
```

The preceding query retrieves all the properties for the specified Person label:

+		 	+
	label(n)	name	
+		 	+
	Person	Camille	
	Person	Liam	
	Person	Nikita	
Τ.		 	

You can also specify a PREFIX to avoid duplicate column names in cases where you select all properties using multiple variables. For example:

```
SELECT n.* PREFIX 'n_', e.* PREFIX 'e_', m.* PREFIX 'm_'
FROM MATCH (n:Account) -[e:transaction]-> (m:Account)
ORDER BY "e_amount"
```

The query output is as shown:

```
+-----+
| n_number | e_amount | m_number |
+-----+
| 10039 | 1000.0 | 8021 |
```



	8021	1500.3	- 1	1001	
	8021	3000.7		1001	
	2090	9900.0		10039	
	1001	9999.5	-	2090	
+-		 			+

16.3.2 Unnesting of Variable-Length Path Queries

Unnesting of variable-length path queries (such as, SHORTEST or CHEAPEST paths) to obtain a separate row for each vertex or edge along a path is supported.

You can unnest a path aggregation using one of the following options:

- ONE ROW PER MATCH (default option)
- ONE ROW PER VERTEX (vertex variable)
- ONE ROW PER STEP(edge_source_variable,edge_variable,edge_destination_variable)

For example, the following PGQL query uses the ONE ROW PER STEP option:

```
SELECT v1.ACCT_ID AS src_no, k.TXN_AMOUNT, v2.ACCT_ID AS dest_no FROM MATCH ALL SHORTEST (a:Accounts) -[e:transfers]->+ (b:Accounts) ONE ROW PER STEP( v1,k,v2 ) WHERE a.ACCT ID = 284 AND b.ACCT ID = 616
```

It is important to note that the ONE ROW PER STEP option only supports paths with a minimal hop greater than 0 and hence * quantifier is not supported with this option.

On execution, the preceding query retrieves one row for every edge on the path that is bound by the corresponding source and destination vertices:

+-						+
 -	src_no		TXN_AMOUNT		dest_no	 -
1	 744	. — ·	1000.0	. — ·	 616	+
i	772	i	1000.0	i	744	i
ĺ	284		1000.0		772	
	744		1000.0		616	
	772		1500.0		744	
	284		1000.0		772	
+-						+

You can also use the Graph Visualization tool to visualize edges using ONE ROW PER STEP along a path:



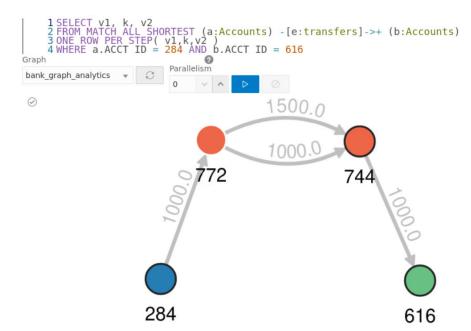


Figure 16-1 Visualizing Unnesting of Variable-Length Path Queries

An example for a query with the ONE ROW PER VERTEX option is as follows:

```
SELECT k.acct_id AS id, k.acct_name AS name
FROM MATCH ANY SHORTEST (a:Accounts) ((src:Accounts)-[e:transfers]->)
{1,3} (b:Accounts)
ONE ROW PER VERTEX(k)
WHERE a.acct_id=284 AND b.acct_id=616
```

On execution, the preceding query retrieves one row per vertex along a path:

+-		 	+
	id	name	
+-		 	+
	616	Account4	
	744	Account3	
	772	Account2	
	284	Account1	
+-		 +	-

Built-in Function Support for Recursive Path Unnesting Queries

PGQL supports the following two built-in functions, which can be used in combination with any of the path unnesting option (ONE ROW PER VERTEX, ONE ROW PER STEP or ONE ROW PER MATCH):

• MATCH_NUMBER(k): Returns a unique per-path identifier for each unnested path (that is, if two rows come from the same path, they have the same MATCH_NUMBER(k)).



• **ELEMENT_NUMBER (k):** Returns the element number of a vertex or an edge along a path. Vertices are numbered with odd numbers, the leftmost vertex is numbered 1, the second 3, then 5 and so on. Edges are assigned with even numbers, starting with 2 for the leftmost edge, 4 for the next one, and so on.

For example, the following PGQL query uses the MATCH_NUMBER(k) and ELEMENT_NUMBER(k) functions with ONE ROW PER VERTEX option:

```
SELECT k.*, match_number(k), element_number(k)
FROM MATCH ANY SHORTEST (a:Accounts) -[e:transfers]->* (b:Accounts) ONE ROW
PER VERTEX ( k )
WHERE a.acct id = 284 AND b.acct id = 616
```

The preceding query produces the following output on execution. Note that the <code>element_number(k)</code> returned for the vertices are odd numbered values. Since the preceding query uses <code>ANY</code> path pattern, there is only one arbitrary path displayed in the output. Therefore <code>match number(k)</code> is the same for all the rows in the path.

+		 	 	 	+
1	ACCT_ID	ACCT_NAME	<pre>match_number(k)</pre>	<pre>element_number(k)</pre>	I
+		 	 	 	+
	616	Account	0	7	
	744	Account	0	5	
	772	Account	0	3	
	284	Account	0	1	
+		 	 	 	+

The following example shows a PGQL query using ${\tt MATCH_NUMBER(k)}$ and ${\tt ELEMENT\ NUMBER(k)}$ functions with ONE ROW PER STEP option:

```
SELECT v1.acct_id AS src_no,k.txn_amount,v2.acct_id AS dest_no,
match_number(k), element_number(k)
FROM MATCH ALL SHORTEST (a:Accounts) -[e:transfers]->+ (b:Accounts)
ONE ROW PER STEP( v1,k,v2 )
WHERE a.acct_id = 284 AND b.acct_id = 616
```

The preceding query output is as shown. Note that there are two paths identified by match_number(k) and the edges are displayed with even numbered element_number(k) values.

+	src_no	 	txn_amount	 	dest_no	 	match_number(k)	 	element_number(k)	+
	744 772		1000.0	 	616 744		0		6 4	
İ	284 744		1000.0		772 616		0 1		2	
	772 284		1500.0 1000.0	 	744 772		1		4 2	



16.3.3 Using INTERVAL Literals in PGQL Queries

You can use INTERVAL literals in PGQL queries to add or subtract intervals to or from PGQL temporal data types respectively.

See the PGQL 1.5 Specification for the supported temporal data types. An INTERVAL type is a period of time, which consists of the keyword "INTERVAL" followed by a numeral and a temporal unit. For example, INTERVAL '1' DAY.

The following table shows the valid temporal units that are supported in INTERVAL values:

Table 16-3 Valid values for fields in INTERVAL values

Keyword	Supported Valid Values
YEAR	Unconstrained except by <interval field="" leading="" precision=""></interval>
MONTH	Months (within years) (0-11)
DAY	Unconstrained except by <interval field="" leading="" precision=""></interval>
HOUR	Hours (within days) (0-23)
MINUTE	Minutes (within hours) (0-59)
SECOND	Seconds (within minutes) (0-59.999)

The following INTERVAL operations are supported on a temporal data type:

- TEMPORAL TYPE + INTERVAL
- INTERVAL + TEMPORAL TYPE
- TEMPORAL TYPE INTERVAL

For example, the following PGQL query retrieves persons where n.birthdate + INTERVAL '20' YEAR > TIMESTAMP '2000-01-01 00:00:00':

- JShell
- Java
- Python

JShell

opg4j> graph.queryPgql("SELECT n.name, n.birthdate FROM MATCH
(n:Person) WHERE n.birthdate + INTERVAL '20' YEAR > TIMESTAMP
'2000-01-01 00:00:00'").print()



Java

```
graph.queryPgql("SELECT n.name, n.birthdate FROM MATCH (n:Person) WHERE
n.birthdate + INTERVAL '20' YEAR > TIMESTAMP '2000-01-01 00:00:00'").print();
```

Python

```
graph.query_pgql("SELECT n.name, n.birthdate FROM MATCH (n:Person) WHERE
n.birthdate + INTERVAL '20' YEAR > TIMESTAMP '2000-01-01 00:00:00'").print()
```

On execution, the query output is as shown:

16.3.4 Limitations on Quantifiers

Although all quantifiers such as *, +, and $\{1,4\}$ are supported for reachability and shortest path patterns, the only quantifier that is supported for cheapest path patterns is * (zero or more).

16.3.5 Limitations on WHERE and COST Clauses in Quantified Patterns

The WHERE and COST clauses in quantified patterns, such as reachability patterns or shortest and cheapest path patterns, are limited to referencing a single variable only.

The following are examples of queries that are not supported because the WHERE or COST clauses reference two variables e and x instead of zero or one:

```
... PATH p AS (n) -[e] \rightarrow (m) WHERE e.prop > m.prop ... 
 ... SHORTEST ( (n) (-[e] \rightarrow) (x) WHERE e.prop + x.prop > 10) * (m) ) ... 
 ... CHEAPEST ( (n) (-[e] \rightarrow) (x) COST e.prop + x.prop ) * (m) ) ...
```

The following query is supported because the subquery only references a single variable $\tt a$ from the outer scope, while the variable $\tt c$ does not count since it is newly introduced in the subquery:

```
... PATH p AS (a) -> (b)

WHERE EXISTS ( SELECT * FROM MATCH (a) -> (c) ) ...
```



16.4 Java APIs for Executing CREATE PROPERTY GRAPH Statements

The easiest way to execute a CREATE PROPERTY GRAPH statement is through the PgxSession.executePgql (String statement) method.

Example 16-2 Executing a CREATE PROPERTY GRAPH statement

```
String statement =
     "CREATE PROPERTY GRAPH hr simplified "
   + " VERTEX TABLES ( "
        hr.employees LABEL employee "
            PROPERTIES ARE ALL COLUMNS EXCEPT ( job id, manager id,
department id ), "
   + " hr.departments LABEL department "
          PROPERTIES ( department id, department name ) "
   + " ) "
   + " EDGE TABLES ( "
   + " hr.employees AS works_at "
   + "
          SOURCE KEY ( employee id ) REFERENCES employees
(employee id) "
   + " DESTINATION departments "
   + "
          PROPERTIES ( employee_id ) "
   + " )";
session.executePgql(statement);
PgxGraph g = session.getGraph("HR SIMPLIFIED");
/**
* Alternatively, one can use the prepared statement API, for example:
PgxPreparedStatement stmnt = session.preparePgql(statement);
stmnt.execute();
stmnt.close();
PgxGraph g = session.getGraph("HR SIMPLIFIED");
```

16.5 Python APIs for Executing CREATE PROPERTY GRAPH Statements

You can create a property graph by executing the CREATE PROPERTY GRAPH statement through the Python API.

Creating a Property Graph Using the Python Client

Launch the Python client:

```
./bin/opg4py --base url https://localhost:7007 --user customer 360
```



Define and execute the CREATE PROPERTY GRAPH statement as shown:

```
statement = (
    "CREATE PROPERTY GRAPH "+ "<graph_name>" + " " +
    "VERTEX TABLES ( " +
    "bank_accounts " +
    "KEY(acct_id) " +
    "LABEL Account PROPERTIES (acct_id) " +
    ")" +
    "EDGE TABLES ( " +
    "bank_txns " +
    "KEY (txn_id) " +
    "SOURCE KEY (from_acct_id) REFERENCES bank_accounts (acct_id) " +
    "DESTINATION KEY (to_acct_id) REFERENCES bank_accounts (acct_id) " +
    "LABEL Transfer PROPERTIES(amount) " +
    ")")
>>> session.prepare pgql(statement).execute()
```

where *<graph_name>* is the name of the graph.

The graph gets created and you can verify through the get graph method:

```
>>> graph = session.get_graph("<graph_name>")
>>> graph
PgxGraph(name:<graph_variable>, v: 1000, e: 5001, directed: True,
memory(Mb): 0)
```

16.6 Java APIs for Executing SELECT Queries

This section describes the APIs to execute SELECT queries in the graph server (PGX).

- Executing SELECT Queries Against a Graph in the Graph Server (PGX)
 The PgxGraph.queryPgql (String query) method executes the query in the current session. The method returns a PgqlResultSet.
- Executing SELECT Queries Against a PGX Session
 The PgxSession.queryPgql (String query) method executes the given query in the session and returns a PgqlResultSet.
- Iterating Through a Result Set
 There are two ways to iterate through a result set: in a JDBC-like manner or using the Java Iterator interface.
- Printing a Result Set
 The following methods of PgqlResultSet (package oracle.pgx.api) are used to print a result set:

16.6.1 Executing SELECT Queries Against a Graph in the Graph Server (PGX)

The PgxGraph.queryPgql (String query) method executes the query in the current session. The method returns a PgqlResultSet.

The ON clauses inside the MATCH clauses can be omitted since the query is executed directly against a PGX graph. For the same reason, the INTO clauses inside the INSERT clauses can be omitted. However, if you want to explicitly specify graph names in the ON and INTO clauses, then those graph names have to match the actual name of the graph (PgxGraph.getName()).

16.6.2 Executing SELECT Queries Against a PGX Session

The PgxSession.queryPgql(String query) method executes the given query in the session and returns a PgqlResultSet.

The ON clauses inside the MATCH clauses, and the INTO clauses inside the INSERT clauses, must be specified and cannot be omitted. At this moment, all the ON and INTO clauses of a query need to reference the same graph since joining data from multiple graphs in a single query is not yet supported.

16.6.3 Iterating Through a Result Set

There are two ways to iterate through a result set: in a JDBC-like manner or using the Java Iterator interface.

For JDBC-like iterations, the methods in PgqlResultSet (package oracle.pgx.api) are similar to the ones in <code>java.sql.ResultSet</code>. A noteworthy difference is that PGQL's result set interface is based on the new date and time library that was introduced in Java 8, while <code>java.sql.ResultSet</code> is based on the legacy <code>java.util.Date</code>. To bridge the gap, PGQL's result set provides <code>getLegacyDate(..)</code> for applications that still use <code>java.util.Date</code>.

A PgqlResultSet has a cursor that is initially set before the first row. Then, the following methods are available to reposition the cursor:

```
next() : boolean
previous() : boolean
beforeFirst()
afterLast()
first() : boolean
last() : boolean
absolute(long row) : boolean
relative(long rows) : boolean
```

After the cursor is positioned at the desired row, the following getters are used to obtain values:

```
    getObject(int columnIdx) : Object
    getObject(String columnName) : Object
    getString(int columnIdx) : String
    getString(String columnName) : String
    getInteger(int columnIdx) : Integer
    getInteger(String columnName) : Integer
```



```
getLong(int columnIdx) : Long
getLong(String columnName) : Long
 getFloat(int columnIdx) : Float
 getFloat(String columnName) : Float
 getDouble(int columnIdx) : Double
 getDouble(String columnName) : Double
 getBoolean(int columnIdx) : Boolean
 getBoolean(String columnName) : Boolean
 getVertexLabels(int columnIdx) : Set<String>
 getVertexLabels(String columnName) : Set<String>
 getDate(int columnIdx) : LocalDate
 getDate(String columnName) : LocalDate
 getTime(int columnIdx) : LocalTime
 getTime(String columnName) : LocalTime
 getTimestamp(int columnIdx) : LocalDateTime
 getTimestamp(String columnName) : LocalDateTime
 getTimeWithTimezone(int columnIdx) : OffsetTime
 getTimeWithTimezone(String columnName) : OffsetTime
 getTimestampWithTimezone(int columnIdx) : OffsetDateTime
 getTimestampWithTimezone(String columnName) : OffsetDateTime
 getLegacyDate(int columnIdx) : java.util.Date
 getLegacyDate(String columnName) : java.util.Date
 getVertex(int columnIdx) : PgxVertex<ID>
 getVertex(String columnName) : PgxVertex<ID>
 getEdge(int columnIdx) : PgxEdge
 getEdge(String columnName) : PgxEdge
```

See the Java Documentation for more details.

Finally, there is a PgqlResultSet.close() which releases the result set's resources, and there is a PgqlResultSet.getMetaData() through which the column names and column count can be retrieved.

An example for result set iteration is as follows:

```
PgqlResultSet resultSet = g.queryPgql(
    " SELECT owner.name AS account_holder, SUM(t.amount) AS
total_transacted_with_Nikita "
    + " FROM MATCH (p:Person) -[:ownerOf]-> (account1:Account) "
    + " , MATCH (account1) -[t:transaction]- (account2) "
    + " , MATCH (account2:Account) <-[:ownerOf]- (owner:Person|Company)"</pre>
```

```
+ " WHERE p.name = 'Nikita' "
+ " GROUP BY owner");

while (resultSet.next()) {
   String accountHolder = resultSet.getString(1);
   long totalTransacted = resultSet.getLong(2);
   System.out.println(accountHolder + ": " + totalTransacted);
}

resultSet.close();
```

The output of the above example will look like:

```
Oracle: 4501
Camille: 1000
```

In addition, the PgqlResultSet is also iterable via the Java Iterator interface. An example of a "for each loop" over the result set is as follows:

```
for (PgxResult result : resultSet) {
   String accountHolder = result.getString(1);
   long totalTransacted = result.getLong(2);
   System.out.println(accountHolder + ": " + totalTransacted);
}
```

The output of the above example will look like:

```
Oracle: 4501
Camille: 1000
```

Note that the same getters that are available for PgqlResultSet are also available for PgxResult.

16.6.4 Printing a Result Set

The following methods of PgqlResultSet (package oracle.pgx.api) are used to print a result set:

```
• print() : PgqlResultSet
```

- print(long numResults) : PgqlResultSet
- print(long numResults, int from) : PgqlResultSet
- print(PrintStream printStream, long numResults, int from):
 PgqlResultSet

For example:

```
g.queryPgql("SELECT COUNT(*) AS numPersons FROM MATCH
(n:Person)").print().close()
+----+
| numPersons |
+-----+
```



```
| 3 |
```

Another example:

16.7 Java APIs for Executing UPDATE Queries

The UPDATE queries make changes to existing graphs using the INSERT, UPDATE, and DELETE operations as detailed in the section Graph Modification of the PGQL 1.3 specification.

Note that INSERT allows you to insert new vertices and edges into a graph, UPDATE allows you to update existing vertices and edges by setting their properties to new values, and DELETE allows you to delete vertices and edges from a graph.

- Updatability of Graphs Through PGQL
 Graph data that is loaded from the Oracle RDBMS or from CSV files into the PGX is not updatable through PGQL right away.
- Executing UPDATE Queries Against a Graph in the Graph Server (PGX)

 To execute UPDATE queries against a graph, use the PgxGraph.executePgql (String query) method.
- Executing UPDATE Queries Against a PGX Session

 For now, there is no support for executing UPDATE queries against a PgxSession and therefore, updates always have to be executed against a PgxGraph. To obtain a graph from a session, use the PgxSession.getGraph(String graphName) method.
- Altering the Underlying Schema of a Graph
 The INSERT operations can only insert vertices and edges with known labels and properties. Similarly, UPDATE operations can only set values of known properties. Thus, new data must always conform to the existing schema of the graph.

16.7.1 Updatability of Graphs Through PGQL

Graph data that is loaded from the Oracle RDBMS or from CSV files into the PGX is not updatable through PGQL right away.

First, you need to create a copy of the data through the PgxGraph.clone() method. The resulting graph is fully updatable.

Consider the following example:

Additionally, there is also a PgxGraph.cloneAndExecutePgql(String query, String graphName) method that combines the last two steps from above example into a single step:

Note that graphs that are created through PgxGraph.clone() are local to the session. However, they can be shared with other sessions through the PgxGraph.publish(...) methods but then they are no longer updatable through PGQL. Only session-local graphs are updatable but persistent graphs are not.

16.7.2 Executing UPDATE Queries Against a Graph in the Graph Server (PGX)

To execute UPDATE queries against a graph, use the PgxGraph.executePgql(String query) method.

The following is an example of INSERT query:



Note that the INTO clause of the INSERT can be omitted. If you use an INTO clause, the graph name in the INTO clause must correspond to the name of the PGX graph (PgxGraph.getName()) that the guery is executed against.

The following is an example of UPDATE query:

The following is an example of DELETE query:

16.7.3 Executing UPDATE Queries Against a PGX Session

For now, there is no support for executing UPDATE queries against a PgxSession and therefore, updates always have to be executed against a PgxGraph. To obtain a graph from a session, use the PgxSession.getGraph(String graphName) method.

16.7.4 Altering the Underlying Schema of a Graph

The INSERT operations can only insert vertices and edges with known labels and properties. Similarly, UPDATE operations can only set values of known properties. Thus, new data must always conform to the existing schema of the graph.

However, some PGX APIs exist for updating the schema of a graph: while no APIs exist for adding new labels, new properties can be added through the

PgxGraph.createVertexProperty(PropertyType type, String name) and PgxGraph.createEdgeProperty(PropertyType type, String name) methods. The new properties are attached to each vertex/edge in the graph, irrespective of their labels. Initially the properties are assigned a default value but then the values can be updated through the UPDATE statements.

Consider the following example:



16.8 PGQL Queries with Partitioned IDs

You can retrieve partitioned IDs using the id() function in PGQL.

PGQL SELECT Queries

The following are a few examples to retrieve partitioned IDs using PGQL SELECT queries:

```
g.queryPgql("SELECT id(n) FROM MATCH(n)").print().close()
```

This prints an output similar to:

The output is printed as shown:

```
+-----+
| name |
+-----+
| User1 |
+-----+

g.queryPgql("SELECT LABEL(n), n.name from MATCH(n) WHERE n.id =
1").print().close()
```

The output is printed as shown:

```
+-----+
| label(n) | name |
+-----+
| Accounts | User1 |
+-----+
```

PGX automatically creates a unique index for keys so that queries with predicates such as $\mathtt{WHERE}\ id(n) = \mathtt{'Accounts}(1)\mathtt{'}$ and $\mathtt{WHERE}\ n.id = 1$ can be efficiently processed by retrieving the vertex in constant time.



Using Bind Variables

Partitioned IDs can also be passed as bind values into a PgxPreparedStatement.

For example:

```
PgxPreparedStatement statement = g.preparePgql("SELECT n.name FROM MATCH (n)
WHERE id(n) = ?")
statement.setString(1, "Accounts(1)")
statement.executeQuery().print().close()
```

This prints the output as shown:

```
+----+
| name |
+----+
| User1 |
```

PGQL INSERT Queries

In INSERT queries, you must provide a value for the key property if a key property exists. The value is then used for the vertex or edge key.

For example you can execute an INSERT as shown:

```
g.executePgql("INSERT VERTEX v LABELS (Accounts) PROPERTIES (v.id = 1001,
v.name = 'User1001')")
```

The inserted values can be verified as shown:

```
g.queryPgql("SELECT id(n), n.name FROM MATCH(n) WHERE n.id =
1001").print().close()
```

This prints the output:

+		 	+
	id(n)	name	
+		 	+
	Accounts (1001)	User1001	

16.9 Security Tools for Executing PGQL Queries

To safeguard against query injection, bind variables can be used in place of literals while printIdentifier (String identifier) can be used in place of identifiers like graph names, labels, and property names.

Using Bind Variables

There are two reasons for using bind variables:



Using Identifiers in a Safe Manner

When you create a query through string concatenation, not only literals in queries pose a security risk, but also identifiers like graph names, labels, and property names do. The only problem is that bind variables are not supported for such identifier. Therefore, if these identifiers are variable from the application's perspective, then it is recommended to protect against query injection by passing the identifier through the

oracle.pgql.lang.ir.PgqlUtils.printIdentifier(String identifier)
method.

16.9.1 Using Bind Variables

There are two reasons for using bind variables:

- It protects against query injection.
- It speeds up query execution because the same bind variables can be set multiple times without requiring recompilation of the query.

To create a prepared statement, use one of the following two methods:

- PgxGraph.preparePgql(String query) : PgxPreparedStatement
- PgxSession.preparePgql(String query): PgxPreparedStatement

The PgxPreparedStatement (package oracle.pgx.api) returned from these methods have setter methods for binding the bind variables to values of the designated data type.

```
PreparedStatement stmnt = g.preparePgql(
   "SELECT v.id, v.dob " +
   "FROM MATCH (v) " +
   "WHERE v.firstName = ? AND v.lastName = ?");
stmnt.setString(1, "Camille");
stmnt.setString(2, "Mullins");
ResultSet rs = stmnt.executeQuery();
```

Each bind variable in the query needs to be set to a value using one of the following setters of PgxPreparedStatement:

- setBoolean(int parameterIndex, boolean x)
- setDouble(int parameterIndex, double x)
- setFloat(int parameterIndex, float x)
- setInt(int parameterIndex, int x)
- setLong(int parameterIndex, long x)
- setDate(int parameterIndex, LocalDate x)
- setTime(int parameterIndex, LocalTime x)
- setTimestamp(int parameterIndex, LocalDateTime x)
- setTimeWithTimezone(int parameterIndex, OffsetTime x)
- setTimestampWithTimezone(int parameterIndex, OffsetDateTime x)



setArray(int parameterIndex, List<?> x)

Once all the bind variables are set, the statement can be executed through:

- PgxPreparedStatement.executeQuery()
 - For SELECT queries only
 - Returns a ResultSet
- PgxPreparedStatement.execute()
 - For any type of statement
 - Returns a Boolean to indicate the form of the result: true in case of a SELECT query, false otherwise
 - In case of SELECT, the ResultSet can afterwards be accessed through PgxPreparedStatement.getResultSet()

In PGQL, bind variables can be used in place of literals of any data type, including array literals. An example query with a bind variable to is set to an instance of a String array is:

```
List<String> countryNames = new ArrayList<String>();
countryNames.add("Scotland");
countryNames.add("Tanzania");
countryNames.add("Serbia");

PreparedStatement stmnt = g.preparePgql(
   "SELECT n.name, n.population " +
   "FROM MATCH (c:Country) " +
   "WHERE c.name IN ?");
ResultSet rs = stmnt.executeQuery();
```

Finally, if a prepared statement is no longer needed, it is closed through PgxPreparedStatement.close() to free up resources.

16.9.2 Using Identifiers in a Safe Manner

When you create a query through string concatenation, not only literals in queries pose a security risk, but also identifiers like graph names, labels, and property names do. The only problem is that bind variables are not supported for such identifier. Therefore, if these identifiers are variable from the application's perspective, then it is recommended to protect against query injection by passing the identifier through the

```
oracle.pgql.lang.ir.PgqlUtils.printIdentifier(String identifier) method.
```

Given an identifier string, the method automatically adds double quotes to the start and end of the identifier and escapes the characters in the identifier appropriately.

Consider the following example:

```
String graphNamePrinted = printIdentifier("my graph name with \" special %
characters ");
PreparedStatement stmnt = g.preparePgql(
    "SELECT COUNT(*) AS numVertices FROM MATCH (v) ON " + graphNamePrinted);
```



16.10 Best Practices for Tuning PGQL Queries

This section describes best practices regarding memory allocation, parallelism, and query planning.

Memory Allocation

The graph server (PGX) has on-heap and off-heap memory, the earlier being the standard JVM heap while the latter being a separate heap that is managed by PGX. Just like graph data, intermediate and final results of PGQL queries are partially stored on-heap and partially off-heap. Therefore, both heaps are needed.

Parallelism

By default, all available processor threads are used to process PGQL queries. However, if needed, the number of threads can be limited by setting the parallelism option of the graph server (PGX).

Query Plan Explaining

The PgxGraph.explainPgql (String query) method is used to get insight into the query plan of the query. The method returns an instance of Operation (package oracle.pgx.api) which has the following methods:

16.10.1 Memory Allocation

The graph server (PGX) has on-heap and off-heap memory, the earlier being the standard JVM heap while the latter being a separate heap that is managed by PGX. Just like graph data, intermediate and final results of PGQL queries are partially stored on-heap and partially off-heap. Therefore, both heaps are needed.

In case of the on-heap memory, the default maximum is chosen upon startup of the JVM, but it can be overwritten through the -Xmx option.

In case of the off-heap, there is no maximum set by default and the off-heap memory usage, therefore, keeps increasing automatically until it depletes the system resources, in which case the operation is canceled, it's memory is released, and an appropriate exception is passed to the user. If needed, a maximum off-heap size can be configured through the <code>max_off_heap_size</code> option in the graph server (PGX).

A ratio of 1:1 for on-heap versus off-heap is recommended as a good starting point to allow for the largest possible graphs to be loaded and queried. See Configuring On-Heap Limits for the steps to configure the on-heap memory size.

16.10.2 Parallelism

By default, all available processor threads are used to process PGQL queries. However, if needed, the number of threads can be limited by setting the parallelism option of the graph server (PGX).

See Configuration Parameters for the Graph Server (PGX) Engine for more information on the graph server configuration parameters.



16.10.3 Query Plan Explaining

The PgxGraph.explainPgql (String query) method is used to get insight into the query plan of the query. The method returns an instance of Operation (package oracle.pgx.api) which has the following methods:

- print(): for printing the operation and its child operations
- getOperationType(): for getting the type of the operation
- getPatternInfo(): for getting a string representation of the operation
- getCostEstimate(): for getting the cost of the operation
- getTotalCostEstimate(): for getting the cost of the operations and its child operations
- getCardinatlityEstimate(): for getting the expected number of result rows
- getChildren(): for accessing the child operations

Consider the following example:

```
g.explainPgql("SELECT COUNT(*) FROM MATCH (n) -[e1]-> (m) -[e2]->
(o)").print()
\--- GROUP BY GroupBy {"cardinality":"42", "cost":"42",
"accumulatedCost":"58.1"}
    \--- (m) -[e2]-> (o) NeighborMatch {"cardinality":"3.12",
"cost":"3.12", "accumulatedCost":"16.1"}
    \--- (n) -[e1]-> (m) NeighborMatch {"cardinality":"5", "cost":"5",
"accumulatedCost":"13"}
    \--- (n) RootVertexMatch {"cardinality":"8", "cost":"8",
"accumulatedCost":"8"}
```

In the above example, the print () method is used to print the query plan.

If a query plan is not optimal, it is often possible to rewrite the query to improve its performance. For example, a SELECT query may be split into an UPDATE and a SELECT query as a way to improve the total runtime.

Note that the graph server (PGX) does not provide a hint mechanism.

Also, printing the guery plan shows the filters used in the guery. For example:

```
g.explainPgql("SELECT id(n) FROM MATCH (n)-[e]->(m) WHERE " +
...> "id(n) > 500 " +
...> "AND id(n) < 510 " +
...> "AND id(n) <> 509 " +
...> "AND id(n) <> 507 ").print()
\--- Projection {"cardinality":"146", "cost":"0", "accumulatedCost":"175"}
   \--- (n) -[e]-> (m) NeighborMatch {"cardinality":"146", "cost":"146",
"accumulatedCost":"175"}
   \--- (n) RootVertexMatch {"cardinality":"29.2", "cost":"29.2",
"accumulatedCost":"29.2"}
   WHERE $filter1
filter1: (id(n) <> 509) AND
   (id(n) <> 507) AND
```



```
(id(n) > 500) AND (id(n) < 510)
```

In the preceding example, since the query has filters that spans more than three lines, the filters are shown displayed below the query plan. If the filters are less than three lines, then the filters are shown directly within the query plan tree as shown:



17

REST Endpoints for the Graph Server

This section explains the Graph Server REST endpoints:

The following are the available REST endpoints:

Note:

The examples shown in the REST endpoints assume that:

- The PGX server is up and running on https://localhost:7007.
- Linux with cURL is installed. cURL is used to demonstrate how to access the graph.publish API using the CA certificate for verifying the graph server.
- Login
- List Graphs
- Run a PGQL Query
- Get User
- Logout
- Asynchronous REST Endpoints

17.1 Login

HTTP Request: POST https://localhost:7007/ui/v1/login/

Authentication: Uses cookie-based authentication.

Table 17-1 Parameters

Parameter	Parameter Type	Value	Required
Content-type	Header	application/json	Yes
username	Body	<username></username>	Yes
password	Body	<password></password>	Yes



Table 17-1 (Cont.) Parameters

Parameter	Parameter Type	Value	Required
baseUrl	Body	<pre><baseurl> to point to the graph server (PGX) or the database</baseurl></pre>	Optional. If empty, the pgx.base_ur l parameter value in the web.xml file will be used. See Table 18-1 for the location of the web.xml file.
pgqlDriver	Body	Valid PGQL driver configuration values are: • pgxDriver: for PGQL on the graph server (PGX) • pgqlDriver: for PGQL on Oracle Database	Yes
sessionId	Body	sessionId from graph server (PGX)	Optional

Request

The following curl command signs the user in to the graph server:

```
curl --cacert /etc/oracle/graph/ca_certificate.pem -c cookie.txt -X
POST -H "Content-Type: application/json" -d '{"username":
   "<username>", "password": "<password>", "pgqlDriver":
   "<pgqlDriver>","baseUrl": "<baseUrl>", "sessionId": "<sessionId>" }'
https://localhost:7007/ui/v1/login/
```

Response: The username used for the login. For example:

"oracle"

On successful login, the server session cookie is stored in a cookie file, <code>cookie.txt</code>. Use this cookie file, in the subsequent calls to the API.

17.2 List Graphs

GET /v2/graphs

HTTP Request: GET https://localhost:7007/ui/v2/graphs
Request

The following \mathtt{curl} command lists all the graphs to which the user has access along with the schema information:

curl --cacert /etc/oracle/graph/ca_certificate.pem -b cookie.txt
'https://localhost:7007/ui/v2/graphs'



Response: The list of graphs available for the current user along with the schema details. For example:

Also, note that the schema parameter will be NULL for graphs created in the graph server (PGX).

GET /v1/graphs



The /v1/graphs endpoint may be deprecated in a future release of Graph Server and Client. Therefore, it is recommended that you use the /v2/graphs endpoint to list all the graphs for a user.

HTTP Request: GET https://localhost:7007/ui/v1/graphs
Request

The following curl command lists all the graphs that belong to the user:

curl --cacert /etc/oracle/graph/ca_certificate.pem -b cookie.txt 'https://
localhost:7007/ui/v1/graphs'

Response: The list of graphs available for the current user. For example:

["hr", "bank graph analytics"]

17.3 Run a PGQL Query

HTTP Request: GET https://localhost:7007/ui/v1/query?
pgql=<PGQL_query>&graph=<graph_name>¶llelism=<parallelism_value>&size=<size_
value>&formatter=<formatter_value>

Table 17-2 Query Parameters

Parameter	Description	Values	Required
pgql	PGQL query string	<pgql_query></pgql_query>	Yes
graph	Name of the graph	<graph_name></graph_name>	Optional, only if the pgql query parameter contains the graph name. Otherwise, it is required.



Table 17-2 (Cont.) Query Parameters

Parameter	Description	Values	Required
parallelism	Degree of Parallelism	<pre><parallelism_value></parallelism_value></pre>	Optional. Default value depends on the PGQL driver configuration: • pgxDriver: <number-of-cpus> See parallelism in Table 20-1. • pgqlDriver: 1</number-of-cpus>
size	Fetch size (= the number of rows) of the query result	<size_value></size_value>	Optional. Default size value is 100.
formatter	Formatter of the graph	<formatter_value></formatter_value>	Optional. Supported formatter options are: datastudio gvt Default value is datastudio.

Request

The following curl command executes PGQL Query on a property graph:

```
curl --cacert /etc/oracle/graph/ca_certificate.pem -b cookie.txt
'https://localhost:7007/ui/v1/query?pgql=SELECT%20e%0AMATCH%20()-
%5Be%5D-%3E()%0ALIMIT%205&graph=hr&size=100'
```

Response: The PGQL query result in JSON format.

```
"name": "bank graph analytics 2",
"resultSetId": "pgql 14",
"graph": {
  "idType": "number",
  "vertices": [
      " id": "1",
      "p": [],
      "1": [
       "Accounts"
      ],
      "g": [
       "anonymous 1"
    },
      " id": "418",
      "p": [],
      "1": [
        "Accounts"
      ],
```



```
"g": [
    "anonymous 2"
  },
  {
    " id": "259",
    "p": [],
   "1": [
    "Accounts"
    ],
    "g": [
    "anonymous_2"
],
"edges": [
   " id": "0",
    "p": [
     {
      "n": "AMOUNT",
      "v": "1000.0",
      "s": false
     }
    ],
    "1": [
    "Transfers"
    ],
    "g": [
    "e"
    ],
    "s": "1",
    "d": "259",
    "u": false
    " id": "1",
    "p": [
     {
      "n": "AMOUNT",
"v": "1000.0",
      "s": false
     }
    ],
    "1": [
    "Transfers"
    ],
    "g": [
    "e"
    ],
    "s": "1",
    "d": "418",
    "u": false
],
```



```
"paths": [],
   "totalNumResults": 2
},
   "table":
"e\nPgxEdge[provider=Transfers,ID=0]\nPgxEdge[provider=Transfers,ID=1]"
}
```

17.4 Get User

HTTP Request: GET https://localhost:7007/ui/v1/user

Request

The following curl command gets the name of the current user:

```
curl --cacert /etc/oracle/graph/ca_certificate.pem -b cookie.txt
'https://localhost:7007/ui/v1/user'
```

Response: The name of the current user. For example:

```
"oracle"
```

17.5 Logout

HTTP Request: POST https://localhost:7007/ui/v1/logout/

Request

The following curl command is to successfully log out from the Graph Visualization application:

```
curl --cacert /etc/oracle/graph/ca_certificate.pem -b cookie.txt -X
POST 'https://localhost:7007/ui/v1/logout/'
```

Response: None

On successful logout, the server returns HTTP status code 200 and the session token from the cookie.txt file will no longer be valid.

17.6 Asynchronous REST Endpoints

The graph server REST endpoints support cancellation of queries.

In order to be able to cancel queries, you need to send the query using the following asynchronous REST endpoints:

- Run a PGQL Query Asynchronously
- Check a Query Completion
- Cancel a Query Execution
- Retrieve a Query Result



17.6.1 Run a PGQL Query Asynchronously

HTTP Request: GET https://localhost:7007/ui/v1/async-query?pgql=<PGQL
query>&graph><graph>¶llelism=<value>&size=<size value>

See Table 17-2 for more information on query parameters.

Request

The following curl command executes a PGQL query asynchronously on a property graph:

curl --cacert /etc/oracle/graph/ca_certificate.pem -b cookie.txt 'https://
localhost:7007/ui/v1/async-query?pgql=SELECT%20e%0AMATCH%20()-%5Be%5D-%3E()
%0ALIMIT%205&graph=hr¶llelism=&size=100'

Response: None.



An error message will be returned in case the query is malformed or if the graph does not exist.

17.6.2 Check a Query Completion

HTTP Request: GET https://localhost:7007/ui/v1/async-query-complete

Request

The following curl command checks if the PGQL query execution is completed:

curl --cacert /etc/oracle/graph/ca_certificate.pem -b cookie.txt 'https://
localhost:7007/ui/v1/async-query-complete'

Response: A boolean that indicates if the query execution is completed. For example,

true



You do not have to specify any request ID, as the currently executing query is attached to your HTTP session. You can only have one query executing per session. For concurrent query execution, create multiple HTTP sessions by logging in multiple times.

17.6.3 Cancel a Query Execution

HTTP Request: DELETE https://localhost:7007/ui/v1/async-query

Request

The following curl command cancels a currently executing PGQL Query on a property graph:

```
curl -X DELETE --cacert /etc/oracle/graph/ca_certificate.pem -b
cookie.txt 'https://localhost:7007/ui/v1/async-query'
```

Response: Confirmation of the cancellation or an error message if the query has already completed execution.

17.6.4 Retrieve a Query Result

HTTP Request: GET https://localhost:7007/ui/v1/async-result



The endpoint, GET https://localhost:7007/ui/v1/async-result?
pgql=<PGQL query>&graph=<graph>¶llelism=<value>&size=<size
value>, to retrieve a query result is deprecated:

```
curl --cacert /etc/oracle/graph/ca_certificate.pem -b
cookie.txt 'https://localhost:7007/ui/v1/async-result?
pgql=SELECT%20e%0AMATCH%20()-%5Be%5D-%3E()
%0ALIMIT%205&graph=hr&parallelism=&size=100'
```

Request

The following curl command retrieves the result of a successfully completed query:

```
curl --cacert /etc/oracle/graph/ca_certificate.pem -b cookie.txt
'https://localhost:7007/ui/v1/async-result'
```

Response: The PGQL query result in JSON format.



```
},
 {
   " id": "418",
   "p": [],
   "1": [
    "Accounts"
   ],
   "g": [
    "anonymous 2"
 },
 {
   " id": "259",
   "p": [],
   "1": [
    "Accounts"
   "g": [
   "anonymous 2"
],
"edges": [
   "_id": "0",
   "p": [
     {
     "n": "AMOUNT",
     "v": "1000.0",
      "s": false
    }
   ],
   "1": [
    "Transfers"
   ],
   "g": [
    "e"
   ],
   "s": "1",
   "d": "259",
   "u": false
   " id": "1",
   "p": [
    {
     "n": "AMOUNT",
"v": "1000.0",
      "s": false
    }
   ],
   "1": [
    "Transfers"
   ],
   "g": [
```

```
"e"
        ],
        "s": "1",
        "d": "418",
        "u": false
    ],
   "paths": [],
    "totalNumResults": 2
  },
  "table":
"e\nPgxEdge[provider=Transfers,ID=0]\nPgxEdge[provider=Transfers,ID=1]"
}
```



Part VI

Graph Visualization Application

The Graph Visualization application enables interactive exploration and visualization of property graphs. It can also visualize graphs stored in the database.

- About the Graph Visualization Application
 The Graph Visualization application is a single-page web application that works with the graph server (PGX).
- Using the Graph Visualization Application
 Depending on the PGQL driver selected at the time of logging in, the Graph Visualization application is either connected to the database or to the graph server (PGX).



About the Graph Visualization Application

The Graph Visualization application is a single-page web application that works with the graph server (PGX).

The graph server can be deployed in embedded mode or in Apache Tomcat or Oracle WebLogic Server. Graph Visualization application takes PGQL queries as an input and renders the result visually. A rich set of client-side exploration and visualization features can reveal new insights into your graph data.

Graph Visualization application works with the graph server (PGX). It can visualize graphs that are have been loaded into the graph server (PGX), either preloaded when the graph server is started, or loaded at run-time by a client application and made available through the graph.publish() API.

- How does the Graph Visualization Application Work
 The Graph Visualization application exposes its own web interface and REST endpoint
 and can execute PGQL queries against the graph server (PGX) or the Oracle Database
 (PGQL on RDBMS).
- Kerberos Enabled Authentication for the Graph Visualization Application
 The Graph Visualization application can authenticate users with Kerberos authentication enabled.
- Embedding the Graph Visualization Library in a Web Application
 You can integrate the graph visualization component in a web application to visualize graph data.

18.1 How does the Graph Visualization Application Work

The Graph Visualization application exposes its own web interface and REST endpoint and can execute PGQL queries against the graph server (PGX) or the Oracle Database (PGQL on RDBMS).

By default, it uses PGX and therefore requires a running PGX server to function. Alternatively, you can configure Graph Visualization application to directly talk to the database via PGQL on RDBMS. Graph Visualization application does not have any UI to create graphs, it can only visualize graphs which are already loaded into PGX or Oracle Database. See REST Endpoints for the Graph Server for more information on the graph visualization REST endpoints.

See Running the Graph Visualization Application in Standalone Mode for more information on starting the Graph Visualization application.

18.2 Kerberos Enabled Authentication for the Graph Visualization Application

The Graph Visualization application can authenticate users with Kerberos authentication enabled.

Graph Visualization provides two different drivers to log in:

- Graph Server (PGX) Driver: To send your credentials (Kerberos ticket) to Graph Server.
- Database Driver: To send your credentials (Kerberos ticket) directly to the database.
- Prerequisite Requirements for Kerberos Authentication
- Preparing the Graph Visualization Application for Kerberos Authentication



Kerberos Enabled Authentication for the Graph Server (PGX)

18.2.1 Prerequisite Requirements for Kerberos Authentication

The system requirements for the respective PGQL drivers are as follows:

- **Graph Server (PGX) Driver:** See Prerequisite Requirements for enabling Kerberos authentication on the graph server (PGX).
- Database Driver:
 - The database must have Kerberos authentication enabled. See Configuring Kerberos Authentication for more information.
 - Both the database and the Kerberos Authentication Server need to be reachable from the host where the Graph Visualization application is running.
 - The database must be prepared for graph server authentication. That is, relevant graph roles have been granted to users who will log into the Graph Visualization application.

18.2.2 Preparing the Graph Visualization Application for Kerberos Authentication

In order to use Kerberos authentication, you must enter your Active Directory credentials in the Graph Visualization application login page. To enable Kerberos authentication for the Graph Visualization application, follow the steps shown:

1. Locate the web.xml file for your installation.

You can locate the WEB-INF/web.xml inside the Graph Visualization WAR file for your installation as shown in the following table:



Table 18-1 Location of WEB-INF/web.xml file

Type of Installa tion	WAR file	Location	
Standal one installati on (RPM)	graphviz- <version>- pgviz<graphviz-version>.war</graphviz-version></version>	/opt/oracle/graph/graphviz	
Apache Tomcat Deploy ment:	graphviz- <version>- pgviz<graphviz-version>- tomcat.war <version> denotes the downloaded Oracle Graph Server and Client version.</version></graphviz-version></version>	a.	Download oracle-graph-webapps- <version>.zip from Oracle Software Delivery Cloud</version>
		b.	Unzip the file into a directory of your choice.
		C.	Locate the .war file for deploying the Graph Visualization application to Tomcat. It follows the naming pattern: graphviz- <version>-pgviz<graphviz-version>-tomcat.war</graphviz-version></version>
Oracle WebLog ic Server	graphviz- <version>- pgviz<graphviz-version>-wls.war <version> denotes the downloaded Oracle Graph Server and Client version.</version></graphviz-version></version>	a.	Download oracle-graph-webapps- <version>.zip from Oracle Software Delivery Cloud</version>
Deploy ment	Graph Gerver and Gherit version.	b.	Unzip the file into a directory of your choice.
		C.	Locate the .war file for deploying the Graph Visualization application to Oracle WebLogic Server. It follows the naming pattern: graphviz- <version>-pgviz<graphviz- version="">-wls.war</graphviz-></version>

2. Extract the appropriate WAR file to a directory of your choice by executing the following command:

```
unzip graphviz-*.war -d <war-file-extraction-path>
```

3. Locate and open the WEB-INF/web.xml file for update using any file editor of your choice. For example:

```
cd <war-file-extraction-path>
vi WEB-INF/web.xml
```

4. Enable the graphviz.driver.auth.kerberos **parameter as shown**:



Setting this flag **true** initiates the Graph Visualization application to install its own okinit package.

5. Optionally. set the cache directory that will be used by the Graph Visualization application to temporarily store Kerberos tickets given by clients as shown

The default value is /dev/shm/graph_cache. If the directory does not exist, it will be automatically created upon server startup.

6. Optionally, set the maximum amount of concurrent Kerberos active sessions in the Graph Visualization application.

7. Optionally, modify the directory where okinit package will be installed, by updating the following parameter:

Note:

The default value is /tmp and you must have executable permission for the directory.

8. Optionally, set the following parameter if there is a location for an existing okinit package on your machine. In this case, the GraphVisualization application will not install its own okinit package.



Note:

The GraphVisualization application must have executable permission for the directory location.

9. Finally, after all the preceding updates, repackage the WAR file by executing the following commands:

```
cd <war-file-extraction-path>
jar -cvf <war-file-name> *
```

10. Redeploy the WAR file to the appropriate directory for your installation.

Kerberos authentication is enabled for the Graph Visualization Application.

18.3 Embedding the Graph Visualization Library in a Web Application

You can integrate the graph visualization component in a web application to visualize graph data.

The Oracle Graph Server and Client deployment contains a JavaScript library for the Graph Visualization component in the oracle-graph-visualization-library-23.1.0.zip file.

The Graph Visualization interface in the library supports:

- Custom vertex and edge styling based on its properties
- Interactive actions for graph exploration
- Tooltip with vertex and edge details
- Automatic legend
- Multiple graph layouts

See the Graph JavaScript API Reference for Property Graph Visualization for more information.

You can download the <code>oracle-graph-visualization-library-23.1.0.zip</code> file from Oracle Software Delivery Cloud and integrate the library in you web application.

See the demo application on GitHub for an example.



Using the Graph Visualization Application

Depending on the PGQL driver selected at the time of logging in, the Graph Visualization application is either connected to the database or to the graph server (PGX).

In both cases, the principal points of entry for the Graph Visualization application are the query editor and the graph list which displays the list of graphs existing either in the graph server (PGX) or in the database.

The following sections explain the application user interface and running PGQL queries for visualization in detail:

- Visualizing PGQL Queries on Graphs Loaded Into the Graph Server (PGX)
 To run PGQL queries on a graph loaded into the graph server (PGX), you must login to the Graph Visualization application by selecting the Graph Server advanced option in the login screen.
- Visualizing PGQL and SQL Graph Queries on Graphs in the Database
 To run PGQL or SQL graph queries on graphs in the database, you must login to the
 Graph Visualization application by selecting the Database advanced option in the login
 screen.
- Graph Visualization Modes
 The buttons on the right let you switch between two modes: Graph Manipulation and Zoom/Move.
- Graph Visualization Settings
 You can click the Settings gear icon to display the Graph Visualization settings window.
- Using the Geographical Layout
 The Graph Visualization application offers a choice of layouts for rendering graphs. One of them is the Geographical layout that will show the graph (vertices and edges) on a global map.
- Using Live Search
 Live Search lets you to search the displayed graph and add live fuzzy search score to
 each item, so you can create a Highlight which visually shows the results of the search in
 the graph immediately.
- Using URL Parameters to Control the Graph Visualization Application
 You can provide the Graph Visualization application input data through URL parameters
 instead of using the form fields of the user interface.

19.1 Visualizing PGQL Queries on Graphs Loaded Into the Graph Server (PGX)

To run PGQL queries on a graph loaded into the graph server (PGX), you must login to the Graph Visualization application by selecting the **Graph Server** advanced option in the login screen.

See Configuring the Graph Visualization Application for PGQL on Graph Server (PGX) for more information.

To run queries against a graph, select that graph. The query editor lets you write PGQL queries that can be visualized. (PGQL is the SQL-like query language supported by the Graph Visualization application.)



For graphs loaded into the graph server (PGX), the graph visualization application supports only PGQL SELECT queries. Modifying a graph using a PGQL INSERT, UPDATE or DELETE query is not supported.

Once the query is ready and the desired graph is selected, click the **Run** icon to execute the query. The following figure shows a query visualization identifying all edges that are directed edges from any vertex in the graph to any other vertex.

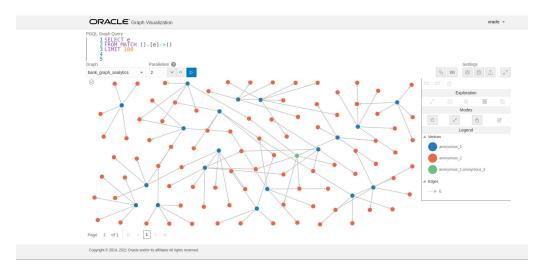


Figure 19-1 Query Visualization

When a query is successful, the graph visualization is displayed, including nodes and their connections. You can right-click a node or connection to display tooltip information, and you can drag the nodes around.

19.2 Visualizing PGQL and SQL Graph Queries on Graphs in the Database

To run PGQL or SQL graph queries on graphs in the database, you must login to the Graph Visualization application by selecting the **Database** advanced option in the login screen.

See Configuring the Graph Visualization Application for PGQL on Database for more information.

The user interface for the Graph Visualization application can vary depending on the database version you are using.

For Oracle Database 23ai: The following two tab options are displayed:



- PGQL: To run PGQL queries on property graph views. See Graph Pattern Matching queries in the PGQL specification for more details.
- SQL/PGQ: To run SQL graph queries on SQL property graphs. See SQL GRAPH_TABLE Queries for more details.

For Oracle Database 21c or earlier: Only the PGQL option in the preceding list is supported.

The following sections explain with examples visualization of PGQL and SQL graph queries.

- Visualizing PGQL Queries on PG Views
 You can create, query, modify and visualize property graph views (PG Views) in the
 database using the Graph Visualization application.
- Visualizing Graph Queries on SQL Property Graphs
 You can query and visualize a SQL property graph in the database using the Graph
 Visualization application.

19.2.1 Visualizing PGQL Queries on PG Views

You can create, query, modify and visualize property graph views (PG Views) in the database using the Graph Visualization application.

When connected to the database, you can run the following PGQL queries in the **PGQL** tab of the application.

CREATE PROPERTY GRAPH: To create a new property graph as shown:

Figure 19-2 Creating a PG View



• INSERT, UPDATE and DELETE: To modify an existing graph. For example:



Figure 19-3 Updating an Edge in a PG View

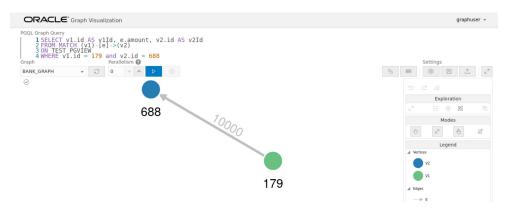


Figure 19-4 Deleting an Edge in a PG View



• SELECT: To query a property graph as shown:

Figure 19-5 Querying a PG View



DROP PROPERTY GRAPH: To delete a property graph as shown:

Figure 19-6 Dropping a PG View





19.2.2 Visualizing Graph Queries on SQL Property Graphs

You can query and visualize a SQL property graph in the database using the Graph Visualization application.

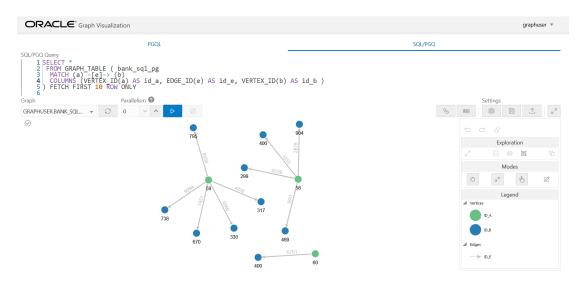
However, in order to visualize the vertices and edges of a <code>GRAPH_TABLE</code> query together with their IDs and all their labels and properties, the query must return the vertex ID, or edge ID, or both.

For example, the following figure shows the visualization of a SQL GRAPH_TABLE query on a SQL property graph. Note that the COLUMNS clause in the query uses the VERTEX_ID and EDGE ID operators.



In addition to the privileges mentioned in Privileges to Query a SQL Property Graph, you must also have the CREATE VIEW and CREATE MATERIALIZED VIEW privileges to query and visualize a SQL property graph in the Graph Visualization application.

Figure 19-7 GRAPH TABLE Query on SQL Property Graph



In the preceding example, note that the name of the graph is provided in the ${\tt GRAPH_TABLE}$ query.



SQL GRAPH_TABLE Queries for more information



19.3 Graph Visualization Modes

The buttons on the right let you switch between two modes: Graph Manipulation and Zoom/Move.

- **Graph Manipulation** mode lets you execute actions that modify the visualization. These actions include:
 - Drop removes selected vertices from visualization. Can also be executed from the tooltip.
 - Group selects multiple vertices and collapses them into a single one.
 - Ungroup selects a group of collapsed vertices and ungroups them.
 - Expand retrieves a configurable number of neighbors (hops) of selected vertices. Can also be executed from the tooltip.
 - Focus, like Expand, retrieves a configurable number of neighbors, but also drops all other vertices. Can also be executed from the tooltip.
 - Undo undoes the last action.
 - Redo redoes the last action.
 - Reset resets the visualization to the original state after the query.
- Zoom/Move mode lets you zoom in and out, as well as to move to another part of the visualization. The Pan to Center button resets the zoom and returns the view to the original one.

An additional mode, called **Sticky** mode, lets you cancel the action of dragging the nodes around.

19.4 Graph Visualization Settings

You can click the **Settings** gear icon to display the Graph Visualization settings window.

The settings window lets you modify some parameters for the visualization, and it has tabs for General, Visualization, and Highlights. The following figure shows this window, with the Visualization tab selected.



Ⅲ General > Visualization √ Highlights General Theme ☆ Light (Dark Edge Style Straight Curved Edge Marker ← Arrow None Similar Edges Collect Keep Page Size 100 Layouts Layout Force Edge Distance 120 Force Strength -30 0.3 Velocity Decay Vertex Padding 40

Figure 19-8 Graph Visualization Settings Window

The **General tab** includes the following:

- Number of hops: The configurable number of hops for the expand and focus actions.
- Truncate label: Truncates the label if it exceeds the maximum length.
- Max. visible label length: Maximum length before truncating.
- Show Label On Hover: Controls whether the label is shown on hover.
- Display the graph legend: Controls whether the legend is displayed.

The **Visualization tab** includes the following:

- Theme: Select a light or dark mode.
- Edge Style: Select straight or curved edges.
- Edge Marker: Select arrows or no edge marker. This only applies to directed edges.
- Similar Edges: Select keep or collect.
- Page Size: Specify how many vertices and edges are displayed per page.
- Layouts: Select between different layouts (random, grid, circle, concentric, ...).
- Vertex Label: Select which property to use as the vertex label.
- **Vertex Label Orientation**: Select the relative position of the vertex label.
- Edge Label: Select which property to use as the edge label.



The **Highlights tab** includes customization options that let you modify the appearance of edges and vertices. Highlighting can be applied based on conditions (filters) on single or multiple elements. The following figure shows a condition (country = United States) and visual highlight options for vertices.

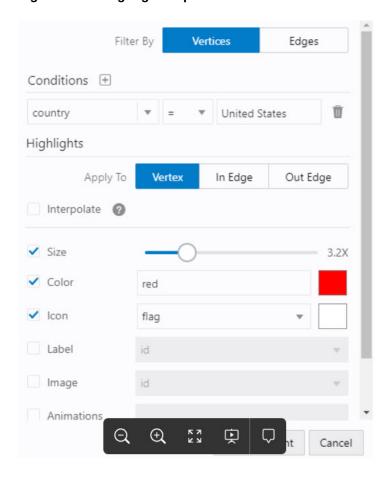


Figure 19-9 Highlights Options for Vertices

A filter for highlights can contain multiple conditions on any property of the element. The following conditions are supported.

- = (equal to)
- < (less than)
- <= (less than or equal to)</p>
- > (greater than)
- >= (greater than or equal to)
- != (not equal to)
- ~ (filter is a regular expression)
- * (any: like a wildcard, can match to anything)

The visual highlight customization options include:



- Edges:
 - Width
 - Color
 - Label
 - Style
 - Animations
- Vertices:
 - Size
 - Color
 - Icon
 - Label
 - Image
 - Animations

You can export and import highlight options by clicking the Save and Import buttons in the main window. **Save** lets you persist the highlight options, and **Load** lets you apply previously saved highlight options.

When you click **Save**, a file is saved containing a JSON object with the highlights configuration. Later, you can load that file to restore the highlights of the saved session.

19.5 Using the Geographical Layout

The Graph Visualization application offers a choice of layouts for rendering graphs. One of them is the Geographical layout that will show the graph (vertices and edges) on a global map.

The following figure shows a graph rendered on a geographical layout in the Graph Visualization application:

Figure 19-10 Geographical Layout





In order to view your vertices on a map, they must include a geographical location, in the form of a pair of properties that contain the longitude and latitude coordinates for that vertex. For example:

+.		 	 			-+
1	iata	city	longitude		latitude	
+.		 	 			-+
	SIN	Singapore	103.994003		1.35019	
	LAX	Los Angeles	-118.4079971		33.94250107	
	MUC	Munich	11.7861		48.353802	
	CDG	Paris	2.55		49.012798	
	LHR	London	-0.461941		51.4706	
+.		 	 			-+



You can use any name for the longitude and latitude properties (such as X and Y, or long and lat). But, you must ensure that the longitude/latitude pair are in the WGS84 system (GPS coordinates), and the coordinates are expressed in decimal degrees.

You can select the geographical layout in the Graph Visualization settings window as shown:

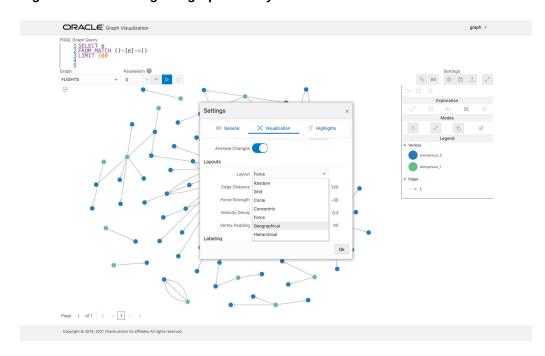


Figure 19-11 Setting Geographical Layout

Then, select the properties in your vertices that contain the geographical coordinates as shown:

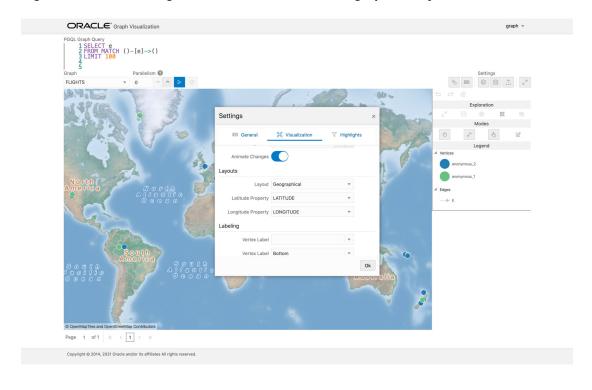


Figure 19-12 Selecting the Coordinates for the Geographical layout

You can now move around the map and zoom in/out using your mouse or trackpad. From now on, whenever you enter a new PGQL query, the map will automatically center and zoom the vertices returned by the query.

19.6 Using Live Search

Live Search lets you to search the displayed graph and add live fuzzy search score to each item, so you can create a Highlight which visually shows the results of the search in the graph immediately.

If you run a query, and a graph is displayed, you can add the live search, which is on the settings dialog. On the bottom of the General tab, you will see these options.

- **Enable Live Search:** Enables the Live Search feature, adds the search input to the visualization, and lets you further customize the search.
- **Enable Search In:** You can select whether you want to search the properties of Vertices, Edges, or both.
- **Properties To Search:** Based on what you selected for Enable Search In, you can set one or more properties to search in. For example, if you disable the search for edges but you had a property from edges selected, it will be stored and added back when you enable search for the edges again. (This also works for vertices.)
- **Advanced Settings:** You can fine-tune the search even more. Each of the advanced options is documented with context help, visible upon enabling.
 - Location: Determines approximately where in the text the pattern is expected to be found.



- Distance: Determines how close the match must be to the fuzzy location (specified by location). An exact letter match which is distance characters away from the fuzzy location would score as a complete mismatch. A distance of 0 requires the match be at the exact location specified, a distance of 1000 would require a perfect match to be within 800 characters of the location to be found using a threshold of 0.8.
- Maximum Pattern Length: The maximum length of the pattern. The longer the pattern (that is, the search query), the more intensive the search operation will be. Whenever the pattern exceeds this value, an error will be thrown.
- Min Char Match: The minimum length of the pattern. Whenever the pattern length is below this value, an error will be thrown.

When the search is enabled, the input will be displayed in the top left part of the Graph Visualization component. If you start typing, the search will add a score to every vertex or edge, based on the settings and the search match.

To be able to see the results visually, you have to add a **Highlight** with interpolation set to a **Live Search** score and other settings based on the desired visual change.

19.7 Using URL Parameters to Control the Graph Visualization Application

You can provide the Graph Visualization application input data through URL parameters instead of using the form fields of the user interface.

If you supply the parameters in the URL, the Graph Visualization application automatically executes the specified query and hides the input form fields from the screen, so only the resulting visualization output is visible. This feature is useful if you want to embed the resulting graph visualization into an existing application, such as through an iframe. However, it is important to note that the application must run on the same domain as the graph visualization application.

The following table specifies the available URL parameters:

Table 19-1 Available URL Parameters

Parameter Name	Value (must be URL encoded)	Туре	Optional?
graph	Graph name	string	No
parallelism	Degree of parallelism desired	number	Yes (defaults to server-side default parallelism)
query	PQL query	string	No

The following URL shows an example of visualizing the PGQL query SELECT v, e MATCH (v) -[e]-> () LIMIT 10 on graph myGraph with parallelism 4:

https://myhost:7007/ui/?query=SELECT%20v%2C%20e%20MATCH%20%28v%29%20-%5Be%5D-%3E%20%28%29%20LIMIT%2010&graph=myGraph¶llelism=4



Part VII

Graph Server (PGX) Advanced User Guide

Part II provides in-depth information on using the graph server (PGX) for advanced users.

Part II contains the following chapters:

- Configuring the Graph Server (PGX) and the Graph Client
 This chapter explains the configuration options for the graph server (PGX) and the graph client.
- Deploying Oracle Graph Server Behind a Load Balancer
 You can deploy multiple graph servers (PGX) behind a load balancer and connect clients to the servers through the load balancer.
- Namespaces and Sharing
 The graph server (PGX) supports separate namespaces that help you to organize your entities.
- PGX Programming Guides
 You can avail all the PGX functionalities through asynchronous Java APIs. Each asynchronous method has a synchronous equivalent, which blocks the caller thread until the server produces a response.
- Working with Files Using the Graph Server (PGX)
 This chapter describes in detail about working with different file formats to perform various actions like loading, storing, or exporting a graph using the Graph Server (PGX).
- Log Management in the Graph Server (PGX)
 The graph server (PGX) internally uses the SLF4J interface with Logback as the default logger implementation.



20

Configuring the Graph Server (PGX) and the Graph Client

This chapter explains the configuration options for the graph server (PGX) and the graph client.

- Configuration Parameters for the Graph Server (PGX) Engine
 You can configure the graph server (PGX) engine and the PGX run-time library by
 assigning a single JSON file to the graph server (PGX) at start up.
- Configuration Parameters for Connecting to the Graph Server (PGX)
 You can configure the graph server (PGX) to use the required options at startup.
- Configuration Parameters for the Graph Client
 You can configure the PGX graph client. All the parameters are available as command line options also.

20.1 Configuration Parameters for the Graph Server (PGX) Engine

You can configure the graph server (PGX) engine and the PGX run-time library by assigning a single JSON file to the graph server (PGX) at start up.

To pass the PGX engine configuration file to the graph server (PGX), see Passing the Configuration File to the Graph Server (PGX).

The PGX engine parameters are shown in the following table:

Table 20-1 Configuration Parameters for the Graph Server (PGX) Engine

Parameter	Туре	Description	Default
admin_request_cache_timeout	integer	After how many seconds admin request results get removed from the cache. Requests which are not done or not yet consumed are excluded from this timeout. Note: This is only relevant if PGX is deployed as a webapp.	
allow_idle_timeout_overwrite	boolean	If true, sessions can overwrite the default idle timeout.	true
allow_override_scheduling_information	boolean	If true, allow all users to override scheduling information like task weight, task priority, and number of threads	true
allow_task_timeout_overwrite	boolean	If true, sessions can overwrite the default task timeout.	true



Table 20-1 (Cont.) Configuration Parameters for the Graph Server (PGX) Engine

Parameter	Туре	Description	Default
allow_user_auto_refresh	boolean	If true, users may enable auto refresh for graphs they load. If false, only graphs mentioned in preload_graphs can have auto refresh enabled.	false
allowed_remote_loading_locations	array of string	Allow loading graphs into the PGX engine from remote locations (http, https, ftp, ftps, s3, hdfs). If empty, as by default, no remote location is allowed. If "*" is specified in the array, all remote locations are allowed. Only the value "*" is currently supported. Note that pre-loaded graphs are loaded from any location, regardless of the value of this setting. Note that this parameter reduces security and therefore use it only when needed.	[]
basic_scheduler_config	object	Configuration parameters for the fork join pool backend.	null
bfs_iterate_que_task_size	integer	Task size for BFS iterate QUE phase.	128
bfs_threshold_parent_read_based	number	Threshold of BFS traversal level items to switch to parent-read-based visiting strategy.	0.05
bfs_threshold_read_based	integer	Threshold of BFS traversal level items to switch to read-based visiting strategy.	1024
bfs_threshold_single_threaded	integer	Until what number of BFS traversal level items vertices are visited single-threaded.	128
character_set	string	Standard character set to use throughout PGX. UTF-8 is the default. Note: Some formats may not be compatible.	utf-8
<pre>cni_diff_factor_default</pre>	integer	Default diff factor value used in the common neighbor iterator implementations.	8
cni_small_default	integer	Default value used in the common neighbor iterator implementations, to indicate below which threshold a subarray is considered small.	128
<pre>cni_stop_recursion_default</pre>	integer	Default value used in the common neighbor iterator implementations, to indicate the minimum size where the binary search approach is applied.	96



Table 20-1 (Cont.) Configuration Parameters for the Graph Server (PGX) Engine

Parameter	Туре	Description	Default
dfs_threshold_large	integer	Value that determines at which number of visited vertices the DFS implementation will switch to data structures that are optimized for larger numbers of vertices.	4096
enable_csrf_token_checks	boolean	If true, the PGX webapp will verify the Cross-Site Request Forgery (CSRF) token cookie and request parameters sent by the client exist and match. This is to prevent CSRF attacks.	true
enable_gm_compiler	boolean	If true, enable dynamic compilation of PGX Algorithm API (or Green-Marl code) during runtime.	true
enable_shutdown_cleanup_hook	boolean	If true, PGX will add a JVM shutdown hook that will automatically shutdown PGX at JVM shutdown. Notice: Having the shutdown hook deactivated and not explicitly shutting down PGX may result in pollution of your temp directory.	true
enterprise_scheduler_config	object	Configuration parameters for the enterprise scheduler.	null
enterprise_scheduler_flags	object	[relevant for enterprise_scheduler] Enterprise scheduler-specific settings.	null
explicit_spin_locks	boolean	true means spin explicitly in a loop until lock becomes available. false means using JDK locks which rely on the JVM to decide whether to context switch or spin. Setting this value to true usually results in better performance.	true
file_locations	array of object	The file locations that can be used in the authorization-config.	[]
graph_algorithm_language	enum[GM_LEGA CY, GM, JAVA]	Front-end compiler to use.	JAVA
graph_validation_level	enum[low, high]	Level of validation performed on newly loaded or created graphs.	low
<pre>ignore_incompatible_backend_operations</pre>	boolean	If true, only log when encountering incompatible operations and configuration values in RTS or FJ pool. If false, throw exceptions.	false



Table 20-1 (Cont.) Configuration Parameters for the Graph Server (PGX) Engine

Parameter	Туре	Description	Default
<pre>in_place_update_consistency_model</pre>	enum[ALLLOW_INCONSISTENCIES, CANCEL_TASKS]	Consistency model used when in-place updates occur. Only relevant if in-place updates are enabled. Currently updates are only applied in place if the updates are not structural (Only modifies properties). Two models are currently implemented, one only delays new tasks when an update occurs, the other also delays running tasks.	ALLOW_I NCONSIS TENCIES
<pre>init_pgql_on_startup</pre>	boolean	If true PGQL is directly initialized on start-up of PGX. Otherwise, it is initialized during the first use of PGQL.	true
<pre>interval_to_poll_max</pre>	integer	Exponential backoff upper bound (in ms) to which -once reached, the job status polling interval is fixed	1000
<pre>java_home_dir</pre>	string	The path to Java's home directory. If set to <system-java-home-dir>, use the java.home system property.</system-java-home-dir>	<system -java-="" dir="" home-=""></system>
large_array_threshold	integer	Threshold when the size of an array is too big to use a normal Java array. This depends on the used JVM. (Defaults to Integer.MAX_VALUE - 3)	2147483 644
max_active_sessions	integer	Maximum number of sessions allowed to be active at a time.	1024
max_distinct_strings_per_pool	integer	[only relevant if string_pooling_strategy is indexed] Number of distinct strings per property after which to stop pooling. If the limit is reached, an exception is thrown.	65536
max_http_client_request_size	long	Maximum size in bytes of any http request sent to the PGX server over the REST API. Setting it to -1 allows requests of any size.	1048576



Table 20-1 (Cont.) Configuration Parameters for the Graph Server (PGX) Engine

Parameter	Туре	Description	Default
max_off_heap_size	integer	Maximum amount of off-heap memory (in megabytes) that PGX is allowed to allocate before an OutOfMemoryError will be thrown. Note that this limit is not	<availa ble- physica l- memory></availa
		guaranteed to never be exceeded, because of rounding and synchronization trade-offs. It only serves as threshold when PGX starts to reject new memory allocation requests.	
max_queue_size_per_session	integer	The maximum number of pending tasks allowed to be in the queue, per session. If a session reaches the maximum, new incoming requests of that session get rejected. A negative value means infinity or unlimited	-1
max_snapshot_count	integer	Number of snapshots that may be loaded in the engine at the same time. New snapshots can be created via auto or forced update. If the number of snapshots of a graph reaches this threshold, no more autoupdates will be performed, and a forced update will result in an exception until one or more snapshots are removed from memory. A value of zero indicates to support an unlimited amount of snapshots.	0
memory_allocator	<pre>enum[basic_a llocator, enterprise_a llocator]</pre>	The memory allocator to use.	basic_a llocato r
memory_cleanup_interval	integer	Memory cleanup interval in seconds.	5
min_array_compaction_threshold	number	Minimum value (only relevant for graphs optimized for updates) that can be used for the array_compaction_threshol d value in graph configuration. If a graph configuration attemps to use a value lower than the one specified by min_array_compaction_thre shold, it will use min_array_compaction_thre shold instead.	0.2



Table 20-1 (Cont.) Configuration Parameters for the Graph Server (PGX) Engine

Parameter	Туре	Description	Default
min_fetch_interval_sec	integer	For delta-refresh (only relevant if the graph format supports delta updates), the lowest interval at which a graph source is queried for changes. You can tune this value to prevent PGX from hanging due to too frequent graph delta-refreshing.	2
min_update_interval_sec	integer	For auto-refresh, the lowest interval after which a new snapshot is created, either by reloading the entire graph or if the format supports delta-updates, out of the cached changes (only relevant if the format supports delta updates). You can tune this value to prevent PGX from hanging due to too frequent graph auto-refreshing.	2
ms_bfs_frontier_type_strategy	<pre>enum[auto_gr ow, short, int]</pre>	The type strategy to use for MS-BFS frontiers.	auto_gr ow
num_spin_locks	integer	Number of spin locks each generated app will create at instantiation. Trade-off: a small number implies less memory consumption; a large number implies faster execution (if algorithm uses spin locks).	1024
parallelism	integer	Number of worker threads to be used in thread pool. Note: If the caller thread is part of another thread-pool, this value is ignored and the parallelism of the parent pool is used.	<number -of- cpus></number
<pre>pattern_matching_supernode_cache_thresh old</pre>	integer	Minimum number of a node's neighbor to be a supernode. This is for the pattern matching engine.	1000
pgx_realm	object	Configuration parameters for the realm.	null
pgx_server_base_url	string	This is used when deploying the graph server behind a load balancer to make clients before 21.3 backward compatible. The value should be set to the load balancer address.	null



Table 20-1 (Cont.) Configuration Parameters for the Graph Server (PGX) Engine

Parameter	Туре	Description	Default
pooling_factor	number	[only relevant if string_pooling_strategy is on_heap] This value prevents the string pool to grow as big as the property size, which could render the pooling ineffective.	0.25
preload_graphs	array of object	List of graph configs to be registered at start-up. Each item includes path to a graph config, the name of the graph and whether it should be published.	[]
random_generator_strategy	<pre>enum[non_det erministic, deterministi c]</pre>	Method of generating random numbers in PGX.	non_det erminis tic
random_seed	long	[relevant for deterministic random number generator only] Seed for the deterministic random number generator used in pgx. The default is -24466691093057031.	-244666 9109305 7031
readiness_memory_usage_ratio	number	Memory limit ratio that should be considered to detect if PGX server is ready. This is used by isReady API and the default value is 1.0	1.0
release_memory_threshold	number	Threshold percentage (decimal fraction) of used memory after which the engine starts freeing unused graphs. Examples: A value of 0.0 means graphs get freed as soon as their reference count becomes zero. That is, all sessions which loaded that graph were destroyed/timed out. A value of 1.0 means graphs never get freed, and the engine will throw OutOfMemoryErrors as soon as a graph is needed which does not fit in memory anymore. A value of 0.7 means the engine keeps all graphs in memory as long as total memory consumption is below 70% of total available memory, even if there is currently no session using them. When consumption exceeds 70% and another graph needs to get loaded, unused graphs get freed until memory consumption is below 70% again.	0.0
revisit_threshold	integer	Maximum number of matched results from a node to be cached.	4096



Table 20-1 (Cont.) Configuration Parameters for the Graph Server (PGX) Engine

Parameter	Туре	Description	Default
running_memory_usage_ratio	number	Memory limit ratio that should be considered to detect if PGX server is running. This is used by isRunning API and the default value is 1.0	1.0
scheduler	enum[basic_s cheduler, enterprise_s cheduler, low_latency_ scheduler]	The scheduler to use. basic_scheduler: uses a scheduler with basic features enterprise_scheduler: uses a scheduler with advanced enterprise features for running multiple tasks concurrently and providing better performance low_latency_scheduler: uses a scheduler that privileges latency of tasks over throughput or fairness across multiple sessions. The low_latency_scheduler is only available in embedded mode.	enterpr ise_sch eduler
session_idle_timeout_secs	integer	Timeout of idling sessions in seconds. Zero (0) means infinity or no timeout.	14400
session_task_timeout_secs	integer	Timeout in seconds to interrupt long-running tasks submitted by sessions (algorithms, I/O tasks). Zero (0) means infinity or no timeout.	0
small_task_length	integer	Task length if the total amount of work is smaller than default task length (only relevant for task-stealing strategies).	128
<pre>strict_mode</pre>	boolean	If true, exceptions are thrown and logged with ERROR level whenever the engine encounters configuration problems, such as invalid keys, mismatches, and other potential errors. If false, the engine logs problems with ERROR/WARN level (depending on severity) and makes best guesses and uses sensible defaults instead of throwing exceptions.	true
string_pooling_strategy	<pre>enum[indexed , on_heap, none]</pre>	•	on_heap



Table 20-1 (Cont.) Configuration Parameters for the Graph Server (PGX) Engine

Parameter	Туре	Description	Default
task_length	integer	Default task length (only relevant for task-stealing strategies). Should be between 100 and 10000. Trade-off: a small number implies more fine-grained tasks are generated, higher stealing throughput; a large number implies less memory consumption and GC activity.	
<pre>tmp_dir</pre>	string	Temporary directory to store compilation artifacts and other temporary data. If set to <system-tmp-dir>, uses the standard tmp directory of the underlying system (/tmp on Linux).</system-tmp-dir>	"/tmp"
udf_config_directory	string	Directory path containing UDF config files.	null
<pre>use_index_for_reachability_queries</pre>	<pre>enum[auto, off]</pre>	Create index for reachability queries.	auto
<pre>use_memory_mapper_for_reading_pgb</pre>	boolean	If true, use memory mapped files for reading graphs in PGB format if possible; if false, always use a stream-based implementation.	
<pre>use_memory_mapper_for_storing_pgb</pre>	boolean	If true, use memory mapped files for storing graphs in PGB format if possible; if false, always use a stream-based implementation.	true

The default values of the runtime configuration fields are optimized to deliver the best performance across a wide set of algorithms. Depending on your workload you may be able to improve performance further by experimenting with different strategies, sizes, and thresholds.

Enterprise Scheduler Parameters

The following parameters are relevant only if the advanced scheduler is used. (They are ignored if the basic scheduler is used.)



Parameter	Туре	Description	Default
analysis_task_config	object	Configuration for analysis tasks	weight <no-of- CPUs></no-of-
			priority MEDIUM
			max_thread s <no-of- CPUs></no-of-
fast_analysis_task_config	object	Configuration for fast analysis tasks	weight
			priority HIGH
			max_thread s <no-of- CPUs></no-of-
max_num_concurrent_io_tasks	integer	Maximum number of concurrent I/O tasks at a time	3
num_io_threads_per_task	integer	Number of I/O threads to use per task	<no-of- cpus></no-of-

Basic Scheduler Parameters

The following parameters are relevant only if the basic scheduler is used. (They are ignored if the advanced scheduler is used.)

Field	Туре	Description	Default
num_workers_analysis	integer	This specifies how many worker threads to use for analysis tasks.	<no-of- cpus></no-of-
<pre>num_workers_fast_track_analysis</pre>	integer	This specifies how many worker threads to use for fast-track analysis tasks.	1



Field	Туре	Description	Default
num_workers_io	integer	This specifies how many worker threads to use for I/O tasks (load/refresh/write from/to disk). This value does not impact file-based loaders, as they are always single-threaded. Database loaders will open a new connection for each I/O worker.	<no-of- cpus></no-of-

Example 20-1 Minimal Graph Server (PGX) Configuration

The following example causes the graph server (PGX) to initialize its analysis thread pool with 32 workers. (Default values are used for all other parameters.)

```
{
  "enterprise_scheduler_config": {
     "analysis_task_config": {
        "max_threads": 32
      }
  }
}
```

Example 20-2 Two Pre-loaded Graphs

This example sets more fields and specifies two fixed graphs for loading into memory during the graph server (PGX) startup.

```
"enterprise_scheduler_config": {
  "analysis task config": {
   "max_threads": 32
  "fast analysis task config": {
   "max threads": 32
  }
},
"memory cleanup interval": 600,
"max active sessions": 1,
"release memory threshold": 0.2,
"preload_graphs": [
    "path": "graph-configs/my-graph.bin.json",
    "name": "my-graph"
  },
    "path": "graph-configs/my-other-graph.adj.json",
   "name": "my-other-graph",
    "publish": false
```



```
"authorization": [{
    "pgx_role": "GRAPH_DEVELOPER",
    "pgx_permissions": [{
        "preloaded_graph": "my-graph",
        "grant": "read"
    },
    {
        "preloaded_graph": "my-other-graph",
        "grant": "read"
    }]
},
    ....
]
```

Relative paths in parameter values are always resolved relative to the parent directory of the configuration file in which they are specified. For example, if the preceding JSON is in /pgx/conf/pgx.conf, then the file path graph-configs/my-graph.bin.json inside that file would be resolved to /pgx/conf/graph-configs/my-graph.bin.json.

- Configuration of the Graph Server (PGX) Run-Time Parameters
- Passing the Configuration File to the Graph Server (PGX)
- Memory Consumption by the Graph Server (PGX)
 The graph server (PGX) loads the graph into main memory in order to carry out analysis on the graph and its properties.

20.1.1 Configuration of the Graph Server (PGX) Run-Time Parameters

You can configure the following graph server (PGX) run-time fields.

Table 20-2 Graph Server (PGX) Run-Time Parameters

Parameter	Туре	Description	Default
bfs_iterate_que_task_size	integer	Task size for BFS iterate QUE phase.	128
bfs_threshold_parent_read_base d	number	Threshold of BFS traversal level items above which to switch to parent-read-based visiting strategy.	0.05
bfs_threshold_read_based	integer	Threshold of BFS traversal level items above which to switch to read-based visiting strategy.	1024
bfs_threshold_single_threaded	integer	Number until which BFS traversal level items vertices are visited single-threaded.	128
character_set	string	Standard charset to use throughout PGX, UTF-8 will be used as default. Note: Some formats may not be compatible.	utf-8
cni_diff_factor_default	integer	Default diff factor value used in the common neighbor iterator implementations.	8
cni_small_default	integer	Default value used in the common neighbor iterator implementations, to indicate below which threshold a subarray is considered small.	128



Table 20-2 (Cont.) Graph Server (PGX) Run-Time Parameters

Parameter	Туре	Description	Default
cni_stop_recursion_default	integer	Default value used in the common neighbor iterator implementations, to indicate the minimum size where the binary search approach is applied.	96
dfs_threshold_large	integer	Value that determines at which number of visited vertices, the DFS implementation will switch to data-structures that are more optimized for larger numbers of vertices.	4096
enterprise_scheduler_flags	object	[relevant for enterprise_scheduler] Enterprise scheduler specific settings.	null
explicit_spin_locks	boolean	true means spin explicitly in a loop until lock becomes available. false means using JDK locks which rely on the JVM to decide whether to context switch or spin. Our experiments showed that setting this value to true results in better performance.	true
graph_validation_level	enum[lo w, high]	Level of validation performed on newly loaded or created graphs.	low
max_distinct_strings_per_pool	integer	[only relevant if string_pooling_strategy is indexed] Amount of distinct strings per property after which to stop pooling. If the limit is reached an exception is thrown.	65536
max_off_heap_size	integer	Maximum amount of off-heap memory PGX is allowed to allocate in megabytes, before an OutOfMemoryError will be thrown.	<availa ble- physica 1-</availa
		This limit is not guaranteed to never be exceeded because of rounding and synchronization trade-offs. It only serves as threshold when PGX starts to reject new memory allocation requests.	memory>
memory_allocator	enum[ba sic_all ocator, enterpr ise_all ocator]	Denotes which memory allocator to use.	basic_a llocato r



Table 20-2 (Cont.) Graph Server (PGX) Run-Time Parameters

Parameter	Туре	Description	Default
ms_bfs_frontier_type_strategy	<pre>enum[au to_grow , short, int]</pre>	The type strategy to use for MS-BFS frontiers.	auto_gr ow
num_spin_locks	integer	Number of spin locks each generated app will create at instantiation. Trade-off: small number implies less memory consumption. Big number implies faster execution (if algorithm uses spin locks).	1024
<pre>pattern_matching_supernode_cac he_threshold</pre>	integer	Minimum number of a node's neighbor to be a supernode. This is for pattern matching engine.	1000
pooling_factor	number	[only relevant if string_pooling_strategy is on_heap] This value prevents the string pool to grow as big as the property size which could render the pooling ineffective.	0.25
random_generator_strategy	enum[no n_deter ministi c, determi nistic]	Method of generating random numbers in PGX.	non_det erminis tic
random_seed	long	[relevant for deterministic random number generator only] Seed for the deterministic random number generator used in PGX. The default is -24466691093057031.	-244666 9109305 7031
revisit_threshold	integer	Maximum number of matched results from a node to be cached.	4096
scheduler	enum[ba sic_sch eduler, enterpr ise_sch eduler, low_lat ency_sc heduler]	Denotes which scheduler to use. basic_scheduler: use scheduler with basic features. enterprise_scheduler: use scheduler with advanced, enterprise features for running multiple tasks concurrently and increased performance. low_latency_scheduler: use scheduler that privileges latency of tasks over throughput or fairness across multiple sessions. The low_latency_scheduler is only available in embedded mode	enterpr ise_sch eduler
small_task_length	integer	Task length, if total amount of work is small than default task length (only relevant for task-stealing strategies).	128
string_pooling_strategy	<pre>enum[in dexed, on_heap , none]</pre>	Denotes which string pooling strategy to use.	on_heap



Table 20-2 (Cont.) Graph Server (PGX) Run-Time Parameters

Parameter	Туре	Description	Default
task_length	integer	Default task length (only relevant for task-stealing strategies). F/J pool documentation says this value should be between 100 and 10000. Trade-off: small number implies more fine-grained tasks are generated, higher stealing throughput. High number implies less memory consumption and GC activity.	4096
<pre>use_index_for_reachability_que ries</pre>	enum[au to, off]	Create index for reachability queries.	auto
<pre>use_memory_mapper_for_reading_ pgb</pre>	boolean	If true, use memory mapped files for reading graphs in PGB format if possible; false always use s stream based implementation.	true
<pre>use_memory_mapper_for_storing_ pgb</pre>	boolean	If true, use memory mapped files for storing in PGB format if possible; if false always use a stream based implementation.	true

20.1.2 Passing the Configuration File to the Graph Server (PGX)

The PGX engine configuration file is parsed by the graph server at startup-time whenever ServerInstance#startEngine (or any of its variants) is called. You can pass the path to your configuration file to the graph server (PGX) or perform it programmatically. This topic explains the different ways to pass the configuration file to the graph server (PGX):

Programmatically

All configuration fields exist as Java enums. Example:

```
Map<PgxConfig.Field, Object> pgxCfg = new HashMap<>();
pgxCfg.put(PgxConfig.Field.MEMORY_CLEANUP_INTERVAL, 600);
ServerInstance instance = ...
instance.startEngine(pgxCfg);
```

All parameters not explicitly set will get default values.

Explicitly Using a File

Instead of a map, you can pass the graph server (PGX) configuration JSON file from the local filesystem or from the classpath:

instance.startEngine("path/to/pgx.conf"); // file on local filesystem
instance.startEngine("classpath:/path/to/pgx.conf"); // file on current
classpath



For all other protocols, you can directly pass the JSON file as an input stream:

```
InputStream is = ...
instance.startEngine(is);
```

Implicitly Using a File

If startEngine() is called without an argument, then the graph server (PGX) looks for a configuration file at the following places and stops when it finds the file:

- File path found in the Java system property pgx_conf. Example: java Dpgx conf=conf/my.pgx.config.json ...
- A file named pgx.conf in the root directory of the current classpath
- A file named pgx.conf in the root directory relative to the current System.getProperty("user.dir") directory

Note: Providing a configuration is optional. A default value for each field will be used if the field cannot be found in the given configuration file, or if no configuration file is provided.

Using the Shell in Embedded Mode

To change how the shell configures the embedded (local) graph server (PGX) instance, edit etc/oracle/graph/conf/pgx.conf. Changes will be reflected the next time you invoke the OPG4J shell CLI.

You can also change the location of the configuration file as in the following example:

```
./bin/opg4j --pgx conf path/to/my/other/pgx.conf
```

Setting System Properties

Any graph server (PGX) engine or runtime parameter can be set using Java system properties by writing -Dpgx.<FIELD>=<VALUE> arguments to the JVM that the graph server (PGX) is running on. Note that setting system properties will overwrite any other configuration. The following example sets the maximum off-heap size to 256 GB, regardless of what any other configuration says:

```
java -Dpgx.max_off_heap_size=256000 ...
```

You can also set nested configuration fields, as used for the enterprise scheduler configuration using system properties. The <FIELD> is formed as <CONFIG FIELD1> <CONFIG FIELD2>.

Setting Environment Variables

Also, any graph server (PGX) engine or runtime parameter can be set using environment variables by adding 'PGX_' to the graph server (PGX) JVM environment. Note that setting environment variables will overwrite any other configuration. However, if both system property and an environment variable are set for the same parameter, then the system property value is used. The following example sets the maximum off-heap size to 256 GB using an environment variable:

```
PGX MAX OFF HEAP SIZE=256000 java ...
```



20.1.3 Memory Consumption by the Graph Server (PGX)

The graph server (PGX) loads the graph into main memory in order to carry out analysis on the graph and its properties.

The memory consumed by the graph server for a graph is split between the memory to store the topology of the graph (the information to indicate what are the vertices and edges in the graph without their attached properties), and the memory for the properties attached to the vertices and edges. Internally, the graph server (PGX) stores the graph topology in compressed sparse row (CSR) format, a data structure which has minimal memory footprint while providing very fast read access.

Memory Management

20.1.3.1 Memory Management

The graph server (PGX) requires both on-heap and off-heap memory to store graph data.

The allocation of memory for the graph data is as shown:

- · Graph indexes and graph topology are stored off-heap.
- All primitive properties (integer, long, double, float, boolean, date, local_date, timestamp, time, point2d) are stored off-heap.
- String properties are stored on-heap.

Default Configuration of Memory Limits

You can configure both on-heap and off-heap memory limits. In case of the on-heap, if you don't explicitly set a maximum then it will default to the maximum on-heap size determined by Java Hotspot, which is based on various factors, including the total amount of physical memory available. In case of the off-heap, if you don't explicitly set a maximum then it will default to the total physical available memory on the machine.

- Configuring On-Heap Limits
- Configuring Off-Heap Limits

20.1.3.1.1 Configuring On-Heap Limits

The on-heap memory limits for the graph server (PGX) can be configured by updating the systemd configuration file for the PGX service. However, there is a risk of losing the updates to the configuration file, the next time you upgrade the graph server (PGX). Therefore, it is recommended that you provide the on-heap memory configuration in a drop-in file. All directives in the drop-in file are dynamically merged with the directives in the main configuration file (/etc/systemd/system/pgx.service) during the graph server (PGX) startup.

You can perform the following steps to create a drop-in file and configure the on-heap memory size:

- 1. Navigate to the /etc/systemd/system/pgx.service.d directory. If the pgx.service.d directory does not exist in the file path, then create one.
- 2. Create a drop-in file (.conf file) with any name in /etc/systemd/system/pgx.service.d. Skip this step, if one already exists.



3. Edit the drop-in file as a root user or with sudo command and add the on-heap memory option in the [Service] section as shown:

sudo vi /etc/systemd/system/pgx.service.d/setup.conf The following example displays the added on-heap memory setting in the setup.conf file:

```
[Service]
# Java on-heap memory setting
Environment="JAVA TOOL OPTIONS=-Xms1G -Xmx2G"
```

This option sets the initial heap space to 1GB and allows it to grow up to 2GB.

The supported options for configuring the on-heap memory are:

- -Xmx: to set the maximum on-heap size of the JVM.
- -Xms: to set the initial on-heap size of the JVM.
- -XX:NewSize: to set the initial size of the young generation
- -XX:MaxNewSize: to set the maximum size of the young generation

See the java command documentation for more information on these options.

4. Add the JAVA_HOME environment variable to ensure that the graph server (PGX) is using the appropriate JDK.

```
[Service]
# JAVA_HOME variable
Environment=JAVA_HOME=/usr/java/jdk-15.0.1/
# Java on-heap memory setting
Environment="JAVA TOOL OPTIONS=-Xms1G -Xmx2G"
```

Note that the comments begin with # and you can optionally comment any specific option in order to test your configuration.

5. Reload the PGX service to use the updated settings by running the following command:

```
sudo systemctl daemon-reload
```

6. Restart the graph server (PGX):

```
sudo systemctl restart pgx
```

7. Verify that the new memory setting is correctly passed to the graph server (PGX):

```
sudo systemctl show pgx -p Environment
Environment=PGX_SERVER_KEYSTORE_PASSWORD=changeit JAVA_HOME=/usr/
java/jdk-15.0.1/ JAVA TOOL OPTIONS=-Xms1G\x20-Xmx2G
```

8. Verify that the new memory setting is used by the graph server (PGX):

```
$ sudo journalctl -u pgx -n 50 | grep JAVA_TOOL_OPTIONS
Sep 26 10:52:46 localvm.localdomain bash[25206]: Picked up
JAVA_TOOL_OPTIONS: -Xms1G -Xmx2G
```



9. Finally, use the server-state REST endpoint to confirm the new memory usage. For example:

```
$ BASE URL=https://localhost:7007
$ USERNAME=graph
$ PASSWORD=graph
$ PGX RESPONSE=`curl -s -k -X POST -H 'Content-Type: application/json' -d
'{"username": "'"${USERNAME}"", "password": "'"${PASSWORD}"""}' $
{BASE URL}/auth/token`
$ PGX ACCESS TOKEN=`echo $PGX RESPONSE | jq -r '.access token'`
$ echo `curl -s -k -H 'Authorization: Bearer '"$
{PGX ACCESS TOKEN}" $BASE URL/control/v1/serverState|jq '.entity.memory'`
Гjq
  "localvm.localdomain": [
      "free heap mb": 830,
      "current heap mb": 989,
      "used heap mb": 159,
      "maximum heap mb": 1979,
      "type": "java heap"
      "used off heap mb": 0,
      "maximum off heap mb": 1741,
      "current off heap mb": 1741,
      "free off heap mb": 1741,
      "type": "unmanaged"
  ]
```

20.1.3.1.2 Configuring Off-Heap Limits

You can specify the off-heap limit by setting the $max_off_heap_size$ field in the graph server (PGX) configuration. See Configuration Parameters for the Graph Server (PGX) Engine for more information on the $max_off_heap_size$ parameter. Note that the off-heap limit is not guaranteed to never be exceeded because of rounding and synchronization trade-offs.

20.2 Configuration Parameters for Connecting to the Graph Server (PGX)

You can configure the graph server (PGX) to use the required options at startup.

See Configuring the Graph Server (PGX)

20.3 Configuration Parameters for the Graph Client

You can configure the PGX graph client. All the parameters are available as command-line options also.

Table 20-3 Configuration Parameters for the Graph Client

Parameter	Туре	Description	Default
access_token	string	The authentication token.	null
base_url	string	The base url in the format host [: port] [/path] of the PGX server REST end-point. If the base_url is null, the default will be used which points to embedded PGX instance.	null
cctrace_out	string	[relevant for enable_cctrace] When cctrace is enabled, this option specifies a path to a file where cctrace should log to. If null it will use the default PGX logger on level TRACE. If it is the special value:stderr: it will log to stderr.	null
cctrace_print_stacktraces	boolean	[relevant for enable_cctrace] When cctrace is enabled, this flag prints the stacktrace for each request and result.	false
<pre>client_server_interaction mode</pre>	<pre>enum[async _polling, blocking]</pre>	If async_polling the PGX client would poll the status of the future until it is completed. If blocking, the PGX client would send a request to directly get the value of the future and the server would block until the future result is ready.	<pre>async_poll ing</pre>
enable_cctrace	boolean	If true log every call to a Control or Core interface.	false
keystore	string	The path to the keystore to use for client connections. The keystore is used to authenticate this client at the PGX server if two-way SSL/TLS is enabled.	null
<pre>max_client_http_connectio ns</pre>	integer	Maximum number of connections to open to the PGX server.	2
password	string	Keystore password only.	null
prefetch_size	integer	Number of items to be prefetched in remote iterators.	2048
realm_client_config	object	Implementation dependent configuration options for the realm client.	null
<pre>remote_future_pending_ret ry_interval</pre>	integer	Number of milliseconds to wait before sending another request in case a GET request for a PgxRemoteFuture receives a 202 - Accepted response.	500



 Table 20-3
 (Cont.) Configuration Parameters for the Graph Client

Parameter	Туре	Description	Default
remote_future_timeout	integer	Time that a GET request for a PgxRemoteFuture will be alive, until it times out and tries again. Time in milliseconds, set it to zero for an infinite timeout. See HTTP Client SO_TIMEOUT for more details.	300000
tls_version	string	TLS version to be used by the client. For example, TLSv1.2.	tlsv1.2
truststore	string	Path to the truststore to use for client connections. The truststore is used to validate the server certificate if communicating over SSL/TLS.	null
upload_batch_size	integer	Number of items to be uploaded in a batch. This is used in Core#addAllToCollection() and Core#setProperty().	65536
username	string	Name of the user.	null

Example 20-3 Configure the Graph Client Using the Graph PGX Shell

This following is an example to configure the graph client:

```
cd /opt/oracle/graph
./bin/opg-jshell --base_url https://myhost:8080/pgx --username scott --prefetch_size
1024 --upload_batch_size 5000 --remote_future_timeout 20000 --pending_retry_interval
800
```

Example 20-4 Configure the Graph Client Using the Java API

The following is an example to configure the graph client programatically using the Pqx.qetInstance methods:

```
public static ServerInstance getInstance(String baseUrl, String username,
String password, Integer prefetchSize,
   Integer uploadBatchSize, Integer remoteFutureTimeout, Integer
remoteFuturePendingRetryInterval)
```

To specify key store and trust store for SSL connections use the standard JDK system properties:

```
System.setProperty("javax.net.ssl.trustStore","<truststore>");
System.setProperty("javax.net.ssl.keyStore","<keystore>");
System.setProperty("javax.net.ssl.keyStorePassword","<password>");
```



Deploying Oracle Graph Server Behind a Load Balancer

You can deploy multiple graph servers (PGX) behind a load balancer and connect clients to the servers through the load balancer.

Using Session Persistence with a Load Balancer

You can use the Load Balancer sticky cookie feature since the graph server (PGX) is not stateless. This implies that when you configure load balancer cookie stickiness, the load balancer inserts a cookie to identify the server and the client requests are always directed to the same backend server.

The graph client supports all sessions that belong to a serverInstance to be sent to the same server. You must set the cookie name as PGX INSTANCE STICKY COOKIE.

You can use one of the following options to deploy different graph servers behind a load balancer:

- Using HAProxy for PGX Load Balancing and High Availability
 HAProxy is a high-performance TCP/HTTP load balancer and proxy server that allows
 multiplexing incoming requests across multiple web servers.
- Deploying Graph Server (PGX) Using OCI Load Balancer
 You can deploy multiple graph servers (PGX) behind a load balancer using Oracle Cloud Infrastructure (OCI) Load Balancing Service.
- · Health Check in the Load Balancer

21.1 Using HAProxy for PGX Load Balancing and High Availability

HAProxy is a high-performance TCP/HTTP load balancer and proxy server that allows multiplexing incoming requests across multiple web servers.

You can use HAProxy with multiple instances of the graph server (PGX) for high availability. The following example uses the OPG4J shell to connect to PGX.

The following instructions assume you have already installed and configured the graph server (PGX), as explained in Starting the Graph Server (PGX).

 If HAProxy is not already installed on Big Data Appliance or your Oracle Linux distribution, run this command:

yum install haproxy

Start the graph server instances.
 For example, if you want to load balance PGX across 4 nodes (such as bda02, bda03, bda04, and bda05) in the Big Data Appliance, start PGX on each of these nodes.
 Configure PGX to listen for connections on port 7007.

3. Configure HAProxy:

- Locate the haproxy.cfg file in /etc/haproxy directory on the host where you installed HAProxy.
- **b.** Add a frontend section with the following parameters:
 - bind: to set the listening IP address and port
 - mode: http Or https
 - default backend: to set the name of the backend to be used

For example, the following frontend configuration receives HTTP traffic on all IP addresses assigned to the server at port 7008:

```
frontend graph_server_front
  bind *:7008
  mode http
  default backend graph server
```

- c. Add a backend section with the following parameters:
 - mode: http Or https
 - cookie: name of the cookie to be used for session persistence
 - server: list of servers running behind the load balancer

For example, the following backend configuration uses the ${\tt PGX}$ INSTANCE STICKY COOKIE:

```
backend graph_server
  mode http
  cookie PGX_INSTANCE_STICKY_COOKIE insert indirect nocache
  server graph_server_1 host_name_graph_server_1:port check
cookie graph_server_1 # Notice that the name at the end must be
the same as the server name
  server graph_server_2 host_name_graph_server_2:port check
cookie graph_server_2
  option httpchk GET /isReady
  http-check expect string true
```

In the preceding configuration file, the <code>option</code> <code>httpchk</code> clause instructs the load balancer to check the readiness of the server. The <code>http-check</code> clause specifies that the load balancer must expect a <code>true</code> response in order to determine that the server is healthy and capable of handling more requests. See <code>Health Check</code> in the <code>Load Balancer</code> for supported health check endpoints.

4. Start the load balancer.

Start HAProxy using systemctl:

```
sudo systemctl start haproxy
```

5. Test the load balancer.



From any host you can test connectivity to the HAProxy server by passing in the host and port of the server running HAProxy as the <code>base_url</code> parameter to the graph client shell CLI. For example:

```
cd /opt/oracle/graph
./bin/opg4j --base url http://localhost:7008 -u <username>
```



The PGX in-memory state is lost if the server goes down. HAProxy will route commands to another server, but the client must reload all graph data.

It is recommended that you run a series of PGX commands to test routing. Stop the server and restart the graph shell CLI to confirm that HAProxy redirects the request to a new server.

21.2 Deploying Graph Server (PGX) Using OCI Load Balancer

You can deploy multiple graph servers (PGX) behind a load balancer using Oracle Cloud Infrastructure (OCI) Load Balancing Service.

You can enable cookie-based session persistence with a load balancer to direct all requests from a single client to a specific backend server.

You can you perform the following steps to deploy multiple graph servers using the OCI load balancer.

As a prerequisite requirement, you must ensure that two or more graph servers are running on different machines on the same port (7007 by default).

- 1. Sign in to OCI console using your Oracle Cloud Account.
- 2. Open the navigation menu, click **Networking** and then **Load Balancers**.
- 3. Click Create Load Balancer.

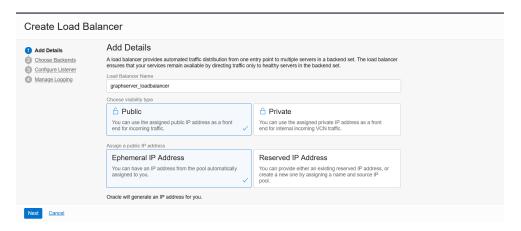
The **Select Load Balancer Type** window opens.

4. Select Load Balancer and click Create Load Balancer.

The **Add Details** page opens as shown:



Figure 21-1 Configuring Load Balancer Details



- 5. Optionally, edit the following details:
 - Load Balancer Name
 - · Choose visibility type
 - Choose IP address type
- Under Choose Networking section, select the Virtual Cloud Network where the graph server instances are running.
- 7. Accept the default values for all other fields and click Next.

The Choose Backends page opens.

- 8. Select Weighted Round Robin as the Load Balancing Policy.
- 9. Click Add Backends to add the backend servers.

The **Add Backends** slider opens as shown.

Figure 21-2 Adding Backends to Load Balancer



 Select as many backend graph server instances as available and click Add Selected Backends.

The selected backend set appear in the **Select Backend Servers** table.

- 11. Specify the following values for the parameters under **Specify Health Check Policy**:
 - Protocol: HTTP
 - Port: backend port used by all the graph servers
 - Interval in milliseconds: default value
 - Timeout in milliseconds: default value
 - Number of Retries: default value



Status Code: 200

URL Path: /isReady

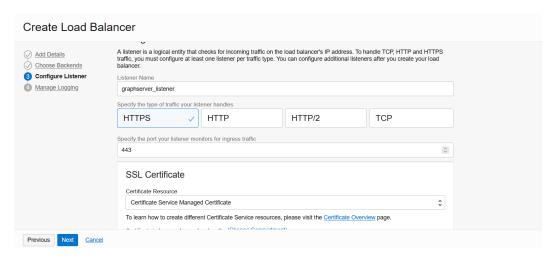
See Health Check in the Load Balancer for supported health check endpoints.

Response Body RegEx: true

12. Click Next.

The Configure Listener page opens as shown:

Figure 21-3 Configuring a Listener for the Load Balancer



- 13. Optionally, edit the Listener Name.
- **14.** Specify **HTTPS** or **HTTP** as the type of traffic handled by the listener.
- **15.** Specify the listener port value to either **443** or **80**.
- **16.** Upload **SSL Certificate** if you specified **HTTPS** communication.
- 17. Click Next.

The Manage Logging page opens as shown.

18. Accept all the default values on this page and click **Submit**.

The load balancer is provisioned and it appears on the table in the **Load Balancers** page.

- 19. Click on the provisioned load balancer to view the Load Balancer Details.
- 20. Click Backend Sets under Resources.
- 21. Click the backend set you want to edit.

The Backend Set Details page opens.

22. Click Edit.

The **Edit Backend Set** dialog opens as shown:



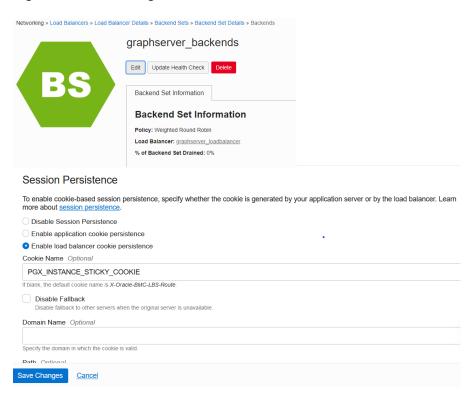


Figure 21-4 Enabling Session Persistence

- 23. Select Enable load balancer cookie persistence.
- **24.** Set the **Cookie Name** to PGX_INSTANCE_STICKY_COOKIE and click **Save Changes**. Your work request gets submitted.

You can now send requests to the load balancer and your session will be persisted on the respective server to which you are logged in.

21.3 Health Check in the Load Balancer

To configure health check in the load balancer, the graph server(PGX) exposes the isReady and isRunning endpoints.



By default, the isReady and isRunning endpoints are unprotected. See Public Health Endpoint Security to enable protection for the health check API.

The load balancer can check the following health status of the graph servers:

Readiness of the graph server: The isReady endpoint detects if the graph server
(PGX) is capable of handling more requests. See the
readiness_memory_usage_ratio system parameter in Configuration Parameters
for the Graph Server (PGX) Engine for more details.



• Liveness of the graph server: The isRunning endpoint detects if the graph server (PGX) is running and alive. See the running_memory_usage_ratio system parameter in Configuration Parameters for the Graph Server (PGX) Engine for more details.

By default, both the endpoints do not require authentication. If the server is running or ready, they return true in the HTTP body with HTTP status code 200. If the server is **not** running or ready, they return false with HTTP status code 200.



Namespaces and Sharing

The graph server (PGX) supports separate namespaces that help you to organize your entities.

Each client session has its own session-private namespace and can choose any name without affecting other sessions. There is also a public namespace for published graphs (for example, published via the publishWithSnapshots() or the publish() methods).

Similarly, each published graph defines a public namespace for published properties as well as a private namespace per session. So different sessions can create properties with the same name on a published graph.

- Defining Graph Names
- · Retrieving Graphs by Name
- Checking Used Names
- Property Name Resolution and Graph Mutations

22.1 Defining Graph Names

Graphs that are created in a session either through loading (for example, calling readGraphWithProperties()) or through mutations will take up a name in the session-private namespace. A graph will be placed in the public namespace only through publishing (that is, when calling the publishWithSnapshots() or the publish() methods). Publishing a graph will move its name from the session-private namespace to the public namespace.

There can only be one graph with a given name in a given namespace, but a name can be used in different namespaces to refer to different graphs. An operation that creates a new graph (for example, readGraphWithProperties()) will fail if the chosen name of the new graph already exists in the session-private namespace. Publishing a graph fails if there is already a graph in the public namespace with the same name.

22.2 Retrieving Graphs by Name

You can retrieve a graph by name by the following two ways:

- getGraph(Namespace, String): with explicitly mentioning the namespace
- getGraph(String): without explicitly mentioning the namespace

With getGraph(Namespace, String), you need to provide the namespace (either session private or public). In this case, the graph will be looked up in the given namespace only.

With getGraph(String), the provided name will be first looked up in the private namespace. If no graph with the given name is found there, then the graph name will be looked up in the public namespace. In other words, if a graph with the same name is defined in both the public and the private namespaces, getGraph(String) will return the private graph and you need to use getGraph (Namespace, String) to get hold of the public graph with that name.

22.3 Checking Used Names

To see the currently used names in a namespace you can use the PgxSession.getGraphs(Namespace) method, which will list all the names in the given namespace. The names in the returned collection can be used in a getGraph(Namespace, String) call to retrieve the corresponding PgxGraph.

22.4 Property Name Resolution and Graph Mutations

Property names behave in a similar way as graph names. All property names of a non-published graph are in the session-private namespace. Once a graph is published with PgxGraph.publishWithSnapshots() or the PgxGraph.publish() methods, its properties are published as well and their names move into the public namespace.

Once a graph is published, newly created properties will still be private to the session and their names will be in the private namespace. Those properties can be published individually with the Property.publish() method, as long as no other property with the same name is already published for that graph.

Additionally, new private properties can be created with the same name of an already-published properties (since the names are part of separate namespaces). To handle such situations and retrieve the correct property, the PGX API offers the getVertexProperty(Namespace, String) and the getEdgeProperty(Namespace, String) methods, which allow specifying the namespace where the property name should be looked up.

Similar to graphs, if you search a property without specifying the namespace, the private namespace is searched first and if the property is not found, the search proceeds to the public namespace. This case applies for getVertexProperty(String)) or the getVertexProperty(String)) methods and for PGQL queries.

Likewise, when a mutation on a graph reads or writes a property referred to by name and two properties exist with the same name, the property in the private namespace is selected. To override the default selection, some mutation mechanisms accept a collection of specific Property objects to be copied into the mutated graph. For example, such mechanism is supported for filter expressions. See Creating Subgraphs for more details.



PGX Programming Guides

You can avail all the PGX functionalities through asynchronous Java APIs. Each asynchronous method has a synchronous equivalent, which blocks the caller thread until the server produces a response.

These APIs may perform one or any combination of:

- Complex, non-blocking Java applications on top of PGX
- Simple, sequential Java scripts executed by JShell
- ShellPerforming interactive graph analysis in the JShell

Layers of PGX API

The PGX API is composed of a few different Java interfaces. Each interface provides a distinct layer of abstraction for PGX, as shown in the following table:

Table 23-1 PGX API Interface

Interface	Description
ServerInstance	The ServerInstance class encapsulates access to a PGX server instance and can be used to create sessions, start and stop the PGX engine, monitor the engine status and perform other administrative tasks. If the instance points to a remote instance, access to the administrative functions requires special authorization on the HTTP level by default.
PgxSession	A PgxSession represents an active user currently connected to an instance. Each session gets its own workspace on the server side which can be used to read graphs, create in-memory data structures, hold analysis results and custom algorithms. The PgxSession class provides various methods to create new transient data (currently collections). If a session is idling for too long, the PGX engine will automatically destroy it to ensure no resources are wasted.
PgxGraph	A PgxGraph represents a client-side handle to the graph data managed by the PGX server. A graph may contain an arbitrary amount of properties of type VertexProperty and/or EdgeProperty.



The PGX currently only supports nonpartitioned graphs, meaning every vertex/ edge has the same properties with the same names and types as all the other vertices/edges.

PgxGraph class provides various methods to create new transient data (including maps and collections) as well as graph mutation operations, such as undirecting, sorting and filtering.



Table 23-1 (Cont.) PGX API Interface

Interface	Description
Analyst	The Analyst API contains all of the built-in algorithms PGX provides. Analyst objects keep track of all the transient data they created during algorithm invocations to hold analysis results. Once an Analyst gets destroyed, all the results it created get freed on the server-side automatically.
CompiledProgram	The CompiledProgram class (PGX Algorithm API) encapsulates runtime-compiled custom algorithms and allows invocation of those algorithms using PGX data objects, such as PgxGraph or VertexProperty, as arguments.

Please see the oracle.pgx.api package in the Javadoc for more details.

- Design of the Graph Server (PGX) API
 This guide focuses on the design of the graph server (PGX) API.
- Data Types and Collections in the Graph Server (PGX)
 This guide provides you the list of the supported data types and collections in the graph server (PGX).
- Handling Asynchronous Requests in Graph Server (PGX)
 This guide explains in detail the asynchronous methods supported by the PGX API.
- Graph Client Sessions

The graph server (PGX) assumes there may be multiple concurrent clients, and each client submits request to the shared PGX server independently.

Graph Mutation and Subgraphs

This guide discusses the several methods provided by the graph server (PGX) for mutating graph instances.

- Graph Builder and Graph Change Set
- Managing Transient Data

This guide discusses how to handle transient properties and collections.

Graph Versioning

This guide describes the different ways to work with graph snapshots.

Labels and Properties

You can perform various actions on the graph property and label values by executing PGQL queries.

Filter Expressions

This guide explains the usage of filter expressions.

- Advanced Task Scheduling Using Execution Environments
 This guide shows how you can use the advanced scheduling features of the enterprise scheduler.
- Admin API

This guide shows how to use the graph server (PGX) Admin API to inspect the server state including sessions, graphs, tasks, memory and thread pools.

PgxFrames Tabular Data-Structure

23.1 Design of the Graph Server (PGX) API

This guide focuses on the design of the graph server (PGX) API.

The design of the PGX API reflects consideration of the following situations:

- Multiple clients may concurrently be accessing a single running instance of PGX, sharing its resources. Each client needs to maintain its own isolated workspace (session).
- Graph and property data can be large in size and therefore that data only resides on the server side.
- Some graph analysis may take a significant amount of time.
- Clients may not reside in the same address space (JVM) as PGX. Actually, clients may not even be Java applications.

Client Sessions

In PGX, each client maintains its own session, an isolated, private workspace. Therefore, clients first have to obtain a PgxSession object from a PGX ServerInstance before they can perform any analysis.

Asynchronous Execution

The PGX API is designed for asynchronous execution. That means that each computationally intensive method in the PGX API *immediately* returns a PgxFuture object without waiting for the request to finish. The PgxFuture class implements the Future interface, which can be used to retrieve the result of a computation at some point in the future.



The asynchronous execution aspect of this design facilitates multiple (remote) clients submitting requests to a single server. A request from one client may be queued up to wait until PGX resources become available. The asynchronous API allows the client (or calling thread) to work on other tasks until PGX completes the request.

No Direct References

The PGX API does not return objects with direct reference to PGX internal objects (such as the graph or its properties) to the client. This is because:

- The client might not be in the same JVM as the server
- The graph instance might be shared by multiple clients

Instead, the PGX API only returns lightweight, stateless pointer objects to those objects. These pointer objects only holds the ${\tt ID}({\tt name})$ of the server-side object to which they are pointing.

Resource Management Considerations

The graph server (PGX), being an *in-memory* analytic engine, might allocate large amounts of memory to hold the graph data of clients. Therefore, it is important that client sessions



clean up their resources once they have ended. The PGX API supports several features to make this easier:

• Every object returned by the PGX API pointing to a server-side resource implements the <code>Destroyable</code> interface, which means all memory-consuming client-side objects can be destroyed the same way. For example:

```
PgxGraph myGraph = ...
myGraph.destroyAsync(); // request destruction of myGraph, don't
wait for response
try {
   myGraph.destroy(); // blocks caller thread until destruction
was done
} catch (ExecutionException e) {
   // destruction failed
}
```

 Destroyable extends AutoClosable, so users can leverage Java's built-in resource management syntax:

```
try (PgxGraph myGraph = session.readGraphWithProperties(config)) {
   // do something with myGraph
}
// myGraph is destroyed
```

Session time out. In some cases, the PGX server will remove the session and all
its data automatically. This can occur when a client fails to destroy either the data
or its session, or if it does not hear from the session after a configurable timeout.
See Configuration Parameters for the Graph Server (PGX) Engine for more
information to configure timeout parameters.

23.2 Data Types and Collections in the Graph Server (PGX)

This guide provides you the list of the supported data types and collections in the graph server (PGX).

Primitive Data Types

The following section explains the primitive data types supported by the graph server (PGX) and their limitations.

PGX supports the following primitive data types.:

- **Numeric Types**: integer, long, float, and double. These types have the same size, range and precision of the corresponding Java primitive data type.
- **Boolean Type**: The boolean data type has only two possible values, true and false. As with Java and C++, its size is not precisely defined.
- **String**: String is a primitive data type in PGX. PGX follows the Java conventions for String representation.
- Datetime Types: date, time, timestamp, time with time zone, and timestamp with time zone. These types correspond to the Java types shown in Table 23-2 from the standard library package java.util.time.



Vertex and Edge: The type vertex or edge of the graph itself is a proper type in PGX.

Note:

- vertex and edge is itself a valid primitive data type. For instance, in a path-finding algorithm, each vertex can have a temporary property predecessor that stores which incoming neighbor is the predecessor vertex in the path. Such a property would have the type vertex.
- local_date must be used instead of date in the graph configuration file. See
 Using Datetime Data Types for more examples on usage of datetime data
 types.

All properties and scalar variables must be one of the above preceding data types. See Managing Transient Data for more information on handling transient properties and scalar variables.

The following table presents the overview of the supported data types, their integration in different languages and APIs and their minimum and maximum value limitations.

Note:

- For float and double types, the smallest absolute value is included in the table, the minimum value is the negative of maximum value for these types.
- For string values, PGX supports arbitrary long strings.

Table 23-2 Overview of Data types

Data Type	Loading & Storing	PGX Java API	PGQL and Filter Expression	Minimum Value Limitation	Maximum Value Limitation	
string	string	String	STRING	_	-	
int/integer	int/integer	int	INT/INTEGER	-2147483648	2147483647	
long	long	long	LONG	-92233720368547 75808	-92233720368547 75807	
float	float	float	FLOAT	1.4E-45	3.4028235e+38	
double	double	double	DOUBLE	4.9E-324	1.7976931348623 157E308	
boolean	boolean	boolean	BOOLEAN	_	_	
date	local_date	LocalDate	DATE	-5877641-06-23	5881580-07-11	
time	time	LocalTime	TIME	00:00:00.000	23:59:59.999	
timestamp	timestamp	LocalDateTi me	TIMESTAMP	-292275055-05-1 7 00:00:00.000	292278994-08-17 07:12:55.807	
time with time zone	time_with_t imezone	OffsetTime	TIME WITH TIME ZONE	00:00:00.000+18 :00	23:59:59.999-18 :00	



Data Type Loading & Storing		PGX Java API	PGQL and Filter Expression	Minimum Value Limitation	Maximum Value Limitation		
timestamp with time zone	timestamp_w ith_timezon e	OffsetDateT ime	TIMESTAMP WITH TIME ZONE	-292275055-05-1 7 00:00:00.000+18 :00	292278994-08-17 07:12:55.807-18 :00		
vertex	_	PgxVertex	_	_	-		
edge	_	PgxEdge	_	_	-		

Collections

The graph server (PGX) supports three different collection types: sequence, set and order. All of these collections can contain values of the vertex type, but each has different semantics regarding uniqueness and preserving the order of its elements:

- **Sequence**: a sequence works basically like a list. It preserves the order of the elements added to it, and the same element can appear multiple times.
- Set: a set can contain the same value once at the most. Adding a value that is already in the set will have no effect. set does not preserve the order of the elements it contains.
- Order: just like the set, the order collection will contain each element once at the most. But the order preserves the order of the elements inserted into it (that is, it is a FIFO data structure).

See Collection Data Types for examples on creation and usage of the different collections.

Immutable Collections

Some operations, like PgxGraph.getVertices() and PgxGraph.getEdges() return immutable collections. These collections behave like normal collections, but cannot be modified by operations like addAll or removeAll and clear.

An immutable collection can be transformed into a mutable collection by using the toMutable method, which returns a mutable copy of the collection. If toMutable is called on a collection that is already mutable, the method has the same result as the method clone.

To check if a collection is mutable, use the isMutable method.

Maps

PGX provides the following two kinds of maps:

- Graph-bound maps can hold mappings between types in PropertyType. This is the kind of maps to use if the key or value types are graph-related like VERTEX and EDGE otherwise using session-bound maps is recommended.
- Session-bound maps can map between non graph-related types and are directly bound to the session.

See Map Data Types for examples on creation and usage of maps.



- Using Collections and Maps
- Using Datetime Data Types

23.2.1 Using Collections and Maps

This section explains with examples, the creation and usages of collections and maps.

You must first create a session before getting started with the collection and map data types.

- JShell
- Java
- Python

JShell

```
cd /opt/oracle/graph/
./bin/opg-jshell // starting the shell will create an implicit session
```

Java

```
import oracle.pgx.api.*;
...
PgxSession session=Pgx.createSession("<session name>");
```

Python

```
from pypgx import get_session
session = get session(session name="<session name>")
```

- Collection Data Types
- Map Data Types

23.2.1.1 Collection Data Types

The graph server (PGX) defines two types of collections:

- **Graph-bound collections**: such as vertex and edge collections. These collections belong to the graph.
- Session-bound collections: belong to the session.
- · Graph-Bound Collections
- Session-Bound Collections



23.2.1.1.1 Graph-Bound Collections

The following describes the usage of graph-bound collections.

You must first load the graph to work with vertex and edge collections as shown in Reading Graphs from Oracle Database into the Graph Server (PGX).

Vertex Collections

You can create a vertex collection as shown in the following code:

- JShell
- Java
- Python

JShell

```
v0 = graph.getVertex(100) // 'graph' is the loaded graph object. '100'
-> '103' are vertex ids that supposedly
v1 = graph.getVertex(101) // exist in the graph
v2 = graph.getVertex(102)
v3 = graph.getVertex(103)

myVertexSet = graph.createVertexSet("myVertexSet") // A name is
automatically generated if none given
myVertexSet.add(v0) // Adds vertex
'v0' to the set
myVertexSet.addAll([v1, v2, v3]) // Supports
variadic parameter as well: myVertexSet.addAll(v1, v2, v3)
```

```
import java.util.Arrays;
import oracle.pgx.api.*;
...
PgxVertex v0 = graph.getVertex(100);
PgxVertex v1 = graph.getVertex(101);
PgxVertex v2 = graph.getVertex(102);
PgxVertex v3 = graph.getVertex(103);

VertexSet myVertexSet = graph.createVertexSet("myVertexSet"); // A name is automatically generated if none given myVertexSet.add(v0);
myVertexSet.addAll(Arrays.asList(v1, v2, v3));
```



Python

```
v0 = graph.get_vertex(100)
v1 = graph.get_vertex(101)
v2 = graph.get_vertex(102)
v3 = graph.get_vertex(103)

my_vertex_set = graph.create_vertex_set("myVertexSet")
my_vertex_set.add(v0)
my_vertex_set.add_all([v1,v2,v3])
```

Edge Collections

You can create an edge collection as shown in the following code:

- JShell
- Java
- Python

JShell

```
e0 = graph.getEdge(100) // 'graph' is the loaded graph object. '100' ->
'103' are edge ids that supposedly
e1 = graph.getEdge(101) // exist in the graph
e2 = graph.getEdge(102)
e3 = graph.getEdge(103)

myEdgeSequence = graph.createEdgeSequence("myEdgeSequence")
myEdgeSequence.add(e0)
myEdgeSequence.addAll([e1, e2, e3])
```

```
import java.util.Arrays;
import oracle.pgx.api.*;
...
PgxEdge e0 = graph.getEdge(100);
PgxEdge e1 = graph.getEdge(101);
PgxEdge e2 = graph.getEdge(102);
PgxEdge e3 = graph.getEdge(103);

EdgeSequence myEdgeSequence = graph.createEdgeSequence("myEdgeSequence");
myEdgeSequence.add(e0);
myEdgeSequence.addAll(Arrays.asList(e1, e2, e3));
```



Python

```
e0 = graph.get_edge(100)
e1 = graph.get_edge(101)
e2 = graph.get_edge(102)
e3 = graph.get_edge(103)

my_edge_sequence = graph.create_edge_sequence("my_edge_sequence")
my_edge_sequence.add(e0)
my_edge_sequence.add_all([e1, e2, e3])
```

23.2.1.1.2 Session-Bound Collections

You can create and manipulate collections directly in the session without the need for a graph. Session-bound collections can be further passed as parameters to graph algorithms or used like any other collection object. The following sub-sections describe the currently supported types for these collections.

Scalar Collections

Scalar collections contain simple data types like Integer, Long, Float, Double and Boolean. They can be managed by the PgxSession APIs:

Creation of a Scalar Collection

You can use <code>createSet()</code> and <code>createSequence()</code> methods to create a scalar collection as shown in the following code:

- JShell
- Java

JShell

```
myIntSet = session.createSet(PropertyType.INTEGER, "myIntSet")
myDoubleSequence = session.createSequence(PropertyType.DOUBLE) // A
name will be automatically generated if none is provided.
println myDoubleSequence.getName() //
Display the generated name.
```

```
import oracle.pgx.api.*;
import oracle.pgx.common.types.*;
...
ScalarSet myIntSet = session.createSet(PropertyType.INTEGER,
"myIntSet");
ScalarSequence myDoubleSequence =
```



```
session.createSequence(PropertyType.DOUBLE);
System.out.println(myDoubleSequence.getName());
```

Run Operations on a Scalar Collection

You can run several operations on a scalar collection as shown in the following code:

- JShell
- Java

JShell

Java

```
import java.util.Arrays;
import oracle.pgx.api.*;
...
myIntSet.add(10);
myIntSet.addAll(Arrays.asList(0, 1, 2, 3, 4, 5, 6, 7, 8, 9));
myIntSet.addAll(Arrays.asList(0, 1, 2));

myIntSet.contains(1); // Returns `true`.
myIntSet.remove(10);
myIntSet.removeAll(Arrays.asList(4, 5, 6, 7, 8, 9));
```

Traversal of a Scalar Collection

You can traverse a scalar collection either using an iterator or using the new Stream API. You can add elements of a sequence to a set, traverse a sequence and filter out elements not required, and then add the rest to another scalar collection.



- JShell
- Java

JShell

```
myIntSet.forEach({x -> print x + "\n"})
myIntSet.stream().filter({x -> x % 2 == 0}).forEach({x -> myDoubleSequence.add(x)})
println myDoubleSequence
```

Java

```
import java.util.Iterator;
import java.util.stream.Stream;
import oracle.pgx.api.*;
...
myIntSet.forEach(x -> System.out.println(x));
myIntSet.stream().filter(x -> x % 2 ==
0).forEach(myDoubleSequence::add);
```

23.2.1.2 Map Data Types

The graph server (PGX) defines two types of maps:

- **Graph-bound maps**: These maps support any key or value type and are created using a graph object.
- **Session-bound maps**: Keys or values in these maps are of any type except from graph-related types (that is, vertices or edges). These maps belong to the session.
- Graph-Bound Maps
- Session-Bound Maps

23.2.1.2.1 Graph-Bound Maps

Some data types like VERTEX or EDGE depend on the graph. Consequently, mappings involving these data types also depend on the graph. PGX provides PgxGraph and PgxMap APIs to manage such maps.

The following describes the usage of graph-bound maps.

You must first load the graph to work with vertex and edge maps.

You can create a graph-bound map using vertices as keys as shown in the following code:

- JShell
- Java



Python

JShell

```
v0 = graph.getVertex(100)
v1 = graph.getVertex(101)
v2 = graph.getVertex(102)
v3 = graph.getVertex(103)

vertexToLongMap = graph.createMap(PropertyType.VERTEX, PropertyType.LONG,
"vertexToLongMap")
vertexToLongMap.put(v0, v0.getDegreeAsync().get())
vertexToLongMap.put(v1, v1.getDegreeAsync().get())
vertexToLongMap.put(v2, v2.getDegreeAsync().get())
vertexToLongMap.put(v3, v3.getDegreeAsync().get())
```

Java

```
import java.util.Arrays;
import oracle.pgx.api.*;
...
PgxVertex v0 = graph.getVertex(100);
PgxVertex v1 = graph.getVertex(101);
PgxVertex v2 = graph.getVertex(102);
PgxVertex v3 = graph.getVertex(103);

PgxMap<PgxVertex, Long> vertexToLongMap =
graph.createMap(PropertyType.VERTEX, PropertyType.LONG, "vertexToLongMap");
vertexToLongMap.put(v0, v0.getDegree());
vertexToLongMap.put(v1, v1.getDegree());
vertexToLongMap.put(v2, v2.getDegree());
vertexToLongMap.put(v3, v3.getDegree());
```

Python

```
v0 = graph.get_vertex(100)
v1 = graph.get_vertex(101)
v2 = graph.get_vertex(102)
v3 = graph.get_vertex(103)

vertex_to_long_map = graph.create_map("vertex", "long", "vertex_to_long_map")
vertex_to_long_map.put(v0, v0.degree)
vertex_to_long_map.put(v1, v1.degree)
vertex_to_long_map.put(v2, v2.degree)
vertex_to_long_map.put(v3, v3.degree)
```

You can create graph-bound maps using edges as keys as shown in the following code:



- JShell
- Java
- Python

JShell

```
e0 = graph.getEdge(100)
e1 = graph.getEdge(101)
e2 = graph.getEdge(102)
e3 = graph.getEdge(103)

edgeToVertexMap = graph.createMap(PropertyType.EDGE,
PropertyType.VERTEX, "edgeToVertexMap")
edgeToVertexMap.put(e0, e0.getSource())
edgeToVertexMap.put(e1, e1.getSource())
edgeToVertexMap.put(e2, e2.getSource())
edgeToVertexMap.put(e3, e3.getSource())
```

Java

```
import java.util.Arrays;
import oracle.pgx.api.*;
...
PgxEdge e0 = graph.getEdge(100);
PgxEdge e1 = graph.getEdge(101);
PgxEdge e2 = graph.getEdge(102);
PgxEdge e3 = graph.getEdge(103);

PgxMap<PgxEdge, PgxVertex> edgeToVertexMap = graph.createMap(PropertyType.EDGE, PropertyType.VERTEX, "edgeToVertexMap");
edgeToVertexMap.put(e0, e0.getSource());
edgeToVertexMap.put(e1, e1.getSource());
edgeToVertexMap.put(e2, e2.getSource());
edgeToVertexMap.put(e3, e3.getSource());
```

Python

```
e0 = graph.get_edge(100)
e1 = graph.get_edge(101)
e2 = graph.get_edge(102)
e3 = graph.get_edge(103)

edge_to_long_map = graph.create_map("edge", "long",
   "edge_to_long_map")
edge_to_long_map.put(e0, e0.source)
edge_to_long_map.put(e1, e1.source)
edge_to_long_map.put(e2, e2.source)
edge_to_long_map.put(e3, e3.source)
```





If you destroy the graph you will lose the map. Consider using a session-bound maps instead if your map does not involve any graph-related key or value type.

23.2.1.2.2 Session-Bound Maps

You can directly create maps in the session. But, you cannot use any graph-related data type as the map key or value type. Session-bound maps can be further passed as parameters to graph algorithms or used like any other map object. They are managed by PgxSession and PgxMaps APIs.

Scalar collections contain simple data types like Integer, Long, Float, Double and Boolean. They can be managed by the PgxSession APIs.

Creation of a Session-bound Map

You can use createMap() method and its overloads to create a session-bound map.

- JShell
- Java

JShell

```
intToDouble = session.createMap(PropertyType.INTEGER, PropertyType.DOUBLE,
"intToDouble")
intToTime = session.createMap(PropertyType.INTEGER, PropertyType.TIME) // A
name will be automatically generated.
println intToTime.getName()
println intToTime.getSessionId()
println intToTime.getGraph() //
`null`: Not bound to a graph.
println intToTime.getKeyType()
println intToTime.getValueType()
```

```
import java.time.LocalTime;
import oracle.pgx.api.*;
import oracle.pgx.common.types.*;
...
PgxMap<Integer, Double> intToDouble =
session.createMap(PropertyType.INTEGER, PropertyType.DOUBLE, "intToDouble");
PgxMap<Integer, LocalTime> intToTime =
session.createSequence(PropertyType.INTEGER, PropertyType.TIME);
System.out.println(intToTime.getName());
System.out.println(intToTime.getSessionId());
```



```
System.out.println(intToTime.getGraph()); // `null`: Not bound to a
graph.
System.out.println(intToTime.getKeyType());
System.out.println(intToTime.getValueType());
```

Run Operations on a Session-bound Map

You can run important operations such as setting, removing and checking existence of entries on a session-bound map as shown in the following code:

- JShell
- Java

JShell

```
intToDouble.put(0, 0.314)
intToDouble.put(1, 3.14)
intToDouble.put(2, 31.4)
intToDouble.put(3, 314)
println intToDouble.size()
                                     // 4
println intToDouble.get(1)
println intToDouble.get(3)
println intToDouble.get(10)
                                     // null
                                     // `true`
println intToDouble.containsKey(0)
intToDouble.remove(0)
println intToDouble.containsKey(0)
                                     // `false`
println intToDouble.containsKey(10) // `false`
intToDouble.remove(10)
println intToDouble.containsKey(10) // `false`
println intToDouble.put(1, 999)
                                     // previous mapped value
(`3.14`) is replaced by `999`
intToDouble.destroy()
```

```
import java.util.Arrays;
import oracle.pgx.api.*;
...
intToDouble.put(0, 0.314);
intToDouble.put(1, 3.14);
intToDouble.put(2, 31.4);
intToDouble.put(3, 314);
```



```
System.out.println(inToDouble.size());
                                                 // 4
System.out.println(intToDouble.get(1));
System.out.println(intToDouble.get(3));
System.out.println(intToDouble.get(10));
                                                 // null
System.out.println(intToDouble.containsKey(0));
                                                 // `true`
intToDouble.remove(0);
                                                 // `false`
System.out.println(intToDouble.containsKey(0));
System.out.println(intToDouble.containsKey(10)); // `false`
intToDouble.remove(10);
System.out.println(intToDouble.containsKey(10)); // `false`
System.out.println(intToDouble.put(1, 999)); // previous mapped value
(`3.14`) is replaced by `999`
intToDouble.destroy();
```

Traversal of a Session-bound Map

You can traverse a session-bound map, using entries() method to get an iterable of map entries and keys() method to get an iterable of map keys.

- JShell
- Java

JShell

```
intToDouble.entries().forEach {it -> println (it)}
intToDouble.keys().forEach {it -> println (it)}
```

Java

```
import java.util.Iterable;
import java.util.stream.Stream;
import oracle.pgx.api.*;
...
Iterable<Map.Entry> entries = intToDouble.entries();
entries.forEach(System.out::println);
Iterable<Map.Entry> keys = intToDouble.keys();
keys.forEach(System.out::println);
```

23.2.2 Using Datetime Data Types

This section explains in detail working of datetime data types such as date, time and timestamp.

Overview of Datetime Data Types in Graph Server (PGX)

Table 23-3 presents the overview of the five datetime data types supported by PGX along with example values.



PGX also supports custom format specification when loading data into PGX.

Table 23-3 Overview of Datetime Data Types in PGX

Data Type	Loading and Storing	PGX Java API	PGQL and Filter Expression	Example Value-1	Example Value-1	
date	local_date	al_date LocalDate DATE		2001-01-29	2018-10-08	
time	time	LocalTime	TIME	10:15	10:30:01.000	
timestamp	timestamp	amp LocalDateT TIMEST ime		2001-01-29 10:15	2018-10-08 10:30:01.000	
time with time zone	time_with_ timezone	OffsetTime	TIME WITH	10:15+01:00	10:30:01.000- 08:00	
timestamp with time zone	timestamp_ with_timez one	OffsetDate Time	TIMESTAMP WITH TIME ZONE	2001-01-29 10:15+01:00	2018-10-08 10:30:01.000- 08:00	

- Loading Datetime Data
- Specifying Custom Datetime Formats
- APIs for Accessing Datetime Data
- Querying Datetime Data Using PGQL
- Accessing Datetimes from PGQL Result Sets

23.2.2.1 Loading Datetime Data

You must first load a graph to work with datetime data. See Reading Graphs from Oracle Database into the Graph Server (PGX) for more information on graph loading.

The following example shows how to load a graph that has three vertices representing persons and zero edges.

Example 23-1 Loading Datetime Data

1. Create an EDGE LIST file persons.edge list as shown:

1*Judy,1989-01-15,1989-01-15 10:15-08:00 2*Klara,2001-01-29,2001-01-29 21:30-08:00 3*Pete,1995-08-01,1995-08-01 03:00-08:00



2. Create a corresponding graph configuration file persons.edge list.json as shown:

```
{
    "format": "edge list",
    "uri": "persons.edge list",
    "vertex id type": "long",
    "vertex props":[
        {
            "name":"name",
            "type": "string"
        },
            "name": "date of birth",
            "type": "local date"
        },
            "name": "timestamp of birth",
            "type": "timestamp with timezone",
            "format":["yyyy-MM-dd H[H]:m[m][:s[s]][XXX]"]
        }
    ],
    "edge props":[
    "separator":","
}
```

3. You can now load the data as shown in the following code:

- JShell
- Java
- Python

JShell

```
opg4j> var graph = session.readGraphWithProperties("persons.edge_list.json",
"people_graph")
```

```
import oracle.pgx.api.*;
...
PgxGraph graph =
session.readGraphWithProperties("persons.edge list.json","people graph");
```



Python

```
graph =
session.read_graph_with_properties("persons.edge_list.json",graph_name=
"people graph")
```

23.2.2.2 Specifying Custom Datetime Formats

You can also manually specify the datetime format(s) of your data.

By default, PGX tries to parse datetime values using a set of predefined formats. If this fails, an exception like the following is thrown:

```
property timestamp_of_birth: could not parse value at line 1 for
property of temporal type OffsetDateTime using any of the given formats
```

In such a case, you can custom format the datetime data.

There are two ways of specifying datetime formats:

- on a per-property basis
- on a per-type basis

Property-Specific Datetime format:

You can custom format the property timestamp_of_birth used in Example 23-1 to the format yyyy-MM-dd H[H]:m[m][:s[s]][XXX] as shown:

Example 23-2 Specifying Property-Specific Datetime format:

```
{
    "name":"timestamp_of_birth",
    "type":"timestamp_with_timezone",
    "format":["yyyy-MM-dd H[H]:m[m][:s[s]][XXX]"]
}
```

where yyyy-MM-dd H[H]:m[m][:s[s]][XXX] specifies that the timestamp values consist of:

- a four-digit year
- a hyphen followed by a two-digit month
- a hyphen followed by a two-digit day
- a space
- an hour, specified as either one or two digits
- a colon followed by a minute, specified as either one or two digits
- an optional part that consists of a colon followed by a second that is specified as either one or two digits
- an optional timezone



Note:

- H[H]:m[m] allows the value 01:15 as well as the value 1:15.
- yyyy-MM-dd allows the value 1989-01-15 but not the value 1989-1-15. However, if two-digit months and days are needed, a format like yyyy-M[M]-d[d] can be used.

Also the format specification takes a *list* of formats. In the preceding example, the list contains only a single format, but you may specify any number of formats. If more than one format is specified, then when parsing the datetime data, the formats are tried from left to right until parsing succeeds. In this way, you can even load data that contains a mixture of values in different formats.

Type-Specific Datetime format:

You can also specify datetime formats on a *per-type* basis. This is useful in cases when there are multiple properties that have the same type as well as the same format because you will only need to specify the datetime format only once.

In case of the per-type specification, the format is used for each vertex or edge property that has the particular type.

The following example shows two type-specific formats (local_date_format and timestamp with timezone format):

Example 23-3 Specifying Type-Specific Datetime format:

```
"edge_props":[
],
   "separator":",",
   "local_date_format":["yyyy-MM-dd"],
   "timestamp_with_timezone_format":["yyyy-MM-dd H[H]:m[m][:s[s]][XXX]"]
}
```

In the example, properties of type date (local_date) have the format yyyy-MM-dd while properties of type timestamp with time zone (timestamp_with_timezone) have the format yyyy-MM-dd H[H]:m[m][:s[s]][XXX].

Note:

Property-specific formats always overrides type-specific formats. If you specify a type-specific format, and the property of the particular type also has a property-specific format, then only the property-specific format is used to parse the datetime data.

23.2.2.3 APIs for Accessing Datetime Data

The graph server (PGX) uses the new Java 8 temporal data types for accessing datetime data through the Java API:

- date in PGX maps to LocalDate in Java
- time in PGX maps to LocalTime in Java
- timestamp in PGX maps to LocalDateTime in Java
- time with time zone in PGX maps to OffsetTime in Java
- timestamp with time zone in PGX maps to OffsetDateTime in Java

You can retrieve a date as shown in the following code:

- JShell
- Java
- Python

JShell

```
opg4j> var dateOfBirthProperty =
graph.getVertexProperty("date_of_birth")
opg4j> var birthdayOfJudy = dateOfBirthProperty.get(1)
```

Java

```
import java.time.LocalDate;
import oracle.pgx.api.*;
...
VertexProperty<LocalDate> dateOfBirthProperty =
graph.getVertexProperty("date_of_birth");
LocalDate birthdayOfJudy = dateOfBirthProperty.get(1);
```

Python

```
date_of_birth_property = graph.get_vertex_property("date_of_birth")
birthday_of_judy = date_of_birth_property.get(1)
```

23.2.2.4 Querying Datetime Data Using PGQL

You can perform various operations such as *extracting* values from datetimes, *comparing* datetime values, and, *converting* between different datetime types. on datetime data using PGQL.

The following are example PGQL queries that show different operations that involve datetime data:



Retrieving Datetime Properties

The following query retrieves the date_of_birth and timestamp_of_birth properties from the all the persons in the graph.

```
SELECT n.name AS name, n.date_of_birth AS birthday, n.timestamp_of_birth
AS timestamp
   FROM MATCH (n) ON people_graph
ORDER BY birthday
```

The result of the query is as follows:

Comparing Datetime Values

The following query provides an overview of persons who are older than other persons in the graph:

```
SELECT n.name AS person1, 'is older than' AS relation, m.name AS person2
   FROM MATCH (n) ON people_graph, (m) ON people_graph
   WHERE n.date_of_birth > m.date_of_birth
ORDER BY person1, person2
```

The result of the query is as follows:

Extracting Values from Datetimes

The following query extracts the year, month, and day from the date of birth values:

```
SELECT n.name AS name
, n.date_of_birth AS dob
, EXTRACT(YEAR FROM n.date_of_birth) AS year
, EXTRACT(MONTH FROM n.date_of_birth) AS month
, EXTRACT(DAY FROM n.date_of_birth) AS day
FROM MATCH (n) ON people_graph
ORDER BY name
```



The result of the query is as follows:

name		dob		year		month		day	
Klara	i	1989-01-15 2001-01-29 1995-08-01	İ	2001	İ	1	İ	15 29 1	İ

Converting Between Different Types of Datetime Values

The following query converts the timestamp_of_birth property into values of the following three datetime types:

- a timestamp (without time zone)
- a time with time zone
- a time (without time zone)

```
SELECT n.name AS name
    , n.timestamp_of_birth AS original_timestamp
    , CAST(n.timestamp_of_birth AS TIMESTAMP) AS utc_timestamp
    , CAST(n.timestamp_of_birth AS TIME WITH TIME ZONE) AS
timezoned_time
    , CAST(n.timestamp_of_birth AS TIME) AS utc_time
    FROM MATCH (n) ON people_graph
ORDER BY original timestamp
```

The result of the query is as follows:

```
+-----+
| name | original_timestamp | utc_timestamp | timezoned_time |
utc_time |
+-------+
| Judy | 1989-01-15T10:15-08:00 | 1989-01-15T18:15 | 10:15-08:00 |
18:15 |
| Pete | 1995-08-01T03:00-08:00 | 1995-08-01T11:00 | 03:00-08:00 |
11:00 |
| Klara | 2001-01-29T21:30-08:00 | 2001-01-30T05:30 | 21:30-08:00 |
05:30 |
```

23.2.2.5 Accessing Datetimes from PGQL Result Sets

You can use the following APIs for retrieving datetime values from PGQL result sets.

```
LocalDate getDate(int elementIdx)
LocalDate getDate(String variableName)
LocalTime getTime(int elementIdx)
```



```
LocalTime getTime(String variableName)

LocalDateTime getTimestamp(int elementIdx)

LocalDateTime getTimestamp(String variableName)

OffsetTime getTimeWithTimezone(int elementIdx)

OffsetTime getTimeWithTimezone(String variableName)

OffsetDateTime getTimestampWithTimezone(int elementIdx)

OffsetDateTime getTimestampWithTimezone(String variableName)
```

The following example prints the birthdays of all the persons in the graph is as follows:

- JShell
- Java

JShell

```
opg4j> var resultSet = session.queryPgql("""
    SELECT n.name, n.date_of_birth
        FROM MATCH (n) ON people_graph
ORDER BY n.name
""")
opg4j> while (resultSet.next()) {
    ...> System.out.println(resultSet.getString(1) + " has birthday " + resultSet.getDate(2));
    ...> }
opg4j> resultSet.close()
```

```
import java.time.LocalDate;
import oracle.pgx.api.*;
...

PgqlResultSet resultSet = session.queryPgql(
    " SELECT n.name, n.date_of_birth\n" +
    " FROM MATCH (n) ON people_graph\n" +
    "ORDER BY n.name");

while (resultSet.next()) {
    System.out.println(resultSet.getString(1) + " has birthday " +
    resultSet.getDate(2));
}

resultSet.close();
```



The result of the query is as follows:

```
Judy has birthday 1989-01-15
Klara has birthday 2001-01-29
Pete has birthday 1995-08-01
```

In addition to the Java types from the new java.time package, the legacy java.util.Date is also supported through the following APIs:

```
Date getLegacyDate(int elementIdx)
Date getLegacyDate(String variableName)
```



The legacy <code>java.util.Date</code> can store dates, times, as well as timestamps, so these two APIs can be used for accessing values of any of the five datetime types.

23.3 Handling Asynchronous Requests in Graph Server (PGX)

This guide explains in detail the asynchronous methods supported by the PGX API.

The PGX API is designed to be asynchronous. This means that all of its core methods ending with *Async* **do not** block the caller thread until the request is completed. Instead, a PgxFuture object is instantly returned.

You can perform the following three actions on the returned PgxFuture object:

- Block
- Chain
- Cancel
- Blocking Operation
- Chaining Operation
- Cancelling Operation
- Handling Concurrent Asynchronus Operations

23.3.1 Blocking Operation

You can easily get the result by calling the get() method on the PgxFuture. The get() blocks the caller thread until the result is available:

```
PgxFuture<PgxSession> sessionPromise = instance.createSessionAsync("my-
session");
try {
    // block caller thread
    PgxSession session = sessionPromise.get();
```



```
// do something with session
...
} catch (InterruptedException e) {
    // caller thread was interrupted while waiting for result
} catch (ExecutionException e) {
    // an exception was thrown during asynchronous computation
    Throwable cause = e.getCause(); // the actual exception is nested
}
```

PGX provides blocking convenience methods for every *Async* method, which calls the <code>get()</code> method. Typically, those methods have the same name as the asynchronous method they wrap, but without the *Async* suffix. For example, the preceding code snippet is equal to:

```
try {
    // block caller thread
    PgxSession session = instance.createSession("my-session");
    // do something with session
    ...
} catch (InterruptedException e) {
    // caller thread was interrupted while waiting for result
} catch (ExecutionException e) {
    // an exception was thrown during asynchronous computation
    Throwable cause = e.getCause(); // the actual exception is nested
}
```

23.3.2 Chaining Operation

The graph server (PGX) ships a version of Java 8's CompletableFuture named PgxFuture, a monadic enhancement of the Future interface.

The CompletableFuture allows chaining of asynchronous computations without polling or the need of deeply nested callbacks (also known as callback hell). All PgxFuture instances returned by PGX APIs are instances of CompletableFuture and can be chained without the need of Java 8.

```
import java.util.concurrent.CompletableFuture

...

final GraphConfig graphConfig = ...
instance.createSessionAsync("my-session")
   .thenCompose(new Fun<PgxSession, CompletableFuture<PgxGraph>>() {
   @Override
   public CompletableFuture<PgxGraph> apply(PgxSession session) {
      return session.readGraphWithPropertiesAsync(graphConfig);
   }
}).thenAccept(new Action<PgxGraph>() {
   @Override
   public void accept(PgxGraph graph) {
      // do something with loaded graph
   }
});
```



The asynchronous chaining in the preceding example is explained as follows:

- The first line in the code makes an asynchronous call to <code>createSessionAsync()</code> to create a session.
 - Once the promise is resolved, it returns a PgxFuture object, which is the newly created PgxSession.
- The code then calls the .thenCompose() handler by passing a function which takes the PgxSession object as an argument.
 Inside the function, there is another asynchronous readGraphWithPropertiesAsync() request which return another PgxFuture object.
 - The outer PgxFuture object returned by .thenCompose() gets resolved when the readGraphWithPropertiesAsync() request completes.
- This is followed by the .thenAccept() handler. The function that is passed to .thenAccept() does not return anything. Therefore, the future return type of .thenAccept() is PgxFuture<Void>.

Blocking Versus Chaining

For most use cases, you can block the caller thread. However, blocking can quickly lead to poor performance or deadlocks once things get more complex. As a rule, use blocking to quickly analyze selected graphs in a sequential manner, for example, in shell scripts or during interactive analysis using the interactive PGX shell.

Use chaining for applications built on top of PGX.

23.3.3 Cancelling Operation

You can cancel a pending request by invoking the cancel method of the returned PqxFuture instance.

For example:

```
PgxFuture<Object> promise=...
// do something else
promise.cancel(); // will cancel computation
```

Any subsequent calls to promise.get() will result in a CancellationException being thrown.



Due to Java's cooperative threading model, it might take some time before PGX actually stops the computation.

23.3.4 Handling Concurrent Asynchronus Operations

Using the PgxSession#runConcurrently API provided by the graph server (PGX), you can submit a list of suppliers of asynchronous APIs to run concurrently in the PGX server.

For example:

```
import oracle.pgx.api.*;
    Supplier<PgxFuture<?>> asyncRequest1 = () ->
session.readGraphWithPropertiesAsync(...);
    Supplier<PgxFuture<?>> asyncRequest2 = () ->
session.getAvailableSnapshotsAsync(...);

    List<Supplier<PgxFuture<?>>> supplierList = Arrays.asList(asyncRequest1,
asyncRequest2);

    //executing the async requests with the enabled optimization feature
    List<?> results = session.runConcurrently(supplierList);

    //the supplied requests are mapped to their results and orderly collected
    PgxGraph graph = (PgxGraph) results.get(0);
    Deque<GraphMetaData> metaData = (Deque<GraphMetaData>) results.get(1);
```

23.4 Graph Client Sessions

The graph server (PGX) assumes there may be multiple concurrent clients, and each client submits request to the shared PGX server independently.

Each session has its own workspace in PGX and is isolated from other sessions.

You can share graphs or properties among sessions.

Creating Sessions

The following methods in the ServerInstance class are used to create sessions:

- Java
- Python

Java

```
PgxSession createSession(String source)
PgxSession createSession(String source, long idleTimeout, long taskTimeout,
TimeUnit unit)
```

The preceding methods accept the following arguments:

- source is any arbitrary string that describes the client. Currently, this string is only used for logging purposes.
- The user can specify the idle timeout (idleTimeout) and task timeout (taskTimeout)
 when creating a new session. If these values are not specified, default values are used.
 See Configuration Parameters for the Graph Server (PGX) Engine for more information
 on graph server (PGX) configuration options.



Python

```
import pypgx
session = pypgx.get_session()
```

Destroying Sessions

To destroy a session, simply call:

- JShell
- Java
- Python

JShell

```
session.destroyAsync();
```

Java

session.destroy();

Python

session.destroy()

Administrators can destroy sessions by ID using the following code:

instance.killSession(sessionId);



Calling administrative methods by default requires special authorization in client/server mode.

When a session is destroyed, PGX reclaims all of the resources associated with the session. Specifically, all transient data is destroyed immediately. See Managing Transient Data for more information on transient data.

However, PGX may choose to keep the loaded graph instance in memory for caching purposes, especially if a graph instance is shared by multiple clients. In summary, every graph remains in memory until no client is using it.

Note:

A session can be destroyed automatically via the session time-out mechanism. See Configuration Parameters for the Graph Server (PGX) Engine for more information on graph server (PGX) configuration options.

23.5 Graph Mutation and Subgraphs

This guide discusses the several methods provided by the graph server (PGX) for mutating graph instances.

You can use the mutation and subgraph methods that are defined in the PgxGraph class, to mutate a graph.

Note:

All of the mutating methods create a new graph or snapshot instance as the mutated version of the original graph, rather than mutating the original graph directly.

- · Altering Graphs
- Simplifying and Copying Graphs
- Transposing Graphs
- Undirecting Graphs
- Advanced Multi-Edge Handling
- Creating a Subgraph
- Creating a Bipartite Subgraph
- · Creating a Sparsified Subgraph

23.5.1 Altering Graphs

This section explains the graph alteration mutation used to add or remove vertex and edge providers of a graph.

You can add or remove vertex and edge providers in a graph that has been loaded or created previously. Providers can be added from existing datasources, or new empty providers can be created. The mutation can either create a new independent graph, or create a new snapshot for the graph.

The following topics explain in detail on adding and removing vertex and edge providers:

You must first create a graph-alteration builder to start altering an existing graph. For example, the following code shows how to start a graph alteration on a graph that is stored in a variable graph:



- JShell
- Java
- Python

JShell

```
opg-jshell> var alterationBuilder = graph.alterGraph()
```

Java

```
import oracle.pgx.api.*;
import oracle.pgx.api.graphalteration.GraphAlterationBuilder;
GraphAlterationBuilder alterationBuilder = graph.alterGraph();
```

Python

```
alteration builder = graph.alter graph()
```

Loading Or Removing Additional Vertex or Edge Providers

23.5.1.1 Loading Or Removing Additional Vertex or Edge Providers

You can alter your graph by adding or removing vertex or edge providers from a specific datasource. Alternatively you can also add empty vertex or edge providers.

Keys in Additionally Loaded Providers

The vertex and edge providers that are loaded must provide the respective keys in accordance with the vertex ID and edge ID strategy of the graph being altered. If the ID strategy is <code>KEYS_AS_IDS</code>, the provider must create a key mapping. But, if the ID strategy is <code>UNSTABLE GENERATED IDS</code>, it must not create the key mapping.

- Loading Vertex Providers
- Loading Edge Providers
- Adding Additional Empty Vertex or Edge Providers
- Removing Vertex or Edge Providers
- Applying the Alteration and Building a Graph or Snapshot

23.5.1.1.1 Loading Vertex Providers

You can add a vertex provider by calling

```
alterationBuilder.addVertexProvider(EntityProviderConfig
vertexProviderConfig).
```

vertexProviderConfig is a vertex provider configuration and it provides configuration details such as:

- location of the datasource to load from
- the stored format
- properties of the vertex provider

Adding a Vertex Provider from a JSON Configuration

You can add the provider by calling alterationBuilder.addVertexProvider(String pathToVertexProviderConfig) where pathToVertexProviderConfig points to a file accessible from the client that contains a JSON representation of a vertex provider configuration.

For example, a vertex provider configuration can be stored in a JSON file as shown:

You can then add the vertex provider as shown in the following example:

- JShell
- Java
- Python

JShell

```
// Loading by indicating the path to the JSON file
opg4j> alterationBuilder.addVertexProvider("<path-to-vertex-provider-
configuration>")
$9 ==>
oracle.pgx.api.graphalteration.internal.GraphAlterationBuilderImpl@48d464cf

// Or by first loading the content of a JSON file into an
EntityProviderConfig object
opg4j> var vertexProviderConfig = new
AnyFormatEntityProviderConfigFactory().fromPath("<path-to-vertex-provider-
configuration>")
```



```
vertexProviderConfig ==>
{"format":"rdbms", "name":"Accounts", "database_table_name":"BANK_ACCOUNT
S", "loading":{"create_key_mapping":true}, "key_type":"integer", "props":
[{"type":"integer", "name":"ID"},
{"type":"string", "name":"NAME"}], "key_column":"ID"}
opg4j> alterationBuilder.addVertexProvider(vertexProviderConfig)
$15 ==>
oracle.pgx.api.graphalteration.internal.GraphAlterationBuilderImpl@77e2
a5d3
```

Java

```
// Loading by indicating the path to the JSON file
alterationBuilder.addVertexProvider("<path-to-vertex-provider-
configuration>");

// Or by first loading the content of a JSON file into an
EntityProviderConfig object
EntityProviderConfig vertexProviderConfig = new
AnyFormatEntityProviderConfigFactory().fromPath("<path-to-vertex-
provider-configuration>");
alterationBuilder.addVertexProvider(vertexProviderConfig);
```

Python

```
# Loading by indicating the path to the JSON file
alterationBuilder.add_vertex_provider("<path-to-vertex-provider-
configuration>");
```

Adding a Vertex Provider Programmatically Using an API

Alternatively, the vertex provider configuration can be built programmatically:

- JShell
- Java

JShell



```
opg4j> var vertexProviderConfig = vertexProviderConfigBuilder.build()
vertexProviderConfig ==> {"error_handling":
{},"format":"rdbms","name":"Accounts","database_table_name":"BANK_ACCOUNTS","
loading":{"create_key_mapping":true},"attributes":
{},"key_type":"long","props":
[{"dimension":0,"type":"integer","name":"ID"}],"key_column":"ID"}
opg4j> alterationBuilder.addVertexProvider(vertexProviderConfig)
$24 ==>
oracle.pqx.api.graphalteration.internal.GraphAlterationBuilderImpl@7b303608
```

Java

```
RdbmsEntityProviderConfigBuilder vertexProviderConfigBuilder = new
RdbmsEntityProviderConfigBuilder()
    .setName("Accounts")
    .setKeyColumn("ID")
    .setDatabaseTableName("BANK_ACCOUNTS")
    .addProperty("ID", PropertyType.INTEGER);
EntityProviderConfig vertexProviderConfig =
vertexProviderConfigBuilder.build();
alterationBuilder.addVertexProvider(vertexProviderConfig);
```

23.5.1.1.2 Loading Edge Providers

You can add an edge provider by calling

alterationBuilder.addEdgeProvider(EntityProviderConfig edgeProviderConfig) where edgeProviderConfig. edgeProviderConfig is an edge provider configuration and it provides configuration details such as:

- location of the datasource to load from
- the stored format
- properties of the edge provider

The source and destination vertex providers to which it is linked must either be already in the base graph (and not removed in the alteration), or added with the alteration.

Adding an Edge Provider from a JSON Configuration

You can also add the provider by calling alterationBuilder.addEdgeProvider(String pathToEdgeProviderConfig) where pathToEdgeProviderConfig points to a file accessible from the client that contains a JSON representation of an edge provider configuration.

For example, an edge provider configuration can be stored in a JSON file as shown:

```
{
  "name": "Transfers",
  "format": "rdbms",
  "database_table_name": "BANK_EDGES_AMT",
  "key_column": "ID",
  "source_column": "SRC_ID",
```



You can then add the edge provider as shown in the following example:

- JShell
- Java
- Python

JShell

```
// Loading by indicating the path to the JSON file
opg4j> alterationBuilder.addEdgeProvider("<path-to-edge-provider-
configuration>")
$10 ==>
oracle.pgx.api.graphalteration.internal.GraphAlterationBuilderImpl@48d4
64cf
// Or by first loading the content of a JSON file into an
EntityProviderConfig object
opg4j> EntityProviderConfig edgeProviderConfig = new
AnyFormatEntityProviderConfigFactory().fromPath("<path-to-edge-
provider-configuration>")
edgeProviderConfig ==>
{"format":"rdbms", "source vertex provider":"Accounts", "name":"Transfers
","database table name": "BANK EDGES AMT", "loading":
{"create key mapping":false}, "source column": "SRC ID", "destination colu
mn":
"DEST ID", "key type": "long", "destination vertex provider": "Accounts", "p
rops":[{"type":"float", "name":"AMOUNT"}], "key column":"ID"}
opg4j> alterationBuilder.addEdgeProvider(edgeProviderConfig)
$26 ==>
oracle.pgx.api.graphalteration.internal.GraphAlterationBuilderImpl@7b30
3608
```

```
// Loading by indicating the path to the JSON file
alterationBuilder.addEdgeProvider("<path-to-edge-provider-
configuration>");
```

```
// Or by first loading the content of a JSON file into an
EntityProviderConfig object
EntityProviderConfig edgeProviderConfig = new
AnyFormatEntityProviderConfigFactory().fromPath("<path-to-edge-provider-configuration>");
alterationBuilder.addEdgeProvider(edgeProviderConfig);
```

Python

```
# Loading by indicating the path to the JSON file
alterationBuilder.add_edge_provider("<path-to-edge-provider-configuration>");
```

Adding an Edge Provider Programmatically Using an API

Alternatively, the edge provider configuration can be built programmatically:

- JShell
- Java

JShell

```
opg4j> RdbmsEntityProviderConfigBuilder edgeProviderConfigBuilder = new
RdbmsEntityProviderConfigBuilder().
...>
                                                   setName("Transfers").
...>
                                                   setKeyColumn("id").
...>
                                                   setSourceColumn("src id").
...>
setDestinationColumn("dest id").
setSourceVertexProvider("Accounts").
setDestinationVertexProvider("Accounts").
...>
                                                   createKeyMapping(true).
setDatabaseTableName("bank txns").
addProperty("from acct id", PropertyType.LONG).
                                                   addProperty("to_acct_id",
...>
PropertyType.LONG).
...>
                                                   addProperty("amount",
PropertyType.LONG)
edgeProviderConfigBuilder ==>
oracle.pgx.config.RdbmsEntityProviderConfigBuilder@5a5f65b9
opg4j> EntityProviderConfig edgeProviderConfig =
edgeProviderConfigBuilder.build()
edgeProviderConfig ==> {"error handling":
```



```
{},"attributes{},"destination_column":"dest_id","key_type":"long","dest
ination_vertex_provider":"Accounts","key_column":"id","format":"rdbms",
"source_vertex_provider":
"Accounts","name":"Transfers","database_table_name":"bank_txns","loadin
g":{"create_key_mapping":true},"source_column":"src_id","props":
[{"dimension":0,"type":"long","name":"from_acct_id"},
{"dimension":0,"type":"long",
"name":"to_acct_id"},{"dimension":0,"type":"long","name":"amount"}]}

opg4j> alterationBuilder.addEdgeProvider(edgeProviderConfig)
$30 ==>
oracle.pgx.api.graphalteration.internal.GraphAlterationBuilderImpl@441c
cfd7
```

Java

```
RdbmsEntityProviderConfigBuilder edgeProviderConfigBuilder = new
RdbmsEntityProviderConfigBuilder()
.setName("Transfers")
.setKeyColumn("id")
.setSourceColumn("src id")
.setDestinationColumn("dest id").
.setSourceVertexProvider("Accounts")
.setDestinationVertexProvider("Accounts")
.createKeyMapping(true)
.setDatabaseTableName("bank txns")
.addProperty("from acct id", PropertyType.LONG)
.addProperty("to_acct_id", PropertyType.LONG)
.addProperty("amount", PropertyType.LONG);
EntityProviderConfig edgeProviderConfig =
edgeProviderConfigBuilder.build();
alterationBuilder.addEdgeProvider(edgeProviderConfig);
```

23.5.1.1.3 Adding Additional Empty Vertex or Edge Providers

You can also add empty vertex or edge providers, without having the providers connected to any specific datasource.

The names and types of the properties of each empty provider can be specified programmatically. Similarly, you can also specify if a key mapping for the providers needs to be created.

Adding Additional Empty Vertex Providers

You can add an empty vertex provider by calling

alterationBuilder.addEmptyVertexProvider(String vertexProviderName). You can then add properties, specify the key column, create the key mapping programmatically as shown in the following example.

See the **GraphAlterationEmptyVertexProviderBuilder** Interface in the Javadoc for more details.

- JShell
- Java

Java

Adding Additional Empty Edge Providers

You can add an empty edge provider by calling

alterationBuilder.addEmptyEdgeProvider(String providerName, String sourceProvider, String destProvider). You can then add properties, specify the key column, create the key mapping programmatically as shown in the following example.

See the **GraphAlterationEmptyEdgeProviderBuilder** Interface in the Javadoc for more details.

- JShell
- Java

JShell



Java

```
alterationBuilder.addEmptyEdgeProvider("TransactionProvider",
"Accounts", "Accounts")
.setLabel("Transfers")
.createKeyMapping(false)
.addProperty("Description", PropertyType.STRING);
```

23.5.1.1.4 Removing Vertex or Edge Providers

You can remove an edge provider by calling

alterationBuilder.removeEdgeProvider(String edgeProviderName), where edgeProviderName is the name of the edge provider to be removed from the graph.

Similarly, calling alterationBuilder.removeVertexProvider(String vertexProviderName) will result in the graph to not contain that specific vertex provider. If that vertex provider was the source or destination provider for some edge providers in the base graph, those edge providers should also be removed before the application of the alteration or an exception will be thrown.

It is possible to indicate that the edge providers associated to a removed vertex provider should be automatically removed by calling

alterationBuilder.cascadeEdgeProviderRemovals(boolean cascadeEdgeProviderRemovals) **With** cascadeEdgeProviderRemovals **Set to** true.

23.5.1.1.5 Applying the Alteration and Building a Graph or Snapshot

You must call <code>alterationBuilder.build()</code>, once the different vertex and edge providers have been added or removed in the alteration to actually apply the operation. By calling <code>alterationBuilder.build()</code>, a new graph is created and that graph contains all the providers of the base graph excluding the removed providers, and the additionally loaded providers.

You can also call <code>alterationBuilder.buildNewSnapshot()</code>, in which case, a new snapshot for the base graph is created and that snapshot contains all the providers of the base graph excluding the removed providers, and the additionally loaded providers.

23.5.2 Simplifying and Copying Graphs

You can create a simplified version of the graph by calling the simplify() method.

- Java
- Python

Java



```
SelfEdges selfEdges, TrivialVertices trivialVertices, Mode mode, String newGraphName)
```

Python

The first two arguments (vertexProps and edgeProps) list which properties will be copied into the newly created simplified graph instance. PGX provides convenience constants VertexProperty.ALL, EdgeProperty.ALL and VertexProperty.NONE, EdgeProperty.NONE to specify all properties or none properties to be stored, respectively.

The next three arguments determine which operations will be performed to simplify the graph.

- multiEdges: if MultiEdges.REMOVE_MULTI_EDGES, eliminate multiple edges between a source vertex and a destination vertex, that is, leave at most one edge between two vertices. MultiEdges.KEEP_MULTI_EDGES indicates to keep them. By default, PGX picks one edge out of the multi-edges and takes its properties. See Advanced Multi-Edge Handling for more fine-grained control over the edge properties during simplification.
- selfEdges: if SelfEdges.REMOVE_SELF_EDGES, eliminate every edge whose source and destination are the same vertex. SelfEdges.KEEP MULTI EDGES indicates to keep them.
- trivialVertices: if TrivialVertices.REMOVE_TRIVIAL_VERTICES, eliminate all the vertices that have neither incoming edges nor outgoing edges.
 TrivialVertices.KEEP TRIVIAL VERTICES indicates to keep them.

The mode argument, if set to <code>Mode.MUTATE_IN_PLACE</code>, requests that the mutation occurs directly on the specified graph instance without creating a new one. If set to <code>Mode.CREATE_COPY</code>, the method will create a new graph instance with the new name in <code>newGraphName</code>. If <code>newGraphName</code> is omitted (or <code>null</code>), PGX will generate a unique graph name.

The return value of this method is the simplified PgxGraph instance.

The Mode.MUTATE_IN_PLACE option is only applicable if the graph is marked as mutable. Every graph is immutable by default when loaded into PGX. To make a PgxGraph mutable, the client should create a private copy of the graph first, using one of the following methods:

- Java
- Python

Java

```
PgxGraph clone()
PgxGraph clone(String newGraphName)
PgxGraph clone(Collection<VertexProperty<?, ?>> vertexProps,
Collection<EdgeProperty<?>> edgeProps, String newGraphName)
```



Python

clone(self, vertex properties=True, edge properties=True, name=None)

As with <code>simplify()</code>, the user can specify optional properties of the graph to copy with <code>vertexProps</code> and <code>edgeProps</code>. If no properties are specified, all of the original graph's properties will be copied into the new graph instance. The user can specify the name of the newly created graph instance with <code>newGraphName</code>.

23.5.3 Transposing Graphs

You can create a transposed version of the graph.

- Java
- Python

Java

Python

The edgeLabelMapping argument can be used to rename edge labels. If any key in the given map does not exist as an edge label, it will be ignored.

edgeLabelMapping argument can also be an empty Map or null.

- null: if argument is null, edge labels from source graph will be removed on transposed graph. (default behavior when using convenience methods).
- empty Map: if argument is an empty Map, edge labels from source graph will be neither removed or renamed. Instead, it will be kept as it is in source graph.

See Simplifying and Copying Graphs for the meaning of the other parameters.

Additionally, the graph server (PGX) provides the following convenience methods from the PgxGraph class for the common operation of copying all vertex and edge properties into the transposed graph instance:



- transpose (Mode mode, String newGraphName)
- transpose (String newGraphName)
- transpose (Mode mode)

23.5.4 Undirecting Graphs

The following methods create the undirected version of a graph instance:

- Java
- Python

Java

```
PgxGraph undirect()
PgxGraph undirect(String newGraphName)
PgxGraph undirect(MultiEdges multiEdges, SelfEdges selfEdges,
TrivialVertices trivialVertices, Mode mode, String newGraphName)
PgxGraph undirect(Collection<VertexProperty<?, ?>> vertexProps,
Collection<EdgeProperty<?>> edgeProps, MultiEdges multiEdges, SelfEdges
selfEdges, Mode mode, String newGraphName)
```

Python

The first two methods create an undirected version of the graph while copying all of the vertex properties. <code>newGraphName</code> is an optional argument to specify the name of the newly created graph instance.

In contrast, the third and fourth methods concurrently perform *undirecting* and *simplifying* of a graph. See Simplifying and Copying Graphs for the meaning of each parameter.

All methods return an object of the undirected PgxGraph type.

An undirected graph has some restrictions. Some algorithms are only supported on directed graphs or are not yet supported for undirected graphs. Further, PGX does not support to store undirected graphs nor reading from undirected formats. Since the edges do not have a direction anymore, the behavior of pgxEdge.getSource() or pgxEdge.getDestination() can be ambiguous. In order to provide deterministic results, PGX will always return the vertex with the smaller internal id as source and the other as destination vertex.



23.5.5 Advanced Multi-Edge Handling

Both <code>simplify()</code> and <code>undirect()</code> support the removal of multi-edges using <code>MultiEdges.REMOVE_MULTI_EDGES</code>. If this parameter is set, all multi-edges in this graph are removed, that is, collapsed. Whenever several multi-edges with edge properties are collapsed into one edge, you can choose one of the following two strategies supported by the graph server (PGX) to decide how to treat the corresponding properties:

- Picking
- Merging

If you choose picking, the graph server (PGX) picks one edge out of every set of multiedges and copies all its properties including the edge label and key into the new graph. In the case of merging, the graph server (PGX) creates a completely new edge out for every set of multi-edges. PGX determines the properties of these new edges by applying a MergingFunction on every property of the multi-edges.

If there are no multi-edges between two vertices, that is, zero or only one edge, the chosen strategy does not have an effect on the outcome. The edge is kept with all its properties as it is.

- Picking
- Merging
- StrategyBuilder in General

23.5.5.1 Picking

This strategy can be used to pick an edge out of multi-edges. The graph server (PGX) allows the user to define several picking criteria. You can pick by:

- Property
- Label
- Edge-ID

Every picking criteria has to be combined with a PickingStrategyFunction. PGX supports either PickingStrategyFunction.MIN and PickingStrategyFunction.MAX, which picks the edge whose property/label/id is either minimal or maximal. If one does not specify a picking criteria, PGX will non-deterministically pick an edge out of the multi-edges.

A PickingStrategy can be created using a PickingStrategyBuilder, which can be retrieved by calling createPickingStrategyBuilder() on the target graph.

You can call one of the following functions as per your chosen picking criteria:

PickingStrategyBuilder setPickByEdgeId(PickingStrategyFunction pickingStrategyFunction)
PickingStrategyBuilder setPickByLabel(PickingStrategyFunction pickingStrategyFunction)
PickingStrategyBuilder setPickByProperty(EdgeProperty edgeProperty, PickingStrategyFunction pickingStrategyFunction)
PickingStrategyBuilder setPickByProperty(String propertyName, PickingStrategyFunction pickingStrategyFunction)



The following figure shows how PGX picks the edge with the *minimal* cost and takes all its properties.

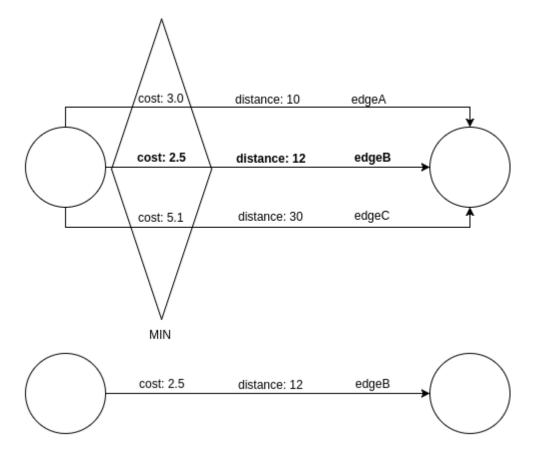


Figure 23-1 Picking Strategy

23.5.5.2 Merging

This strategy can be used to merge the properties of multi-edges. The graph server (PGX) allows the user to define a MergingFunction for every property. Currently, PGX supports the following functions:

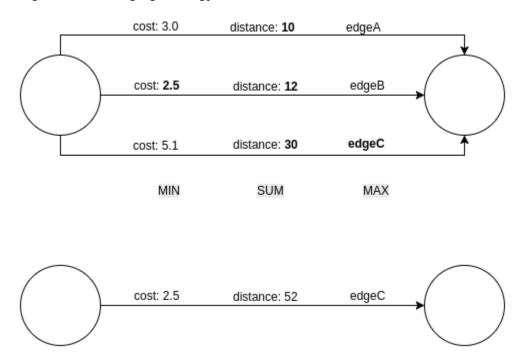
- MergingFunction.MIN
- MergingFunction.MAX
- MergingFunction.SUM



SUM is only defined on numeric properties.

The following figure shows how the graph server (PGX) merges the different edge properties and labels. It takes the *minimal* cost, the *sum* of distances and the *maximal* edge label.

Figure 23-2 Merging Strategy



23.5.5.3 StrategyBuilder in General

By default, both the StrategyBuilders use the same values as in the convenience methods of simplify() and undirect(). This includes that all properties are kept by default. If one wants to drop specific properties, one can either use the dropVertexProperty() or dropEdgeProperty() functions.

```
MutationStrategyBuilder setNewGraphName(String newGraphName)
MutationStrategyBuilder setCopyMode(Mode mode)
MutationStrategyBuilder setTrivialVertices(TrivialVertices
trivialVertices)
MutationStrategyBuilder setSelfEdges(SelfEdges selfEdges)
MutationStrategyBuilder setMultiEdges(MultiEdges multiEdges)
MutationStrategyBuilder
dropVertexProperties(Collection<VertexProperty<?, ?>> vertexProperty)
MutationStrategyBuilder dropEdgeProperties(Collection<EdgeProperty<?>>> edgeProperty)
MutationStrategyBuilder dropVertexProperty(VertexProperty<?, ?>> vertexProperty)
MutationStrategyBuilder dropVertexProperty(EdgeProperty<?>> edgeProperty)
MutationStrategyBuilder dropEdgeProperty(EdgeProperty<?>> edgeProperty)
MutationStrategyBuilder dropEdgeProperty(EdgeProperty<?>> edgeProperty)
MutationStrategy build()
```

Simplify() and undirect() can be called using a MutationStrategy as follows:

MutationStrategy strategy = strategyBuilder.build()
PgxGraph simplifiedGraph graph.simplify(strategy)
//OR
PgxGraph undirectedGraph graph.undirect(strategy)



23.5.6 Creating a Subgraph

PGX provides the following methods for creating subgraphs via a filter (see Filter Expressions for more information) expression:

- Java
- Python

Java

```
PgxGraph filter(GraphFilter graphFilter)
PgxGraph filter(GraphFilter graphFilter, String newGraphName)
PgxGraph filter(Collection<VertexProperty<?, ?>> vertexProps,
Collection<EdgeProperty<?>> edgeProps, GraphFilter graphFilter, String
newGraphName)
```

Python

filter(self, graph_filter, vertex_properties=True, edge_properties=True,
name=None)

As in the other graph mutating methods, the user has the option to specify the name of the subgraph with the <code>newGraphName</code> parameter and of choosing the vertex and edge properties to be copied into the subgraph (<code>vertexProps</code> and <code>edgeProps</code>). All of the preceding methods return a <code>PgxGraph</code> object which represents the created subgraph.

All filter methods require a <code>GraphFilter</code> argument containing a filter expression. Fundamentally, the filter expression is a Boolean expression that is evaluated for every vertex and edge in the original graph (in parallel). If the expression is evaluated as <code>true</code> for the vertex or edge, then that vertex or edge is included in the subgraph.

See Creating Subgraphs for more information on how to create subgraphs from graphs loaded into memory.

23.5.7 Creating a Bipartite Subgraph

The graph server (PGX) enables the client to create a bipartite subgraph. The following methods return the created BipartiteGraph instance:

- Java
- Python



Java

```
BipartiteGraph bipartiteSubGraphFromLeftSet(VertexSet<?> vertexSet)
BipartiteGraph bipartiteSubGraphFromLeftSet(VertexSet<?> vertexSet,
String newGraphName)
BipartiteGraph
bipartiteSubGraphFromLeftSet(Collection<VertexProperty<?, ?>>
vertexProps, Collection<EdgeProperty<?>> edgeProps, VertexSet<?>
vertexSet, String newGraphName)
BipartiteGraph
bipartiteSubGraphFromLeftSet(Collection<VertexProperty<?, ?>>
vertexProps, Collection<EdgeProperty<?>> edgeProps, VertexSet<?>
vertexProps, Collection<EdgeProperty<?>> edgeProps, VertexSet<?>
vertexSet, String newGraphName, String isLeftPropName)
```

Python

bipartite_sub_graph_from_left_set(self, vset, vertex_properties=True,
edge properties=True, name=None, is left name=None)

These methods require an additional argument <code>vertexSet</code>, which points to a set of vertices (see Using Collections and Maps for more information) whose elements (vertices) would contain the left vertices (that is, vertices on the left side of the bipartite graph that have only edges to vertices on the right side) in the resulting bipartite graph.

When creating the bipartite subgraph, PGX automatically inserts an additional boolean vertex property <code>isLeft</code>. The value of this property is set <code>true</code> for the left vertices and <code>false</code> for the right vertices in the bipartite subgraph. The name of the <code>isLeft</code> vertex property can be obtained with <code>getIsLeftPropertyAsync()</code> on the returned <code>BipartiteGraph</code> object.

The user has the option to specify a name for the newly created graph (newGraphName) as well as a custom name for the Boolean left-vertex indicating property (isLeftPropName). The user can also specify the vertex and edge properties to be copied into the newly created graph instance (vertexProps and edgeProps).

23.5.8 Creating a Sparsified Subgraph

The graph server (PGX) supports creating a sparsified subgraph of a graph:

- Java
- Python

Java

```
PgxGraph sparsify(double e)
PgxGraph sparsify(double e, String newGraphName)
```



PgxGraph sparsify(Collection<VertexProperty<?, ?>> vertexProps,
Collection<EdgeProperty<?>> edgeProps, double e, String newGraphName)

Python

sparsify(self, sparsification, vertex_properties=True, edge_properties=True,
name=None)

The double argument e is the sparsification coefficient with a value between 0.0 and 1.0.

The user again has the option to specify the name for the newly created graph (newGraphName) as well as the vertex and edge properties to be copied into the newly created graph instance (vertexProps and edgeProps).

The returned PgxGraph object represents a sparsified subgraph which has fewer edges than the original graph.

23.6 Graph Builder and Graph Change Set

This guide explains the GraphBuilder API used for creating graphs and the GraphChangeSet interface used for modifying loaded graphs.

- Building Graphs Using GraphBuilder Interface
- Modifying Loaded Graphs Using ChangeSet

23.6.1 Building Graphs Using GraphBuilder Interface

Using the GraphBuilder interface, you can create graphs programmatically.

The basic work flow for creating graphs from scratch is:

- 1. Acquire a modifiable graph builder to accumulate all the new vertices and edges
- 2. Add vertices and edges to the graph builder
- 3. Create a PgxGraph out of the accumulated changes
- Creating a Simple Graph
- Adding a Vertex Property
- · Using Strings as Vertex Identifiers
- Referencing a Vertex for Creating Edges
- · Adding an Edge Property and a Label
- Using Graph Builder with Implicit IDs

23.6.1.1 Creating a Simple Graph

This section shows an example of creating a simple graph using the createGraphBuilder() method.



- JShell
- Java
- Python

```
opg4j> var builder = session.createGraphBuilder()
builder ==> GraphBuilderImpl[session=cd201ac9-e73f-447c-9cec-
cd929293acc3, vertexChanges=0, edgeChanges=0]
opg4j> builder.addEdge(1, 2)
opg4j> builder.addEdge(2, 3)
opg4j> builder.addEdge(2, 4)
opg4j> builder.addEdge(3, 4)
opg4j> builder.addEdge(4, 2)
opg4j> var graph = builder.build()
graph ==>
PgxGraph[name=anonymous graph 16, N=4, E=5, created=1629805890550]
Java
import oracle.pgx.api.*;
PgxSession session = Pgx.createSession("example");
GraphBuilder<Integer> builder = session.createGraphBuilder();
builder.addEdge(1, 2);
builder.addEdge(2, 3);
builder.addEdge(2, 4);
builder.addEdge(3, 4);
builder.addEdge(4, 2);
PgxGraph graph = builder.build();
Python
from pypgx import get session
session = get_session(session_name="example")
builder = session.create graph builder()
builder.add edge(1, 2)
builder.add edge(2, 3)
builder.add edge(2, 4)
builder.add edge(3, 4)
builder.add edge(4, 2)
```



graph = builder.build()

Also, note that the following:

- A call to addEdge consists of the new unique edge ID, the source vertex ID and the
 destination vertex ID.
- No graph configuration is required.
- When adding edges, all vertices that do not already exist are created on the fly as edges are created.
- GraphBuilder supports only the following two generation strategies for creating vertices and edge IDs:
 - USER_IDS (the default value)
 - AUTO_GENERATED

23.6.1.2 Adding a Vertex Property

You can also add vertices separately and assign property values to them.

The following example shows how to add a vertex property using the GraphBuilder interface.

- JShell
- Java
- Python

JShell

```
opg4j> var builder = session.createGraphBuilder()
opg4j> builder.addVertex(1).setProperty("double-prop", 0.1)
opg4j> builder.addVertex(2).setProperty("double-prop", 2.0)
opg4j> builder.addVertex(3).setProperty("double-prop", 0.3)
opg4j> builder.addVertex(4).setProperty("double-prop", 4.56789)
opg4j> builder.addEdge(1, 2)
opg4j> builder.addEdge(2, 3)
opg4j> builder.addEdge(2, 4)
opg4j> builder.addEdge(3, 4)
opg4j> builder.addEdge(4, 2)
opg4j> var graph = builder.build()
```

Java

```
import oracle.pgx.api.*;

PgxSession session = Pgx.createSession("example");
GraphBuilder<Integer> builder = session.createGraphBuilder();

builder.addVertex(1).setProperty("double-prop", 0.1);
builder.addVertex(2).setProperty("double-prop", 2.0);
```



```
builder.addVertex(3).setProperty("double-prop", 0.3);
builder.addVertex(4).setProperty("double-prop", 4.56789);
builder.addEdge(1, 2);
builder.addEdge(2, 3);
builder.addEdge(2, 4);
builder.addEdge(3, 4);
builder.addEdge(4, 2);
PgxGraph graph = builder.build();
Python
from pypgx import get session
session = get session(session name="example")
builder = session.create graph builder()
```

builder.add vertex(1).set property("double-prop", 0.1) builder.add vertex(2).set property("double-prop", 2.0) builder.add vertex(3).set property("double-prop", 0.3) builder.add vertex(4).set property("double-prop", 4.56789)

```
graph=builder.build()
```

If the value for a property is missing for a vertex or an edge, a default value is assumed as shown:

Table 23-4 Default Property Values

builder.add edge(1, 2) builder.add edge(2, 3) builder.add edge(2, 4) builder.add edge(3, 4) builder.add edge(4, 2)

Properties	Default Values
Numeric	0 (or the respective equivalent)
Boolean	false
Date	1.1.1970 00:00:00
String	null



🕜 Tip:

Multiple calls to setProperty can be chained to set multiple property values at once.

23.6.1.3 Using Strings as Vertex Identifiers

By default, integer vertex IDs are used to identify a vertex. But, the type of the vertex ID can also be a long or a string.

In order to implement this, you must specify the vertex ID type when creating the graph using the GraphBuilder as shown:

- JShell
- Java
- Python

JShell

opg4j> GraphBuilder<String> builder =
session.createGraphBuilder(IdType.STRING)

PgxGraph graph = builder.build();

```
opg4j> builder.addVertex("vertex 1").setProperty("double-prop", 0.1)
opg4j> builder.addVertex("vertex 2").setProperty("double-prop", 2.0)
opq4j> builder.addVertex("vertex 3").setProperty("double-prop", 0.3)
opg4j> builder.addVertex("vertex 4").setProperty("double-prop", 4.56789)
opg4j> builder.addEdge("vertex 1", "vertex 2")
opg4j> builder.addEdge("vertex 2", "vertex 3")
opg4j> builder.addEdge("vertex 2", "vertex 4")
opg4j> builder.addEdge("vertex 3", "vertex 4")
opg4j> builder.addEdge("vertex 4", "vertex 2")
opg4j> var graph = builder.build()
Java
import oracle.pgx.api.*;
import oracle.pqx.common.types.IdType;
PgxSession session = Pgx.createSession("example");
GraphBuilder<String> builder = session.createGraphBuilder(IdType.STRING);
builder.addVertex("vertex 1").setProperty("double-prop", 0.1);
builder.addVertex("vertex 2").setProperty("double-prop", 2.0);
builder.addVertex("vertex 3").setProperty("double-prop", 0.3);
builder.addVertex("vertex 4").setProperty("double-prop", 4.56789);
builder.addEdge("vertex 1", "vertex 2");
builder.addEdge("vertex 2", "vertex 3");
builder.addEdge("vertex 2", "vertex 4");
builder.addEdge("vertex 3", "vertex 4");
builder.addEdge("vertex 4", "vertex 2");
```



Python

```
from pypgx import get_session

session = get_session(session_name="example")
builder = session.create_graph_builder(id_type='string')
builder.add_vertex("vertex 1").set_property("double-prop", 0.1)
builder.add_vertex("vertex 2").set_property("double-prop", 2.0)
builder.add_vertex("vertex 3").set_property("double-prop", 0.3)
builder.add_vertex("vertex 4").set_property("double-prop", 4.56789)

builder.add_edge("vertex 1", "vertex 2")
builder.add_edge("vertex 2", "vertex 3")
builder.add_edge("vertex 2", "vertex 4")
builder.add_edge("vertex 3", "vertex 4")
builder.add_edge("vertex 4", "vertex 2")

graph = builder.build()
```

23.6.1.4 Referencing a Vertex for Creating Edges

You can also avoid entering the full vertex ID when adding an edge. For this, you must obtain a reference to the vertex that is created, which can be later used in the <code>addEdge</code> statement.

- JShell
- Java
- Python

JShell

```
opg4j> GraphBuilder<String> builder =
session.createGraphBuilder(IdType.STRING)

opg4j> var v1 = builder.addVertex("vertex 1").setProperty("double-
prop", 0.1)
opg4j> var v2 = builder.addVertex("vertex 2").setProperty("double-
prop", 2.0)
opg4j> var v3 = builder.addVertex("vertex 3").setProperty("double-
prop", 0.3)
opg4j> var v4 = builder.addVertex("vertex 4").setProperty("double-
prop", 4.56789)

opg4j> builder.addEdge(v1, v2)
opg4j> builder.addEdge(v2, v3)
opg4j> builder.addEdge(v2, v4)
opg4j> builder.addEdge(v2, v4)
opg4j> builder.addEdge(v3, v4)
```



```
opg4j> builder.addEdge(v4, v2)
opg4j> var graph = builder.build()
Java
import oracle.pgx.api.*;
import oracle.pqx.common.types.IdType;
PgxSession session = Pgx.createSession("example");
GraphBuilder<String> builder = session.createGraphBuilder(IdType.STRING);
VertexBuilder<String> v1 = builder.addVertex("vertex 1").setProperty("double-
prop", 0.1);
VertexBuilder<String> v2 = builder.addVertex("vertex 2").setProperty("double-
prop", 2.0);
VertexBuilder<String> v3 = builder.addVertex("vertex 3").setProperty("double-
prop", 0.3);
VertexBuilder<String> v4 = builder.addVertex("vertex 4").setProperty("double-
prop", 4.56789);
builder.addEdge(v1, v2);
builder.addEdge(v2, v3);
builder.addEdge(v2, v4);
builder.addEdge(v3, v4);
builder.addEdge(v4, v2);
PgxGraph graph = builder.build();
Python
from pypgx import get session
session = get session(session name="example")
builder = session.create graph builder(id type='string')
v1 = builder.add vertex("vertex 1").set property("double-prop", 0.1)
v2 = builder.add vertex("vertex 2").set property("double-prop", 2.0)
v3 = builder.add vertex("vertex 3").set_property("double-prop", 0.3)
v4 = builder.add vertex("vertex 4").set property("double-prop", 4.56789)
builder.add edge(v1, v2)
builder.add edge(v2, v3)
builder.add edge(v2, v4)
builder.add edge(v3, v4)
builder.add edge(v4, v2)
graph = builder.build()
```



23.6.1.5 Adding an Edge Property and a Label

The following examples show how to add an edge property and a label to a graph.

- JShell
- Java
- Python

JShell

```
opg4j> var builder = session.createGraphBuilder(IdType.STRING)
opg4j> var v1 = builder.addVertex("vertex 1").setProperty("double-
prop", 0.1)
opg4j> var v2 = builder.addVertex("vertex 2").setProperty("double-
prop", 2.0)
opg4j> var v3 = builder.addVertex("vertex 3").setProperty("double-
prop", 0.3)
opg4j> var v4 = builder.addVertex("vertex 4").setProperty("double-
prop", 4.56789)
opg4j> builder.addEdge(v1, v2).setProperty("edge-prop",
"edge_prop_1_2").setLabel("label")
opg4j> builder.addEdge(v2, v3).setProperty("edge-prop",
"edge prop 2 3").setLabel("label")
opg4j> builder.addEdge(v2, v4).setProperty("edge-prop",
"edge prop 2 4").setLabel("label")
opg4j> builder.addEdge(v3, v4).setProperty("edge-prop",
"edge prop 3 4").setLabel("label")
opg4j> builder.addEdge(v4, v2).setProperty("edge-prop",
"edge prop 4 2").setLabel("label")
opg4j> var graph = builder.build()
Java
import oracle.pgx.api.*;
import oracle.pgx.common.types.IdType;
PgxSession session = Pgx.createSession("example");
GraphBuilder<String> builder =
session.createGraphBuilder(IdType.STRING);
VertexBuilder<String> v1 = builder.addVertex("vertex
1").setProperty("double-prop", 0.1);
```

VertexBuilder<String> v2 = builder.addVertex("vertex

VertexBuilder<String> v3 = builder.addVertex("vertex

2").setProperty("double-prop", 2.0);

3").setProperty("double-prop", 0.3);



```
VertexBuilder<String> v4 = builder.addVertex("vertex 4").setProperty("double-
prop", 4.56789);
builder.addEdge(v1, v2).setProperty("edge-prop",
"edge prop 1 2").setLabel("label");
builder.addEdge(v2, v3).setProperty("edge-prop",
"edge prop 2 3").setLabel("label");
builder.addEdge(v2, v4).setProperty("edge-prop",
"edge prop 2 4").setLabel("label");
builder.addEdge(v3, v4).setProperty("edge-prop",
"edge prop 3 4").setLabel("label");
builder.addEdge(v4, v2).setProperty("edge-prop",
"edge prop 4 2").setLabel("label");
PgxGraph graph = builder.build();
Python
from pypqx import get session
session = get session(session name="example")
builder = session.create graph builder(id type='string')
v1 = builder.add vertex("vertex 1").set property("double-prop", 0.1)
v2 = builder.add vertex("vertex 2").set property("double-prop", 2.0)
v3 = builder.add vertex("vertex 3").set property("double-prop", 0.3)
v4 = builder.add vertex("vertex 4").set property("double-prop", 4.56789)
builder.add edge(v1, v2).set property("edge-prop",
"edge prop 1 2").set label("label")
builder.add edge(v2, v3).set property("edge-prop",
"edge_prop_2_3").set_label("label")
builder.add edge(v2, v4).set property("edge-prop",
"edge prop 2 4").set label("label")
builder.add edge(v3, v4).set property("edge-prop",
"edge prop 3 4").set label("label")
builder.add edge(v4, v2).set property("edge-prop",
"edge prop 4 2").set label("label")
```

23.6.1.6 Using Graph Builder with Implicit IDs

graph = builder.build()

The GraphBuilder supports an $AUTO_GENERATED$ generation strategy that allows to omit the edge or vertex IDs.

In this generation strategy, the graph server (PGX) will automatically assign IDs to the entities being added to the changeset. PgxSession supports

createGraphBuilder(IdGenerationStrategy vertexIdGenerationStrategy,
IdGenerationStrategy edgeIdGenerationStrategy) and createGraphBuilder(IdType

The following example illustrates creating a graph with three vertices and three edges using the GraphBuilder interface.

- JShell
- Java
- Python

JShell

```
opg4j> var builder =
session.createGraphBuilder(IdGenerationStrategy.AUTO_GENERATED,
IdGenerationStrategy.AUTO_GENERATED)

opg4j> var v1 = builder.addVertex()
opg4j> var v2 = builder.addVertex()
opg4j> var v3 = builder.addVertex()
opg4j> builder.addEdge(v1, v2)
opg4j> builder.addEdge(v1, v3)
opg4j> builder.addEdge(v3, v2)
opg4j> var graph = builder.build()
```

Java

```
import oracle.pgx.api.*;

PgxSession session = Pgx.createSession("example");
GraphBuilder<Integer> builder =
session.createGraphBuilder(IdGenerationStrategy.AUTO_GENERATED,
IdGenerationStrategy.AUTO_GENERATED);

VertexBuilder<Integer> v1 = builder.addVertex();
VertexBuilder<Integer> v2 = builder.addVertex();
VertexBuilder<Integer> v3 = builder.addVertex();
builder.addEdge(v1, v2);
builder.addEdge(v1, v3);
builder.addEdge(v1, v3);
builder.addEdge(v3, v2);

PgxGraph graph = builder.build();
```

Python

```
>>> builder =
session.create_graph_builder(vertex_id_generation_strategy='auto_genera
ted', edge_id_generation_strategy='auto_generated')
>>> v1 = builder.add_vertex()
```



```
>>> v2 = builder.add_vertex()
>>> v3 = builder.add_vertex()
>>> builder.add_edge(v1, v2)
>>> builder.add_edge(v1, v3)
>>> builder.add_edge(v3, v2)
>>> graph = builder.build()
```

23.6.2 Modifying Loaded Graphs Using ChangeSet

This guide explains how to add and remove vertices and edges from already loaded graphs.

As a prerequisite, you must have a graph already loaded into the graph server (PGX). See Reading Graphs from Oracle Database into the Graph Server (PGX) for more information.

You can now use the GraphChangeSet interface to modify the loaded graphs.



Modifying undirected graphs is not supported in graph server (PGX) 21.3.

- Modifying Vertices
- Adding Edges
- GraphChangeSet with Partitioned IDs
- Error Handling when Using a ChangeSet

23.6.2.1 Modifying Vertices

You can add, remove and modify vertices using the GraphChangeSet object.

- JShell
- Java
- Python

JShell

```
opg4j> var changeSet = graph.<Integer>createChangeSet()
opg4j> changeSet.addVertex(42).setProperty("prop", 23)
opg4j> changeSet.updateVertex(128).setProperty("prop", 5)
opg4j> changeSet.removeVertex(1908)
opg4j> var updatedGraph = changeSet.build()
```



```
opg4j> updatedGraph.hasVertex(42) // Evaluates to: true
opg4j> updatedGraph.hasVertex(1908) // Evaluates to: false

Java
import oracle.pgx.api.*;
GraphChangeSet<Integer> changeSet = graph.createChangeSet();
changeSet.addVertex(42).setProperty("prop", 23);
changeSet.updateVertex(128).setProperty("prop", 5);
changeSet.removeVertex(1908);

PgxGraph updatedGraph = changeSet.build();

Python

from pypgx.api import *
change_set = graph.create_change_set()
change_set.add_vertex(42).set_property("prop", 23)
changeSet.update_vertex(128).set_property("prop", 5)
changeSet.remove vertex(1908)
```

23.6.2.2 Adding Edges

You can also add edges to a graph using GraphChangeSet.

updated graph = change set.build()

- JShell
- Java
- Python

JShell

```
opg4j> var changeSet2 = updatedGraph.<Integer>createChangeSet()
opg4j> changeSet2.addEdge(333, 42).setProperty("cost", 42.3)
opg4j> changeSet2.addEdge(42, 99)
opg4j> var updatedGraph2 = changeSet2.build()
```



Java

```
import oracle.pgx.api.*;
GraphChangeSet<Integer> changeSet2 = graph.createChangeSet();
changeSet2.addEdge(333, 42).setProperty("cost", 42.42);
changeSet2.addEdge(42, 99);

PgxGraph updatedGraph2 = changeSet2.build();

Python

from pypgx.api import *
change_set_2 = graph.create_change_set()
changeSet2.add_edge(333, 42).set_property("cost", 42.42)
changeSet2.add_edge(42, 99)
updated graph 2 = change set 2.build()
```

Note that by calling changeSet2.build(), you created a brand new graph with a unique name assigned by the graph server (PGX). If need be, you can specify a name argument to the build() method.

Additionally, you can create a new snapshot on top of the current graph with the buildNewSnapshot() method. See Creating a Snapshot via ChangeSet for more information.

23.6.2.3 GraphChangeSet with Partitioned IDs

You can use the <code>GraphChangeSet</code> API with graph with partitioned IDs. Ensure to set both the vertex ID generation strategy as well as the edge ID generation strategy to <code>IdGenerationStrategy.USER_IDS</code>. Furthermore, make sure to set the vertex ID type to <code>String</code>. An edge ID type does not need to be specified.

You can add, update and remove vertices and edges as shown in the following examples:

- Java
- Python

Java

```
GraphChangeSet<String> changeSet =
g.createChangeSet(IdGenerationStrategy.USER_IDS,
IdGenerationStrategy.USER_IDS);
changeSet.addVertex("Accounts(1002)").setProperty("NAME","User1002");
changeSet.updateVertex("Accounts(4)").setProperty("NAME","User4");
changeSet.removeVertex("Accounts(3)");
changeSet.addEdge("Transfers(5002)", "Accounts(5)",
```



```
"Accounts(6)").setProperty("AMOUNT", 12.50);
changeSet.updateEdge("Transfers(5)").setProperty("DESCRIPTION",
'Transfer from User');
changeSet.removeEdge("Transfers(5001)");
PgxGraph g1 = changeSet.build();
```

Python

```
change_set = graph.create_change_set(vertex_id_generation_strategy =
'user_ids', edge_id_generation_strategy = 'user_ids')
change_set.add_vertex("Accounts(1002)").set_property("NAME",
"User1002")
change_set.update_vertex("Accounts(4)").set_property("NAME", "User4")
change_set.remove_vertex("Accounts(3)")
change_set.remove_edge("Transfers(5001)")
PgxGraph g1 = change_set.build()
```

Note:

You cannot use the setLabel() API when IDs are partitioned. The vertex or edge will be labelled automatically based on the label attached to the provider (for which the name is provided as part of the ID). Similarly, you cannot set the vertex or edge key properties through the setProperty() API as the value is already extracted from the vertex or edge ID.

23.6.2.4 Error Handling when Using a ChangeSet

Error handling while populating a ChangeSet or while applying a ChangeSet to the existing graph can be configured by setting the InvalidChangePolicy. The options are:

- OnInvalidChange.ERROR: throws an exception (This is the default configuration)
- OnInvalidChange.IGNORE: ignores the issue and continues
- OnInvalidChange.IGNORE_AND_LOG: ignores the issue, logs in DEBUG log level and continues
- OnInvalidChange.IGNORE_AND_LOG_ONCE: only logs the first occurrence of each issue type

Issues that can be ignored with InvalidChangePolicy include trying to remove a vertex or an edge that does not exist in the graph, property type mismatch, updates to non existing properties, providing a vertex ID with wrong type or invalid vertex or edge providers.

The following example, tries to remove vertex 9032 which does not exist in the graph. By configuring <code>IGNORE_AND_LOG</code>, this action will be ignored while the property value update for vertex 99 will be applied successfully.



- JShell
- Java

```
opg4j> var changeSet3 = updatedGraph2.<Integer>createChangeSet()
opg4j> changeSet3.setInvalidChangePolicy(OnInvalidChange.IGNORE_AND_LOG)

opg4j> changeSet3.removeVertex(9032)
opg4j> changeSet3.updateVertex(99).setProperty("prop1", 17)
opg4j> var updatedGraph3 = changeSet3.build() // will log that a vertex removal was ignored

opg4j> var prop1Val = updatedGraph3.getVertex(99).getProperty("prop1") // evaluates to 17
```

Java

```
import oracle.pgx.api.*;

GraphChangeSet<Integer> changeSet3 = graph.createChangeSet();
changeSet3.setInvalidChangePolicy(OnInvalidChange.IGNORE_AND_LOG);
changeSet3.removeVertex(9032);
changeSet3.updateVertex(99).setProperty("prop1", 17);
PgxGraph updatedGraph3 = changeSet3.build(); // will log that a vertex removal was ignored

int prop1Val = updatedGraph3.getVertex(99).getProperty("prop1"); // evaluates to 17
```

Note:

When connecting to a remote graph server (PGX), error handling log messages will not be relayed to the client. In such a case, you need access to the server logs to determine which issues have been ignored. For this, you must update the default Logback configuration file in /etc/oracle/graph/logback.xml and the graph server (PGX) logger configuration file in /etc/oracle/graph/logback-server.xml to log the DEBUG logs. You can then view the ignored issues in /var/opt/log/pgx-server.log file.

Add Existing Edges and Vertices

The error handling for adding a vertex or an edge where its ID is already used in the graph or in an incompatible ChangeSet action can be configured with AddExistingVertexPolicy and AddExistingEdgePolicy.





The default setting for AddExistingVertexPolicy and AddExistingEdgePolicy is IGNORE. This is different from InvalidChangePolicy where the default is ERROR.

23.7 Managing Transient Data

This guide discusses how to handle transient properties and collections.

The graph server (PGX) allows each client to maintain its own isolated workspace, called session. Clients may create additional data objects in their own session, which they can then use for analysis.

- Managing Transient Properties
- Managing Collections and Scalars

23.7.1 Managing Transient Properties

The graph server (PGX) adopts the Property Graph data model. Once a graph is loaded into PGX, the graph instance itself and its original properties are set as immutable. However, the client can create and attach additional properties to the graph dynamically. These extra properties are referred to as *transient* properties and are mutable by the client

The methods for creating transient properties are available in PgxGraph:

- Java
- Python

Java

VertexProperty<ID, V> createVertexPropertyAsync(PropertyType type)
VertexProperty<ID, V> createVertexPropertyAsync(PropertyType type,
String name)
EdgeProperty<V> createEdgePropertyAsync(PropertyType type)
EdgeProperty<V> createEdgePropertyAsync(PropertyType type, String name)

In the preceding code:

- PropertyType: is an enum for the data type of the property, which must be one of the primitive types supported by PGX.
- name: is an optional argument to assign a unique name to the newly created property. If no name is specified, PGX will assign one to the client.





Names must be unique. There cannot be two different vertex or edge properties for the same graph and with the same name.

Python

```
create_vertex_property(self,data_type,name=None)
```

All methods return a Property object, which represent the newly created transient property. Both of the underlying classes, VertexProperty<ID, V> and EdgeProperty<V>, are parametrized with the value type V the property holds. V matches the given PropertyType. VertexProperty<ID, V> is additionally parametrized with the vertex ID type. This is due to PGX support of several types of vertex identifiers. See our graph configuration chapter on how to specify the vertex ID type of a graph. EdgeProperty<V> is not parametrized with the edge ID type, because PGX only supports edge identifiers of type long.

- Java
- Python

Java

```
GraphConfig config = GraphConfigBuilder.forFileFormats(...)
    ...
    .setVertexIdType(IdType.LONG)
    ...
    .build();

PgxGraph G = session.readGraphWithProperties(config);
VertexProperty<Long, String> p1 =
G.createVertexProperty(PropertyType.STRING);
EdgeProperty<Double> p2 = G.createEdgeProperty(PropertyType.DOUBLE);
```

Python

```
G = session.read_graph_with_properties(config)
p1 = G.create_vertex_property("string")
p2 = G.create_edge_property("double")
```

To delete a transient property from the session, call destroyAsync() (or destroy()) on the property object.



23.7.2 Managing Collections and Scalars

The client can create graph-bound vertex and edge collections to use during the analysis with the following methods in PgxGraph:

- Java
- Python

Java

```
VertexSequence<E> createVertexSequence()
VertexSequence<E> createVertexSequence(String name)
VertexSet<E> createVertexSet()
VertexSet<E> createVertexSet(String name)
EdgeSequence createEdgeSequence()
EdgeSequence createEdgeSequence(String name)
EdgeSet createEdgeSet()
EdgeSet createEdgeSet(String name)
```

Python

```
create_edge_sequence(self, name=None)
create_vertex_sequence(self, name=None)
create_edge_set(self, name=None)
create_edge_sequence(self, name=None)
```

PGX also supports scalar collections such as set and sequence. Each of these collections can hold elements of various primitive data types like INTEGER, LONG, FLOAT, DOUBLE or BOOLEAN. Scalar collections are session-bound and can be created with the following methods in PgxSession:

```
ScalarSet<T> createSet(PropertyType contentType, String name)
ScalarSequence<T> createSequence(PropertyType contentType, String name)
ScalarSet<T> createSet(PropertyType contentType)
ScalarSequence<T> createSequence(PropertyType contentType)
```

In the preceding code, the optional argument (name) specifies the name of the newly created collection. If omitted, PGX chooses a name for the client. As with properties, the collections holding vertices are parametrized with the ID type of the vertices. Refer to graph configuration chapter to learn how to specify the vertex ID type of a graph.

The return value is the collection object which points to the newly created empty collection.

To drop a collection from the session, call destroy() on the collection object.



To check which collections are currently allocated for a graph you can use the following method:

- Java
- Python

Java

Map<String, PgxCollection<? extends PgxEntity<?>, ?>> getCollections()

Python

get_collections(self)

The returned map contains the names of the collections as keys and the collections as values. The collections can be casted to the matching collection subclass.

PGX supports special Map collection types and allows users to map between different data types (oracle.pgx.common.types.PropertyType). Maps can be created using PgxGraph or PgxSession APIs, the difference is that the latter supports only non graph-related types, and that the created maps directly depend on the session:

```
PgxMap<K, V> createMap(PropertyType keyType, PropertyType valType)
PgxMap<K, V> createMap(PropertyType keyType, PropertyType valType, String mapName)
```

Similarly, scalar variables can be created in the client session using the following methods:

- Java
- Python

Java

```
Scalar<T> createScalar(PropertyType type, String newScalarName)
Scalar<T> createScalar(PropertyType type)
```

Python

create scalar(self,data type,name=None)



These collections and scalar variables can then be passed as arguments to graph algorithms. See Using Custom PGX Graph Algorithms for more information.

23.8 Graph Versioning

This guide describes the different ways to work with graph snapshots.

A graph can have multiple snapshots associated with it, reflecting different versions of the graph. All snapshots of a graph have the same graph configuration associated.

The following topics explains the various operations you can perform on graph snapshots:

- Configuring the Snapshots Source
- Creating a Snapshot via Refreshing
- Creating a Snapshot via ChangeSet
- Checking Out the Latest Snapshots of a Graph
- · Checking Out Different Snapshots of a Graph
- Directly Loading a Specific Snapshot of a Graph

23.8.1 Configuring the Snapshots Source

Snapshots can be created from two sources: Refreshing and ChangeSet.

Refreshing is available for graphs that are read from a persistent data source, that is, a file. When the data source has changed with respect to the version stored in the graph server (PGX), it can be read again manually by calling the

PgxSession.readGraphWithProperties() method. Similarly, if auto-refresh is set for the graph, the graph server (PGX) automatically reads the data source and creates new snapshots when the data source has changed.

Instead, a ChangeSet is a set of changes to a graph that the user creates and populates via the PGX ChangeSet API. Once a ChangeSet is created and populated with the desired changes, the user can simply call

GraphChangeSet.buildNewSnapshot() to create a new snapshot for the graph. In this way, you are empowered to integrate changes coming from any source into the graph and build snapshots out of them.

Only one source of snapshots is allowed for a single graph and is chosen during graph configuration via the snapshots_source option, which can be set to either REFRESH or CHANGE_SET. In case the snapshots_source option is not explicitly set by the user, the following default settings apply:

- If the graph is from a persistent data source, the default value is REFRESH, so that snapshots can be created only by calling

 PgxSession.readGraphWithProperties() (or via auto-refresh, if configured).
- If the graph is transient, that is, built from a graph builder, the default value is
 CHANGE_SET, since the graph is not backed by a persistent data source from which
 changes can be read. It is for this reason, CHANGE_SET is the only admissible value
 for transient graphs.

Additionally, the following restrictions apply:



- If auto-refresh is enabled, then snapshots come from reading the backing data source and hence only REFRESH is admissible for the snapshots source option.
- If the user attempts to create snapshots in a way that is different from the configuration (for example, by calling GraphChangeSet.buildNewSnapshot() when the graph's snapshots source is REFRESH), the operation is invalid and an exception is thrown.

23.8.2 Creating a Snapshot via Refreshing

You can create a snapshot via refreshing by performing the following steps:

- 1. Create a session and load the graph into memory.
- Check the available snapshots of the graph with PgxSession.getAvailableSnapshots() method.
 - JShell
 - Java
 - Python

JShell

```
opg4j> session.getAvailableSnapshots(G)
==> GraphMetaData [getNumVertices()=4, getNumEdges()=4, memoryMb=0,
dataSourceVersion=1453315103000, creationRequestTimestamp=1453315122669
(2016-01-20 10:38:42.669), creationTimestamp=1453315122685 (2016-01-20 10:38:42.685), vertexIdType=integer, edgeIdType=long]
```

Java

```
Deque<GraphMetaData> snapshots = session.getAvailableSnapshots(G);
for( GraphMetaData metaData : snapshots ) {
   System.out.println( metaData );
}
```

Python

```
snapshots = session.get_available_snapshots(G)
for metadata in snapshots:
    print(metadata)
```

- 3. Edit the source file to contain an additional vertex and an additional edge or insert two rows in the database.
- 4. Reload the updated graph within the same session as you loaded the original graph. A new snapshot is created.



- JShell
- Java
- Python

```
opg4j> var G = session.readGraphWithProperties( G.getConfig(),
true )
==> PGX Graph named 'sample 2' bound to PGX session
'a1744e86-65fb-4bd1-b2dc-5458b20954a9' registered at PGX Server
Instance running in embedded mode
opg4j> session.getAvailableSnapshots(G)
==> GraphMetaData [getNumVertices()=4, getNumEdges()=4, memoryMb=0,
dataSourceVersion=1453315103000,
creationRequestTimestamp=1453315122669 (2016-01-20 10:38:42.669),
creationTimestamp=1453315122685 (2016-01-20 10:38:42.685),
vertexIdType=integer, edgeIdType=long]
==> GraphMetaData [getNumVertices()=5, getNumEdges()=5, memoryMb=3,
dataSourceVersion=1452083654000,
creationRequestTimestamp=1453314938744 (2016-01-20 10:35:38.744),
creationTimestamp=1453314938833 (2016-01-20 10:35:38.833),
vertexIdType=integer, edgeIdType=long]
```

Java

```
G = session.readGraphWithProperties( G.getConfig(), true );
Deque<GraphMetaData> snapshots = session.getAvailableSnapshots( G );
```

Python

```
G =
session.read_graph_with_properties(G.config,update_if_not_fresh=True)
```

Note that there are two GraphMetaData objects in the call for available snapshots, one with 4 vertices and 4 edges and one with 5 vertices and 5 edges.

- 5. Verify that the graph variable points to the newly loaded graph using getNumVertices() and getNumEdges() methods.
 - JShell
 - Java
 - Python



```
opg4j> G.getNumVertices()
==> 5
opg4j> G.geNumEdges()
==> 5
```

Java

```
int vertices = G.getNumVertices();
long edges = G.getNumEdges();
```

Python

```
vertices = G.num_vertices
edges = G.num edges
```

23.8.3 Creating a Snapshot via ChangeSet

You can create a graph snapshot with ChangeSet via the PGX Java API. When you want to create the graph from a persistent data source, you can use

PgxSession.readGraphWithProperties() with the snapshots_source configuration option set to CHANGE SET.

You can create a snapshot via ChangeSet by performing the following steps:

- 1. Create a snapshot of a transient graph from database:
 - JShell
 - Java
 - Python

JShell

```
opg4j> var builder = session.createGraphBuilder()
opg4j> builder.addEdge(1, 2)
opg4j> builder.addEdge(2, 3)
opg4j> builder.addEdge(2, 4)
opg4j> builder.addEdge(3, 4)
opg4j> builder.addEdge(4, 2)
opg4j> var graph = builder.build()
```

Java

```
import oracle.pgx.api.*;
```



```
GraphBuilder<Integer> builder = session.createGraphBuilder();
builder.addEdge(1, 2);
builder.addEdge(2, 3);
builder.addEdge(2, 4);
builder.addEdge(3, 4);
builder.addEdge(4, 2);

PgxGraph graph = builder.build();

Python
builder = session.create_graph_builder();
builder.add_edge(1, 2)
builder.add_edge(2, 3)
builder.add_edge(2, 4)
builder.add_edge(3, 4)
builder.add_edge(4, 2)
graph = builder.build()
```

- 2. Create a ChangeSet from graph and populate it. The following example shows adding a new edge between vertices 1 and 4:
 - JShell
 - Java
 - Python

```
opg4j> var changeSet = graph.<Integer>createChangeSet()
opg4j> changeSet.addEdge(6, 1, 4)
```

Java

```
import oracle.pgx.api.*;
GraphChangeSet<Integer> changeSet = graph.createChangeSet();
changeSet.addEdge(6, 1, 4);
```

Python

```
changeSet = graph.create_change_set()changeSet.add_edge(1,4,6)
```

- 3. Create a second snapshot using GraphChangeSet.buildNewSnapshot() as shown in the following code:
 - JShell
 - Java
 - Python

JShell

```
opg4j> var secondSnapshot = changeSet.buildNewSnapshot()
opg4j> session.getAvailableSnapshots(secondSnapshot).size()
==> 2
```

Java

```
PgxGraph secondSnapshot = changeSet.buildNewSnapshot();
System.out.println( session.getAvailableSnapshots(secondSnapshot).size() );
```

Python

```
second_snapshot = change_set.build_new_snapshot()
print(len(session,get available snapshots()))
```

Thus two snapshots, referenced via the variables graph and secondSnapshot are created.

23.8.4 Checking Out the Latest Snapshots of a Graph

With multiple snapshots of a graph being available and regardless of their source, you can check out a specific snapshot using the PgxSession.setSnapshot() method. You can use the LATEST_SNAPSHOT constant of PgxSession to easily check out the latest available snapshot, as shown in the following example:

- JShell
- Java

```
opg4j> session.setSnapshot( G, PgxSession.LATEST_SNAPSHOT )
==> null
```



```
opg4j> session.getCreationTimestamp()
==> 1453315122685
```

```
session.setSnapshot( G, PgxSession.LATEST_SNAPSHOT );
System.out.println(session.getCreationTimestamp());
```

See the printed timestamp to verify the most recent snapshot.

23.8.5 Checking Out Different Snapshots of a Graph

You can also check out a specific snapshot, again using the PgxSession.setSnapshot().

For example, consider the following two snapshots of a graph:

```
==> GraphMetaData [getNumVertices()=4, getNumEdges()=4, memoryMb=0,
dataSourceVersion=1453315103000,
creationRequestTimestamp=1453315122669 (2016-01-20 10:38:42.669),
creationTimestamp=1453315122685 (2016-01-20 10:38:42.685),
vertexIdType=integer, edgeIdType=long]
==> GraphMetaData [getNumVertices()=5, getNumEdges()=5, memoryMb=3,
dataSourceVersion=1452083654000,
creationRequestTimestamp=1453314938744 (2016-01-20 10:35:38.744),
creationTimestamp=1453314938833 (2016-01-20 10:35:38.833),
vertexIdType=integer, edgeIdType=long]
```

To check out a specific snapshot of the graph, you must pass the creationTimestamp of the snapshot you want to load to setSnapshot().

For example, if G is pointing to the newest graph with 5 vertices and 5 edges, but you want to analyze the older graph, you need to set the snapshot to 1453315122685.

- JShell
- Java
- Python

```
opg4j> G.getNumVertices()
==> 5
opg4j> G.getNumEdges()
==> 5
opg4j> session.setSnapshot( G, 1453315122685 )
==> null
```



```
opg4j> G.getNumVertices()
==> 4
opg4j> G.getNumEdges()
==> 4
```

```
session.setSnapshot(G,1453315122685);
```

Python

```
session.set snapshot(G,1453315122685)
```

Note that setting the snapshot, changes the number of vertices and edges from 5 to 4.

Alternatively, you can also retrieve the creation timestamp of each snapshot from its associated <code>GraphMetaData</code> object via the <code>GraphMetaData.getCreationTimestamp()</code> method. The easiest way to get the <code>GraphMetaData</code> information of all the snapshots is to use the <code>PgxSession.getAvailableSnapshots()</code> method, which returns a collection of <code>GraphMetaData</code> information of each snapshot ordered by creation timestamp from the most recent to the oldest.

23.8.6 Directly Loading a Specific Snapshot of a Graph

You can also load a specific snapshot of a graph directly using the ${\tt PgxSession.readGraphAsOf()} \ method. \ This is a shortcut for loading a graph with {\tt readGraphWithProperties()} \ followed by a {\tt setSnapshot()}.$

Consider two snapshots of a graph that are already loaded into the PGX session. The following example shows how to get a reference to a specific snapshot:

- 1. Get a graph configuration for the graph:
 - JShell
 - Java
 - Python

```
opg4j> var config =
GraphConfigFactory.forAnyFormat().fromPath("<path_to_json>")
==> {"format":"adj list", ... }
```



```
GraphConfig config =
GraphConfigFactory.forAnyFormat().fromPath("<path to json>");
```

Python

```
config =
GraphConfigFactory.for_any_format().from_path("<path_to_json>")
```

2. Check the loaded snapshots for this graph config using

```
getAvailableSnapshots():
```

- JShell
- Java
- Python

JShell

```
opg4j> session.getAvailableSnapshots(G)
==> GraphMetaData [getNumVertices()=4, getNumEdges()=4, memoryMb=0,
dataSourceVersion=1453315103000,
creationRequestTimestamp=1453315122669 (2016-01-20 10:38:42.669),
creationTimestamp=1453315122685 (2016-01-20 10:38:42.685),
vertexIdType=integer, edgeIdType=long]
==> GraphMetaData [getNumVertices()=5, getNumEdges()=5, memoryMb=3,
dataSourceVersion=1452083654000,
creationRequestTimestamp=1453314938744 (2016-01-20 10:35:38.744),
creationTimestamp=1453314938833 (2016-01-20 10:35:38.833),
vertexIdType=integer, edgeIdType=long]
```

Java

Deque<GraphMetaData> snapshots = session.getAvailableSnapshots(G);

Python

```
session.get available snapshots(G)
```

3. Check out the snapshot of the graph which has 4 vertices and 4 edges and having the timestamp 1453315122685:



- JShell
- Java
- Python

JShell

```
opg4j> var G = session.readGraphAsOf( config, 1453315122685 )
==> PGX Graph named 'sample' bound to PGX session 'a1744e86-65fb-4bd1-
b2dc-5458b20954a9' registered at PGX Server Instance running in embedded
mode
opg4j> G.getNumVertices()
==> 4
opg4j> G.getNumEdges()
==> 4
```

Java

```
PgxGraph G = session.readGraphAsOf( config, 1453315122685 );
```

Python

```
G = read graph as of(config, creation timestamp=1453315122685)
```

23.9 Labels and Properties

You can perform various actions on the graph property and label values by executing PGQL queries.

- Setting and Getting Property Values
- Getting Label Values

23.9.1 Setting and Getting Property Values

Getting Property Values

You can obtain the vertex or edge property values by executing a SELECT PGQL query on the graph.

For example:

- JShell
- Java



JShell

```
opg4j> session.queryPgql("SELECT e.src_id, e.dest_id, e.amount FROM
MATCH (n:Account) -[e:Transfers]-> (m:Account) on bank graph").print()
```

Java

```
...
...
PgxGraph g = session.getGraph("bank_graph");
String query =
     "SELECT e.src_id, e.dest_id, e.amount FROM MATCH (n:Account) -
[e:Transfers]-> (m:Account)";
g.queryPgql(query).print();
```

The resulting property values may appear as:

+-		 	 +
	src_id	dest_id	amount
+-		 	 +
	1	259	1000
	1	418	1000
	1	584	1000
	1	644	1000
	1	672	1000
	2	493	1000
	2	546	1000
	2	693	1000
	2	833	1000
	2	840	1000
+-		 	 +

Setting Property Values

You can set the vertex or edge property values by executing insert or update PGQL queries on the graph.

For example, to set a new vertex account ID on a graph using INSERT query:

- JShell
- Java

```
opg4j> PgxGraph g = session.getGraph("bank_graph_analytics")
g ==>
PgxGraph[name=bank_graph_analytics, N=1000, E=5001, created=1616312153556]
opg4j> PgxGraph g_mutable = g.clone("bank_graph_analytics_copy")
```



```
g_mutable ==>
PgxGraph[name=bank_graph_analytics_copy, N=1000, E=5001, created=1616312413799]
opg4j> g_mutable.executePgql("INSERT VERTEX v LABELS (Accounts) PROPERTIES
( v.id = 1001)")
```

```
...

PgxGraph g1 = session.readGraphWithProperties("bank_graph_analytics.json");

PgxGraph g2 = g1.clone("bank_graph_analytics_copy");

g2.executePgq1("INSERT VERTEX v " +

" LABELS ( Accounts ) " +

" PROPERTIES ( v.id = 1001 )");
```

23.9.2 Getting Label Values

You can retrieve the vertex or edge label values of a graph as shown:

The result may appear as shown:

```
+-----+
| LABEL(n) | COUNT(*) |
+-----+
| ACCOUNT | 1000 |
```

23.10 Filter Expressions

This guide explains the usage of filter expressions.

Filter expressions are applied in the following scenarios:

- Path-Finding: Include only specific vertices and edges in a path
- Sub-Graphs: Include only specific vertices and edges in a subgraph
- Set creation: Create a vertex or edge set and include only specific vertices or edges

There are two types of filter expressions:

• Vertex filters:: Evaluated on each vertex



Edge filters: Evaluated on each edge, including the two vertices it connects.

These filter expressions will evaluate to true if the current edge or vertex matches the expression or to false if it does not. Filter expressions are stateless and side-effect free.

The following short example below will evaluate to true for all edges where the source vertex's string property name is "PGX".

```
src.name="PGX"
```

- Syntax
- Type System
- Path Finding Filters
- Subgraph Filters
- Operations on Filter Expressions

23.10.1 Syntax

Trivial Expressions

Always evaluates to true:

true

Always evaluates to false:

false

Constants

Legal constants are integer, long and floating point numbers of single and double precision as well as strings literals and true and false. Long constants need to be suffixed with 1 or L. Floating point numbers are treated as double precision numbers by default. To force a certain precision you can use f or F for single precision and f or f for double precision floating point numbers. String literals are UTF-8 character sequences, surrounded by single or double quotation marks.

```
25
4294967296L
0.62f
0.33d
"Double quoted string"
'Single quoted string'
```

Vertex and Edge Identifiers

Depending on the filter type, different identifiers are valid.

Vertex Filter



Vertex filter expressions have only one keyword that addresses the vertex in the current context.

vertex denotes the vertex that is currently being evaluated by the filter expression.

vertex

Edge Filter

Edge filter expressions have several keywords that addresses the edge or its vertices in the current context.

edge denotes the edge that is currently being evaluated by the filter expression.

edge

 ${\tt dst}$ denotes the destination vertex of the current edge. ${\tt dst}$ is only valid in the subgraph context.

dst

src denotes the source vertex of the current edge. src is only valid in the subgraph context.

src

Properties

Filter expressions can access the values of vertex and edge properties.

<id>.<property>

where:

- <id>: is any vertex or edge identifier (that is, src, dst, vertex, edge).
- property>: is the name of a vertex or edge property.



This has to be the name of an edge property if the identifier is edge. Otherwise it has to be a vertex property.

If the property name is a reserved name in the filter expression syntax or contains spaces, it must be quoted in single or double quotes.

The following code accesses the 'cost' property of the source vertex.

src.cost

Temporal properties support values comparison (constants and property values) using special constructors. The default temporal formats are shown in the following table:

Table 23-5 Default Temporal Formats

Property Type	Constructor
DATE	date ('yyyy-MM-dd HH:mm:ss')
LOCAL_DATE	date 'yyyy-MM-dd'



Table 23-5 (Cont.) Default Temporal Formats

Property Type	Constructor
TIME	time 'HH:mm:ss'
TIME_WITH_TIMEZONE	time 'HH:mm:ss+/-XXX'
TIMESTAMP	timestamp 'yyyy-MM-dd HH:mm:ss'
TIMESTAMP_WITH_TIMEZONE	timestamp 'yyyy-MM-dd HH:mm:ss+/-XXX'

The following expression accesses the property 'timestamp_withTZ' of an edge and checks if it is equal to 3/27/2007 06:00+01:00.

edge.timestamp_withTZ = timestamp'2007-03-2706:00:00+01:00'



Properties of type *date* can only be checked for equality. *date* type usage is deprecated since version 2.5, instead use *local date* or *timestamp* types that support all operations.

Methods

Filter expressions support the following functions:

Degree Functions

1. outDegree() returns the number of outgoing edges of the vertex identifier. degree() is a synonym for outDegree.

```
int <id>.degree()
int <id>.outDegree()
```

The following example determines whether the out-degree of the source vertex is greater than three:

```
src.degree() > 3
```

2. inDegree () returns the number of incoming edges of the vertex identifier.

```
int <id>.inDegree()
```

Label Functions

1. hasLabel() checks if a vertex has a label.

```
boolean <id>.hasLabel('<label>')
```

The following example determines whether a vertex has the label "city":

```
vertex.hasLabel('city')
```

2. label() returns the label of an edge.

```
string <id>.label()
```

The following expression checks whether the label of an edge is "clicked_by":

```
edge.label() = 'clicked_by'
```



Relational Expressions

To compare values (e.g., property values or constants), filter expressions provide the comparison operators listed below. Note: Both == and = are synonyms.

== = != < < <= >

The following example checks whether the "cost" property of the source vertex is lower than or equals to 1.23.

src.cost <= 1.23</pre>

Vertex ID Comparison

It is also possible to filter for vertices with a specific vertex ID.

```
\langle id \rangle = \langle vertex id \rangle
```

The following example determines whether the source vertex of an edge has the vertex ID "San Francisco"

```
src = "San Francisco"
```

Regular Expressions

Strings can be matched using regular expressions.

```
<string expression> =~ '<regularexpression>'
```

The following example checks if the edge label starts with a lowercase letter and ends with a number:

```
edge.label() =~ '^[a-z].*[0-9]$'
```



The syntax followed for the pattern on the right-hand side, is Java REGEX.

Type Conversions

The following syntax allows converting the type of <expression> to <type>.

```
(<type>) <expression>
```

The following example converts the value of the 'cost' property of the source vertex to an integer value:

```
(int) src.cost
```

Boolean Expressions

Filter expressions can be composed to form other filter expressions. This can be done using the Boolean operators && (and), $|\cdot|$ (or) and ! (not).





Only boolean operands can be composed.

```
(! true) || false
edge.cost < INF && dst.visited = false
src.degree() < 10 || !(dst.visited)</pre>
```

Arithmetic Expressions

Any numeric expression can be combined using arithmetic expressions. The available arithmetic operators are: +, -, *, /, \$.



These operators only work on numeric operands.

```
1 + 5
-vertex.degree()
edge.cost * 2 > 5
src.value * 2.5 = (dst.inDegree() + 5) / dst.outDegree()
```

Operator Precedence

Operator precedences are shown in the following list, from highest precedence to the lowest. An operator on a higher level is evaluated before an operator on a lower level.

- 1. + (unary plus), (unary minus)
- 2. *,/, %
- **3.** +, -
- **4.** =,!=, <, >, <=, >=, =~
- **5.** NOT
- 6. AND
- **7.** OR

Syntactic Sugar

both and any denote the source and destination vertex of the current edge. They can be used to express a condition that should be true for both or at least either one of the two vertices. These keywords are only valid in an edge filter expression. To use them in a vertex filter results in a runtime type-checking exception.

```
both any
```

The filter expressions inside the following examples are equivalent:

```
both.property = 1
src.property = 1 && dst.property = 1
any.degree() > 1
src.degree() > 1 || dst.degree() > 1
```



23.10.2 Type System

Filter expressions are a very simple type system. There are only the following 13 types:

- integer (can be abbreviated in expressions with int)
- 2. long
- 3. float
- 4. double
- 5. boolean
- 6. string
- 7. date
- 8. time
- 9. time with timezone
- 10. timestamp
- 11. timestamp with timezone
- 12. vertex
- **13.** edge

Conversions are only allowed from one numeric type to another numeric type (i.e. integer, float, double, long).

Comparisons require both sides to be of the same (or convertible) type.

23.10.3 Path Finding Filters

Filters can be used to limit the analyzed edges when searching for a shortest path between a source and destination vertex in a graph.

An edge filter expression is evaluated against each edge that is visited during the traversal of the graph. If the filter evaluates to false on an edge, this edge will be ignored and will not appear in the resulting shortest path.

It is also possible to use a vertex filter for path finding.

A vertex filter expression is evaluated against each vertex that is visited during the traversal of the graph, except for the source and destination vertex.

If the filter evaluates to false on a vertex, the edge to this vertex and all outgoing edges of the vertex will be ignored. The vertex will not appear in the resulting shortest path.

The source and destination vertex can be any vertex in the graph and the filter is not evaluated for them.

23.10.4 Subgraph Filters

Edge Filters

An edge filter expression is evaluated for each edge in the graph. The edge filter has access to the source and destination vertex of each edge and all of its properties.



If the filter expression evaluates to true, the edge and both the source and destination vertex will appear in the subgraph.

Vertex Filters

A vertex filter expression is evaluated for every vertex in the graph.

Every vertex for which the filter expression evaluates to true will appear in the subgraph.

Every edge connecting two vertices for which the expression evaluates to true will also appear in the subgraph.

Result Set Filters

Result set edge and vertex filters allow the creation of edge and vertex sets out of a given PGQL result set.

Vertex and Edge Collection Filters

Vertex and edge collection filters allow the creation of edge and vertex filters out of a given vertex and edge collection.

23.10.5 Operations on Filter Expressions

This section explains the various operations that you can perform on filter expressions.

- Defining Filter Expressions
- Defining Result Set Filters
- · Creating a Subgraph from PGQL Result Set
- Defining Collection Filters
- Creating a Subgraph from Collection Filters
- Combining Filter Expressions
- Creating a Subgraph Using Filter Expressions with Partitioned IDs

23.10.5.1 Defining Filter Expressions

You can define a new vertex filter, as shown in the following code:

- JShell
- Java
- Python

JShell

opg4j> var vertexFilter = VertexFilter.fromExpression("vertex.name =
'PGX'")



```
VertexFilter vertexFilter = VertexFilter.fromExpression("vertex.name =
'PGX'");
```

Python

```
vertex_filter = VertexFilter("vertex.name = 'PGX'")
```

You can define a new edge filter, as shown in the following code:

- JShell
- Java
- Python

JShell

```
opg4j> var edgeFilter = EdgeFilter.fromExpression("edge.cost > 5")
```

Java

```
EdgeFilter edgeFilter = EdgeFilter.fromExpression("edge.cost > 5");
```

Python

```
vertex filter = EdgeFilter("edge.cost > 5")
```

23.10.5.2 Defining Result Set Filters

You can define a result set vertex filter, as shown in the following code:

- JShell
- Java

```
// Evaluates query on graph g to obtain a result set
opg4j> var resultSet = g.queryPgql("SELECT x FROM MATCH (x) WHERE x.age >
```



```
24")
// Define a filter on the result set for the column "x"
opg4j> var vertexFilter = VertexFilter.fromPgqlResultSet(resultSet,
"x")
// Obtain a vertex set
opg4j> var vertexSet = g.getVertices(vertexFilter)
Java
```

```
// Evaluates query on graph g to obtain result set
PgqlResultSet resultSet = g.queryPgql("SELECT x FROM MATCH (x) WHERE
x.age > 24");
// Define a filter on the result set for the column "x"
VertexFilter vertexFilter = VertexFilter.fromPgqlResultSet(resultSet,
"x");
// Obtain a vertex set
VertexSet vertexSet = g.getVertices(vertexFilter);
```

You can define a result set edge filter, as shown in the following code:

- JShell
- Java

JShell

```
// Evaluates query on graph g to obtain result set
opg4j> var resultSet = g.queryPgql("SELECT e FROM MATCH ()-[e]->()
WHERE e.weight >= 8")
// Define a filter on the result set for the column "e"
opg4j> var edgeFilter = EdgeFilter.fromPgqlResultSet(resultSet, "e")
// Obtain an edge set
opg4j> var edgeSet = g.getEdges(edgeFilter)
```

Java

```
// Evaluates query on graph g to obtain result set
PgqlResultSet resultSet = g.queryPgql("SELECT e FROM MATCH ()-[e]->()
WHERE e.weight >= 8");
// Define a filter on the result set for the column "e"
EdgeFilter edgeFilter = EdgeFilter.fromPgqlResultSet(resultSet, "e");
// Obtain an edge set
EdgeSet edgeSet = g.getEdges(edgeFilter);
```



23.10.5.3 Creating a Subgraph from PGQL Result Set

A subgraph can be obtained from a PGQL result set using result set filters.

You can create a subgraph from a result set vertex filter, as shown in the following code:

- JShell
- Java

JShell

```
// Evaluates query on graph g to obtain result set
opg4j> var resultSet = g.queryPgql("SELECT x FROM MATCH (x) WHERE x.age >
24")
// Define a filter on the result set for the column "x"
opg4j> var resultSetVertexFilter = VertexFilter.fromPgqlResultSet(resultSet,
"x")
// Create a subgraph of g containing the matched vertices in the resultSet
and the edges that connect them if any.
opg4j> var newGraph = g.filter(resultSetVertexFilter)
```

Java

```
// Evaluates query on graph g to obtain result set
PgqlResultSet resultSet = g.queryPgql("SELECT x MATCH (x) WHERE x.age > 24");
// Define a filter on the result set for the column "x"
VertexFilter resultSetVertexFilter =
VertexFilter.fromPgqlResultSet(resultSet, "x");
// Create a subgraph of g containing the matched vertices in the resultSet and the edges that connect them if any.
PgxGraph newGraph = g.filter(resultSetVertexFilter);
```

You can create a subgraph from a result set edge filter, as shown in the following code:

- JShell
- Java

```
// Evaluates query on graph g to obtain result set
opg4j> var resultSet = g.queryPgql("SELECT e FROM MATCH ()-[e]->() WHERE
e.cost < 100")
// Define a filter on the result set for the column "e"</pre>
```



```
opg4j> var resultSetEdgeFilter =
EdgeFilter.fromPgqlResultSet(resultSet, "e")
// Create a subgraph of g containing the matched edges in the
resultSet and their corresponding source and destination vertices.
opg4j> var newGraph = g.filter(resultSetEdgeFilter)
```

```
// Evaluates query on graph g to obtain result set
PgqlResultSet resultSet = g.queryPgql("SELECT e FROM MATCH ()-[e]->()
WHERE e.cost < 100");
// Define a filter on the result set for the column "e"
EdgeFilter resultSetEdgeFilter =
EdgeFilter.fromPgqlResultSet(resultSet, "e");
// Create a subgraph of g containing the matched edges in the
resultSet and their corresponding source and destination vertices.
PgxGraph newGraph = g.filter(resultSetEdgeFilter);</pre>
```

23.10.5.4 Defining Collection Filters

You can define a vetex collection filter, as shown in the following code:

- JShell
- Java

JShell

```
// Obtain a vertex collection from an algorithm, query execution or
any other way
opg4j> VertexCollection<?> vertexCollection = ...
// Define a filter from the collection
opg4j> var vertexFilter = VertexFilter.fromCollection(vertexCollection)
```

Java

```
// Obtain a vertex collection from an algorithm, query execution or
any other way
VertexCollection<?> vertexCollection = ...
// Define a filter from the collection
VertexFilter vertexFilter =
VertexFilter.fromCollection(vertexCollection);
```

You can define a edge collection filter, as shown in the following code:

- JShell
- Java

JShell

```
// Obtain an edge collection from an algorithm, query execution or any other
way
opg4j> EdgeCollection edgeCollection = ...
// Define a filter from the collection
opg4j> var edgeFilter = EdgeFilter.fromCollection(edgeCollection)
```

Java

```
// Obtain an edge collection from an algorithm, query execution or any other
way
EdgeCollection edgeCollection = ...
// Define a filter from the collection
EdgeFilter edgeFilter = EdgeFilter.fromCollection(edgeCollection);
```

23.10.5.5 Creating a Subgraph from Collection Filters

A subgraph can be obtained by using vertex or edge collection filters.

You can create a subgraph from vertex collection filter, as shown in the following code:

- JShell
- Java

JShell

```
// Obtain a vertex collection from an algorithm, query execution or any
other way
opg4j> VertexCollection<?> vertexCollection = ...
// Define a filter from the collection
opg4j> var vertexFilter = VertexFilter.fromCollection(vertexCollection)
// Create a subgraph of g containing the matched vertices in the vertex
collection and the edges that connect them if any.
opg4j> var newGraph = g.filter(vertexFilter)
```

Java

```
// Obtain a vertex collection from an algorithm, query execution or any
other way
VertexCollection<?> vertexCollection = ...
```



```
// Define a filter from the collection
VertexFilter vertexFilter =
VertexFilter.fromCollection(vertexCollection);
// Create a subgraph of g containing the matched vertices in the vertex collection and the edges that connect them if any.
PgxGraph newGraph = g.filter(vertexFilter);
```

You can create a subgraph from edge collection filter, as shown in the following code:

- JShell
- Java

JShell

```
// Obtain an edge collection from an algorithm, query execution or any
other way
opg4j> EdgeCollection edgeCollection = ...
// Define a filter from the collection
opg4j> var edgeFilter = EdgeFilter.fromCollection(edgeCollection)
// Create a subgraph of g containing the matched edges in the
collection and their corresponding source and destination vertices.
opg4j> var newGraph = g.filter(edgeFilter)
```

Java

```
// Obtain an edge collection from an algorithm, query execution or any
other way
EdgeCollection edgeCollection = ...
// Define a filter from the collection
EdgeFilter edgeFilter = EdgeFilter.fromCollection(edgeCollection);
// Create a subgraph of g containing the matched edges in the
collection and their corresponding source and destination vertices.
PgxGraph newGraph = g.filter(edgeFilter);
```

23.10.5.6 Combining Filter Expressions

Any filter expression used for subgraph filtering, can be combined with any other filter expression to form a new filter expression.

Filters can be combined using the following operations:

- intersection
- union



The intersection of two filters will only keep a vertex or edge, if both filters would accept it.



The intersection of two filters will not behave as an AND in the filter expression.

The union of two filters will keep a vertex or edge, if one of the filters would accept it.



The union of filters will not behave as an OR in the filter expression.

In the following example an edge filter is intersected with a vertex filter. The resulting subgraph will only include vertices that have the name 'PGX' and will only include edges that have a cost greater than 5.

- JShell
- Java
- Python

JShell

```
opg4j> var edgeFilter = EdgeFilter.fromExpression("edge.cost > 5")
opg4j> var vertexFilter = VertexFilter.fromExpression("vertex.name = 'PGX'")
opg4j> var combinedFilter = edgeFilter.intersect(vertexFilter)
```

Java

```
EdgeFilter edgeFilter = EdgeFilter.fromExpression("edge.cost > 5");
VertexFilter vertexFilter = VertexFilter.fromExpression("vertex.name = 'PGX'");
GraphFilter combinedFilter = edgeFilter.intersect(vertexFilter);
```

Python

```
edge_filter = EdgeFilter("edge.cost > 5")
vertex_filter = VertexFilter("vertex.name = 'PGX'")
combined_filter = edge_filter.intersect(vertex_filter)
```

In contrast, the subgraph created by the union of those filters will include vertices that either have the name 'PGX' or that has an incoming or outgoing edge with a cost greater than 5. It

will also include edges with a cost greater than 5, as well as edges for which the source and destination vertex have the name 'PGX'.

23.10.5.7 Creating a Subgraph Using Filter Expressions with Partitioned IDs

You can create a subgraph using filter expressions with partitioned IDs.

For example, the following creates a subgraph that contains only a single vertex with ID Account(1):

- JShell
- Java
- Python

JShell

```
opg4j> PgxGraph subgraph =
g.filter(VertexFilter.fromExpression("vertex = 'Accounts(1)'"))
subgraph ==> PgxGraph[name=sub-graph 26,N=1,E=0,created=1630414040396]
```

Java

```
PgxGraph subgraph = g.filter(VertexFilter.fromExpression("vertex =
'Accounts(1)'"));
```

Python

```
subgraph = graph.filter(VertexFilter.from_expression("vertex =
'Accounts(1)'"))
```

The following example creates a subgraph that contains only a single edge with ID Transfers(1), and two accompanying vertices:

- JShell
- Java
- Python



JShell

```
opg4j> PgxGraph subgraph = g.filter(EdgeFilter.fromExpression("edge =
'Transfers(1)'"))
subgraph ==> PgxGraph[name=sub-graph_27,N=2,E=1,created=1630414144529]
```

Java

```
PgxGraph subgraph = g.filter(EdgeFilter.fromExpression("edge =
'Transfers(1)'"));
```

Python

```
subgraph = graph.filter(EdgeFilter.from_expression("edge = 'Transfers(1)'"))
```

23.11 Advanced Task Scheduling Using Execution Environments

This guide shows how you can use the advanced scheduling features of the enterprise scheduler.

The enterprise scheduler features of the graph server (PGX) are currently only available for Linux (x86_64), macOS (x86_64) and Solaris (x86_64, sparc).

The following topics provide more detailed information on enabling and scheduling tasks using the execution environment:

- Enterprise Scheduler Configuration Guide
- Enabling Enterprise Scheduler Features
- · Retrieving and Inspecting the Execution Environment
- · Modifying and Submitting Tasks Under an Updated Environment
- Using Lambda Syntax

23.11.1 Enterprise Scheduler Configuration Guide

This chapter describes the extra configuration options for the enterprise scheduler.



These configuration options are only available if the scheduler configuration variable is set to enterprise_scheduler in Configuration Parameters for the Graph Server (PGX) Engine.

The configuration is divided into the following two parts:



- enteprise_scheduler_config: for setting details about how tasks should be scheduled
- 2. enterprise_scheduler_flags: where you can configure the enterprise scheduler in more detail

Enterprise Scheduler Fields

Field	Туре	Description	Default
analysis_ta sk_config	object	Configuration for analysis tasks.	weight <no-of-cpus></no-of-cpus>
			<pre>priority medium</pre>
			<pre>max_threads <no-of-cpus></no-of-cpus></pre>
<pre>fast_analys is_task_con fig</pre>	object	Configuration for fast analysis tasks.	weight 1
			priority high
			<pre>max_threads <no-of-cpus></no-of-cpus></pre>
<pre>max_num_con current_io_ tasks</pre>	integer	Maximum number of concurrent io tasks at a time.	3
<pre>num_io_thre ads_per_tas k</pre>	integer	Number of io threads to use per task.	<no-of-cpus></no-of-cpus>

Analysis Task Config Fields

Field	Туре	Description	Default
max_threads	integer	A hard limit on the number of threads to use for a task.	required
priority	<pre>enum[high, medium, low]</pre>	The priority of the task. Threads are given to the task with the highest priority at the moment of execution. If there are more threads that have the highest priority, threads are given to the tasks according to their weight	required
weight	integer	The weight of the task. Threads are given to tasks proportionally to their weight. Tasks with higher weight will get more threads than tasks with lower weight. Tasks with the same weight will get the same amount of threads.	required



Enterprise Scheduler Flags

Field	Туре	Description	Default
show_allocat ions	boolean	If true show memory allocation information.	false
show_environ ment	boolean	If true show version numbers and main environment settings at startup.	false
show_logging	boolean	If true enable summary logging. This is available even in non-debug builds and includes information such as the machine hardware information obtained at start-up, and per-job / per-loop information about the workload.	false
show_profili	boolean	If true show profiling information.	false
show_schedul er_state	boolean	If true dump scheduler state on each update.	false
show_warning s	boolean	If true enable warnings. These are non-fatal errors. For example, if a NUMA-aware allocation cannot be placed on the intended socket.	true

Example 23-4 Custom Enterprise Scheduler Configuration

This configuration sets the number of io threads per task to 16, increases the maximum number of concurrent io tasks to 5. It also sets the configuration for fast analysis tasks to have a weight of 1, priority of "high" and sets a limit to the maximum number of threads used to 1.

```
{
  "enterprise_scheduler_config": {
    "num_io_threads_per_task": 16,
    "max_num_concurrent_io_tasks": 5,
    "fast_analysis_task_config": {
        "weight": 1,
        "priority": "high",
        "max_threads": 1
     }
}
```

Example 23-5 Using the Enterprise Scheduler Flags

This configuration enables extra logging output from the enterprise scheduler.

```
{
   "enterprise_scheduler_flags": {
      "show_logging": true
   }
}
```



23.11.2 Enabling Enterprise Scheduler Features

You can enable the enterprise scheduler features, by setting the flag allow_override_scheduling_information of the the graph server (PGX) configuration file to true:

```
{"allow override scheduling information":true}
```

See Configuration Parameters for the Graph Server (PGX) Engine for all configuration options of the graph server (PGX).

23.11.3 Retrieving and Inspecting the Execution Environment

Execution environments are bound to a session. You can retrieve the execution environment for a session by calling <code>getExecutionEnvironment()</code> on a <code>PgxSession</code>:

- JShell
- Java

JShell

```
opg4j> execEnv.getValues()
==> [analysis-pool.max_num_threads=4, analysis-pool.weight=4, analysis-
pool.priority=MEDIUM, io-pool.num_threads_per_task=4, fast-track-
analysis-pool.max_num_threads=4, fast-track-analysis-pool.weight=1,
fast-track-analysis-pool.priority=HIGH]
```

Java

```
import oracle.pgx.api.*;
import java.util.List;
import java.util.Map.Entry;

List<Entry<String, Object>> currentValues = execEnv.getValues();
for (Entry<String, Object> value : currentValues) {
   System.out.println(value.getKey() + " = " + value.getValue());
}
```

See Enterprise Scheduler Configuration Guide for the values of an unmodified execution environment.

To retrieve the sub-environments use the <code>getIoEnvironment()</code>, <code>getAnalysisEnvironment()</code> and <code>getFastAnalysisEnvironment()</code> methods. Each sub-environment has their own <code>getValues()</code> method for retrieving the configuration of the sub-environment.



- JShell
- Java

```
opg4j> var ioEnv = execEnv.getIoEnvironment()
ioEnv ==> IoEnvironment[pool=io-pool]
opg4j> ioEnv.getValues()
$5 ==> {num_threads_per_task=4}

opg4j> var analysisEnv = execEnv.getAnalysisEnvironment()
analysisEnv ==> CpuEnvironment[pool=analysis-pool]
opg4j> analysisEnv.getValues()
$7 ==> {max_num_threads=4, weight=4, priority=MEDIUM}

opg4j> var fastAnalysisEnv = execEnv.getFastAnalysisEnvironment()
fastAnalysisEnv ==> CpuEnvironment[pool=fast-track-analysis-pool]
opg4j> fastAnalysisEnv.getValues()
$9 ==> {max_num_threads=4, weight=1, priority=HIGH}

Java
```

```
import oracle.pgx.api.*;
import oracle.pgx.api.executionenvironment.*;
import java.util.Map;

IoEnvironment ioEnv = execEnv.getIoEnvironment();
CpuEnvironment analysisEnv = execEnv.getAnalysisEnvironment();
CpuEnvironment fastAnalysisEnv = execEnv.getFastAnalysisEnvironment();

for (Entry<String, Object> value : ioEnv.getValues().getEntrySet()) {
    System.out.println(value.getKey() + " = " + value.getValue());
}

for (Entry<String, Object> value : analysisEnv.getValues().getEntrySet()) {
    System.out.println(value.getKey() + " = " + value.getValue());
}

for (Entry<String, Object> value :
fastAnalysisEnv.getValues().getEntrySet()) {
    System.out.println(value.getKey() + " = " + value.getValue());
}
```



23.11.4 Modifying and Submitting Tasks Under an Updated Environment

You can modify an Input/Output (IO) environment in the number of threads by using the <code>setNumThreadsPerTask()</code> method of the <code>IoEnvironment</code>. The value is updated immediately and all tasks that are submitted after updating it are executed with the updated value.

- JShell
- Java

JShell

```
opg4j> ioEnv.setNumThreadsPerTask(8)
opg4j> var g = session.readGraphWithProperties(...)
==> PgxGraph[name=graph, N=3, E=6, created=0]
```

Java

```
import oracle.pgx.api.*;
import oracle.pgx.api.executionenvironment.*;
ioEnv.setNumThreadsPerTask(8);
PgxGraph g = session.readGraphWithProperties(...);
```

You can reset an environment to their initial values by calling the <code>ioEnv.reset()</code> method. Additionally, you can reset all environments at once by calling <code>execEnv.reset()</code> on the <code>ExecutionEnvironment</code> class.

You can modify CPU environments in their weight, priority and maximum number of threads using the <code>setWeight()</code>, <code>setPriority()</code> and <code>setMaxThreads()</code> methods:

- JShell
- Java

```
opg4j> analysisEnv.setWeight(50)
opg4j> fastAnalysisEnv.setMaxNumThreads(1)
opg4j> var rank = analyst.pagerank(g)
rank ==> VertexProperty[name=pagerank,type=double,graph=my-graph]
```



```
import oracle.pgx.api.*;
import oracle.pgx.api.executionenvironment.*;
analysisEnv.setWeight(50);
fastAnalysisEnv.setMaxThreads(1);
Analyst analyst = session.createAnalyst();
VertexProperty rank = analyst.pagerank(g);
```

23.11.5 Using Lambda Syntax

Generally you can perform the following actions in the environment:

- 1. Set up the execution environment
- 2. Execute task
- 3. Reset execution environment

All these actions can be combined and performed in a single step using the set method. For each set method there is a method using the with prefix which takes the updated value and a lambda which should be executed using the updated value.

For example, use withNumThreadsPerTask() instead of setNumThreadsPerTask() as shown:

- JShell
- Java

JShell

```
opg4j> var g = ioEnv.withNumThreadsPerTask(8, () ->
session.readGraphWithProperties(...))
==> PgxGraph[name=graph, N=3, E=6, created=0]
```

Java

```
import oracle.pgx.api.*;
import oracle.pgx.api.executionenvironment.*;

PgxGraph g = ioEnv.withNumThreadsPerTask(8, () ->
session.readGraphWithProperties(...));
```



The preceding code execution is equivalent to the following sequence of actions:

```
var oldValue = ioEnv.getNumThreadsPerTask()
ioEnv.setNumThreadsPerTask(currentValue)
var g = session.readGraphWithProperties(...)
ioEnv.setNumThreadsPerTask(oldValue)
```

23.12 Admin API

This guide shows how to use the graph server (PGX) Admin API to inspect the server state including sessions, graphs, tasks, memory and thread pools.

- Get a Server Instance
- Get Inspection Data
- · Get Active Sessions
- Get Cached Graphs
- · Get Published Graphs
- Get Currently Loading Graphs
- Get Tasks
- · Get Available Memories

23.12.1 Get a Server Instance

You can get a PGX Instance as shown in the following code:

- Java
- Python

Java

```
import oracle.pgx.api.*;
ServerInstance instance = Pgx.getInstance(Pgx.EMBEDDED URL);
```

Python

```
instance = pypgx.get session(base url = "url")
```

23.12.2 Get Inspection Data

Inspection data is information about the server state.



You can get the inspection data using the following code. Note that you must the PGX SERVER GET INFO permission to access the server state data.

- JShell
- Java
- Python

JShell

```
var serverState = instance.getServerState()
```

Java

```
JsonNode serverState = instance.getServerState();
```

Python

```
server state = instance.get server state()
```

This returns a JsonNode which contains all the administration information, such as number of graphs loaded, number of sessions, memory usage for graphs, properties, and so on.

```
{
    "cached_graphs": [],
    "published_graphs": [],
    "graphs_currently_loading": [],
    "sessions": [],
    "tasks": [],
    "pools": [],
    "memory": {}
}
```

Note that the sessions parameter lists all the sessions and the memory used by the sessions along with the user information for each session.



23.12.3 Get Active Sessions

serverState.get("sessions") returns an array of current active sessions. Each entry contains information about a session.

```
"session id": "530b5f9a-75c4-4838-9cc3-44df44b035c5",
"source": "testServerState",
"user": "user1",
"task timeout ms":0,
"idle timeout ms":0,
"alive ms":237,
"total_analysis_time_ms":115,
"state": "RELEASED",
"private graphs":[
      "name": "anonymous graph 1",
      "creation timestamp":1589317879755,
      "is transient":true,
      "memory":{
         "topology bytes":46,
         "key mapping_bytes":30,
         "persistent property mem bytes":0,
         "transient_property_mem_bytes":0
      "vertices num":1,
      "edges num":0,
      "persistent vertex properties":[
      "persistent edge properties":[
      "transient vertex properties":[
      "transient edge properties":[
      1
],
"published graphs":[
      "name": "multigraph",
      "creation timestamp":1589317879593,
      "is transient":false,
      "memory":{
         "topology bytes":110,
         "key mapping bytes":56,
         "persistent property mem bytes":64,
         "transient_property_mem_bytes":0
      },
      "vertices num":2,
```



```
"edges num":6,
         "persistent vertex properties":[
               "loaded":true,
               "mem_size_bytes":16,
               "name":"tProp",
               "type":"string"
            }
         ],
         "persistent_edge_properties":[
               "loaded":true,
               "mem size bytes":48,
               "name":"cost",
               "type": "double"
            }
         ],
         "transient vertex properties":[
         ],
         "transient_edge_properties":[
  ]
}
```

The following table explains session information fields:

Table 23-6 Session Information Options

Field	Description
sessionID	Session ID generated by the graph server (PGX)
source	Descriptive string identifying the client session
user	Session owner
task_timeout_ms	Timeout to interrupt long-running tasks submitted by sessions (algorithms, I/O tasks) in milliseconds. Set to zero for infinity/no timeout.
idle_timeout_ms	Timeout of idling sessions in milliseconds. Set to zero for infinity/no timeout.
alive_ms	Session's age in milliseconds
total_analysis_time_ms	Total session's executing time in milliseconds
state	Current session of the session can be Idle, Submitted, Released or Terminating
private_graphs	Session bounded graphs
published_graphs	Published graphs pointed to from the session



Note:

The is_transient field indicates if the graph is transient. A graph is transient if it is not loaded from an external source.

23.12.4 Get Cached Graphs

The server state contains also cached graph information serverState.get("cached_graphs") which returns a collection of graphs cached in memory. Each entry contains information about a graph as shown:

```
"name": "sf-1589317879394",
   "creation timestamp":1589317879394,
   "vertex properties":[
         "loaded":true,
         "mem size bytes":478504,
         "name": "prop1",
         "type": "double"
   ],
   "edge_properties":[
      {
         "loaded":true,
         "mem size bytes":1197720,
         "name": "cost",
         "type": "double"
      },
         "loaded":true,
         "mem size bytes":598860,
         "name":"0",
         "type": "integer"
      }
   ],
   "memory":{
      "topology bytes":3921814,
      "key mapping bytes":1407466,
      "property mem bytes":2275084
   "vertices num":59813,
   "edges num":149715
}
```

The following table explains graph information fields:

Table 23-7 Graph Information

Field	Description
name	Name of the graph.



Table 23-7 (Cont.) Graph Information

Field	Description
creation_timestamp	Creation timestamp of the graph.
vertex_properties	List of vertex properties, each entry contains the name, type, memory size used by the property, and a boolean flag to indicate if the property is loaded into memory.
edge_properties	List of edges properties, similar to vertex properties.
memory	Memory size used by the whole graph (topology, key mappings and properties).
vertices_num	Number of vertices.
edges_num	Number of edges.

23.12.5 Get Published Graphs

serverState.get("published_graphs") returns a list of published graphs.

Each graph entry contains information about the published graph, similar to cached_graphs.

23.12.6 Get Currently Loading Graphs

serverState.get("graphs_currently_loading") returns progress information about graphs which are currently loading.

Each entry, corresponding to one graph, is shown as follows:

```
"name": "anonymous graph 1",
    "session id": "530b5f9a-75c4-4838-9cc3-44df44b035c5",
    "start loading timestamp": 1605468453030,
    "elapsed loading time ms": 281742,
    "num vertices read": 10000000,
    "num edges read": 196500000,
    "num_edge_providers loaded": 1,
    "num edge providers remaining": 9,
    "num vertex providers loaded": 1,
    "num vertex providers remaining": 0,
    "loading phase": "reading edges",
    "loading phase start timestamp": 1605468453085,
    "loading phase elapsed time ms": 281687,
    "loading_phase_state": "current vertex provider index: 1, number of
vertices read for prorvider: 0, current edge provider index: 1, number of
edges read for prorvider: 76,500,000"
```

The name field contains a temporary name of the graph. It may not be equal to the name that is assigned to graph after loading.

Fields indicating the number of read vertices and edges are updated in regular intervals of 10,000 entities.

The field <code>loading_phase</code> indicates the current phase during graph loading. Valid values are "reading edges" or "building graph indices". For some loading phases, the field <code>loading_phase_state</code> contains a string with additional information on the phase. However, not all loading phases provide this additional information.

Note:

graphs_currently_loading is supported for data formats CSV, ADJ_LIST, EDGE_LIST, TWO_TABLES and PG (FLAT_FILE) for homogeneous graphs and for formats CSV and RDBMS for partitioned graphs.

23.12.7 Get Tasks

serverState.get("tasks") returns the last 100 queued tasks.

Each task has a type, the pool to be executed on (the task might be already executed) and other status fields ({Queued|Started|Done} time), and a sessionid if the task belongs to a session.

23.12.8 Get Available Memories

This section contains a map of available memories, the key is the hostname and the value is a list of current available memories (managed and unmanaged). Each entry contains how much memory is free, used and the maximum available memory.

23.13 PgxFrames Tabular Data-Structure

PgxFrame is a data-structure to load, store and manipulate tabular data. It contains rows and columns. A PgxFrame can contain multiple columns where each column consist of elements of the same data type, and has a name. The list of the columns with their names and data types defines the schema of the frame. (The number of rows in the PgxFrame is not part of the schema of the frame.)

PgxFrame provides some operations that also output PgxFrames (described later in the tutorial). Those operations can be performed in-place (meaning that the frame is mutated during the operation) in order to save memory. In place operations should be used whenever possible. However, we provide out-place variants, i.e., a new frame is created during the operation.

The following table lists all the in-place operations along with the respective out-place operations:

Table 23-8 Mapping between In-Place and Out-Place Operations

In-place operations	Out-place operations	
headInPlace	head	
tailInPlace	tail	
flattenAllInPlace	flattenAll	
renameColumnInPlace	renameColumn	
renameColumnsInPlace	renameColumns	



Table 23-8 (Cont.) Mapping between In-Place and Out-Place Operations

In-place operations	Out-place operations
selectInPlace	select

- Converting PgqlResultSet to a PgxFrame
- Storing a PgxFrame to a Database
- Storing a PgxFrame to a CSV File
- Union of PGX Frames
- Joining PGX Frames
- Printing the Content of a PgxFrame
- Destroying a PgxFrame
- Loading and Storing Vector Properties
- Flattening Vector Properties
- PgxFrame Helpers
- Converting a PgxFrame to PgqlResultSet
- PgxFrame to Pandas DataFrame Conversions
- Loading a PgxFrame from a Database
- Loading a PgxFrame from a CSV File
- Loading a PgxFrame from Client-Side Data
- Creating a Graph from Multiple PgxFrame Objects

23.13.1 Converting PgqlResultSet to a PgxFrame

The following example describes how to save the PgqlResultSet to a PgxFrame.

- JShell
- Java
- Python

JShell



Java

```
import oracle.pgx.api.frames.*;

PgxGraph pg =
session.readGraphByName("BANK_GRAPH_NEW", GraphSource.PG_VIEW);
PgqlResultSet rs = pg.queryPgql("SELECT e.* FROM MATCH (v1:Accounts)-
[e:Transfers]->(v2:Accounts) LIMIT 5");
PgxFrame rsFrame = rs.toFrame();
rsFrame.print();
```

Python

```
>>> pg = session.read graph by name('BANK GRAPH NEW', 'pg view')
>>> rs = pg.query pgql("SELECT e.* FROM MATCH (v1:Accounts)-
[e:Transfers] -> (v2:Accounts) LIMIT 5")
>>> rs frame = rs.to frame()
>>> rs frame.print()
| FROM ACCT ID | TO ACCT ID | AMOUNT | DESCRIPTION |
+----+
           | 418 | 1000.0 | transfer
| 1
           | 584
                      | 1000.0 | transfer
| 1
                      | 1000.0 | transfer
           | 644
| 1
                      | 1000.0 | transfer
1 1
           | 672
           | 259
                      | 1000.0 | transfer
```

Converting PgqlResultSet to pandas DataFrame

You can also save the PgqlResultSet to pandas DataFrame as shown in the following example:



23.13.2 Storing a PgxFrame to a Database

When storing a PgxFrame to a database, the frame is stored as a table, where the columns correspond to the columns of the PgxFrame and the rows correspond to the rows of the PgxFrame. Note that the column order preservation may or may not happen when storing a PgxFrame in the database.

The following example shows how to store the PgxFrame in the database. The example assumes that you are storing the PgxFrame in the current logged in schema.

- JShell
- Java
- Python

JShell

```
opg4j> rsFrame.write().
                                     // select the "format" to be relational
           db().
db
           name("F1").
                                     // name of the frame
                                     // name of the table in which the data
           tablename("T1").
must be stored
                                     // indicates that if there is a table
          overwrite(true).
with the same name, it will be overwritten (truncated)
           connections (16).
                                    // indicates that 16 connections can be
used to store in parallel
           store()
```

Java

```
rsFrame.write()
    .db()
    .name("F1")
    .tablename("T1")
    .overwrite(true)
    .connections(16)
    .store();
```

```
>>> rs_frame.write().db().\
... table_name('T1').\
... overwrite(True).\
... store()
```



Alternatively, you can also store the PgxFrame in a different schema as shown in the following example. Ensure that you have CREATE TABLE privilege when writing to a different schema:

- JShell
- Java
- Python

JShell

```
// store as table in the database using jdbc + username + password
opg4j> rsFrame.write().
           db().
                                     // select the "format" to be
relational db
                                     // name of the frame
           name("framename").
           tablename("tablename"). // name of the table in which the
data must be stored
           overwrite(true).
                                     // indicates that if there is a
table with the same name, it will be overwritten (truncated)
                                     // indicates that 16 connections
           connections (16).
can be used to store in parallel
           jdbcUrl("<jdbcUrl>").
           username("<db username>").
           password("<password>").
           store()
```

Java

```
rsFrame.write()
                              // select the "format" to be relational
    .db()
db
                              // name of the frame
    .name("framename")
    .tablename("tablename")
                              // name of the table in which the data
must be stored
                              // indicates that if there is a table
    .overwrite(true)
with the same name, it will be overwritten (truncated)
                              // indicates that 16 connections can be
    .connections (16)
used to store in parallel
    .jdbcUrl("<jdbcUrl>")
    .username("<db username>")
    .password("<password>")
    .store();
```

```
>>> rs_frame.write().db().\
... table_name('T1').\
... overwrite(True).\
... jdbc_url("<jdbcUrl>").\
```



```
... username("<db_username>").\
... password("<password>").\
... store()
```

23.13.3 Storing a PgxFrame to a CSV File

In order to write a PgxFrame to a CSV file, you first need to explicitly authorize access to the corresponding directories by defining a directory object pointing to the directory (on the graph server) where the file needs to be written.

```
CREATE OR REPLACE DIRECTORY graph_files AS '/tmp';
GRANT READ, WRITE ON DIRECTORY graph files TO GRAPH DEVELOPER;
```

Also, note the following:

- The directory in the CREATE DIRECTORY statement must exist on the graph server (PGX).
- The directory must be writable at the OS level by the graph server (PGX).

The preceding code grants the privileges on the directory to the <code>GRAPH_DEVELOPER</code> role. However, you can also grant permissions to an individual user:

```
GRANT WRITE ON DIRECTORY graph files TO <graph user>;
```

You can then save a PgxFrame to a CSV file as shown in the following example:

- JShell
- Java
- Python

JShell

```
opg4j> rsFrame.write().overwrite(true).csv("/tmp/Transfers.csv")
```

Java

```
rsFrame.write().overwrite(true).csv("/tmp/Transfers.csv");
```

```
>>> rs_frame.store("/tmp/Transfers.csv")
```



23.13.4 Union of PGX Frames

You can join two PgxFrames that have compatible columns (that is, same type and order).

- JShell
- Java
- Python

JShell

```
opg4j> <first-frame>.union(<secondframe>).print()
```

Java

```
<first-frame>.union(<first-frame>).print();
```

Python

```
<first-frame>.union(<first-frame>).print()
```

The rows of the resulting PgxFrame are the union of the rows from the two original frames.

Note that the union operation does not remove duplicate rows that resulted by joining the two frames.

23.13.5 Joining PGX Frames

You can join two frames whose rows are correlated through one of the columns using the join functionality. This allows us to combine frames by checking for equality between rows for a specific column.

The following example shows joining two PgxFrames exampleFrame and moreInfoFrame on the name column by calling the join method.

- JShell
- Java
- Java



```
opg4j> exampleFrame.join(moreInfoFrame, "name", "leftFrame",
    "rightFrame").print()
```

Java

```
exampleFrame.join(moreInfoFrame, "name", "leftFrame", "rightFrame").print();
```

Java

```
example frame.join(moreInfoFrame, "name", "leftFrame", "rightFrame").print()
```

The result may appear as shown:

```
-----+
| leftFrame name | leftFrame age | leftFrame salary | leftFrame married |
leftFrame tax rate | leftFrame random | leftFrame date of birth |
rightFrame name | rightFrame title | rightFrame reports |
              | 27 | 4133300.0 | true
              | 123456782 | 1985-10-18
| Software Engineering Manager | 5
11.0
John | Software Engineering Manager | 5 | Albert | 23 | 5813000.5 | false
          | 124343142 | 2000-01-14 | Sales Manager | 10
12.0
             | Sales Manager | 10
| 24 | 9380080.5 | false
| 128973221 | 1910-07-30
Albert
| Emily
13.0
           | Operations Manager | 20
Emily
```

The joined frame contains the columns of the two frames involved in the operation for the rows with the same name.



The column prefixes specified in the join() call, leftFrame and rightFrame.

23.13.6 Printing the Content of a PgxFrame

You can observe the contents of a frame using the print functionality as shown:



- JShell
- Java
- Python

```
opg4j> exampleFrame.print()
```

Java

```
exampleFrame.print();
```

Python

```
example_frame.print()
```

The output appears as follows:

+.		 	 	 	+
	FROM_ACCT_ID	TO_ACCT_ID	AMOUNT	DESCRIPTION	
+-		 	 	 	+
	2	546	1000.0	transfer	
	2	840	1000.0	transfer	
	2	493	1000.0	transfer	
	2	693	1000.0	transfer	
	2	833	1000.0	transfer	
т.		 	 	 	

23.13.7 Destroying a PgxFrame

PgxFrames consumes a lot of memory on the graph server (PGX) if they have a lot of rows or columns. Hence, it is necessary to close them with the close() operation. After this operation, the content of the PgxFrame is not available anymore.

You can close a frame as shown:

- JShell
- Java
- Python



```
opg4j> exampleFrame.close()

Java
exampleFrame.close();

Python
example frame.close()
```

23.13.8 Loading and Storing Vector Properties

You can load or store vector properties which are fundamental for PgxML functionality in the graph server (PGX) using PgxFrames.

In order to load a PgxFrame with vector properties, follow the steps as shown:

- 1. Create the PgxFrame schema, defining the columns as shown:
 - JShell
 - Java

JShell

```
opg4j> var vecFrameSchema = List.of(
  columnDescriptor("intProp", DataTypes.INTEGER_TYPE),
  columnDescriptor("intProp2", DataTypes.INTEGER_TYPE),
  columnDescriptor("vectProp", DataTypes.vector(DataTypes.FLOAT_TYPE, 3)),
  columnDescriptor("stringProp", DataTypes.STRING_TYPE),
  columnDescriptor("vectProp2", DataTypes.vector(DataTypes.FLOAT_TYPE, 2))
).toArray(new ColumnDescriptor[0])
```

Java

```
ColumnDescriptor[] vecFrameSchema = {
    columnDescriptor("intProp", DataTypes.INTEGER_TYPE),
    columnDescriptor("intProp2", DataTypes.INTEGER_TYPE),
    columnDescriptor("vectProp", DataTypes.vector(DataTypes.FLOAT_TYPE,
3)),
    columnDescriptor("stringProp", DataTypes.STRING_TYPE),
    columnDescriptor("vectProp2", DataTypes.vector(DataTypes.FLOAT_TYPE,
2))
};
```



- 2. Load the PgxFrame with the given schema from the specified path:
 - JShell
 - Java

```
opg4j> var vecFrame = session.readFrame().
    db().
   name("vector PgxFrame").
    tablename("tablename").
                                 // name of the table from where
the data must be loaded
    jdbcUrl("jdbcUrl").
   username("user").
   owner("owner").
                                 // necessary if the table is owned
by another user
                                 // indicates that 16 connections
   connections (16).
can be used to load in parallel
                                 // columns to load
   columns (vecFrameSchema).
   load()
```

Java

```
PgxFrame vecFrame = session.readFrame()
    .db()
    .name("vector PgxFrame")
    .tablename("tablename")
                                 // name of the table from where
the data must be loaded
    .jdbcUrl("jdbcUrl")
    .username("user")
    .owner("owner")
                                 // necessary if the table is owned
by another user
    .connections(16)
                                 // indicates that 16 connections
can be used to load in parallel
    .columns(vecFrameSchema)
                                 // columns to load
    .load();
```

The final result in the PgxFrame may appear as follows:

+	intProp	 intProp2	 vectProp	-	stringProp		vectProp2	·+
+		 	 					+
	0	2	0.1;0.2;0.3		testProp0		0.1;0.2	
	1	1	0.1;0.2;0.3		testProp10		0.1;0.2	
	1	2	0.1;0.2;0.3		testProp20		0.1;0.2	
	2	3	0.1;0.2;0.3		testProp30		0.1;0.2	



23.13.9 Flattening Vector Properties

You can split the vector properties into multiple columns using the ${\tt flattenAll}()$ operation.

For example, you can flatten the vector properties for the example explained in Loading and Storing Vector Properties as shown:

- JShell
- Java

JShell

```
opg4j> vecFrame.flattenAll()
```

Java

vecFrame.flattenAll();

The resulting flattened PgxFrame may appear as shown:

```
| intProp | intProp2 | vectProp 0 | vectProp 1 | vectProp 2 | stringProp |
vectProp2 0 | vectProp2 1 |
+-----
| 1
| 0.1 | 2
| 0.2
| 2
   | 3 | 0.1
            | 0.2
0.1
   | 1
       | 0.1
            1 3
    | 0.2
0.1
----+
```

23.13.10 PgxFrame Helpers

PgxFrame supports the following operations:



- head
- tail
- select
- renameColumns

Head Operation

The head operation can be used to only keep the first rows of a PgxFrame. (The result is deterministic only for ordered PgxFrame.)

- JShell
- Java

JShell

```
opg4j> vecFrame.head(2).print()
```

Java

```
vecFrame.head(2).print();
```

The output appears as follows:

Tail Operation

The tail operation can be used to only keep the last rows of a PgxFrame. (The result is deterministic only for ordered PgxFrame).

- JShell
- Java

JShell

```
opg4j> vecFrame.tail(2).print()
```



Java

```
vecFrame.tail(2).print();
```

The output appears as follows:

Select Operation

The select operation can be used to keep only a specified list of columns of an input PqxFrame.

- JShell
- Java
- Python

JShell

```
opg4j> var vecFrameSelected = vecFrame.select("vectProp2", "vectProp",
"stringProp")
```

Java

```
PgxFrame vecFrameSelected =
vecFrame.select("vectProp2","vectProp","stringProp");
```

Python

```
vec_frame_selected = vec_frame.select("vectProp2","vectProp","stringProp")
```

The result may appear as follows:

```
+-----+
| vectProp2 | vectProp | stringProp |
+------+
| 0.1;0.2 | 0.1;0.2;0.3 | testProp0 |
```



Rename PgxFrame Columns

You can rename the columns in a PgxFrame to customized names as follows:

- JShell
- Java

JShell

```
opg4j> var vecFrameRenamed = vecFrame.renameColumns(
  renaming("vectProp2", "vectProp2_renamed"),
  renaming("vectProp", "vectProp_renamed"),
  renaming("stringProp", "stringProp_renamed"))
```

Java

The renamed PgxFrame appears as follows:

```
| intProp | intProp2 | vectProp renamed | stringProp renamed |
vectProp2 renamed |
<u>+-----</u>
----+
0.1;0.2
       | 0.1;0.2;0.3 | testProp10
| 0.1;0.2;0.3 | testProp20
       | 0.1;0.2;0.3 | testProp30
| 2 | 3
0.1;0.2
```



```
0.1;0.2 |
+-----+
```

23.13.11 Converting a PgxFrame to PgqlResultSet

You can convert a PgxFrame to PgqlResultSet as follows:

- JShell
- Java
- Python

JShell

```
opg4j> var resultSet = exampleFrame.toPgqlResultSet()
```

Java

PgqlResultSet resultSet = exampleFrame.toPgqlResultSet();

Python

```
result_set = example_frame.to_pgql_result_set()
```

You can view the content of the result set through the usual PgqlResultSet APIs. The output appears as follows:

+	 	to_acct_id	 	amount	 	description	+
1 1 1 1 1	 	418 584 644 672 259		1000.0 1000.0 1000.0		transfer transfer transfer transfer transfer	

23.13.12 PgxFrame to Pandas DataFrame Conversions

You can save a PgxFrame to a pandas DataFrame as shown in the following example:

```
>>> pandas data frame = example frame.to pandas()
```



Similarly, you can load a PgxFrame from a pandas DataFrame as shown in the following example:

```
>>> example_frame = session.pandas_to_pgx_frame(pandas_data_frame,
"example frame")
```

23.13.13 Loading a PgxFrame from a Database

You can load a PgxFrame from relational tables in an Oracle database. Each column of the relational table will correspond to a column in the loaded frame. When loading a PgxFrame from the database, the default behavior is to detect the table columns and load them all. If not specified explicitly, the connection details of the current user and session are used and the columns are detected automatically.

The following describes the steps to load a PgxFrame from a database table:

- 1. Create a Session and an Analyst:
 - JShell
 - Java
 - Python

JShell

```
cd /opt/oracle/graph/
./bin/opg4j

// starting the shell will create an implicit session and analyst opg4j> import static oracle.pgx.api.frames.functions.ColumnRenaming.renaming opg4j> import static oracle.pgx.api.frames.schema.ColumnDescriptor.columnDescriptor opg4j> import oracle.pgx.api.frames.schema.*
opg4j> import oracle.pgx.api.frames.schema.datatypes.*
```

Java

```
import oracle.pgx.api.*;
import oracle.pgx.api.frames.*;
import oracle.pgx.api.frames.functions.*;
import oracle.pgx.api.frames.schema.*;
import oracle.pgx.api.frames.schema.datatypes.*;
import static
oracle.pgx.api.frames.functions.ColumnRenaming.renaming;
import static
oracle.pgx.api.frames.schema.ColumnDescriptor.columnDescriptor;

PgxSession session = Pgx.createSession("my-session");
Analyst analyst = session.createAnalyst();
```



Python

```
session = pypgx.get_session(session_name="my-session")
analyst = session.create_analyst()
```

- 2. Load a PgxFrame. The example assumes that you are loading the PgxFrame from the current logged in schema.
 - JShell
 - Java
 - Python

JShell

Java

```
PgxFrame exampleFrame = session.readFrame()
    .db()
    .name("Transfers")
    .tablename("T1")
    .connections(16)
    .load();
```

Python

3. If only a subset of the columns must be loaded, then you can specify the columns as shown in the following example. Note that the following example loads the PgxFrame from a different schema.

- JShell
- Java
- Python

```
opg4j> session.registerKeystore(<pathToKeystore>,
<keystorePassword>)
opg4j> var exampleFrame = session.readFrame().
...>
          db().
          name("Transfers").
...>
                                       // name of the table from
...>
         tablename("T1").
where the data must be loaded
...> jdbcUrl("<jdbcUrl>").
...>
          username("<username>").
          keystoreAlias("<keystore alias>").
...>
...>
          connections (16).
                                       // indicates that 16
connections can be used to load in parallel
...>
          columns(
          columnDescriptor("FROM ACCT ID", DataTypes.INTEGER TYPE),
...>
...>
          columnDescriptor("TO ACCT ID", DataTypes.INTEGER TYPE)
                                     // columns to load
...>
          ) .
...>
          load()
```

Java

```
session.registerKeystore(<pathToKeystore>, <keystorePassword>)
PgxFrame exampleFrame = session.readFrame()
    .db()
    .name("Transfers")
    .tablename("T1")
                                 // name of the table from where
the data must be loaded
    .jdbcUrl("<jdbcUrl>")
    .username("<username>")
    .keystoreAlias("<keystore alias>")
    .connections(16)
                                 // indicates that 16 connections
can be used to load in parallel
    .columns(
              columnDescriptor("FROM ACCT ID",
DataTypes.INTEGER TYPE),
              columnDescriptor("TO ACCT ID", DataTypes.INTEGER TYPE)
                                 // columns to load
    .load();
```



```
... .table_name('T1') \
... .jdbc_url('jdbc:oracle:thin:@localhost:1521/orclpdb') \
... .username('graphuser') \
... .keystore_alias('database3') \
... .columns(
... .[
... ('FROM_ACCT_ID', 'INTEGER_TYPE'),
... .('TO_ACCT_ID', 'INTEGER_TYPE')
... .]
... .]
... .load()
```

You can also create a graph from the PgxFrame(s). See Creating a Graph from Multiple PgxFrame Objects for more information.

23.13.14 Loading a PgxFrame from a CSV File

In order to load a PgxFrame from a CSV file, you first need to explicitly authorize access to the corresponding directories by defining a directory object pointing to the directory (on the graph server) where the file needs to be written.

```
CREATE OR REPLACE DIRECTORY graph_files AS '/tmp';
GRANT READ, WRITE ON DIRECTORY graph files TO GRAPH DEVELOPER;
```

Also, note the following:

- The directory in the CREATE DIRECTORY statement must exist on the graph server (PGX).
- The directory must be readable at the OS level by the graph server (PGX).

The preceding code grants the privileges on the directory to the <code>GRAPH_DEVELOPER</code> role. However, you can also grant permissions to an individual user:

```
GRANT READ ON DIRECTORY graph files TO <graph user>;
```

You can then load a PgxFrame from a CSV file as shown in the following example:

- JShell
- Java
- Python

JShell

```
opg4j> import oracle.pgx.api.frames.schema.datatypes.*
opg4j> import static
oracle.pgx.api.frames.schema.ColumnDescriptor.columnDescriptor
```



```
opg4j> var exampleFrame = session.readFrame().csv().
...> name("transfersFrame").
...> columns(
...> columnDescriptor("from_acct_id", DataTypes.INTEGER_TYPE),
...> columnDescriptor("to_acct_id", DataTypes.INTEGER_TYPE),
...> columnDescriptor("amount", DataTypes.FLOAT_TYPE),
...> columnDescriptor("description", DataTypes.STRING_TYPE)
...> ).
...> load("/tmp/Transfers.csv")
```

Java

```
import oracle.pgx.api.frames.schema.datatypes.*;
import static
oracle.pgx.api.frames.schema.ColumnDescriptor.columnDescriptor;

PgxFrame exampleFrame = session.readFrame().csv().
    name("transfersFrame").
    columns(
        columnDescriptor("from_acct_id", DataTypes.INTEGER_TYPE),
        columnDescriptor("to_acct_id", DataTypes.INTEGER_TYPE),
        columnDescriptor("amount", DataTypes.FLOAT_TYPE),
        columnDescriptor("description", DataTypes.STRING_TYPE)
    ).
    load("/tmp/Transfers.csv");
```

Python

```
>>> example_frame = session.read_frame(). \
... csv(). \
... name('transfers_frame'). \
... columns([('from_acct_id', 'INTEGER_TYPE'),
... ('to_acct_id', 'INTEGER_TYPE'),
... ('amount', 'FLOAT_TYPE'),
... ('description', 'STRING_TYPE')]). \
... load('/tmp/Transfers.csv')
```

23.13.15 Loading a PgxFrame from Client-Side Data

You can also load PgxFrame(s) directly from client-side data. The following describes the steps to load a PgxFrame from client-side data:

- 1. Create a Session and an Analyst:
 - See step-1 in Loading a PgxFrame from a Database for the code examples.
- 2. Define a frame schema to load a PgxFrame from client side data. For example, the following shows a frame schema defined with various data types:



- JShell
- Java
- Python

```
opg4j> var exampleFrameSchema = List.of(
    columnDescriptor("name", DataTypes.STRING_TYPE),
    columnDescriptor("age", DataTypes.INTEGER_TYPE),
    columnDescriptor("salary", DataTypes.DOUBLE_TYPE),
    columnDescriptor("married", DataTypes.BOOLEAN_TYPE),
    columnDescriptor("tax_rate", DataTypes.FLOAT_TYPE),
    columnDescriptor("random", DataTypes.LONG_TYPE),
    columnDescriptor("date_of_birth", DataTypes.LOCAL_DATE_TYPE)
)
```

Java

```
List<ColumnDescriptor> exampleFrameSchema = Arrays.asList(
    columnDescriptor("name", DataTypes.STRING_TYPE),
    columnDescriptor("age", DataTypes.INTEGER_TYPE),
    columnDescriptor("salary", DataTypes.DOUBLE_TYPE),
    columnDescriptor("married", DataTypes.BOOLEAN_TYPE),
    columnDescriptor("tax_rate", DataTypes.FLOAT_TYPE),
    columnDescriptor("random", DataTypes.LONG_TYPE),
    columnDescriptor("date_of_birth", DataTypes.LOCAL_DATE_TYPE));
```

```
example_frame_schema = [
    ("name", "STRING_TYPE"),
    ("age", "INTEGER_TYPE"),
    ("salary", "DOUBLE_TYPE"),
    ("married", "BOOLEAN_TYPE"),
    ("tax_rate", "FLOAT_TYPE"),
    ("random", "LONG_TYPE"),
    ("date_of_birth", "LOCAL_DATE_TYPE")]
```

- 3. Define data as per the schema.
 - JShell
 - Java
 - Python



Java



- 4. Load the frame as shown:
 - JShell
 - Java
 - Python

```
opg4j> var exampleFrame = session.createFrame(exampleFrameSchema,
exampleFrameData, "example frame")
```

Java

```
PgxFrame exampleFrame = session.createFrame(exampleFrameSchema,
exampleFrameData, "example frame");
```

Python

```
example_frame=session.create_frame(example_frame_schema,example_frame_data
,'example frame')
```

- 5. You can also load the frame incrementally as you receive more data:
 - JShell
 - Java
 - Python

JShell

```
opg4j> var exampleFrameBuilder =
session.createFrameBuilder(exampleFrameSchema);
opg4j> exampleFrameBuilder.addRows(exampleFrameData)
opg4j> Map<String, Iterable<?>> exampleFrameDataPart2 = Map.of(
    "name", Arrays.asList("Dave"),
    "age", Arrays.asList(26),
    "salary", Arrays.asList(18000.0),
    "married", Arrays.asList(true),
    "tax_rate", Arrays.asList(0.30),
    "random", Arrays.asList(456783423423L),
    "date_of_birth", Arrays.asList(LocalDate.of(1989, 9, 15))
)
```



```
opg4j> exampleFrameBuilder.addRows(exampleFrameDataPart2)
opg4j> var exampleFrame = exampleFrameBuilder.build("example frame")
```

Java

Python

```
example_frame_builder =
session.create_frame_builder(example_frame_schema)
example_frame_builder.add_rows(example_frame_data)
example_frame_data_part_2 = {
    "name": ["Dave"],
    "age": [26],
    "salary": [18000.0],
    "married": [True],
    "tax_rate": [0.30],
    "random": [456783423423],
    "date_of_birth": [date(1989, 9, 15)]
}
example_frame_builder.add_rows(example_frame_data_part_2)
example_frame = example_frame_builder.build("example_frame")
```

6. Finally, you can also load a frame from a Pandas dataframe in Python as shown:

```
import pandas as pd
example_pandas_dataframe = pd.DataFrame(data=example_frame_data)
example_frame =
session.pandas_to_pgx_frame(example_pandas_dataframe, "example
frame")
```

You can also create a graph from the PgxFrame(s) . See Creating a Graph from Multiple PgxFrame Objects for more information.

23.13.16 Creating a Graph from Multiple PgxFrame Objects

You can create a PgxGraph with vertex PgxFrame(s) and edge PgxFrame(s).

Consider the following PgxFrame objects:

people						
+-				+		
	id		name			
+-				+		
	1		Alice			
	2		Bob			
	3		Charlie			
т.				_		

houses

+-	identification		location	+-
İ	1 2 3		Road 1 Street 5 Avenue 4	İ

knows

+-		 	+
	src	dst	
+-		 	+
	1	1	
	2	3	
	3	2	
+-		 	+

lives

+-				-+
İ	source		destination	İ
+-				+
	1		2	
	2		1	
	3		3	
+-				-+

You can now create a PgxGraph as shown in the following examples:

- JShell
- Java
- Python



```
opg4j> var graphFromFramesCreator =
session.createGraphFromFrames("example graph")
opg4j> graphFromFramesCreator.vertexProvider("people", people)
opg4j> graphFromFramesCreator.vertexProvider("houses",
houses).vertexKeyColumn("identification")
opg4j> graphFromFramesCreator.edgeProvider("knows", "people",
    "people", knows)
opg4j> var edge_provider =
graphFromFramesCreator.edgeProvider("lives", "people", "houses", lives)
opg4j> edge_provider.sourceVertexKeyColumn("source")
opg4j> edge_provider.destinationVertexKeyColumn("destination")
opg4j> graphFromFramesCreator.partitioned(true)
opg4j> var graph = graphFromFramesCreator.create()
```

Java

```
PgxGraphFromFramesCreator graphFromFramesCreator =
session.createGraphFromFrames("example graph");
graphFromFramesCreator.vertexProvider("people", people);
graphFromFramesCreator.vertexProvider("houses",
houses).vertexKeyColumn("identification");
graphFromFramesCreator.edgeProvider("knows", "people", "people",
knows);
PgxEdgeProviderFromFramesCreator edgeProvider =
graphFromFramesCreator.edgeProvider("lives", "people", "houses",
lives);
edgeProvider.sourceVertexKeyColumn("source");
edgeProvider.destinationVertexKeyColumn("destination");
graphFromFramesCreator.partitioned(true);
PgxGraph graph = graphFromFramesCreator.create();
```

```
vertex providers from frames = [
    session.vertex provider from frame ("person",
                                        people),
    session.vertex provider from frame ("house",
                                        frame = houses,
                                        vertex key column =
"identification")
edge providers from frames = [
    session.edge provider from frame ("person knows person",
                                      source provider = "person",
                                      destination provider = "person",
                                      frame = knows),
    session.edge provider from frame("person lives at house",
                                      source provider = "person",
                                      destination provider = "house",
                                      frame = lives,
                                      source vertex column="source",
```



```
destination_vertex_column="destination")
]
graph = session.graph_from_frames("example graph",
vertex_providers_from_frames, edge_providers_from_frames, partitioned=True)
```



Working with Files Using the Graph Server (PGX)

This chapter describes in detail about working with different file formats to perform various actions like loading, storing, or exporting a graph using the Graph Server (PGX).

In order to read or write files, you need to explicitly authorize access to the corresponding directories by defining a directory object pointing to the directory (on the graph server) that contains the files to read or write.

```
CREATE OR REPLACE DIRECTORY graph_files AS '/data/graphs/my_graphs'; GRANT READ, WRITE ON DIRECTORY graph_files TO GRAPH_DEVELOPER;
```

Also, note the following:

- The directory in the CREATE DIRECTORY statement must exist on the graph server (PGX).
- The directory must be readable (and/or writable) at the OS level by the graph server (PGX).

The preceding code grants the privileges on the directory to the <code>GRAPH_DEVELOPER</code> role. However, you can also grant permissions to an individual user:

```
GRANT READ ON DIRECTORY graph_files TO <graph_user>;
```

- Loading Graph Data from Files
- Loading Graph Data in Parallel from Multiple Files
- · Exporting Graphs Into a File
- Exporting a Graph into Multiple Files

24.1 Loading Graph Data from Files

You can load graph data from files by either of the two ways:

- using the header format specified in the files
- · by directly calling the graph builder API

Creating a graph using file header format

The graph server (PGX) uses the header of the files to determine the name and types of the properties to load. It also infers the column to be used as vertex ID, the columns that indicate the source and destination vertex ID for edges, and the column to be loaded as vertex or edge label.

Creating a graph using graph builder API

You can also use PgxSession.readGraphFiles() to load the graph. This method takes the following three arguments:

- path to the vertex file
- path to the edge file
- name of the graph to be created
- JShell
- Java
- Python

JShell

```
opg4j> var loadedGraph = session.readGraphFiles("<path/vertices.csv>",
"<path/edges.csv>", "<graph name>")
```

Java

```
import oracle.pgx.api.PgxSession;
import oracle.pgx.api.PgxGraph;

PgxSession session = Pgx.createSession("NewSession");
PgxGraph loadedGraph = session.readGraphFiles("<path/vertices.csv>",
"<path/edges.csv>", "<graph name>");
```

Python

```
session = pypgx.get_session(session_name="<session_name>")
loaded_graph = session.read_graph_files("<path/vertices.csv>", "<path/edges.csv>", "<graph_name>")
```

The graph server (PGX) supports loading graph data from files for the following data formats

- Plain Text Formats
- XML File Formats
- Binary File Formats
- · Graph Configuration for Loading from File
- · Specifying the File Path
- Supported File Access Protocols



- Plain Text Formats
- XML File Formats
- Binary File Formats

24.1.1 Graph Configuration for Loading from File

The following table presents the graph configuration options to load graph data from all supported file formats to the graph server (PGX).

Table 24-1 Loading from File - Graph Configuration Options

Field	Type	Description	Dofault
array_compaction_threshold	number	[only relevant if the graph is optimized for updates] Threshold used to determined when to compact the delta-logs into a new array. If lower than the engine min_array_compaction_threshold value, min_array_compaction_threshold will	0.2
attributes	object	be used instead. Additional attributes needed to read and write the graph data.	null
detect_gzip	boolean	Enable or disable automatic gzip compression detection when loading graphs.	true
edge_id_strategy	<pre>enum[no_ids, keys_as_ids, unstable_gen erated_ids]</pre>	Indicates what ID strategy should be used for the edges of this graph. If not specified (or set to null), the strategy will be determined during loading or using a default value.	null
edge_id_type	enum[long]	Type of the edge ID. For homogeneous graphs, if not specified (or set to null), it will default to long.	null
edge_props	array of object	Specification of edge properties associated with graph.	[]
edge_uris	array of string	List of unified resource identifiers.	[]
error_handling	object	Error handling configuration.	null
external_stores	array of object	Specification of the external stores where external string properties reside.	[]
format	<pre>enum[pgb, edge_list, adj_list, graphml, pg, rdf, two tables]</pre>	Graph format to be used.	null
header	boolean	First line of file is meant for headers. For example, 'EdgeId, SourceId, DestId, EdgeProp1, EdgeProp2'	false
keystore_alias	string	Alias to the keystore to use when connecting to the database.	null
loading	object	Loading-specific configuration.	null



Table 24-1 (Cont.) Loading from File - Graph Configuration Options

Field	Туре	Description	Default
local_date_forma t	array of string	Array of local_date formats to use when loading and storing local_date properties. See DateTimeFormatter for documentation of the format string.	[]
optimized_for	<pre>enum[read, updates]</pre>	Indicates if the graph must use data- structures optimized for read-intensive scenarios or for fast updates.	read
<pre>partition_while_ loading</pre>	<pre>enum[by_labe l, no]</pre>	Indicates if the graph must partitioned while loading.	null
password	string	Password to use when connecting to database.	null
point2d	string	Longitude and latitude as floating point values separated by a space.	0.0 0.0
separator	string	A series of single-character separators for tokenizing. The characters ", $\{,\}$ and n cannot be used as separators. Default value is ", " for CSV files, and " t " for other formats. The first character will be used as a separator when storing.	null
storing	object	Storing-specific configuration.	null
time_format	array of string	The time format to use when loading and storing time properties. See DateTimeFormatter for documentation of the format string.	[]
<pre>time_with_timezo ne_format</pre>	array of string	The time with timezone format to use when loading and storing time with timezone properties. See DateTimeFormatter for documentation of the format string.	[]
<pre>timestamp_format</pre>	array of string	The timestamp format to use when loading and storing timestamp properties. See DateTimeFormatter for documentation of the format string.	[]
<pre>timestamp_with_t imezone_format</pre>	array of string	The timestamp with timezone format to use when loading and storing timestamp with timezone properties. See DateTimeFormatter for documentation of the format string.	[]
<pre>vector_component _delimiter</pre>	character	Delimiter for the different components of vector properties.	;
<pre>vertex_id_strate gy</pre>	<pre>enum[no_ids, keys_as_ids, unstable_gen erated_ids]</pre>	Indicates what ID strategy should be used for the vertices of this graph. If not specified (or set to null), the strategy will be automatically detected.	null
vertex_id_type	<pre>enum[int, integer, long, string]</pre>	Type of the vertex ID. For homogeneous graphs, if not specified (or set to null), it will default to a specific value (depending on the origin of the data).	null
vertex_props	array of object	Specification of vertex properties associated with graph.	[]
vertex_uris	array of string	List of unified resource identifiers.	[]



In the CSV format, the columns used to specify the vertex ID column, vertex labels column, edge ID column, edge source ID column, edge destination ID column and the edge label column can be configured with the CSV specific fields as shown in the following table:

Table 24-2 CSV Specific Options

Field	Туре	Description	Default
array_compaction _threshold	number	[only relevant if the graph is optimized for updates] Threshold used to determined when to compact the delta-logs into a new array. If lower than the engine min_array_compaction_threshold value, min_array_compaction_threshold will be used instead.	0.2
attributes	object	Additional attributes needed to read and write the graph data.	null
detect_gzip	boolean	Enable or disable automatic gzip compression detection when loading graphs.	true
edge_destination _column	value	Name or index (starting from 1) of column corresponding to edge destination (for CSV format only).	null
edge_id_column	value	Name or index (starting from 1) of column corresponding to edge id (for CSV format only).	null
edge_id_strategy	<pre>enum[no_ids, keys_as_ids, unstable_gen erated_ids]</pre>	Indicates what ID strategy should be used for the edges of this graph. If not specified (or set to null), the strategy will be determined during loading or using a default value.	null
edge_id_type	enum[long]	Type of the edge ID. For homogeneous graphs, if not specified (or set to null), it will default to long.	null
edge_label_colum n	value	Name or index (starting from 1) of column corresponding to edge label (for CSV format only).	null
edge_props	array of object	Specification of edge properties associated with graph.	[]
edge_source_colu mn	value	Name or index (starting from 1) of column corresponding to edge source (for CSV format only).	null
error_handling	object	Error handling configuration.	null
external_stores	array of object	Specification of the external stores where external string properties reside.	[]
format	<pre>enum[pgb, edge_list, adj_list, graphml, pg, rdf, two_tables]</pre>	Graph format to be used.	null
header	boolean	First line of file is meant for headers. For example, 'EdgeId, SourceId, DestId, EdgeProp1, EdgeProp2'.	false

Table 24-2 (Cont.) CSV Specific Options

Field	Туре	Description	Default
keystore_alias	string	Alias to the keystore to use when connecting to database.	null
loading	object	Loading-specific configuration.	null
<pre>local_date_forma t</pre>	array of string	array of local_date formats to use when loading and storing local_date properties. See DateTimeFormatter for documentation of the format string	[]
optimized_for	<pre>enum[read, updates]</pre>	Indicates if the graph should use data- structures optimized for read-intensive scenarios or for fast updates.	read
<pre>partition_while_ loading</pre>	<pre>enum[by_labe l, no]</pre>	Indicates if the graph should be partitioned while loading.	null
password	string	Password to use when connecting to database.	null
point2d	string	Longitude and latitude as floating point values separated by a space.	0.0 0.0
separator	string	a series of single-character separators for tokenizing. The characters ", $\{,\}$ and \n cannot be used as separators. Default value is "," for CSV files, and " \t " for other formats. The first character will be used as a separator when storing.	null
storing	object	Storing-specific configuration.	null
time_format	array of string	The time format to use when loading and storing time properties. See DateTimeFormatter for documentation of the format string	[]
<pre>time_with_timezo ne_format</pre>	array of string	The time with timezone format to use when loading and storing time with timezone properties. See DateTimeFormatter for documentation of the format string.	[]
timestamp_format	array of string	The timestamp format to use when loading and storing timestamp properties. See DateTimeFormatter for documentation of the format string.	[]
<pre>timestamp_with_t imezone_format</pre>	array of string	The timestamp with timezone format to use when loading and storing timestamp with timezone properties. See DateTimeFormatter for documentation of the format string.	[]
<pre>vector_component _delimiter</pre>	character	Delimiter for the different components of vector properties.	;
vertex_id_column	value	Name or index (starting from 1) of column corresponding to vertex id (for CSV format only).	null
<pre>vertex_id_strate gy</pre>	<pre>enum[no_ids, keys_as_ids, unstable_gen erated_ids]</pre>	Indicates what ID strategy should be used for the vertices of this graph. If not specified (or set to null), the strategy will be automatically detected.	null



Field	Туре	Description	Default
vertex_id_type	<pre>enum[int, integer, long, string]</pre>	Type of the vertex ID. For homogeneous graphs, if not specified (or set to null), it will default to a specific value (depending on the origin of the data).	null
vertex_labels_co lumn	value	Name or index (starting from 1) of column corresponding to vertex labels (for CSV format only).	null
vertex_props	array of object	Specification of vertex properties associated with graph.	[]

24.1.2 Specifying the File Path

The following examples show how to specify the file path for various file formats.

For formats that contain vertices and edges specified in one file (for example, EdgeList), use uris as shown in the following code:

```
{"uris":["path/to/file.format"]}
```

For formats that require separate files for edges and vertices (for example, FlatFile), use vertex uris and edge uris as shown in the following code:

```
{"vertex_uris":["vertices1.format","vertices2.format"],"edge_uris":
["edges1.format","edges2.format"]}
```

PGX will parse graphs in most of the plain text formats in parallel if the graph data is split into multiple files, as shown in the following code:

```
{"uris":["file1.format", "file2.format", ..., "fileN.format"]}
```

24.1.3 Supported File Access Protocols

The graph server (PGX) supports loading from graph configuration files and graph data files over various protocols and virtual file systems. The type of file system or protocol is determined by the scheme of the uniform resource identifier (URI):

- local file system (file:) this is also the default if the given URI does not contain any scheme
- classpath (classpath: or res:)
- HDFS (hdfs:)
- HTTPS (https:)
- FTPS (ftps:)



various archive formats (zip:, jar:, tar:, tgz:, tbz2:, gz: and bz2:). The URI format is scheme://arch-file-uri[!absolute-path] (if you would like to use the ! as a literal file-name character it must be escaped using %21).
 For example, jar:../lib/classes.jar!/META-INF/graph.json.

Paths may be nested as in tar:gz:https://anyhost/dir/mytar.tar.gz!/mytar.tar!/path/in/tar/graph.data.



Relative paths are always resolved relative to the parent directory of the configuration file.

24.1.4 Plain Text Formats

The graph server (PGX) supports the following plain-text formats:

- Comma-Separated Values (CSV)
- Adjacency List (ADJ_LIST)
- Edge List (EDGE_LIST)
- Two Tables (TWO TABLES)
- Flat File (FLAT_FILE)

Parsing of Vertices

PGX supports three types of vertex identifies (id): integer, long and string. The type defaults to integer, but can be configured through the vertex_id_type option in the graph configuration.

Parsing of Edges

Of the various formats and protocols supported by graph server (PGX), only CSV and flat file parsing support edge identifiers. For all other data sources, the id of an edge is PGX's internal id, which is an integer from zero to num edges - 1.

Parsing of Properties

string properties, spatial properties (currently only point2d) and temporal properties (date, local_date, time, timestamp, time_with_timezone and timestamp_with_timezone) must be quoted ("<string>") only if they contain a separator character (usually , for CSV and ' ' for Edge List and Adjacency List) or if they contain " or n.

date properties are parsed using Java's SimpleDateFormat utility, instantiated with the format string yyyy-MM-dd HH:mm:ss unless specified otherwise in the graph configuration. All other types of temporal properties are parsed using Java's DateTimeFormatter utility.

point2d can be specified by its longitude followed by its latitude, separated by a space. Both longitude and latitude are doubles. For example, "-74.0445 40.6892" is the representation of a point2d instance representing the location of the Statue of Liberty.



Boolean values are interpreted as true if the value is true (ignoring case), Y (ignoring case) or 1, false otherwise. The suggested notation for false is false (ignoring case), N (ignoring case) or 0. All other types are parsed using the parseXXXX() functions of its corresponding Java type, for example, Integer.parseInt(...) for integer types.

Vector properties are supported in the Adjacency List (ADJ_LIST), Comma-Separated Values (CSV), Edge List (EDGE_LIST), and Two Tables text (TWO_TABLES) formats. Vector properties with vector components of type <code>integer</code>, <code>long</code>, <code>float</code> and <code>double</code> can be loaded from these formats. In order to specify that a vertex or edge property is a vector property, the <code>dimension</code> field of the graph property configuration must be set to the dimension of the vector and be a strictly positive integer value. A vector value is represented in the supported text formats by the list of the vector components values separated by the vector component delimiter. By default the vector component delimiter is <code>;</code>, but this delimiter can be changed by changing the <code>vector_component_delimiter</code> graph configuration entry. Therefore a 3-dimensional vector of doubles could for example look like <code>0.1;0.0004;3.14</code> in the text file if the vector component delimiter is <code>;</code>.

Separators

When using single file formats, IDs and properties are separated with tab or one single space ("\t ") by default, for multiple file formats comma (",") is used instead. However, PGX allows to configure the separator string.

Parallel Loading

The following formats support parallel loading from multiple files:

- CSV (specify multiple files in vertex_uris and/or edge_uris)
- Adjacency List (specify multiple files in uris)
- Edge List (specify multiple files in uris)
- Two Tables (specify multiple files in vertex uris and/or edge uris)
- Flat File (specify multiple files in vertex_uris and/or edge_uris)

Legend

The following abbreviations are used to specify text formats:

- V = Vertex Key
- VG = Neighbor Vertex
- VL = Vertex Labels
- VP = Vertex Property
- VPK = Vertex Property Key
- VPT = Vertex Property Type
- EL = Edge Label
- EP = Edge Property
- EPK = Edge Property Key
- EPT = Edge Property Type

For example <V-2, VG-4> or <V-2, VG-4> denotes the 4th neighbor of the 2nd vertex.

Comma-Separated Values (CSV)



- Adjacency List (ADJ_LIST)
- Edge List (EDGE LIST)
- Two Tables (TWO_TABLES)

24.1.4.1 Comma-Separated Values (CSV)

The CSV format is a text file format with vertices and edges stored in different files. Each line of the files represents a vertex or an edge. The vertex key and labels, the edge key, source, destination and label, and the attached properties are stored in the order specified by the file header (first line) and the configuration.

A graph with V vertices, having N vertex properties and K neighbors each, and E edges, having M edge properties, would be represented in CSV as shown:

Example 24-1 Loading graph from a CSV file with header details

The following examples shows a graph configuration file for loading a graph with two vertices and two edges:

```
key,integer_prop,string_prop
1,33,"Alice"
2,42,"Bob"

edges.csv

source,dest,integer_prop,string_prop
1,2,0,"baz"
2,2,-12,"bat"
```

The corresponding graph configuration file is as shown:

```
{
    "format": "csv",
    "header": true,
    "vertex_id_column": "key",
    "edge_source_column": "source",
    "edge destination column": "dest",
```



```
"vertex uris": ["vertices.csv"],
    "edge uris": ["edges.csv"],
    "vertex props": [
            "name": "integer prop",
            "type": "integer"
        },
            "name": "string_prop",
            "type": "string"
    ],
    "edge props": [
            "name": "integer prop",
            "type": "integer"
        },
            "name": "string prop",
            "type": "string"
    ]
}
```

Example 24-2 Loading graph from a CSV file without header details

The following examples shows a graph configuration file for loading a graph with two vertices and two edges:

```
vertices.csv

1,33,"Alice"
2,42,"Bob"

edges.csv

1,2,0,"baz"
2,2,-12,"bat"
```

The corresponding graph configuration file is as shown:



The column indices are given in place of the column names.

```
"format": "csv",
   "header": false,
   "vertex_id_column": 1,
   "edge_source_column": 1,
   "edge destination column": 2,
```



```
"vertex uris": ["vertices.csv"],
    "edge uris": ["edges.csv"],
    "vertex props": [
            "name": "integer prop",
            "type": "integer",
            "column": 2
        },
        {
            "name": "string prop",
            "type": "string",
            "column": 3
    ],
    "edge _props": [
            "name": "integer prop",
            "type": "integer",
            "column": 3
        },
            "name": "string prop",
            "type": "string",
            "column": 4
    ]
}
```

If no column indices are set in the configuration file, the columns are assumed to be in the following order:

- For vertex files: Vertex ID Vertex labels (if present) Vertex properties in the order they are declared in the configuration
- For edge files: Edge ID (if present) Edge source Edge destination Edge label (if present) Edge properties in the order they are declared in the configuration

Therefore the earlier configuration is equivalent to:

24.1.4.2 Adjacency List (ADJ LIST)

The Adjacency List format is a text file format containing a list of neighbors from a vertex, per line. The format is extended to encode properties. The following shows a graph with V vertices, having N vertex properties and M edge properties:

```
<V-1> <V-1, VP-1> ... <V-1, VP-N> <V-1, VG-1> <EP-1> ... <EP-M> <V-1, VG-2> <EP-1> ...
<EP-M>
<V-2> <V-2, VP-1> ... <V-2, VP-N> <V-2, VG-1> <EP-1> ... <EP-M> <V-2, VG-2> <EP-1> ...
<EP-M>
...
<V-V> <V-V, VP-1> ... <V-V, VP-N> <V-V, VG-1> <EP-1> ... <EP-M> <V-V, VG-2> <EP-1> ...
<EP-M>
```

Note:

Trailing separators will be considered as errors. For example, if whitespace is used to separate the properties, any trailing whitespace will cause an exception to be raised.

Example 24-3 Graph in Adjacency List Format

This example shows a graph with 4 vertices (1, 2, 3 and 4), each having a double and a string property, and 3 edges, each having a boolean and a date property, encoded in Adjacency List format:

```
1 8.0 "foo"
2 4.3 "bar" 1 false "1985-10-18 10:00:00"
3 6.1 "bax" 2 true "1961-12-30 14:45:14" 4 false "2001-01-15 07:00:43"
4 17.78 "f00"
```

Note:

ADJ_LIST is more space efficient than EDGE_LIST. This is because vertices are first defined and then the edges are being created, indicating that we are repeating each vertex at least once.

24.1.4.3 Edge List (EDGE LIST)

The Edge List format is a text file format starting with a section with one vertex per line, followed by a section with one edge per line. If a vertex does not have any labels or

properties, it is possible to omit the vertex in the first section, but still specify edges for the vertex in the second section.

```
EdgeList := {Vertex '\n'}* '\n' {Edge '\n'}*

Vertex := VertexId '*' VertexLabels? PropertyValue*

VertexId := Integer | Long | String

VertexLabels := '{' String* '}'

Edge := SrcVertex DstVertex EdgeLabel? PropertyValue*

SrcVertex := VertexId

DstVertex := VertexId

EdgeLabel := String

PropertyValue := Integer | Long | Double | Float | Boolean | String | Date
```

The vertices start with an identifier (VertexId), followed by a *, an optional set of vertex labels (VertexLabels?) and the vertex properties (PropertyValue*). A vertex identifier is either an Integer, a Long, or a String. Furthermore, vertex labels are zero or more Strings between curly braces ('{' String* '}').

The edges start with source and destination vertex identifiers (SrcVertex DstVertex), followed by optional edge label (EdgeLabel?) and the edge properties (PropertyValue*). The edge label is a String.

Example 24-4 Graph in Edge List format

This example shows a graph with two vertices and two edges, with labels and properties:

```
1 * { "Person" "Male" } "Mario" 15
2 * { "Person" "Male" } "Luigi" 14
1 2 "likes" 3.5
2 1 "likes" 2.1
```

The two vertices (lines 1-2) have identifiers 1 and 2 and both have the labels "Person" and "Male", a string property ("Mario" and "Luigi") and an integer property (15 and 14). There is an edge from vertex 1 to vertex 2 (line 3) with label "likes" and a double property with value 3.5, and another edge from vertex 2 to vertex 1 with label "likes" and a double property with value 2.1.

The following shows the corresponding graph configuration:

```
{
  "format":"edge_list",
  "uri":"example.edgelist",
  "vertex_id_type":"long",
  "vertex_labels":true,
  "edge_label":true,
  "vertex_props":[
      {
        "name":"name",
        "type":"string"
      },
      {
        "name":"age",
        "type":"int"
```



```
}
],
"edge_props":[
    {
        "name":"rating",
        "type":"double"
    }
],
"loading_options": {
        "load_vertex_labels":true,
        "load_edge_label":true
},
        "separator":" "
}
```

24.1.4.4 Two Tables (TWO_TABLES)

When configured to use file as datastore, the Two Tables format becomes a text file format similar to the Edge List format, with the only difference that the vertices and edges are stored in two different files. The vertices file contains vertex IDs followed by vertex properties. The edges file contains the source vertices and target vertices, followed by edge properties.

A graph with V vertices, having N vertex properties and M edge properties would be represented in two files as shown in the following:

Example 24-5 Graph in Two Tables Text format

The following example shows the graph of 4 vertices (1, 2, 3 and 4), each having a double and a string property, and 3 edges, each having a boolean and a date property, encoded in Two Tables Text format:

```
vertices.ttt:

1 8.0 "foo"
2 4.3 "bar"
3 6.1 "bax"
4 17.78 "f00"

edges.ttt:

2 1 false "1985-10-18 10:00:00"
3 2 true "1961-12-30 14:45:14"
3 4 false "2001-01-15 07:00:43"
```





If you are planning on storing big graphs you must consider Two Tables Text format in order to save disk space.

24.1.5 XML File Formats

Graph ML

The graph server (PGX) supports loading graphs from files using the XML-based Graph ML format. Graphs already in memory may also be exported into GraphML files. See GraphML specification for a detailed description of the XML schema.

PGX GraphML Limitation

PGX does not support all features of the GraphML format. Some of the limitations are:

- If the graph is undirected (edgedefault="undirected"), then edge properties are not supported
- All vertices (edges) must have the same amount and type of vertex (edge) properties
- port, default, and hyperedge are not supported

Example 24-6

The following example graph consists of 3 vertices and 3 edges. Each vertex has an integer property named number and each edge has a string property named label. Note that the edges are directed and that the strings for the property do not have to be put in (double) quotation marks.

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns">
    <key attr.name="number" attr.type="integer" for="node"</pre>
id="number"/>
    <key attr.name="label" attr.type="string" for="edge" id="label"/>
    <graph edgedefault="directed">
        <node id="1">
            <data key="number">2</data>
        </node>
        <node id="2">
            <data key="number">45</data>
        </node>
        <node id="3">
            <data key="number">83</data>
        </node>
        <edge target="2" source="1">
            <data key="label">this graph</data>
        </edge>
        <edge source="3" target="2">
            <data key="label">forms a</data>
        </edge>
        <edge target="1" source="3">
            <data key="label">triangle</data>
```



```
</edge>
</graph>
</graphml>
```



Caution:

Due to the verbose nature of XML, the GraphML format comes with a large overhead compared to other file-based graph formats. You must use a different format if you want to consider the load or store performance and file size as important factors.

24.1.6 Binary File Formats

PGX Binary Format (PGB)

PGX binary format (.pgb) is the proprietary binary format for graph server (PGX), which allows fast and efficient file processing. Fundamentally, the file is a binary dump of the graph and property data. Bytes are written in network byte order (big endian).

Type Encoding

Table 24-3 Type Encoding

Value	Туре	Size in bytes
0	Boolean	1
1	Integer	4
2	Long	8
3	Float	4
4	Double	8
7	String	varies
11	Vertex labels	varies
13	Local date	4
14	Time	4
15	Timestamp	8
16	Time with time zone	8
17	Timestamp with time zone	12
18	Vector property	<pre>variable: <sizeof component-="" type=""> * <dimension></dimension></sizeof></pre>

File Layout

Table 24-4 File Layout

Size in bytes	Description	Require Commo	ent
4	magic word	Yes 0x9919	1191



Table 24-4 (Cont.) File Layout

Size in bytes	Description	Require d	Comment
4	vertex size	Yes	Allowed values are 4 and 8.
4	edge size	Yes	Allowed values are 4 and 8.
<vertex size=""></vertex>	number of vertices	Yes	
<edge size=""></edge>	number of edges	Yes	
<pre><edge size=""> * (<numvertices> + 1)</numvertices></edge></pre>	edge begin array	Yes	
<pre><vertex size=""> * <numedges></numedges></vertex></pre>	destination vertex array	Yes	
1	component bitmap	Yes	• 0x0001: node keys
			• 0x0002: vertex labels
			• 0x0004: edge label
			• 0x0008: edge keys
			 other bits: reserved
4	vertexKey type	No	Only present if <i>component bitmap</i> & $0x0001 == 0x0001$. See Table 24-3 for type encoding.
<pre><vertex key="" layout=""></vertex></pre>	vertex keys	No	Only present if <i>component bitmap</i> & $0 \times 0001 == 0 \times 0001$.
4	edgeKey type	No	Only present if <i>component bitmap</i> & $0x0008 == 0x0008$. See table Table 24-3 for type encoding
<numedges> * 8</numedges>	edge keys	No	Only present if <i>component bitmap</i> & $0x0008 == 0x0008$.
4	number of vertex properties	Yes	
<pre><num properties="" vertex=""> * <pre><pre>cproperty layout></pre></pre></num></pre>	property data	Yes	See Table 24-10.
4	number of edge properties	Yes	
<pre><num edge="" properties=""> * <pre><pre><pre>property layout></pre></pre></pre></num></pre>	property data	Υ	See Edge Property Layout.
<pre><vertex labels="" layout=""></vertex></pre>	vertex labels	No	Only present if <i>component bit</i> & 0×0002 == 0×0002 .
<pre><edge labels="" layout=""></edge></pre>	edge label	No	Only present if <i>component bit</i> & 0×0004 == 0×0004 .
4	number of shared pools	Yes	
<pre><shared pools="" size=""></shared></pre>	•	No	
<pre><pre><pre>property names size></pre></pre></pre>	property names	No	Only present if <i>component bit</i> & 0×0010 == 0×0010 . See Table 24-19.

Vertex Key Layout

The layout of vertex keys depends on the vertex Key type. PGB supports integer, long and string vertex keys.



Table 24-5 Integer Vertex Keys

Size in bytes	Description	Require d	Comment
<pre><numvertices> * 4</numvertices></pre>	key data	Yes	For each vertex, the corresponding integer key value.

Table 24-6 Long Vertex Keys

Size in bytes	Description	Require d	Comment
<numvertices> * 8</numvertices>	key data	Yes	For each vertex, the corresponding long key value.

Table 24-7 String Vertex Keys

Size in bytes	Description	Require d	Comment
4	compression scheme	Yes	reserved (must be 0)
8	property size	Yes	size of each element in bytes in the following data
<pre><number keys="" of=""> * <string element="" key="" layout=""></string></number></pre>	string key data	Yes	content of the vertex keys (see Table 24-5)

Table 24-8 String Key Element Layout

Size in bytes	Description	Require d	Comment
4	string length	Yes	length of the string in bytes
<string length=""></string>	string key data	Yes	content of the string as bytes, No zero- character

Property Layout

The following shows the special layout for string properties, and for vector properties:

Table 24-9 Primitive Type Layout

Size in bytes	Description	Require d	Comment
4	property type	Yes	See Table 24-3 for type encoding.
8	property size	Yes	Size of the property data in bytes
<pre><pre><pre><pre><pre><pre><pre><pre></pre></pre></pre></pre></pre></pre></pre></pre>	property data	Yes	<pre>Stored as <numvertices numedges=""></numvertices></pre>



Table 24-10 Vector Property Layout

Size in bytes	Description	Comment
4	vector type mark	Always equal to 18.
8	size of vector property data and extra fields	<pre>dataSize = <sizeof component-type=""> * <dimension> + 8 (The 8 extra bytes are for the added following 2 extra fields in the vector property header.)</dimension></sizeof></pre>
4	vector component data type	Valid types are integer, long, float, double. Encoded with the value specified in Table 24-3.
4	vector dimension	Number of components per vector value. Must be greater than 0 to be a valid vector property.
dataSize - 8	data	Stored as array of length * ` in which the value of the j-th component of the vector for the i-th entity is at position i * + j`.

Table 24-11 String Type Layout

Size in bytes	Description	Requir ed	Comment
4	property type	Yes	Must be 7.
8	property size	Yes	Size of the following data in bytes.
1	reserved	Yes	Reserved (must be 0).
<dictionary layout=""></dictionary>	dictionary	Yes	String dictionary used in the property
<pre><numvertices numedges=""> * 8</numvertices></pre>	property content	Yes	Content of the string property, stored as IDs that refer to the strings in the dictionary.

Table 24-12 String Dictionary Layout

Size in bytes	Description	Requir ed	Comment
1	reserved	Yes	Reserved (must be 0).
8	number of strings	Yes	Number of strings in the following dictionary.
<pre><number of="" strings=""> * <dictionary element="" layout=""></dictionary></number></pre>	dictionary data	Yes	See Table 24-13.

Table 24-13 String Dictionary Element Layout

Size in bytes	Description	Requir ed	Comment
8	string id	Yes	Unique ID of the string.



Table 24-13 (Cont.) String Dictionary Element Layout

Size in bytes	Description	Requir ed	Comment
4	string length	Yes	Length of the string in bytes.
<string length=""></string>	string data	Yes	Content of the string as bytes, No zero-character

Vertex Labels Layout

Table 24-14 Vertex Labels Layout

Size in bytes	Description	Require d	Comment
4	type	Yes	Must be 11.
8	size	Yes	Size of the following data in bytes.
<dictionary layout=""></dictionary>	dictionary	Yes	String dictionary used in the vertex labels.
<numvertices +="" 1=""> * 8</numvertices>	string id begin array	Yes	<pre><string ids=""> offset array for each vertex.</string></pre>
8	number of string ids	Yes	The number of string ids.
<pre><number ids="" of="" string=""> * 8</number></pre>	string ids	Yes	Array of string ids in the string dictionary.

Edge Label Layout

The edge label layout follows the string type layout.

Shared Pools Layout

Table 24-15 Shared Pools Layout

Size in bytes	Description	Require d	Comment
1	type	Yes	1: enum, 2: prefixed

Table 24-16 Type == Enum

Size in bytes	Description	Require d	Comment
8	num strings	Yes	
<pre><number of="" strings=""> * <string pre="" table<=""></string></number></pre>	dictionary data	Yes	See Table 24-18.
layout>			



Table 24-17 Type == Prefix

Size in bytes	Description	Requir ed	Comment
8	num prefixes	Yes	
<pre><number of="" prefixes=""> * <string layout="" table=""></string></number></pre>	dictionary data	Yes	See Table 24-18.
8	num suffixes	Yes	
<pre><number of="" suffixes=""> * <string layout="" table=""></string></number></pre>	dictionary data	Yes	See Table 24-18.

Table 24-18 String Table for Shared Pools

Size in bytes	Description	Requir ed	Comment
8	string id	Yes	String can be literal (in case of enum) or prefix/suffix (in case of prefix).
4	string length	Yes	
<string length=""></string>	string data	Yes	

Property Names Layout

Table 24-19 Property Names Layout

Size in bytes	Description	Requir ed	Comment
8	size	Yes	String can be literal (in case of enum) or prefix/suffix (in case of prefix).
<pre><sum names="" of="" property="" size="" vertex=""></sum></pre>	vertex property names	No	Follows the String Key Element Layout. See Table 24-8.
<pre><sum edge="" names="" of="" property="" size=""></sum></pre>	edge property names	No	Follows the String Key Element Layout. See Table 24-8.

24.2 Loading Graph Data in Parallel from Multiple Files

You can load a graph in parallel using multiple files.

The following example demonstrates how to load graph data from multiple files.



For example, consider a vertex file split into four partitions as shown:

```
vertex_file1

1,Color,1,red,,
2,Color,1,yellow,,

vertex_file2

3,Color,1,blue,,
4,Color,1,green,,

vertex_file3

5,Color,1,orange,,
6,Color,1,white,,

vertex_file4

7,Color,1,black,,
```

The edge file is split into two partitions as shown:

```
edge_file1

1,1,2,edge1,Weight,4,,1.0,
2,2,3,edge2,Weight,4,,2.0,
3,3,4,edge3,Weight,4,,3.0,

edge_file2

4,4,5,edge4,Weight,4,,4.0,
5,5,6,edge5,Weight,4,,5.0,
6,6,7,edge6,Weight,4,,6.0,
```

The following graph configuration can be used to load the graph data from four vertex files and two edge files into the same graph. Note that all the uris are specified inside the JSON graph configuration.

```
{
  "format": "flat_file",
  "vertex_uris": ["vertex_file1", "vertex_file2", "vertex_file3",
"vertex_file4"],
  "edge_uris": ["edge_file1", "edge_file2"],
  "separator": ",",
  "edge_props": [
     {
        "name": "Weight",
        "type": "double"
```



You can also create a graph configuration with multiple file partitions using Java as shown:

```
FileGraphConfig config = GraphConfigBuilder
    .forFileFormat(Format.FLAT_FILE)
    .setSeparator(",")
    .addVertexUri("vertex_file1")
    .addVertexUri("vertex_file2")
    .addVertexUri("vertex_file3")
    .addVertexUri("vertex_file4")
    .addEdgeUri("edge_file1")
    .addEdgeUri("edge_file2")
    .addVertexProperty("Color", PropertyType.STRING)
    .addEdgeProperty("Weight", PropertyType.DOUBLE)
    .build();
```

Note:

The graph configuration in the preceding codes include one double edge property named "Weight" and one string vertex property named "Color".

You can now load the graph data from the files as explained in Creating a graph using graph builder API.

The graph server (PGX) will automatically load the graph in parallel, using one thread for each file. This means that a graph can be loaded in parallel with as many threads as files are given depending on the configured parallelism for the graph server (PGX) instance.

Note:

Since the graph config will be used for all of the specified files, it is crucial to use the same format for all these files, that is, using the same separator, having the same defined properties, complying with the same format specification.

24.3 Exporting Graphs Into a File

The graph server (PGX) allows the client to export a currently loaded graph into a file.

Using the <code>store()</code> method on any PgxGraph object, the client can specify which file format to store the graph in. The client can also dynamically select the set of properties to be stored with the graph, that is, not all the properties need to be exported. The client can specify a CompressionScheme to use when storing as shown:

Table 24-20 Files CompressionScheme

CompressionScheme	Supported Formats
NONE	All formats
GZIP	ADJ_LIST, EDGE_LIST, FLAT_FILE, TWO_TABLES (text)

The client can export to multiple files as well.

When PGX exports the specified graph into a file, PGX also creates a graph config which the client receives as return value. This is to help loading the created graph instance later.

When exporting graph data into multiple files a FileGraphStoringConfig can be used which contains the following JSON fields:

Table 24-21 Graph Configuration when Exporting Graph into Multiple Files

	_		
Field	Туре	Description	Default
base_path	string	Base path to use for storing a graph; file paths will be constructed using the following format , that is, parent_path/ my_graph_1.edges.	null
compression_sche me	<pre>enum[none, gzip]</pre>	The scheme to use for compression, or none to disable compression.	none
delimiter	character	Delimiter character used as separator when storing. The characters ", {, } and \n cannot be used as delimiters.	null
edge_extension	string	The extension to use when creating edge file partitions.	edges
<pre>initial_partitio n_index</pre>	integer	The value used as initial partition index, that is, initial_partition_i ndex=1024 -> my_graph_1024.edges , my_graph_1025.edges .	1
num_partitions	integer	The number of partitions that should be created, when exporting to multiple files.	1



Table 24-21 (Cont.) Graph Configuration when Exporting Graph into Multiple Files

Field	Туре	Description	Default
row_extension	string	The extension to use when creating row file partitions.	rows
vertex_extension	string	The extension to use when creating vertex file partitions.	nodes

· Exporting a Graph to Disk

24.3.1 Exporting a Graph to Disk

You can save a graph loaded into memory to the disk in various formats. Therefore you can make sub-graphs and graph data computed at run time through analytics persistent, for future use. The resulting file can be used later as input for the graph server (PGX).

Consider the following example where a graph is loaded into memory and PageRank analysis is executed on the graph.

- JShell
- Java
- Python

JShell

```
var g = session.readGraphWithProperties("<path_to_json>")
var rank = analyst.pagerank(g, 0.001, 0.85, 100)
```

Java

```
PgxGraph g = session.readGraphWithProperties("<path_to_json>");
Analyst analyst = session.createAnalyst();
VertexProperty<Integer, Double> rank = analyst.pagerank(g, 0.001, 0.85, 100);
```

Python

```
g = session.read_graph_with_properties("<path_to_json>")
analyst = session.create_analyst()
rank = analyst.pagerank(g, 0.001, 0.85, 100)
```

You can now store the graph, together with the result of the PageRank analysis and all original edge properties, as a file in edge-list format, on disk. When a graph is stored, you need to specify the graph format, a path where the file should be stored, the properties to store and a flag that specifies whether or not a file should be overwritten should a file with the same name already exist.

- JShell
- Java
- Python

JShell

```
var config = g.store(Format.EDGE_LIST, "<file-path>", List.of(rank),
EdgeProperty.ALL, false)
```

Java

```
var config = g.store(Format.EDGE_LIST, "<file-path>", List.of(rank),
EdgeProperty.ALL, false);
```

Python

```
config = g.store('edge_list', "<file-path>", vertex_properties = [rank],
overwrite= False)
```

The graph data can now be found under the file path. The graph configuration returned by the store method can be used to load the new graph back into memory. To persist the graph configuration to disk as well, you can use the config's toString method to get a JSON representation:

- JShell
- Java
- Python

JShell

```
var path = Paths.get("<file-path>")
Files.writeString(path, config.toString())
```



Java

```
import apache.commons.io.*; // PGX contains a version of Apache
Commons IO
...
FileUtils.write(new File("<file-path>"), config.toString());
```

Python

```
with open("<file-path>","w"):
    f.write(str(config))
```

24.4 Exporting a Graph into Multiple Files

You can store a graph into multiple files using the store method. Most parameters are the same, as if storing to a single file. However, the main difference lies in specifying how to partition the data.

You can partition the data in either of the following two ways:

- specifying a FileGraphStoringConfig (see Table 24-21 for more information)
- specifying a base path and the number of partitions

Export into Multiple Files Using FileGraphStoringConfig

You can specify a more detailed way of creating the multiple partitions used to store the graph by using the FileGraphStoringConfig. You can create a FileGraphStoringConfig object using a FileGraphStoringConfigBuilder.

For example, the following code specifies that the storing should be done into four partitions using the specified base path and using zero as the initial index for the partitioning. It also contains the file extension to use for vertex files and for edge files and finally it sets comma as the delimiter to be used when storing the graph data:

```
FileGraphStoringConfig storingConfig = new
FileGraphStoringConfigBuilder(basePath) //
    .setNumPartitions(4) //
    .setInitialPartitionIndex(0) //
    .setVertexExtension(vertexExtension) //
    .setEdgeExtension(edgeExtension) //
    .setDelimiter(',') //
    .build();
```

You can also partition all tables equally using the numPartitions parameter. This implies that all tables are exported into the same number of files.

If you do not want to partition the tables equally, you can either create one PartitionedGraphConfig which contains for each provider a FileGraphStoringConfig (see Table 24-21) or we can use a version of store() that



takes two maps of FileGraphStoringConfigs, one for the vertex tables and one for the edge tables.

For the first option, you can create for each vertex and edge table a FileGraphStoringConfig and put it into a FileEntityProviderConfig using setStoringOptions in the builder of FileEntityProviderConfig. The providers are then added to the PartitionedGraphConfig as edge and vertex providers using addVertexProvider() and addEdgeProvider() in the builder of PartitionedGraphConfig. Later you can use the store() method which takes the PartitionedGraphConfig as parameter.

The second option creates for every edge and vertex table a storing configuration, adds those into a vertex provider and an edge provider map and calls the corresponding <code>store()</code> method with these maps as parameters.

For example:

```
FileGraphStoringConfig vertexStoringConfig1 = new
FileGraphStoringConfigBuilder(basePath + " vertexTable1") //
  .setNumPartitions(4) //
  .setInitialPartitionIndex(0) //
  .setVertexExtension(vertexExtension) //
  .setDelimiter(',') //
  .build();
FileGraphStoringConfig vertexStoringConfig2 = new
FileGraphStoringConfigBuilder(basePath + " vertexTable2") //
  .setNumPartitions(4) //
  .setInitialPartitionIndex(0) //
  .setVertexExtension(vertexExtension) //
  .setDelimiter(',') //
  .build();
FileGraphStoringConfig edgeStoringConfig1 = new
FileGraphStoringConfigBuilder(basePath + " edgeTable1") //
  .setNumPartitions(4) //
  .setInitialPartitionIndex(0) //
  .setEdgeExtension(edgeExtension) //
  .setDelimiter(',') //
  .build();
Map<String, FileGraphStoringConfig> vertexStoringConfigs = new HashMap<>();
vertexStoringConfigs.put("vertexTable1", vertexStoringConfig1);
vertexStoringConfigs.put("vertexTable2", vertexStoringConfig2);
Map<String, FileGraphStoringConfig> edgeStoringConfigs = new HashMap<>();
edgeStoringConfigs.put("edgeTable1", edgeStoringConfig);
```

Export into Multiple Files without FileGraphStoringConfig

If you only need to specify how many partitions are required and the base name to be used, it is simpler to use <code>store()</code> method by only specifying those parameters. Following this procedure, the graph server (PGX) will use defaults for the other fields. See Table 24-21 for more information on default values.

Export into Multiple Files Using a Graph Configuration Object

An alternate way for exporting into multiple files is by creating a FileGraphStoringConfig and putting it into a Graph Configuration object using setStoringOptions in its builder, and then using the corresponding version of the store () method.



Log Management in the Graph Server (PGX)

The graph server (PGX) internally uses the SLF4J interface with Logback as the default logger implementation.

Configuring Logback Logging

25.1 Configuring Logback Logging

The default Logback logging configuration file is located in /etc/oracle/graph/logback-server.xml. This configuration file contains the target location for the logs in /var/log/oracle/graph/. Additionally, the rolling file appenders are also defined in this configuration file.

Note:

- Logback is configured to roll the log files based on both log size (250 MB) and date.
- Log files are automatically saved in a compressed format in subdirectories, one directory per month. There can be multiple files on a given day.
- Also, each startup of the graph server(PGX) triggers a new log file.

The Logback configuration file is picked up automatically by the the graph server(PGX). To use this configuration in your java application, you can set the <code>logback.configurationFile</code> system variable when launching the JVM:

```
java -Dlogback.configurationFile=$PGX HOME/conf/logback.xml ...
```

Changing Logging Level During a JShell Session

When connected to the graph server using JShell, you can use the loglevel (String loggerName, String levelName) function to quickly change the logging level of any logger. For example:

```
loglevel("oracle.pgx", "debug")
loglevel("ROOT", "info")
loglevel("org.apache.hadoop", "off")
```

Logging in a Web Application Server

The graph-server-<version>-pgx<version>.war file in the oracle-graph-webapps-<version>.zip download package contains the logback.xml. This file determines what should be logged in the web application running on the application server of your choice. The

file is located in the folder WEB-INF/classes inside the graph-server-<version>-pgx<version>.war file. By default, only errors are logged. But you can change this file if you want more logging in your web server. You must restart the web server after you change the file, for the change to take effect.



Part VIII

Supplementary Information for Property Graph Support

This document has the following appendixes.

- Using the Property Graph Schema
 This chapter provides conceptual and usage information about creating, storing, and working with property graph data in an Oracle Database environment.
- Mapping Graph Server Roles to Default Privileges
- Disabling Transport Layer Security (TLS) in Graph Server
- Migrating Property Graph Applications from Before Release 21c
 If you are migrating from a previous version of Oracle Spatial and Graph to Release 21c, you may need to make some changes to existing property graph-related applications.
- Upgrading From Graph Server and Client 20.4.x to 21.x If you are upgrading from Graph Server and Client 20.4.x to 21.x version, you may need to create new roles in database and migrate authorization rules from pgx.conf file to the database. Also, starting from Graph Server and Client Release 21.1, TLS is enforced at the time of the RPM file installation.
- Third-Party License Information for Oracle Graph Server and Client
 This appendix contains licensing information about third-party products included with
 Oracle Graph Server and Client.

A

Using the Property Graph Schema

This chapter provides conceptual and usage information about creating, storing, and working with property graph data in an Oracle Database environment.

You can create a property graph and store it in the property graph schema in Oracle Database in one of the following ways:

- 1. Use the CREATE PROPERTY GRAPH statement to create and populate these property graph schema objects.
- Use OPG_APIS.CREATE_PG, to create the property graph schema objects. Then load data
 from the database tables into the schema objects using SQL or using the Data Access
 Layer APIs. The property graph schema provides a flexible schema option for storing
 your graph.



The original database tables remain as-is and the data is copied from the original tables into the property graph schema tables.

- Property Graph Schema Objects for Oracle Database
 The property graph PL/SQL and Java APIs use special Oracle Database schema objects.
- Data Access Layer
- Getting Started with Property Graphs
 Follow these steps to get started with property graphs.
- Using Java APIs for Property Graph Data
 Creating a property graph involves using the Java APIs to create the property graph and objects in it.
- Access Control for Property Graph Data (Graph-Level and OLS)
 Oracle Graph supports two access control and security models: graph level access control, and fine-grained security through integration with Oracle Label Security (OLS).
- SQL-Based Property Graph Query and Analytics
 You can use SQL to query property graph data in Oracle Spatial and Graph.
- Creating Property Graph Views on an RDF Graph
 With Oracle Graph, you can view RDF data as a property graph to execute graph
 analytics operations by creating property graph views over an RDF graph stored in
 Oracle Database.
- Quick Start: Interactively Analyze Graph Data Stored in Property Graph Schema Objects
 This tutorial shows how you can quickly get started using property graph data and learn
 to execute PGQL queries and run graph algorithms on the data and display results.
- Working with Property Graph Objects in SQL Developer
 You can use Oracle SQL Developer to execute PGQL statements and queries directly on
 property graph schema graphs in the database.

- Executing PGQL Queries Against Property Graph Schema Tables
 This topic explains how you can execute PGQL queries directly against the graph stored in property graph schema tables.
- OPG_APIS Package Subprograms
 The OPG_APIS package contains subprograms (functions and procedures) for working with property graphs in an Oracle database.
- OPG_GRAPHOP Package Subprograms
 The OPG_GRAPHOP package contains subprograms for various operations on property graphs in an Oracle database.

A.1 Property Graph Schema Objects for Oracle Database

The property graph PL/SQL and Java APIs use special Oracle Database schema objects.

This topic describes objects related to the property graph schema approach to working with graph data.

Oracle Spatial and Graph lets you store, query, manipulate, and query property graph data in Oracle Database. For example, to create a property graph named myGraph, you can use either the Java APIs (oracle.pg.rdbms.OraclePropertyGraph) or the PL/SQL APIs (MDSYS.OPG APIS package).

With the PL/SQL API:

With the Java API:

OraclePropertyGraph opg = OraclePropertyGraph.getInstance(cfg);

Property Graph Tables (Detailed Information)



- Default Indexes on Vertex (VT\$) and Edge (GE\$) Tables
- Flexibility in the Property Graph Schema

A.1.1 Property Graph Tables (Detailed Information)

SQL> describe myGraphVT\$

After a property graph is established in the database, several tables are created automatically in the user's schema, with the graph name as the prefix and VT\$ or GE\$ as the suffix. For example, for a graph named myGraph, table myGraphVT\$ is created to store vertices and their properties (K/V pairs), and table myGraphGE\$ is created to store edges and their properties.

Additional internal tables are created with IT\$ and GT\$ suffixes, to store text index metadata and graph skeleton (topological structure).

The definitions of tables myGraphVT\$ and myGraphGE\$ are as follows. They are important for SQL-based analytics and SQL-based property graph query. In both the VT\$ and GE\$ tables, VTS, VTE, and FE are reserved columns; column SL is for the security label; and columns K, T, V, VN, and VT together store all information about a property (K/V pair) of a graph element. In the VT\$ table, VID is a long integer for storing the vertex ID. In the GE\$ table, EID, SVID, and DVID are long integer columns for storing edge ID, source (from) vertex ID, and destination (to) vertex ID, respectively.

Name	Nul	1?	Type		
VID K T V VN VT SL VTS VTE FE	 NOT		NUMBER NVARCHAR2 (3100) NUMBER (38) NVARCHAR2 (15000) NUMBER TIMESTAMP (6) WIT NUMBER DATE DATE DATE NVARCHAR2 (4000)		ZONE
SQL> describe myGraph GE\$ Name	Nul	1?	Type		
EID SVID DVID EL K T V VN VT SL	NOT	NULI	NUMBER NUMBER NVARCHAR2 (3100) NVARCHAR2 (3100) NUMBER (38) NVARCHAR2 (15000) NUMBER TIMESTAMP (6) WI NUMBER DATE))	E ZONE



VTE DATE FE NVARCHAR2 (4000)

For simplicity, only simple graph names are allowed, and they are case insensitive.

In both the VT\$ and GE\$ tables, Columns K, T, V, VN, VT together store all information about a property (K/V pair) of a graph element, while SL is used for security label, and VTS, VTE, FE are reserved columns.

In the property graph schema design, a property value is stored in the VN column if the value has numeric data type (long, int, double, float, and so on), in the VT column if the value is a timestamp, or in the V column for Strings, boolean and other serializable data types. For better Oracle Text query support, a literal representation of the property value is saved in the V column even if the data type is numeric or timestamp. To differentiate all the supported data types, an integer ID is saved in the T column.

The K column in both VT\$ and GE\$ tables stores the property key. Each edge must have a label of String type, and the labels are stored in the EL column of the GE\$ table.

The T column in both VT\$ and GE\$ tables is a number representing the data type of the value of the property it describes. For example 1 means the value is a string, 2 means the value is an integer, and so on. Some T column possible values and associated data types are as follows:

- 1: STRING
- 2: INTEGER
- 3: FLOAT
- 4: DOUBLE
- 5: DATE
- 6: BOOLEAN
- 7: LONG
- 8: SHORT
- 9: BYTE
- 10: CHAR
- 20: Spatial data

The **VT\$ table** schema for storing vertices contains these columns:

- VID, a long column denoting the ID of the vertex.
- VL, a string column denoting the label of the vertex.
- K, a string column denoting the name of the property. If there is no property associated to the vertex, the value of this column should be a whitespace.
- T, a long column denoting the type of the property.
- V, a string column denoting the value of the property as a String. If the property type is numeric, a String format version of the value is stored in this column.
 Similarly, if the property is timestamp based, a String format version of the value is stored.



- VN, a numeric column denoting the value of a numeric property. This column stores the property value only if the property type is numeric.
- VT, a timestamp with time zone column storing the value of a date time property. This column stores the property value only if the property type is timestamp based.
- SL, a numeric column reserved for the security label set using Oracle Label Security (for further details on using Security Labels, see Access Control for Property Graph Data (Graph-Level and OLS)).
- VTS, a timestamp with time zone column reserved for future extensions.
- VTE, a timestamp with time zone column reserved for future extensions.
- FE, a string column reserved for future extensions.

The following example inserts rows into a table named CONNECTIONSVT\$. It includes T column values 1 through 10 (representing various data types).

```
INSERT INTO connectionsvt$(vid,k,t,v,vn,vt) VALUES (2001, '1-STRING', 1,
'Some String', NULL, NULL);
INSERT INTO connectionsvt$(vid,k,t,v,vn,vt) VALUES (2001, '2-INTEGER', 2,
NULL, 21, NULL);
INSERT INTO connectionsvt$(vid,k,t,v,vn,vt) VALUES (2001, '3-FLOAT', 3,
NULL, 21.5, NULL);
INSERT INTO connectionsvt$(vid,k,t,v,vn,vt) VALUES (2001, '4-DOUBLE', 4,
NULL, 21.5, NULL);
INSERT INTO connectionsvt$(vid,k,t,v,vn,vt) VALUES (2001, '5-DATE', 5, NULL,
NULL, timestamp'2018-07-20 15:32:53.991000');
INSERT INTO connectionsvt$(vid,k,t,v,vn,vt) VALUES (2001, '6-BOOLEAN', 6,
'Y', NULL, NULL);
INSERT INTO connectionsvt$(vid,k,t,v,vn,vt) VALUES (2001, '7-LONG', 7, NULL,
42, NULL);
INSERT INTO connectionsvt$(vid,k,t,v,vn,vt) VALUES (2001, '8-SHORT', 8,
NULL, 10, NULL);
INSERT INTO connectionsvt$(vid,k,t,v,vn,vt) VALUES (2001, '9-BYTE', 9, NULL,
10, NULL);
INSERT INTO connectionsvt$(vid,k,t,v,vn,vt) VALUES (2001, '10-CHAR', 10,
'A', NULL, NULL);
UPDATE connectionsVT$ SET V = coalesce(v,to nchar(vn),to nchar(vt)) WHERE
vid=2001;
COMMIT;
```

The **GE\$ table** schema for storing edges contains these columns:

- EID, a long column denoting the ID of the edge.
- SVID, a long column denoting the ID of the outgoing (origin) vertex.
- DVID, a long column denoting the ID of the incoming (destination) vertex.
- EL, a string column denoting the label of the edge.
- K, a string column denoting the name of the property. If there is no property associated to the vertex, the value of this column should be a whitespace.
- T, a long column denoting the type of the property.



- V, a string column denoting the value of the property as a String. If the property type is numeric, a String format version of the value is stored in this column.
 Similarly, if the property is timestamp based, a String format version of the value is stored.
- VN, a numeric column denoting the value of a numeric property. This column stores the property value only if the property type is numeric.
- VT, a timestamp with time zone column storing the value of a date time property. This column stores the property value only if the property type is timestamp based.
- SL, a numeric column reserved for the security label set using Oracle Label Security (for further details on using Security Labels, see Access Control for Property Graph Data (Graph-Level and OLS)).
- VTS, a timestamp with time zone column column reserved for future extensions.
- VTE, a timestamp with time zone column reserved for future extensionss.
- FE, a string column reserved for future extensions.

In addition to the VT\$ and GE\$ tables, Oracle Spatial and Graph maintains other internal tables.

An internal graph skeleton table, defined with the **GT\$ suffix**, is used to store the topological structure of a graph, and contains these columns:

- EID, a long column denoting the ID of the edge.
- EL, a string column denoting the label of the edge.
- SVID, a long column denoting the ID of the outgoing (origin) vertex.
- DVID, a long column denoting the ID of the incoming (destination) vertex.
- ELH, a raw column specifying the hash value of an edge label.
- ELS, a integer column specifying the edge label size with respect to total of characters.

An internal table, defined with the **SS\$ suffix**, is created for Oracle internal use only.

A.1.2 Default Indexes on Vertex (VT\$) and Edge (GE\$) Tables

For query performance, several indexes on property graph tables are created by default. The index names follow the same convention as the table names, including using the graph name as the prefix. For example, for the property graph myGraph, the following local (partitioned) indexes are created:

- A unique index myGraphXQV\$ on myGraphVT\$ (VID, K)
- A unique index myGraphXQE\$ on myGraphGE\$ (EID, K)
- An index myGraphXSE\$ on myGraphGE\$ (SVID, DVID, EID, VN)
- An index myGraphXDE\$ on myGraphGE\$ (DVID, SVID, EID, VN)

A.1.3 Flexibility in the Property Graph Schema

The property graph schema design does not use a catalog or centralized repository of any kind. Each property graph is separately stored and managed by a schema of user's choice. A user's schema may have one or more property graphs.



This design provides considerable flexibility to users. For example:

- Users can create additional indexes as desired.
- Different property graphs can have a different set of indexes or compression options for the base tables.
- Different property graphs can have different numbers of hash partitions.
- You can even drop the XSE\$ or XDE\$ index for a property graph; however, for integrity you should keep the unique constraints.

A.2 Data Access Layer

The data access layer provides a set of Java APIs that you can use to create and drop property graphs, add and remove vertices and edges, search for vertices and edges using key-value pairs, create text indexes, and perform other manipulations.

For more information, see:

- Using Java APIs for Property Graph Data
- Property Graph Schema Objects for Oracle Database (PL/SQL and Java APIs) and OPG_APIS Package Subprograms (PL/SQL API).

A.3 Getting Started with Property Graphs

Follow these steps to get started with property graphs.

- 1. The first time you use property graphs, ensure that the software is installed and operational.
- 2. Interact with a graph using one or more of the following options:
 - Use Java APIs in your Java application. The Java APIs can also be run in the JShell Command line interface for prototype and demo purposes.
 - Run PGQL gueries:
 - In the Java application, or
 - In the Graph visualization interface, or
 - In the SQLcl client
 - Run PGQL queries and execute Java APIs in the Apache Zeppelin interpreter
- Required Privileges for Database Users

The database schema that contains the graph tables (either Property Graph schema objects or relational tables that will be directly loaded as a graph in memory) requires certain privileges.

Related Topics

Using Java APIs for Property Graph Data
 Creating a property graph involves using the Java APIs to create the property graph and objects in it.



A.3.1 Required Privileges for Database Users

The database schema that contains the graph tables (either Property Graph schema objects or relational tables that will be directly loaded as a graph in memory) requires certain privileges.

ALTER SESSION
CREATE PROCEDURE
CREATE SEQUENCE
CREATE SESSION
CREATE TABLE
CREATE TRIGGER
CREATE TYPE
CREATE VIEW

A.4 Using Java APIs for Property Graph Data

Creating a property graph involves using the Java APIs to create the property graph and objects in it.

- · Overview of the Java APIs
- · Parallel Retrieval of Graph Data
- Using an Element Filter Callback for Subgraph Extraction
- Using Optimization Flags on Reads over Property Graph Data
- Adding and Removing Attributes of a Property Graph Subgraph
- Getting Property Graph Metadata
- Merging New Data into an Existing Property Graph
- Opening and Closing a Property Graph Instance
- Creating Vertices
- Creating Edges
- Deleting Vertices and Edges
- Reading a Graph from a Database into an Embedded Graph Server (PGX)
- Specifying Labels for Vertices
- Building an In-Memory Graph
- Dropping a Property Graph
- Executing PGQL Queries

A.4.1 Overview of the Java APIs

The Java APIs that you can use for property graphs include the following:

- Oracle Graph Property Graph Java APIs
- Oracle Database Property Graph Java APIs



A.4.1.1 Oracle Graph Property Graph Java APIs

Oracle Graph property graph support provides database-specific APIs for Oracle Database.

To use the Oracle Spatial and Graph API, import the following classes into your Java program:

```
import oracle.pg.common.*;
import oracle.pg.text.*;
import oracle.pg.rdbms.*;
import oracle.pg.rdbms.pgql.*;
import oracle.pgx.config.*;
import oracle.pgx.common.types.*;
```

To compile and run your Java applications, set your classpath to include the jar files in <cli><cli><cli>+dir

For example:

```
javac -cp ".:<client-install-dir>/lib/*" Main.java
java -cp ".:<client-install-dir>/lib/*" Main
```

A.4.1.2 Oracle Database Property Graph Java APIs

The Oracle Database property graph Java APIs enable you to create and populate a property graph stored in Oracle Database.

To use these Java APIs, import the classes into your Java program. For example:

```
import oracle.pg.rdbms.*;
import java.sql.*;
```

A.4.2 Parallel Retrieval of Graph Data

The parallel property graph query provides a simple Java API to perform parallel scans on vertices (or edges). Parallel retrieval is an optimized solution taking advantage of the distribution of the data across table partitions, so each partition is queried using a separate database connection.

Parallel retrieval will produce an array where each element holds all the vertices (or edges) from a specific partition (split). The subset of shards queried will be separated by the given start split ID and the size of the connections array provided. This way, the subset will consider splits in the range of [start, start - 1 + size of connections array]. Note that an integer ID (in the range of [0, N - 1]) is assigned to all the splits in the vertex table with N splits.

The following code loads a property graph, opens an array of connections, and executes a parallel query to retrieve all vertices and edges using the opened connections. The number of calls to the <code>getVerticesPartitioned</code> (<code>getEdgesPartitioned</code>) method is controlled by the total number of splits and the number of connections used.

```
OraclePropertyGraph opg = OraclePropertyGraph.getInstance(args, szGraphName);
// Clear existing vertices/edges in the property graph
opg.clearRepository();
```



```
String szOPVFile = "../../data/connections.opv";
String szOPEFile = "../../data/connections.ope";
// This object will handle parallel data loading
OraclePropertyGraphDataLoader opgdl =
OraclePropertyGraphDataLoader.getInstance();
opgdl.loadData(opg, szOPVFile, szOPEFile, dop);
// Create connections used in parallel query
Oracle[] oracleConns = new Oracle[dop];
Connection[] conns = new Connection[dop];
for (int i = 0; i < dop; i++) {
  oracleConns[i] = opg.getOracle().clone();
  conns[i] = oracleConns[i].getConnection();
long lCountV = 0;
// Iterate over all the vertices' partitionIDs to count all the
for (int partitionID = 0; partitionID <</pre>
opg.getVertexPartitionsNumber();
     partitionID += dop) {
  Iterable<Vertex>[] iterables
        = opg.getVerticesPartitioned(conns /* Connection array */,
                                     true /* skip store to cache */,
                                     partitionID /* starting partition
*/);
  lCountV += consumeIterables(iterables); /* consume iterables using
                                              threads */
// Count all vertices
System.out.println("Vertices found using parallel query: " + lCountV);
long lCountE = 0;
// Iterate over all the edges' partitionIDs to count all the edges
for (int partitionID = 0; partitionID <</pre>
opg.getEdgeTablePartitionIDs();
     partitionID += dop) {
  Iterable<Edge>[] iterables
          = opq.getEdgesPartitioned(conns /* Connection array */,
                                    true /* skip store to cache */,
                                    partitionID /* starting
partitionID */);
  1CountE += consumeIterables(iterables); /* consume iterables using
                                              threads */
// Count all edges
System.out.println("Edges found using parallel query: " + 1CountE);
// Close the connections to the database after completed
for (int idx = 0; idx < conns.length; idx++) {
   conns[idx].close();
}
```

A.4.3 Using an Element Filter Callback for Subgraph Extraction

Oracle Spatial and Graph provides support for an easy subgraph extraction using user-defined element filter callbacks. An element filter callback defines a set of conditions that a vertex (or an edge) must meet in order to keep it in the subgraph. Users can define their own element filtering by implementing the <code>VertexFilterCallback</code> and <code>EdgeFilterCallback</code> API interfaces.

The following code fragment implements a <code>VertexFilterCallback</code> that validates if a vertex does not have a political role and its origin is the United States.

```
* VertexFilterCallback to retrieve a vertex from the United States
* that does not have a political role
private static class NonPoliticianFilterCallback
implements VertexFilterCallback
@Override
public boolean keepVertex(OracleVertexBase vertex)
String country = vertex.getProperty("country");
String role = vertex.getProperty("role");
if (country != null && country.equals("United States")) {
if (role == null || !role.toLowerCase().contains("political")) {
return true;
}
return false;
public static NonPoliticianFilterCallback getInstance()
return new NonPoliticianFilterCallback();
}
```

The following code fragment implements an EdgeFilterCallback that uses the VertexFilterCallback to keep only edges connected to the given input vertex, and whose connections are not politicians and come from the United States.

```
/**
  * EdgeFilterCallback to retrieve all edges connected to an input
  * vertex with "collaborates" label, and whose vertex is from the
  * United States with a role different than political
  */
private static class CollaboratorsFilterCallback
implements EdgeFilterCallback
{
  private VertexFilterCallback m_vfc;
  private Vertex m_startV;

public CollaboratorsFilterCallback (VertexFilterCallback vfc,
  Vertex v)
{
  m_vfc = vfc;
  m startV = v;
```



```
}
@Override
public boolean keepEdge(OracleEdgeBase edge)
if ("collaborates".equals(edge.getLabel())) {
if (edge.getVertex(Direction.IN).equals(m startV) &&
m vfc.keepVertex((OracleVertex)
edge.getVertex(Direction.OUT))) {
return true;
else if (edge.getVertex(Direction.OUT).equals(m startV) &&
m vfc.keepVertex((OracleVertex)
edge.getVertex(Direction.IN))) {
return true;
}
return false;
}
public static CollaboratorsFilterCallback
getInstance(VertexFilterCallback vfc, Vertex v)
return new CollaboratorsFilterCallback(vfc, v);
}
```

Using the filter callbacks previously defined, the following code fragment loads a property graph, creates an instance of the filter callbacks and later gets all of Robert Smith's collaborators who are not politicians and come from the United States.

```
OraclePropertyGraph opg = OraclePropertyGraph.getInstance(
args, szGraphName);
// Clear existing vertices/edges in the property graph
opg.clearRepository();
String szOPVFile = "../../data/connections.opv";
String szOPEFile = "../../data/connections.ope";
// This object will handle parallel data loading
OraclePropertyGraphDataLoader opgdl =
OraclePropertyGraphDataLoader.getInstance();
opgdl.loadData(opg, szOPVFile, szOPEFile, dop);
// VertexFilterCallback to retrieve all people from the United States // who are
not politicians
NonPoliticianFilterCallback npvfc = NonPoliticianFilterCallback.getInstance();
// Initial vertex: Robert Smith
Vertex v = opg.getVertices("name", "Robert Smith").iterator().next();
// EdgeFilterCallback to retrieve all collaborators of Robert Smith
// from the United States who are not politicians
CollaboratorsFilterCallback cefc =
CollaboratorsFilterCallback.getInstance(npvfc, v);
Iterable<<Edge> smithCollabs = opg.getEdges((String[])null /* Match any
of the properties */,
```



```
cefc /* Match the
EdgeFilterCallback */
Iterator<<Edge> iter = smithCollabs.iterator();
System.out.println("\n\n-----Collaborators of Robert Smith from " +
 " the US and non-politician\n');
long countV = 0;
while (iter.hasNext()) {
Edge edge = iter.next(); // get the edge
// check if smith is the IN vertex
if (edge.getVertex(Direction.IN).equals(v)) {
System.out.println(edge.getVertex(Direction.OUT) + "(Edge ID: " +
edge.getId() + ")"); // get out vertex
else {
System.out.println(edge.getVertex(Direction.IN)+ "(Edge ID: " +
edge.getId() + ")"); // get in vertex
countV++;
}
```

By default, all reading operations such as get all vertices, get all edges (and parallel approaches) will use the filter callbacks associated with the property graph using the methods opg.setVertexFilterCallback(vfc) and opg.setEdgeFilterCallback(efc). If there is no filter callback set, then all the vertices (or edges) and edges will be retrieved.

The following code fragment uses the default edge filter callback set on the property graph to retrieve the edges.

```
// VertexFilterCallback to retrieve all people from the United States // who are not
politicians
NonPoliticianFilterCallback npvfc = NonPoliticianFilterCallback.getInstance();
// Initial vertex: Robert Smith
Vertex v = opg.getVertices("name", "Robert Smith").iterator().next();
// EdgeFilterCallback to retrieve all collaborators of Robert Smith
// from the United States who are not politicians
{\tt CollaboratorsFilterCallback.getInstance} \ ({\tt npvfc,\ v)}\ ;
opg.setEdgeFilterCallback(cefc);
Iterable<Edge> smithCollabs = opg.getEdges();
Iterator<Edge> iter = smithCollabs.iterator();
System.out.println("\n\n-----Collaborators of Robert Smith from " +
" the US and non-politician\n\n");
long countV = 0;
while (iter.hasNext()) {
Edge edge = iter.next(); // get the edge
// check if smith is the IN vertex
if (edge.getVertex(Direction.IN).equals(v)) {
System.out.println(edge.getVertex(Direction.OUT) + "(Edge ID: " +
edge.getId() + ")"); // get out vertex
}
System.out.println(edge.getVertex(Direction.IN)+ "(Edge ID: " +
edge.getId() + ")"); // get in vertex
}
```

```
countV++;
}
```

A.4.4 Using Optimization Flags on Reads over Property Graph Data

Oracle Spatial and Graph provides support for optimization flags to improve graph iteration performance. Optimization flags allow processing vertices (or edges) as objects with none or minimal information, such as ID, label, and/or incoming/outgoing vertices. This way, the time required to process each vertex (or edge) during iteration is reduced.

The following table shows the optimization flags available when processing vertices (or edges) in a property graph.

Optimization Flag	Description
DO_NOT_CREATE_OBJ ECT	Use a predefined constant object when processing vertices or edges.
JUST_EDGE_ID	Construct edge objects with ID only when processing edges.
JUST_LABEL_EDGE_ID	Construct edge objects with ID and label only when processing edges.
JUST_LABEL_VERTEX _EDGE_ID	Construct edge objects with ID, label, and in/out vertex IDs only when processing edges
JUST_VERTEX_EDGE_ ID	Construct edge objects with just ID and in/out vertex IDs when processing edges.
JUST_VERTEX_ID	Construct vertex objects with ID only when processing vertices.

The following code fragment uses a set of optimization flags to retrieve only all the IDs from the vertices and edges in the property graph. The objects retrieved by reading all vertices and edges will include only the IDs and no Key/Value properties or additional information.

```
import oracle.pg.common.OraclePropertyGraphBase.OptimizationFlag;
OraclePropertyGraph opg = OraclePropertyGraph.getInstance(
args, szGraphName);
// Clear existing vertices/edges in the property graph
opg.clearRepository();
String szOPVFile = "../../data/connections.opv";
String szOPEFile = "../../data/connections.ope";
// This object will handle parallel data loading
OraclePropertyGraphDataLoader opgdl =
OraclePropertyGraphDataLoader.getInstance();
opgdl.loadData(opg, szOPVFile, szOPEFile, dop);
// Optimization flag to retrieve only vertices IDs
OptimizationFlaq optFlaqVertex = OptimizationFlaq.JUST VERTEX ID;
// Optimization flag to retrieve only edges IDs
OptimizationFlag optFlagEdge = OptimizationFlag.JUST EDGE ID;
// Print all vertices
Iterator<Vertex> vertices =
opg.getVertices((String[])null /* Match any of the
```



```
properties */,
null /* Match the VertexFilterCallback */,
optFlagVertex /* optimization flag */
).iterator();
System.out.println("---- Vertices IDs----");
long vCount = 0;
while (vertices.hasNext()) {
OracleVertex v = vertices.next();
System.out.println((Long) v.getId());
vCount++;
System.out.println("Vertices found: " + vCount);
// Print all edges
Iterator<Edge> edges =
opg.getEdges((String[])null /* Match any of the properties */,
null /* Match the EdgeFilterCallback */,
optFlagEdge /* optimization flag */
).iterator();
System.out.println("---- Edges ----");
long eCount = 0;
while (edges.hasNext()) {
Edge e = edges.next();
System.out.println((Long) e.getId());
eCount++;
System.out.println("Edges found: " + eCount);
```

By default, all reading operations such as get all vertices, get all edges (and parallel approaches) will use the optimization flag associated with the property graph using the method opg.setDefaultVertexOptFlag(optFlagVertex) and opg.setDefaultEdgeOptFlag(optFlagEdge). If the optimization flags for processing vertices and edges are not defined, then all the information about the vertices and edges will be retrieved.

The following code fragment uses the default optimization flags set on the property graph to retrieve only all the IDs from its vertices and edges.

```
import oracle.pg.common.OraclePropertyGraphBase.OptimizationFlag;

// Optimization flag to retrieve only vertices IDs
OptimizationFlag optFlagVertex = OptimizationFlag.JUST_VERTEX_ID;

// Optimization flag to retrieve only edges IDs
OptimizationFlag optFlagEdge = OptimizationFlag.JUST_EDGE_ID;

opg.setDefaultVertexOptFlag(optFlagVertex);
opg.setDefaultEdgeOptFlag(optFlagEdge);

Iterator<Vertex> vertices = opg.getVertices().iterator();
System.out.println("----- Vertices IDs----");
long vCount = 0;
while (vertices.hasNext()) {
OracleVertex v = vertices.next();
System.out.println((Long) v.getId());
vCount++;
}
System.out.println("Vertices found: " + vCount);
```

```
// Print all edges
Iterator<Edge> edges = opg.getEdges().iterator();
System.out.println("----- Edges ----");
long eCount = 0;
while (edges.hasNext()) {
Edge e = edges.next();
System.out.println((Long) e.getId());
eCount++;
}
System.out.println("Edges found: " + eCount);
```

A.4.5 Adding and Removing Attributes of a Property Graph Subgraph

Oracle Spatial and Graph supports updating attributes (key/value pairs) to a subgraph of vertices and/or edges by using a user-customized operation callback. An operation callback defines a set of conditions that a vertex (or an edge) must meet in order to update it (either add or remove the given attribute and value).

You can define your own attribute operations by implementing the <code>VertexOpCallback</code> and <code>EdgeOpCallback</code> API interfaces. You must override the <code>needOp</code> method, which defines the conditions to be satisfied by the vertices (or edges) to be included in the update operation, as well as the <code>getAttributeKeyName</code> and <code>getAttributeKeyValue</code> methods, which return the key name and value, respectively, to be used when updating the elements.

The following code fragment implements a <code>VertexOpCallback</code> that operates over the <code>smithCollaborator</code> attribute associated only with Robert Smith collaborators. The value of this property is specified based on the role of the collaborators.

```
private static class CollaboratorsVertexOpCallback
implements VertexOpCallback
private OracleVertexBase m smith;
private List<Vertex> m smithCollaborators;
public CollaboratorsVertexOpCallback(OraclePropertyGraph opg)
// Get a list of Robert Smith'sCollaborators
m smith = (OracleVertexBase) opg.getVertices("name",
"Robert Smith")
.iterator().next();
Iterable<Vertex> iter = m_smith.getVertices(Direction.BOTH,
"collaborates");
m smithCollaborators = OraclePropertyGraphUtils.listify(iter);
public static CollaboratorsVertexOpCallback
getInstance(OraclePropertyGraph opg)
return new CollaboratorsVertexOpCallback(opg);
 * Add attribute if and only if the vertex is a collaborator of Robert
 * Smith
*/
```



```
@Override
public boolean needOp(OracleVertexBase v)
return m smithCollaborators != null &&
m smithCollaborators.contains(v);
@Override
public String getAttributeKeyName(OracleVertexBase v)
return "smithCollaborator";
 * Define the property's value based on the vertex role
@Override
public Object getAttributeKeyValue(OracleVertexBase v)
String role = v.getProperty("role");
role = role.toLowerCase();
if (role.contains("political")) {
return "political";
else if (role.contains("actor") || role.contains("singer") ||
role.contains("actress") || role.contains("writer") ||
role.contains("producer") || role.contains("director")) {
return "arts";
else if (role.contains("player")) {
return "sports";
else if (role.contains("journalist")) {
return "journalism";
else if (role.contains("business") || role.contains("economist")) {
return "business";
}
else if (role.contains("philanthropist")) {
return "philanthropy";
}
return " ";
}
}
```

The following code fragment implements an EdgeOpCallback that operates over the smithFeud attribute associated only with Robert Smith feuds. The value of this property is specified based on the role of the collaborators.



```
Iterable<Vertex> iter = m smith.getVertices(Direction.BOTH,
m smithFeuds = OraclePropertyGraphUtils.listify(iter);
public static FeudsEdgeOpCallback getInstance(OraclePropertyGraph opg)
return new FeudsEdgeOpCallback(opg);
 * Add attribute if and only if the edge is in the list of Robert Smith's
@Override
public boolean needOp(OracleEdgeBase e)
return m smithFeuds != null && m smithFeuds.contains(e);
@Override
public String getAttributeKeyName(OracleEdgeBase e)
return "smithFeud";
}
* Define the property's value based on the in/out vertex role
 * /
@Override
public Object getAttributeKeyValue(OracleEdgeBase e)
OracleVertexBase v = (OracleVertexBase) e.getVertex(Direction.IN);
if (m smith.equals(v)) {
v = (OracleVertexBase) e.getVertex(Direction.OUT);
String role = v.getProperty("role");
role = role.toLowerCase();
if (role.contains("political")) {
return "political";
else if (role.contains("actor") || role.contains("singer") ||
role.contains("actress") || role.contains("writer") ||
role.contains("producer") || role.contains("director")) {
return "arts";
else if (role.contains("journalist")) {
return "journalism";
else if (role.contains("player")) {
return "sports";
else if (role.contains("business") || role.contains("economist")) {
return "business";
else if (role.contains("philanthropist")) {
return "philanthropy";
return " ";
```

}

Using the operations callbacks defined previously, the following code fragment loads a property graph, creates an instance of the operation callbacks, and later adds the attributes into the pertinent vertices and edges using the addAttributeToAllVertices and addAttributeToAllEdges methods in OraclePropertyGraph.

```
OraclePropertyGraph opg = OraclePropertyGraph.getInstance(
args, szGraphName);
// Clear existing vertices/edges in the property graph
opg.clearRepository();
String szOPVFile = "../../data/connections.opv";
String szOPEFile = "../../data/connections.ope";
// This object will handle parallel data loading
OraclePropertyGraphDataLoader opgdl = OraclePropertyGraphDataLoader.getInstance();
opgdl.loadData(opg, szOPVFile, szOPEFile, dop);
// Create the vertex operation callback
CollaboratorsVertexOpCallback.getInstance(opg);
// Add attribute to all people collaborating with Smith based on their role
opg.addAttributeToAllVertices(cvoc, true /** Skip store to Cache */, dop);
// Look up for all collaborators of Smith
Iterable<Vertex> collaborators = opg.getVertices("smithCollaborator", "political");
System.out.println("Political collaborators of Robert Smith " +
getVerticesAsString(collaborators));
collaborators = opq.getVertices("smithCollaborator", "business");
System.out.println("Business collaborators of Robert Smith " +
getVerticesAsString(collaborators));
// Add an attribute to all people having a feud with Robert Smith to set
// the type of relation they have
FeudsEdgeOpCallback feoc = FeudsEdgeOpCallback.getInstance(opg);
opg.addAttributeToAllEdges(feoc, true /** Skip store to Cache */, dop);
// Look up for all feuds of Smith
Iterable<Edge> feuds = opg.getEdges("smithFeud", "political");
System.out.println("\n\nPolitical feuds of Robert Smith " + getEdgesAsString(feuds));
feuds = opg.getEdges("smithFeud", "business");
System.out.println("Business feuds of Robert Smith " +
getEdgesAsString(feuds));
```

The following code fragment defines an implementation of <code>VertexOpCallback</code> that can be used to remove vertices having value philanthropy for attribute <code>smithCollaborator</code>, then call the API <code>removeAttributeFromAllVertices</code>; It also defines an implementation of <code>EdgeOpCallback</code> that can be used to remove edges having value business for attribute <code>smithFeud</code>, then call the API <code>removeAttributeFromAllEdges</code>.

```
System.out.println("\n\nRemove 'smithCollaborator' property from all the" +
   "philanthropy collaborators");
PhilanthropyCollaboratorsVertexOpCallback pvoc =
PhilanthropyCollaboratorsVertexOpCallback.getInstance();
```



```
opg.removeAttributeFromAllVertices(pvoc);
System.out.println("\n\nRemove 'smithFeud' property from all the" + "business
BusinessFeudsEdgeOpCallback beoc = BusinessFeudsEdgeOpCallback.getInstance();
opg.removeAttributeFromAllEdges(beoc);
/**
 ^{\star} Implementation of a EdgeOpCallback to remove the "smithCollaborators"
 ^{\star} property from all people collaborating with Robert Smith that have a
 * philanthropy role
private static class PhilanthropyCollaboratorsVertexOpCallback implements
VertexOpCallback
 public static PhilanthropyCollaboratorsVertexOpCallback getInstance()
     return new PhilanthropyCollaboratorsVertexOpCallback();
  }
  /**
  ^{\star} Remove attribute if and only if the property value for
   * smithCollaborator is Philanthropy
  */
  @Override
  public boolean needOp(OracleVertexBase v)
    String type = v.getProperty("smithCollaborator");
   return type != null && type.equals("philanthropy");
  @Override
  public String getAttributeKeyName(OracleVertexBase v)
   return "smithCollaborator";
   * Define the property's value. In this case can be empty
  @Override
  public Object getAttributeKeyValue(OracleVertexBase v)
    return " ";
}
 * Implementation of a EdgeOpCallback to remove the "smithFeud" property
 * from all connections in a feud with Robert Smith that have a business role
private static class BusinessFeudsEdgeOpCallback implements EdgeOpCallback
  public static BusinessFeudsEdgeOpCallback getInstance()
    return new BusinessFeudsEdgeOpCallback();
   * Remove attribute if and only if the property value for smithFeud is
```



```
* business
*/
@Override
public boolean needOp(OracleEdgeBase e)
{
   String type = e.getProperty("smithFeud");
   return type != null && type.equals("business");
}

@Override
public String getAttributeKeyName(OracleEdgeBase e)
{
   return "smithFeud";
}

/**
   * Define the property's value. In this case can be empty
   */
@Override
public Object getAttributeKeyValue(OracleEdgeBase e)
{
   return " ";
}
```

A.4.6 Getting Property Graph Metadata

You can get graph metadata and statistics, such as all graph names in the database; for each graph, getting the minimum/maximum vertex ID, the minimum/maximum edge ID, vertex property names, edge property names, number of splits in graph vertex, and the edge table that supports parallel table scans.

The following code fragment gets the metadata and statistics of the existing property graphs stored in an Oracle database.

```
// Get all graph names in the database
List<String> graphNames = OraclePropertyGraphUtils.getGraphNames(dbArgs);
for (String graphName : graphNames) {
OraclePropertyGraph opg = OraclePropertyGraph.getInstance(args,
graphName);
System.err.println("\n Graph name: " + graphName);
System.err.println(" Total vertices: " +
opg.countVertices(dop));
System.err.println(" Minimum Vertex ID: " +
opg.getMinVertexID(dop));
System.err.println(" Maximum Vertex ID: " +
opg.getMaxVertexID(dop));
Set<String> propertyNamesV = new HashSet<String>();
opg.getVertexPropertyNames(dop, 0 /* timeout,0 no timeout */,
propertyNamesV);
System.err.println(" Vertices property names: " +
getPropertyNamesAsString(propertyNamesV));
System.err.println("\n\n Total edges: " + opg.countEdges(dop));
System.err.println(" Minimum Edge ID: " + opg.getMinEdgeID(dop));
```



```
System.err.println(" Maximum Edge ID: " + opg.getMaxEdgeID(dop));
Set<String> propertyNamesE = new HashSet<String>();
opg.getEdgePropertyNames(dop, 0 /* timeout, 0 no timeout */,
propertyNamesE);
System.err.println(" Edge property names: " +
getPropertyNamesAsString(propertyNamesE));
System.err.println("\n\n Table Information: ");
System.err.println("Vertex table number of splits: " +
(opg.getVertexPartitionsNumber()));
System.err.println("Edge table number of splits: " +
(opg.getEdgePartitionsNumber()));
```

A.4.7 Merging New Data into an Existing Property Graph

In addition to loading graph data into an empty property graph in Oracle Database, you can merge new graph data into an existing (empty or non-empty) graph. As with data loading, data merging splits the input vertices and edges into multiple chunks and merges them with the existing graph in database in parallel.

When doing the merging, the flows are different depends on whether there is an overlap between new graph data and existing graph data. *Overlap* here means that the same key of a graph element may have different values in the new and existing graph data. For example, key <code>weight</code> of the vertex with ID 1 may have value 0.8 in the new graph data and value 0.5 in the existing graph data. In this case, you must specify whether the new value or the existing value should be used for the key.

The following options are available for graph data merging: JDB-based, external table-based, and SQL loader-based merging.

- JDBC-Based Graph Data Merging
- External Table-Based Data Merging
- SQL Loader-Based Data Merging

JDBC-Based Graph Data Merging

JDBC-based data merging uses Java Database Connectivity (JDBC) APIs to load the new graph data into Oracle Database and then merge the new graph data into an existing graph.

The following example merges the new graph data from vertex and edge files szOPVFile and szOPEFile in Oracle-defined Flat-file format with an existing graph named opg, using a JDBC-based data merging with a DOP (degree of parallelism) of 48, batch size of 1000, and specified data merging options.



```
"pdml=t, pddl=t, no_dup=t, use_new_val_for_dup_key=t" /*Merge
options*/);
```

To optimize the performance of the data merging operations, a set of flags and hints can be specified in the merging options parameter when calling the JDBC-based data merging. These hints include:

- **DOP:** The degree of parallelism to use when merging the data. This parameter determines the number of chunks to generate when splitting the file, as well as the number of loader threads to use when merging the data into the property graph VT\$ and GE\$ tables.
- Batch Size: An integer specifying the batch size to use for Oracle JDBC statements in batching mode.
- Rebuild index: If set to true, the data loader will disable all the indexes and constraints
 defined over the property graph into which the data will be loaded. After all the data is
 merged into the property graph, all the original indexes and constraints will be rebuilt and
 enabled.
- Merge options: An option (or multiple options separated by commas) to optimize the data merging operations. These options include:
 - PDML=T: enables parallel execution for DML operations for the database session used in the data loader. This hint is used to improve the performance of long-running batching jobs.
 - PDDL=T: enables parallel execution for DDL operations for the database session used in the data loader. This hint is used to improve the performance of long-running batching jobs.
 - NO_DUP=T: assumes the input new graph data does not have invalid duplicates. In a valid property graph, each vertex (or edge) can at most have one value for a given property key. In an invalid property graph, a vertex (or edge) may have two or more values for a particular key. As an example, a vertex, v, has two key/value pairs: name/"John" and name/"Johnny", and they share the same key.
 - OVERLAP=F: assumes there is no overlap between new graph data and existing graph data. That is, there is no key with multiple distinct values in the new and existing graph data.
 - USE_NEW_VAL_FOR_DUP_KEY=T: if there is overlap between new graph data and existing graph data, use the value in the new graph data; otherwise, use the value in the existing graph data.

External Table-Based Data Merging

External table-based data merging uses an external table to load new graph data into Oracle Database and then merge the new graph data into an existing graph.

External-table based data merging requires a directory object, where the files read by the external tables will be stored. This directory can be created using the following SQL*Plus statements:

```
create or replace directory tmp_dir as '/tmppath/';
grant read, write on directory tmp dir to public;
```



The following example merges the new graph data from a vertex and edge files szOPVFile and szOPEFile in Oracle flat-file format with an existing graph opg using an external table-based data merging, a DOP (degree of parallelism) of 48, and specified merging options.

SQL Loader-Based Data Merging

SQL loader-based data merging uses Oracle SQL*Loader to load the new graph data into Oracle Database and then merge the new graph data into an existing graph.

The following example merges the new graph data from a vertex and edge files szOPVFile and szOPEFile in Oracle Flat-file format with an existing graph opg using an SQL loader -based data merging with a DOP (degree of parallelism) of 48 and the specified merging options. To use the APIs, the path to the SQL*Loader needs to be specified.

```
String szUser = "username";
String szPassword = "password";
String szDbId = "db18c"; /*service name of the database*/
String szOPVFile = "../../data/connectionsNew.opv"; 0
String szOPEFile = "../../data/connectionsNew.ope";
String szSQLLoaderPath = "<YOUR ORACLE HOME>/bin/sqlldr";
OraclePropertyGraphDataLoader opgdl =
OraclePropertyGraphDataLoader.getInstance();
opgdl.mergeDataWithSqlLdr(opg, szUser, szPassword, szDbId, szOPVFile,
szOPEFile,
     48 /*DOP*/,
     true /*Use Named Pipe for splitting*/,
     szSQLLoaderPath /* SQL*Loader path: the path to bin/sqlldr */,
     true /*Rebuild index*/,
     "pdml=t, pddl=t, no dup=t, use new val for dup key=t" /*Merge
options*/);
```

A.4.8 Opening and Closing a Property Graph Instance

When describing a property graph, use these Oracle Property Graph classes to open and close the property graph instance properly:

• OraclePropertyGraph.getInstance: Opens an instance of an Oracle property graph. This method has two parameters, the connection information and the graph name.

- OraclePropertyGraph.clearRepository: Removes all vertices and edges from the property graph instance.
- OraclePropertyGraph.shutdown: Closes the graph instance.

For Oracle Database, the <code>OraclePropertyGraph.getInstance</code> method uses an Oracle instance to manage the database connection. <code>OraclePropertyGraph</code> has a set of constructors that let you set the graph name, number of hash partitions, degree of parallelism, tablespace, and options for storage (such as compression). For example:

```
import oracle.pg.rdbms.*;
Oracle oracle = new Oracle(jdbcURL, username, password);

OraclePropertyGraph opg = OraclePropertyGraph.getInstance(oracle, graphName);
opg.clearRepository();
// .

// Graph description
// .

// Close the graph instance
opg.shutdown();
```

If the in-memory analyst functions are required for an application, you should use <code>GraphConfigBuilder</code> to create a graph for Oracle Database, and instantiate <code>OraclePropertyGraph</code> with that graph name as an argument. For example, the following code snippet constructs a graph <code>config</code>, gets an <code>OraclePropertyGraph</code> instance, loads some data into that graph, and gets an in-memory analyst.

```
import oracle.pgx.config.*;
import oracle.pgx.api.*;
import oracle.pqx.common.types.*;
. . .
PqNosqlGraphConfiq cfq = GraphConfiqBuilder. forPropertyGraphRdbms ()
       .setJdbcUrl("jdbc:oracle:thin:@<hostname>:1521:<sid>")
       .setUsername("<username>").setPassword("<password>")
       .setName(szGraphName)
       .setMaxNumConnections(8)
       .addEdgeProperty("lbl", PropertyType.STRING, "lbl")
       .addEdgeProperty("weight", PropertyType.DOUBLE, "1000000")
       .build();
 OraclePropertyGraph opg = OraclePropertyGraph.getInstance(cfg);
  String szOPVFile = "../../data/connections.opv";
 String szOPEFile = "../../data/connections.ope";
  // perform a parallel data load
 OraclePropertyGraphDataLoader opgdl =
OraclePropertyGraphDataLoader.getInstance();
  opgdl.loadData(opg, szOPVFile, szOPEFile, 2 /* dop */, 1000, true,
"PDML=T, PDDL=T, NO DUP=T, ");
  PgxSession session = Pgx.createSession("session-id-1");
```

```
PgxGraph g = session.readGraphWithProperties(cfg);
Analyst analyst = session.createAnalyst();
...
```

A.4.9 Creating Vertices

To create a vertex, use these Oracle Property Graph methods:

- OraclePropertyGraph.addVertex: Adds a vertex instance to a graph.
- OracleVertex.setProperty: Assigns a key-value property to a vertex.
- OraclePropertyGraph.commit: Saves all changes to the property graph instance.

The following code fragment creates two vertices named v1 and v2, with properties for age, name, weight, height, and sex in the opg property graph instance. The v1 properties set the data types explicitly.

```
// Create vertex v1 and assign it properties as key-value pairs
Vertex v1 = opg.addVertex(11);
  v1.setProperty("age", Integer.valueOf(31));
  v1.setProperty("name", "Alice");
  v1.setProperty("weight", Float.valueOf(135.0f));
  v1.setProperty("height", Double.valueOf(64.5d));
  v1.setProperty("female", Boolean.TRUE);

Vertex v2 = opg.addVertex(21);
  v2.setProperty("age", 27);
  v2.setProperty("name", "Bob");
  v2.setProperty("weight", Float.valueOf(156.0f));
  v2.setProperty("height", Double.valueOf(69.5d));
  v2.setProperty("female", Boolean.FALSE);
```

A.4.10 Creating Edges

To create an edge, use these Oracle Property Graph methods:

- OraclePropertyGraph.addEdge: Adds an edge instance to a graph.
- OracleEdge.setProperty: Assigns a key-value property to an edge.

The following code fragment creates two vertices (v1 and v2) and one edge (e1).

```
// Add vertices v1 and v2
Vertex v1 = opg.addVertex(11);
v1.setProperty("name", "Alice");
v1.setProperty("age", 31);

Vertex v2 = opg.addVertex(21);
v2.setProperty("name", "Bob");
v2.setProperty("age", 27);

// Add edge e1
Edge e1 = opg.addEdge(11, v1, v2, "knows");
e1.setProperty("type", "friends");
```



A.4.11 Deleting Vertices and Edges

You can remove vertex and edge instances individually, or all of them simultaneously. Use these methods:

- OraclePropertyGraph.removeEdge: Removes the specified edge from the graph.
- OraclePropertyGraph.removeVertex: Removes the specified vertex from the graph.
- OraclePropertyGraph.clearRepository: Removes all vertices and edges from the property graph instance.

The following code fragment removes edge ${\tt e1}$ and vertex ${\tt v1}$ from the graph instance. The adjacent edges will also be deleted from the graph when removing a vertex. This is because every edge must have an beginning and ending vertex. After removing the beginning or ending vertex, the edge is no longer a valid edge.

```
// Remove edge e1
opg.removeEdge(e1);
// Remove vertex v1
opg.removeVertex(v1);
```

The <code>OraclePropertyGraph.clearRepository</code> method can be used to remove all contents from an <code>OraclePropertyGraph</code> instance. However, use it with care because this action cannot be reversed.

A.4.12 Reading a Graph from a Database into an Embedded Graph Server (PGX)

You can read a graph from Oracle Database into a graph server (PGX) that is embedded in the same client Java application (a single JVM). For the following example, a correct <code>java.io.tmpdir</code> setting is required.

```
int dop = 8;
                                 // need customization
Map<PgxConfig.Field, Object> confPgx = new HashMap<PgxConfig.Field,
Object>();
confPgx.put(PgxConfig.Field.ENABLE GM COMPILER, false);
confPqx.put(PqxConfig.Field.NUM WORKERS IO, dop);
confPgx.put(PgxConfig.Field.NUM WORKERS ANALYSIS, dop); // <= # of physical</pre>
confPgx.put(PgxConfig.Field.NUM WORKERS FAST TRACK ANALYSIS, 2);
confPgx.put(PgxConfig.Field.SESSION TASK TIMEOUT SECS, 0); // no timeout set
confPgx.put(PgxConfig.Field.SESSION IDLE TIMEOUT SECS, 0); // no timeout set
PgRdbmsGraphConfig cfg =
GraphConfigBuilder.forPropertyGraphRdbms().setJdbcUrl("jdbc:oracle:thin:@<you</pre>
r db host>:<db port>:<db sid>")
     .setUsername("<username>")
     .setPassword("<password>")
     .setName("<graph name>")
     .setMaxNumConnections(8)
     .setLoadEdgeLabel(false)
     .build();
```



```
OraclePropertyGraph opg = OraclePropertyGraph.getInstance(cfg);
ServerInstance localInstance = Pgx.getInstance();
localInstance.startEngine(confPgx);
PgxSession session = localInstance.createSession("session-id-1"); //
Put your session description here.

Analyst analyst = session.createAnalyst();

// The following call will trigger a read of graph data from the database
PgxGraph pgxGraph = session.readGraphWithProperties(opg.getConfig());

long triangles = analyst.countTriangles(pgxGraph, false);
System.out.println("triangles " + triangles);

// Remove edge e1
opg.removeEdge(e1);

// Remove vertex v1
opg.removeVertex(v1);
```

A.4.13 Specifying Labels for Vertices

The database and data access layer do not provide labels for vertices; however, you can treat the value of a designated vertex property as one or more labels. Such a transformation is relevant only to the in-memory analyst.

In the following example, a property "country" is specified in a call to setUseVertexPropertyValueAsLabel(), and the comma delimiter "," is specified in a call to setPropertyValueDelimiter(). These two together imply that values of the country vertex property will be treated as vertex labels separated by a comma. For example, if vertex X has a string value "US" for its country property, then its vertex label will be US; and if vertex Y has a string value "UK, CN", then it will have two labels: UK and CN.

```
GraphConfigBuilder.forPropertyGraph...
    .setName("<your_graph_name>")
    ...
    .setUseVertexPropertyValueAsLabel("country")
    .setPropertyValueDelimiter(",")
    .setLoadVertexLabels(true)
    .build();
```

A.4.14 Building an In-Memory Graph

In addition to Store the Database Password in a Keystore, you can create an inmemory graph programmatically. This can simplify development when the size of graph is small or when the content of the graph is highly dynamic. The key Java class is GraphBuilder, which can accumulate a set of vertices and edges added with the addVertex and addEdge APIs. After all changes are made, an in-memory graph instance (PgxGraph) can be created by the GraphBuilder.



The following Java code snippet illustrates a graph construction flow. Note that there are no explicit calls to addVertex, because any vertex that does not already exist will be added dynamically as its adjacent edges are created.

```
import oracle.pgx.api.*;

PgxSession session = Pgx.createSession("example");
GraphBuilder<Integer> builder = session.createGraphBuilder();

builder.addEdge(1, 2);
builder.addEdge(2, 3);
builder.addEdge(2, 4);
builder.addEdge(3, 4);
builder.addEdge(4, 2);

PgxGraph graph = builder.build();
```

To construct a graph with vertex properties, you can use setProperty against the vertex objects created.

```
PgxSession session = Pgx.createSession("example");
GraphBuilder<Integer> builder = session.createGraphBuilder();
builder.addVertex(1).setProperty("double-prop", 0.1);
builder.addVertex(2).setProperty("double-prop", 2.0);
builder.addVertex(3).setProperty("double-prop", 0.3);
builder.addVertex(4).setProperty("double-prop", 4.56789);
builder.addEdge(1, 2);
builder.addEdge(2, 3);
builder.addEdge(2, 4);
builder.addEdge(3, 4);
builder.addEdge(4, 2);

PgxGraph graph = builder.build();
```

To use long integers as vertex and edge identifiers, specify IdType.LONG when getting a new instance of GraphBuilder. For example:

```
import oracle.pgx.common.types.IdType;
GraphBuilder<Long> builder = session.createGraphBuilder(IdType.LONG);
```

During edge construction, you can directly use vertex objects that were previously created in a call to $\mathtt{addEdge}$.

```
v1 = builder.addVertex(1).setProperty("double-prop", 0.5)
v2 = builder.addVertex(2).setProperty("double-prop", 2.0)
builder.addEdge(v1, v2)
```



As with vertices, edges can have properties. The following example sets the edge label by using <code>setLabel</code>:

```
builder.addEdge(v1, v2).setProperty("edge-prop",
"edge prop 1 2").setLabel("label")
```

A.4.15 Dropping a Property Graph

To drop a property graph from the database, use the

OraclePropertyGraphUtils.dropPropertyGraph method. This method has two parameters, the connection information and the graph name. For example:

```
// Drop the graph
Oracle oracle = new Oracle(jdbcUrl, username, password);
OraclePropertyGraphUtils.dropPropertyGraph(oracle, graphName);
```

You can also drop a property graph using the PL/SQL API. For example:

```
EXECUTE opg apis.drop pg('my graph name');
```

A.4.16 Executing PGQL Queries

You can execute PGQL queries directly against Oracle Database with the PgqlStatement and PgqlPreparedStatement interfaces. See Executing PGQL Queries Against Property Graph Schema Tables for details.

A.5 Access Control for Property Graph Data (Graph-Level and OLS)

Oracle Graph supports two access control and security models: graph level access control, and fine-grained security through integration with Oracle Label Security (OLS).

- Graph-level access control relies on grant/revoke to allow/disallow users other than the owner to access a property graph.
- OLS for property graph data allows sensitivity labels to be associated with individual vertex or edge stored in a property graph.

The default control of access to property graph data stored in an Oracle Database is at the graph level: the owner of a graph can grant read, insert, delete, update and select privileges on the graph to other users.

However, for applications with stringent security requirements, you can enforce a fine-grained access control mechanism by using the Oracle Label Security option of Oracle Database. With OLS, for each query, access to specific elements (vertices or edges) is granted by comparing their labels with the user's labels. (For information about using OLS, see *Oracle Label Security Administrator's Guide*.)

With Oracle Label Security enabled, elements (vertices or edges) may not be inserted in the graph if the same elements exist in the database with a stronger sensitivity label. For example, assume that you have a vertex with a very sensitive label, such as:

```
( Vertex ID 1 {name:str:v1} "SENSITIVE" ). This actually prevents a low-
```



privileged (PUBLIC) user from updating the vertex: ($Vertex ID 1 \{name:str:v1\}\}$ "PUBLIC"). On the other hand, if a high-privileged user overwrites a vertex or an edge that had been created with a low-level security label, the newer label with higher security will be assigned to the vertex or edge, and the low-privileged user will not be able to see it anymore.

Applying Oracle Label Security (OLS) on Property Graph Data
 This topic presents an example illustrating how to apply OLS to property graph data.

A.5.1 Applying Oracle Label Security (OLS) on Property Graph Data

This topic presents an example illustrating how to apply OLS to property graph data.

Because the property graph is stored in regular relational tables, this example is no different from applying OLS on a regular relational table. The following shows how to configure and enable OLS, create a security policy with security labels, and apply it to a property graph. The code examples are very simplified, and do not necessarily reflect recommended practices regarding user names and passwords.

 As SYSDBA, create database users named userP, userP2, userS, userTS, userTS2 and pgAdmin.

```
CONNECT / as sysdba;
CREATE USER userP IDENTIFIED BY userPpass;
GRANT connect, resource, create table, create view, create any index TO
userP;
GRANT unlimited TABLESPACE to userP;
CREATE USER userP2 IDENTIFIED BY userP2pass;
GRANT connect, resource, create table, create view, create any index TO
userP2;
GRANT unlimited TABLESPACE to userP2;
CREATE USER userS IDENTIFIED BY userSpass;
GRANT connect, resource, create table, create view, create any index TO
userS;
GRANT unlimited TABLESPACE to userS;
CREATE USER userTS IDENTIFIED BY userTSpass;
GRANT connect, resource, create table, create view, create any index TO
userTS:
GRANT unlimited TABLESPACE to userTS;
CREATE USER userTS2 IDENTIFIED BY userTS2pass;
GRANT connect, resource, create table, create view, create any index TO
userTS2:
GRANT unlimited TABLESPACE to userTS2;
CREATE USER pgAdmin IDENTIFIED BY pgAdminpass;
GRANT connect, resource, create table, create view, create any index TO
GRANT unlimited TABLESPACE to pgAdmin;
```



2. As SYSDBA, configure and enable Oracle Label Security.

```
ALTER USER lbacsys IDENTIFIED BY lbacsys ACCOUNT UNLOCK; EXEC LBACSYS.CONFIGURE_OLS; EXEC LBACSYS.OLS_ENFORCEMENT.ENABLE_OLS;
```

3. As SYSTEM, grant privileges to sec admin and hr sec.

```
CONNECT system/<system-password>
GRANT connect, create any index to sec_admin IDENTIFIED BY password;
GRANT connect, create user, drop user, create role, drop any role
TO hr sec IDENTIFIED BY password;
```

As LBACSYS, create the security policy.

```
CONNECT lbacsys/<lbacsys-password>
BEGIN
SA_SYSDBA.CREATE_POLICY (
  policy_name => 'DEFENSE',
  column_name => 'SL',
  default_options => 'READ_CONTROL, LABEL_DEFAULT, HIDE');
END;
//
```

As LBACSYS, grant DEFENSE_DBA and execute to sec_admin and hr_sec users.

```
GRANT DEFENSE_DBA to sec_admin;
GRANT DEFENSE_DBA to hr_sec;

GRANT execute on SA_COMPONENTS to sec_admin;
GRANT execute on SA_USER ADMIN to hr sec;
```

6. As SEC_ADMIN, create three security levels (For simplicity, compartments and groups are omitted here.)

```
CONNECT sec_admin/<sec_admin-password>;

BEGIN

SA_COMPONENTS.CREATE_LEVEL (
   policy_name => 'DEFENSE',
   level_num => 1000,
   short_name => 'PUB',
   long_name => 'PUBLIC');

END;

/

EXECUTE

SA_COMPONENTS.CREATE_LEVEL('DEFENSE',2000,'CONF','CONFIDENTIAL');

EXECUTE

SA_COMPONENTS.CREATE_LEVEL('DEFENSE',3000,'SENS','SENSITIVE');
```



7. Create three labels.

```
EXECUTE SA_LABEL_ADMIN.CREATE_LABEL('DEFENSE',1000,'PUB');

EXECUTE SA_LABEL_ADMIN.CREATE_LABEL('DEFENSE',2000,'CONF');

EXECUTE SA_LABEL_ADMIN.CREATE_LABEL('DEFENSE',3000,'SENS');
```

8. As HR_SEC, assign labels and privileges.

```
CONNECT hr sec/<hr sec-password>;
BEGIN
SA USER ADMIN.SET USER LABELS (
 policy name => 'DEFENSE',
 user name => 'UT',
 max read label => 'SENS',
 max write label => 'SENS',
 min write label => 'CONF',
 def label => 'SENS',
 row label => 'SENS');
END;
EXECUTE SA USER ADMIN.SET USER LABELS('DEFENSE', 'userTS', 'SENS');
EXECUTE SA USER ADMIN.SET USER LABELS('DEFENSE', 'userTS2', 'SENS');
EXECUTE SA USER ADMIN.SET USER LABELS('DEFENSE', 'users', 'CONF');
EXECUTE SA USER ADMIN.SET USER LABELS ('DEFENSE', userP', 'PUB', 'PUB',
'PUB', 'PUB', 'PUB');
EXECUTE SA USER ADMIN.SET USER LABELS ('DEFENSE', 'userP2', 'PUB', 'PUB',
'PUB', 'PUB', 'PUB');
EXECUTE SA USER ADMIN.SET USER PRIVS ('DEFENSE', 'pgAdmin', 'FULL');
```

9. As SEC_ADMIN, apply the security policies to the desired property graph. Assume a property graph with the name OLSEXAMPLE with userP as the graph owner. To apply OLS security, execute the following statements.

```
CONNECT sec_admin/<password>;

EXECUTE SA_POLICY_ADMIN.APPLY_TABLE_POLICY ('DEFENSE', 'userP', 'OLSEXAMPLEVT$');

EXECUTE SA_POLICY_ADMIN.APPLY_TABLE_POLICY ('DEFENSE', 'userP', 'OLSEXAMPLEGE$');

EXECUTE SA_POLICY_ADMIN.APPLY_TABLE_POLICY ('DEFENSE', 'userP', 'OLSEXAMPLEGT$');

EXECUTE SA_POLICY_ADMIN.APPLY_TABLE_POLICY ('DEFENSE', 'userP', 'OLSEXAMPLEST$');
```

Now Oracle Label Security has sensitivity labels to be associated with individual vertices or edges stored in the property graph.

The following example shows how to create a property graph with name OLSEXAMPLE, and an example flow to demonstrate the behavior when different users with different security labels create, read, and write graph elements.

```
// Create Oracle Property Graph
String graphName = "OLSEXAMPLE";
```



```
Oracle connPub = new Oracle("jdbc:oracle:thin:@host:port:SID",
"userP", "userPpass");
OraclePropertyGraph graphPub =
OraclePropertyGraph.getInstance(connPub, graphName, 48);
// Grant access to other users
graphPub.grantAccess("userP2", "RSIUD"); // Read, Select, Insert,
Update, Delete (RSIUD)
graphPub.grantAccess("userS", "RSIUD");
graphPub.grantAccess("userTS", "RSIUD");
graphPub.grantAccess("userTS2", "RSIUD");
// Load data
OraclePropertyGraphDataLoader opgdl =
OraclePropertyGraphDataLoader.getInstance();
String vfile = "../../data/connections.opv";
String efile = "../../data/connections.ope";
graphPub.clearRepository();
opgdl.loadData(graphPub, vfile, efile, 48, 1000, true, null);
System.out.println("Vertices with user userP and PUBLIC LABEL: " +
graphPub.countVertices()); // 78
System.out.println("Vertices with user userP and PUBLIC LABEL: " +
graphPub.countEdges()); // 164
// Second user with a higher level
Oracle connTS = new Oracle("jdbc:oracle:thin:@host:port:SID",
"userTS", "userTpassS");
OraclePropertyGraph graphTS = OraclePropertyGraph.getInstance(connTS,
"USERP", graphName, 8, 48, null, null);
System.out.println("Vertices with user userTS and SENSITIVE LABEL: " +
graphTS.countVertices()); // 78
System.out.println("Vertices with user userTS and SENSITIVE LABEL: " +
graphTS.countEdges()); // 164
// Add vertices and edges with the second user
long lMaxVertexID = graphTS.getMaxVertexID();
long lMaxEdgeID = graphTS.getMaxEdgeID();
long size = 10;
System.out.println("\nAdd " + size + " vertices and edges with user
userTS and SENSITIVE LABEL\n");
for (long idx = 1; idx \le size; idx++) {
  Vertex v = graphTS.addVertex(idx + lMaxVertexID);
  v.setProperty("name", "v " + (idx + lMaxVertexID));
  Edge e = graphTS.addEdge(idx + lMaxEdgeID, v,
graphTS.getVertex(idx), "edge " + (idx + lMaxEdgeID));
graphTS.commit();
// User userP with a lower level only sees the original vertices and
edges, user userTS can see more
System.out.println("Vertices with user userP and PUBLIC LABEL: " +
graphPub.countVertices()); // 78
System.out.println("Vertices with user userP and PUBLIC LABEL: " +
graphPub.countEdges()); // 164
System.out.println("Vertices with user userTS and SENSITIVE LABEL: " +
```

```
graphTS.countVertices()); // 88
System.out.println("Vertices with user userTS and SENSITIVE LABEL: " +
graphTS.countEdges()); // 174
// Third user with a higher level
Oracle connTS2 = new Oracle("jdbc:oracle:thin:@host:port:SID", "userTS2",
"userTS2pass");
OraclePropertyGraph graphTS2 = OraclePropertyGraph.getInstance(connTS2,
"USERP", graphName, 8, 48, null, null);
System.out.println("Vertices with user userTS2 and SENSITIVE LABEL: " +
graphTS2.countVertices()); // 88
System.out.println("Vertices with user userTS2 and SENSITIVE LABEL: " +
graphTS2.countEdges()); // 174
// Fourth user with a intermediate level
Oracle connS = new Oracle("jdbc:oracle:thin:@host:port:SID", "userS",
"userSpass");
OraclePropertyGraph graphS = OraclePropertyGraph.getInstance(connS, "USERP",
graphName, 8, 48, null, null);
System.out.println("Vertices with user userS and CONFIDENTIAL LABEL: " +
graphS.countVertices()); // 78
System.out.println("Vertices with user userS and CONFIDENTIAL LABEL: " +
graphS.countEdges()); // 164
// Modify vertices with the fourth user
System.out.println("\nModify " + size + " vertices with user userS and
CONFIDENTIAL LABEL\n");
for (long idx = 1; idx <= size; idx++) {</pre>
  Vertex v = graphS.getVertex(idx);
  v.setProperty("security label", "CONFIDENTIAL");
graphS.commit();
// User userP with a lower level that userS cannot see the new vertices
// Users userS and userTS can see them
System.out.println("Vertices with user userP with property security label: "
+ OraclePropertyGraphUtils.size(graphPub.getVertices("security label",
"CONFIDENTIAL"))); // 0
System.out.println("Vertices with user userS with property security label: "
+ OraclePropertyGraphUtils.size(graphS.getVertices("security label",
"CONFIDENTIAL"))); // 10
System.out.println("Vertices with user userTS with property security label:
" + OraclePropertyGraphUtils.size(graphTS.getVertices("security label",
"CONFIDENTIAL"))); // 10
System.out.println("Vertices with user userP and PUBLIC LABEL: " +
graphPub.countVertices()); // 68
System.out.println("Vertices with user userTS and SENSITIVE LABEL: " +
graphTS.countVertices()); // 88
```

The preceding example should produce the following output.

```
Vertices with user userP and PUBLIC LABEL: 78
Vertices with user userP and PUBLIC LABEL: 164
Vertices with user userTS and SENSITIVE LABEL: 78
Vertices with user userTS and SENSITIVE LABEL: 164
```



```
Add 10 vertices and edges with user userTS and SENSITIVE LABEL

Vertices with user userP and PUBLIC LABEL: 78

Vertices with user userTS and SENSITIVE LABEL: 88

Vertices with user userTS and SENSITIVE LABEL: 174

Vertices with user userTS2 and SENSITIVE LABEL: 88

Vertices with user userTS2 and SENSITIVE LABEL: 174

Vertices with user userTS2 and SENSITIVE LABEL: 78

Vertices with user userS and CONFIDENTIAL LABEL: 78

Vertices with user userS and CONFIDENTIAL LABEL: 164

Modify 10 vertices with user userS and CONFIDENTIAL LABEL

Vertices with user userP with property security_label: 0

Vertices with user userTS with property security_label: 10

Vertices with user userTS with property security_label: 10

Vertices with user userP and PUBLIC LABEL: 68

Vertices with user userTS and SENSITIVE LABEL: 88
```

A.6 SQL-Based Property Graph Query and Analytics

You can use SQL to guery property graph data in Oracle Spatial and Graph.

For the property graph support in Oracle Spatial and Graph, all the vertices and edges data are persisted in relational form in Oracle Database. For detailed information about the Oracle Spatial and Graph property graph schema objects, see Property Graph Schema Objects for Oracle Database.

This chapter provides examples of typical graph queries implemented using SQL. The audience includes DBAs as well as application developers who understand SQL syntax and property graph schema objects.

The benefits of querying directly property graph using SQL include:

- There is no need to bring data outside Oracle Database.
- You can leverage the industry-proven SQL engine provided by Oracle Database.
- You can easily join or integrate property graph data with other data types (relational, JSON, XML, and so on).
- You can take advantage of existing Oracle SQL tuning and database management tools and user interface.

The examples assume that there is a property graph named connections in the current schema. The SQL queries and example output are for illustration purpose only, and your output may be different depending on the data in your connections graph. In some examples, the output is reformatted for readability.

- Simple Property Graph Queries
 The examples in this topic query vertices, edges, and properties of the graph.
- Text Queries on Property Graphs
 If values of a property (vertex property or edge property) contain free text, then it might help performance to create an Oracle Text index on the V column.
- Navigation and Graph Pattern Matching
 A key benefit of using a graph data model is that you can easily navigate across entities (people, movies, products, services, events, and so on) that are modeled as vertices, following links and relationships modeled as edges. In addition, graph

matching templates can be defined to do such things as detect patterns, aggregate individuals, and analyze trends.

- Navigation Options: CONNECT BY and Parallel Recursion
 The CONNECT BY clause and parallel recursion provide options for advanced navigation and querying.
- Pivot

The PIVOT clause lets you dynamically add columns to a table to create a new table.

SQL-Based Property Graph Analytics
 In addition to the analytical functions offered by the graph server (PGX), the property graph feature in Oracle Spatial and Graph supports several native, SQL-based property graph analytics.

A.6.1 Simple Property Graph Queries

The examples in this topic query vertices, edges, and properties of the graph.

Example A-1 Find a Vertex with a Specified Vertex ID

This example find the vertex with vertex ID 1 in the connections graph.

```
SQL> select vid, k, v, vn, vt
    from connectionsVT$
    where vid=1;
```

The output might be as follows:

```
1 country United States
1 name Robert Smith
1 occupation CEO of Example Corporation
```

Example A-2 Find an Edge with a Specified Edge ID

This example find the edge with edge ID 100 in the connections graph.

```
SQL> select eid,svid,dvid,k,t,v,vn,vt
    from connectionsGE$
    where eid=1000;
```

The output might be as follows:

```
1000 1 2 weight 3 1 1
```

In the preceding output, the K of the edge property is "weight" and the type ID of the value is 3, indicating a float value.

Example A-3 Perform Simple Counting

This example performs simple counting in the connections graph.



```
299

SQL> -- Get the total number of K/V pairs of all the edges
SQL> select /*+ parallel(8) */ count(1)
          from connectionsGE$;
      164

SQL> -- Get the total number of vertices
SQL> select /*+ parallel */ count(distinct vid)
          from connectionsVT$;

78

SQL> -- Get the total number of edges
SQL> select /*+ parallel */ count(distinct eid)
          from connectionsGE$;

164
```

Example A-4 Get the Set of Property Keys Used

This example gets the set of property keys used for the vertices n the connections graph.

```
SQL> select /*+ parallel */ distinct k
      from connectionsVT$;
company
show
occupation
type
team
religion
criminal charge
music genre
genre
name
role
political party
country
13 rows selected.
SQL> -- get the set of property keys used for edges
SQL> select /*+ parallel */ distinct k
       from connectionsGE$;
weight
```



Example A-5 Find Vertices with a Value

This example finds vertices with a value (of any property) that is of String type, and where and the value contains two adjacent occurrences of a, e, i, o, or u, regardless of case.n the connections graph.

```
SQL> select vid, t, k, v
    from connectionsVT$
    where t=1
        and regexp_like(v, '([aeiou])\1', 'i');

6          1 name Jordan Peele
6         1 show Key and Peele
54         1 name John Green
...
```

It is usually hard to leverage a B-Tree index for the preceding kind of query because it is difficult to know beforehand what kind of regular expression is going to be used. For the above query, you might get the following execution plan. Note that full table scan is chosen by the optimizer.

```
| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time
| Pstart| Pstop | TQ | IN-OUT| PQ Distrib |
______
_____
| 0 | SELECT STATEMENT |
                     | 15 | 795 | 28 (0) | 00:00:01
| Q1,00 | P->S | QC (RAND) |
1 | 8 | Q1,00 | PCWP |
_____
_____
Predicate Information (identified by operation id):
 ._____
 4 - filter(INTERNAL FUNCTION("V") AND REGEXP LIKE ("V",U'([aeiou])\005C1','i') AND
"T"=1 AND INTERNAL FUNCTION("K"))
Note
 - Degree of Parallelism is 2 because of table property
```

If the Oracle Database In-Memory option is available and memory is sufficient, it can help performance to place the table (full table or a set of relevant columns) in memory. One way to achieve that is as follows:

```
SQL> alter table connectionsVT$ inmemory;
Table altered.
```

Now, entering the same SQL containing the regular expression shows a plan that performs a "TABLE ACCESS INMEMORY FULL".



A.6.2 Text Queries on Property Graphs

If values of a property (vertex property or edge property) contain free text, then it might help performance to create an Oracle Text index on the V column.

Oracle Text can process text that is directly stored in the database. The text can be short strings (such as names or addresses), or it can be full-length documents. These documents can be in a variety of textual format.

The text can also be in many different languages. Oracle Text can handle any space-separated languages (including character sets such as Greek or Cyrillic). In addition, Oracle Text is able to handle the Chinese, Japanese and Korean pictographic languages)

Because the property graph feature uses NVARCHAR typed column for better support of Unicode, it is *highly recommended* that UTF8 (AL32UTF8) be used as the database character set.

To create an Oracle Text index on the vertices table (or edges table), the ALTER SESSION privilege is required. For example:

```
SQL> grant alter session to <YOUR USER SCHEMA HERE>;
```

If customization is required, also grant the EXECUTE privilege on CTX_DDL:

```
SQL> grant execute on ctx ddl to <YOUR USER SCHEMA HERE>;
```



The following shows some example statements for granting these privileges to SCOTT.

```
SQL> conn / as sysdba
Connected.
SQL> -- This is a PDB setup --
SQL> alter session set container=orcl;
Session altered.
SQL> grant execute on ctx_ddl to scott;
Grant succeeded.
SQL> grant alter session to scott;
Grant succeeded.
```

Example A-6 Create a Text Index

This example creates an Oracle Text index on the vertices table (V column) of the connections graph in the SCOTT schema. Note that the Oracle Text index created here is for all property keys, not just one or a subset of property keys. In addition, if a new property is added to the graph and the property value is of String data type, then it will automatically be included in the same text index.

The example uses the OPG_AUTO_LEXER lexer owned by MDSYS.

```
SQL> execute opg_apis.create_vertices_text_idx('scott', 'connections',
pref owner=>'MDSYS', lexer=>'OPG AUTO LEXER', dop=>2);
```

If customization is desired, you can use the ctx ddl.create preference API. For example:

```
SQL> -- The following requires access privilege to CTX_DDL
SQL> exec ctx_ddl.create_preference('SCOTT.OPG_AUTO_LEXER', 'AUTO_LEXER');
PL/SQL procedure successfully completed.
SQL> execute opg_apis.create_vertices_text_idx('scott', 'connections', pref_owner=>'scott', lexer=>'OPG_AUTO_LEXER', dop=>2);
PL/SQL procedure successfully completed.
```

You can now use a rich set of functions provided by Oracle Text to perform queries against graph elements.



Note:

If you no longer need an Oracle Text index, you can use the drop_vertices_text_idx or opg_apis.drop_edges_text_idx API to drop it. The following statements drop the text indexes on the vertices and edges of a graph named connections owned by SCOTT:

```
SQL> exec opg_apis.drop_vertices_text_Idx('scott',
'connections');
SQL> exec opg_apis.drop_edges_text_Idx('scott', 'connections');
```

Example A-7 Find a Vertex that Has a Property Value

The following example find a vertex that has a property value (of string type) containing the keyword "Smith".

```
SQL> select vid, k, t, v
    from connectionsVT$
    where t=1
        and contains(v, 'Smith', 1) > 0
    order by score(1) desc
;
```

The output and SQL execution plan from the preceding statement may appear as follows. Note that DOMAIN INDEX appears as an operation in the execution plan.

```
1 name 1 Robert Smith
Execution Plan
Plan hash value: 1619508090
| Id | Operation
                                | Name | Rows | Bytes |
Cost (%CPU) | Time | Pstart| Pstop |
______
 0 | SELECT STATEMENT
                                      | 1 | 56
                                5 (20) | 00:00:01 |
  1 | SORT ORDER BY | 5 (20) | 00:00:01 | |
| 1 | SORT ORDER BY
                                       | 1 | 56
|* 2 | TABLE ACCESS BY GLOBAL INDEX ROWID| CONNECTIONSVT$ | 1 | 56
4 (0) | 00:00:01 | ROWID | ROWID |
| * 3 | DOMAIN INDEX
                               | CONNECTIONSXTV$ |
   4 (0) | 00:00:01 | |
Predicate Information (identified by operation id):
  2 - filter("T"=1 AND INTERNAL FUNCTION("K") AND INTERNAL FUNCTION("V"))
  3 - access("CTXSYS"."CONTAINS"("V",'Smith',1)>0)
```



Example A-8 Fuzzy Match

The following example finds a vertex that has a property value (of string type) containing variants of "ameriian" (a deliberate misspelling for this example) Fuzzy match is used.

```
SQL> select vid, k, t, v
          from connectionsVT$
     where contains(v, 'fuzzy(ameriian,,,weight)', 1) > 0
     order by score(1) desc;
```

The output and SQL execution plan from the preceding statement may appear as follows.

```
1 american business man
   9 role 1 american business man
   4 role 1 american economist
           1 american comedian actor
   6 role
   7 role 1 american comedian actor
   1 occupation 1 44th president of United States of America
6 rows selected.
Execution Plan
Plan hash value: 1619508090
_____
| Id | Operation
                              | Name
                                           | Rows | Bytes | Cost
(%CPU) | Time | Pstart | Pstop |
______
0 | SELECT STATEMENT
                              1 | 56 |
1 |
                                                     56 |
|* 2 | TABLE ACCESS BY GLOBAL INDEX ROWID| CONNECTIONSVT$ |
                                                1 |
  (0) | 00:00:01 | ROWID | ROWID |
|* 3 | DOMAIN INDEX
                              | CONNECTIONSXTV$ |
  (0) | 00:00:01 |
Predicate Information (identified by operation id):
  2 - filter(INTERNAL FUNCTION("K") AND INTERNAL FUNCTION("V"))
```

Example A-9 Query Relaxation

The following example is a sophisticated Oracle Text query that implements **query relaxation**, which enables you to execute the most restrictive version of a query first, progressively relaxing the query until the required number of matches is obtained. Using query relaxation with queries that contain multiple strings, you can provide guidance for determining the "best" matches, so that these appear earlier in the results than other potential matches.



This example searchs for "american actor" with a query relaxation sequence.

```
SQL> select vid, k, t, v
     from connectionsVT$
     where CONTAINS (v,
 '<query>
  <textquery lang="ENGLISH" grammar="CONTEXT">
    cprogression>
     <seq>{american} {actor}</seq>
     <seq>{american} NEAR {actor}</seq>
     <seq>{american} AND {actor}</seq>
     <seq>{american} ACCUM {actor}</seq>
    </textquery>
  <score datatype="INTEGER" algorithm="COUNT"/>
 </query>') > 0;
The output and SQL execution plan from the preceding statement may appear as
follows.
   7 role 1 american comedian actor 6 role 1 american comedian actor
   44 occupation 1 actor
   8 role 1 american business man
   53 occupation 1 actor film producer
   52 occupation 1 actor
   4 role 1 american economist
   47 occupation 1 actor
   9 role
            1 american business man
9 rows selected.
Execution Plan
Plan hash value: 2158361449
______
_____
| Id | Operation
                               | Name | Rows | Bytes | Cost
(%CPU) | Time | Pstart | Pstop |
------
0 | SELECT STATEMENT
                                            | 1 | 56
  4 (0) | 00:00:01 |
|* 1 | TABLE ACCESS BY GLOBAL INDEX ROWID| CONNECTIONSVT$ | 1 | 56
  4 (0) | 00:00:01 | ROWID | ROWID |
  2 | DUMAIN INDEX | CONNECTIONSXTV$ | 4 (0) | 00:00:01 | | |
|* 2 | DOMAIN INDEX
______
Predicate Information (identified by operation id):
  1 - filter(INTERNAL_FUNCTION("K") AND INTERNAL FUNCTION("V"))
  2 - access("CTXSYS"."CONTAINS"("V",'<query> <textquery lang="ENGLISH"
grammar="CONTEXT">
```



Example A-10 Find an Edge

Just as with vertices, you can create an Oracle Text index on the V column of the edges table (GE\$) of a property graph. The following example uses the OPG_AUTO_LEXER lexer owned by MDSYS.

```
SQL> exec opg_apis.create_edges_text_idx('scott', 'connections',
pref_owner=>'mdsys', lexer=>'OPG_AUTO_LEXER', dop=>4);
```

If customization is required, use the ctx_ddl.create_preference API.

A.6.3 Navigation and Graph Pattern Matching

A key benefit of using a graph data model is that you can easily navigate across entities (people, movies, products, services, events, and so on) that are modeled as vertices, following links and relationships modeled as edges. In addition, graph matching templates can be defined to do such things as detect patterns, aggregate individuals, and analyze trends.

This topic provides graph navigation and pattern matching examples using the example property graph named connections. Most of the SQL statements are relatively simple, but they can be used as building blocks to implement requirements that are more sophisticated. It is generally best to start from something simple, and progressively add complexity.

Example A-11 Who Are a Person's Collaborators?

The following SQL ststement finds all entities that a vertex with ID 1 collaborates with. For simplicity, it considers **only** outgoing relationships.

Note:

To find the specific vertex ID of interest, you can perform a text query on the property graph using keywords or fuzzy matching. (For details and examples, see Text Queries on Property Graphs.)

The preceding example's output and execution plan may be as follows.

```
2 collaborates weight 1 1 21 collaborates weight 1 1 22 collaborates weight 1 1 .... 26 collaborates weight 1 1
```



10 rows selected.

```
| Name | Rows |
| Id | Operation
Bytes | Cost (%CPU) | Time | Pstart | Pstop | TQ | | IN-OUT | PQ Distrib |
0 | SELECT STATEMENT
| 460 | 2 (0) | 00:00:01 | 1 | 8 | Q1,00 | PCWC |
|* 4 | TABLE ACCESS BY LOCAL INDEX ROWID BATCHED| CONNECTIONSGE$ | 10
| 460 | 2 (0)| 00:00:01 | 1 | 8 | Q1,00 | PCWP |
|* 5 |
       INDEX RANGE SCAN
                                  | CONNECTIONSXSE$ | 20
    | 1 (0)| 00:00:01 | 1 | 8 | Q1,00 | PCWP |
______
Predicate Information (identified by operation id):
  4 - filter(INTERNAL FUNCTION("EL") AND "EL"=U'collaborates' AND
INTERNAL FUNCTION("K") AND INTERNAL_FUNCTION("V"))
  5 - access("SVID"=1)
```

Example A-12 Who Are a Person's Collaborators and What are Their Occupations?

The following SQL statement finds collaborators of the vertex with ID 1, and the occupation of each collaborator. A join with the vertices table (VT\$) is required.

```
SQL> select dvid, vertices.v
    from connectionsGE$, connectionsVT$ vertices
    where svid=1
        and el='collaborates'
        and dvid=vertices.vid
        and vertices.k='occupation';
```

The preceding example's output and execution plan may be as follows.



```
______
| 0 | SELECT STATEMENT
525 | 7 (0) | 00:00:01 | | |
1
 3 | NESTED LOOPS
    NESTED LOOPS | 7 (0) | 00:00:01 | | | Q1,00 | PCWP |
                                        7 |
525 |
    | 4 |
                                        10 |
250 I
|* 5 | TABLE ACCESS BY LOCAL INDEX ROWID BATCHED| CONNECTIONSGE$
250 | 2 (0)| 00:00:01 | 1 | 8 | Q1,00 | PCWP |
     TABLE ACCESS BY LOCAL INDEX ROWID BATCHED| CONNECTIONSGE$ |
                                        10 |
 |* 6 |
      0 (0)| 00:00:01 | KEY | KEY | Q1,00 | PCWP |
   | * 8 |
      TABLE ACCESS BY LOCAL INDEX ROWID | CONNECTIONSVT$ |
  |* 9 |
```

Predicate Information (identified by operation id):

```
5 - filter(INTERNAL_FUNCTION("EL") AND "EL"=U'collaborates')
6 - access("SVID"=1)
8 - filter(INTERNAL_FUNCTION("VERTICES"."V"))
9 - access("DVID"="VERTICES"."VID" AND "VERTICES"."K"=U'occupation')
    filter(INTERNAL FUNCTION("VERTICES"."K"))
```

Example A-13 Find a Person's Enemies and Aggregate Them by Their Country

The following SQL statement finds enemies (that is, those with the feuds relationship) of the vertex with ID 1, and aggregates them by their countries. A join with the vertices table (VT\$) is required.

```
SQL> select vertices.v, count(1)
    from connectionsGE$, connectionsVT$ vertices
    where svid=1
        and el='feuds'
        and dvid=vertices.vid
        and vertices.k='country'
    group by vertices.v;
```

The example's output and execution plan may be as follows. In this case, the vertex with ID 1 has 3 enemies in the United States and 1 in Russia.



United States

```
______
| 0 | SELECT STATEMENT
  375 | 5 (20) | 00:00:01 |
                                                          1 | PX COORDINATOR
  3 | HASH GROUP BY
   375 | 5 (20) | 00:00:01 | | 4 | PX RECEIVE
                                    | Q1,01 | PCWP |
4 | PX RECEIVE
                                                           |* 9 | TABLE ACCESS BY LOCAL INDEX ROWID BATCHED| CONNECTIONSGE$ | 5 | 125 | 2 (0)| 00:00:01 | 1 | 8 | Q1,00 | PCWP | |
|* 10 | INDEX RANGE SCAN | CONNECTIONSXSE$ | 20 | 1 (0) | 00:00:01 | 1 | 8 | Q1,00 | PCWP | | 11 | PARTITION HASH ITERATOR | 1 | 1 | 0 (0) | 00:00:01 | KEY | KEY | Q1,00 | PCWP | | 1* 12 | TABLE ACCESS BY LOCAL INDEX ROWID | CONNECTIONSVT$ |
Predicate Information (identified by operation id):
_____
  9 - filter(INTERNAL FUNCTION("EL") AND "EL"=U'feuds')
 10 - access("SVID"=1)
```

Example A-14 Find a Person's Collaborators, and aggregate and sort them

The following SQL statement finds the collaborators of the vertex with ID 1, aggregates them by their country, and sorts them in ascending order.

13 - access("DVID"="VERTICES"."VID" AND "VERTICES"."K"=U'country')

```
SQL> select vertices.v, count(1)
    from connectionsGE$, connectionsVT$ vertices
    where svid=1
        and el='collaborates'
        and dvid=vertices.vid
        and vertices.k='country'
    group by vertices.v
    order by count(1) asc;
```

12 - filter(INTERNAL FUNCTION("VERTICES"."V"))

filter(INTERNAL FUNCTION("VERTICES"."K"))

The example output and execution plan may be as follows. In this case, the vertex with ID 1 has the most collaborators in the United States.

Germany 1
Japan 1
Iran 1
United States 7

| Id | Operation Bytes | Cost (%CPU) | Time | Pstart| Pstop | TQ | IN-OUT| PQ Distrib | ______ | 750 | 9 (23) | 00:00:01 | | Q1,02 | PCWP | | | 4 | PX RECEIVE 10 750 | 9 (23) | 00:00:01 | | Q1,02 | PCWP | | 5 | PX SEND RANGE | 750 | 9 (23) | 00:00:01 | | Q1,01 | P->P | RANGE | | 6 | HASH GROUP BY | 6 | HASH GROUP B1 | 750 | 9 (23) | 00:00:01 | | Q1,01 | PCWP | 7 | PX RECEIVE 750 | 9 (23) | 00:00:01 | | Q1,01 | PCWP | | :T010000 7 | | :TQ10000 | | 8 | PX SEND HASH 10 750 | 9 (23) | 00:00:01 | | Q1,00 | P->P | HASH | 1 9 | HASH GROUP BY 10 750 | 9 (23) | 00:00:01 | | Q1,00 | PCWP | 10 | NESTED LOOPS 750 | 7 (0)| 00:00:01 | | Q1,00 | PCWP | 11 | PX PARTITION HASH ALL 10 | 0 (0)| 00:00:01 | KEY | KEY | Q1,00 | PCWP | |* 15 | TABLE ACCESS BY LOCAL INDEX ROWID | CONNECTIONSVT\$ | | CONNECTIONSXQV\$ | | 0 (0)| 00:00:01 | KEY | KEY | Q1,00 | PCWP |

Predicate Information (identified by operation id):



^{12 -} filter(INTERNAL_FUNCTION("EL") AND "EL"=U'collaborates')
13 - access("SVID"=1)

^{15 -} filter(INTERNAL FUNCTION("VERTICES"."V"))

^{16 -} access("DVID"="VERTICES"."VID" AND "VERTICES"."K"=U'country') filter(INTERNAL FUNCTION("VERTICES"."K"))

A.6.4 Navigation Options: CONNECT BY and Parallel Recursion

The CONNECT BY clause and parallel recursion provide options for advanced navigation and querying.

- CONNECT BY lets you navigate and find matches in a hierarchical order. To follow outgoing edges, you can use prior dvid = svid to guide the navigation.
- Parallel recursion lets you perform navigation up to a specified number of hops away.

The examples use a property graph named connections.

Example A-15 CONNECT WITH

The following SQL statement follows the outgoing edges by 1 hop.

```
SQL> select G.dvid
    from connectionsGE$ G
    start with svid = 1
    connect by nocycle prior dvid = svid and level <= 1;</pre>
```

The preceding example's output and execution plan may be as follows.



To extend from 1 hop to multiple hops, change 1 in the preceding example to another integer. For example, to change it to 2 hops, specify: $level \le 2$

Example A-16 Parallel Recursion

The following SQL statement uses recursion within the WITH clause to perform navigation up to 4 hops away, a using recursively defined graph expansion: g_{exp} references g_{exp} in the query, and that defines the recursion. The example also uses the PARALLEL optimizer hint for parallel execution.

```
SQL> WITH g_exp(svid, dvid, depth) as
  (
    select svid as svid, dvid as dvid, 0 as depth
        from connectionsGE$
    where svid=1
    union all
    select g2.svid, g1.dvid, g2.depth + 1
        from g_exp g2, connectionsGE$ g1
        where g2.dvid=g1.svid
        and g2.depth <= 3
    )
select /*+ parallel(4) */ dvid, depth
    from g_exp
    where svid=1
;</pre>
```

The example's output and execution plan may be as follows. Note that CURSOR DURATION MEMORY is chosen in the execution, which indicates the graph expansion stores the intermediate data in memory.

```
22 4
25 4
24 4
1 4
23 4
33 4
22 4
22 4
```

Execution Plan



```
Pstop | TQ |IN-OUT| PQ Distrib |
| 0 | SELECT STATEMENT
  1 | TEMP TABLE TRANSFORMATION
  | 2 | LOAD AS SELECT (CURSOR DURATION MEMORY) |
SYS_TEMP_0FD9D6614_11CB2D2 | | | | | | |
| 3 | UNION ALL (RECURSIVE WITH) BREADTH FIRST
| 4 | PX COORDINATOR
                  | 5 | PX SEND QC (RANDOM)
| :TQ20000 | 2 | 12 | 0 (0) | 00:00:01 |
| Q2,00 | P->S | QC (RAND) |
6 | LOAD AS SELECT (CURSOR DURATION MEMORY) |
| 7 | PX PARTITION HASH ALL
CONNECTIONSXSE$ | 2 | 12 | 0 (0)| 00:00:01 | 1 | 8 | Q2,00 | PCWP | | 9 | PX COORDINATOR
|:TQ10000 | 799 |
                    12M| 12 (0)| 00:00:01 |
| Q1,00 | P->S | QC (RAND) |
| 11 | LOAD AS SELECT (CURSOR DURATION MEMORY)|
SYS_TEMP_0FD9D6614_11CB2D2 | |
                            1
|* 12 | HASH JOIN
              | 799 | 12M| 12 (0)| 00:00:01 |
| Q1,00 | PCWP |
| 13 | BUFFER SORT (REUSE)
             1
                          1
  | Q1,00 | PCWP |
| 14 | PARTITION HASH ALL
            | 164 | 984 | 2 (0) | 00:00:01 | 1
 8 | Q1,00 | PCWC |
| 15 | INDEX FAST FULL SCAN
CONNECTIONSXDE$ | 164 | 984 | 2 (0) | 00:00:01 | 1 | 8 | Q1,00 | PCWP |
| 16 | PX BLOCK ITERATOR
|* 17 | TABLE ACCESS FULL
```



```
SYS TEMP OFD9D6614 11CB2D2 |
                       Q1,00 | PCWP | |
| 18 | PX COORDINATOR
    1
| 19 | PX SEND QC (RANDOM)
                                     | :TO30000
801 | 31239 | 135 (0) | 00:00:01 |
                             | Q3,00 | P->S | QC (RAND)
l* 20 l
       VIEW
801 | 31239 | 135 (0) | 00:00:01 |
                                      Q3,00 | PCWP |
                             | 21 | PX BLOCK ITERATOR
     12M| 135 (0)| 00:00:01|
                             | Q3,00 | PCWC |
801 I
                                    | SYS TEMP OFD9D6614 11CB2D2 |
| 22 |
     TABLE ACCESS FULL
                                   | Q3,00 | PCWP |
801 | 12M| 135 (0) | 00:00:01 |
                             ______
```

Predicate Information (identified by operation id):

```
8 - access("SVID"=1)
12 - access("G2"."DVID"="G1"."SVID")
17 - filter("G2"."INTERNAL_ITERS$"=LEVEL AND "G2"."DEPTH"<=3)
20 - filter("SVID"=1)</pre>
```

A.6.5 Pivot

The PIVOT clause lets you dynamically add columns to a table to create a new table.

The schema design (VT\$ and GE\$) of the property graph is narrow ("skinny") rather than wide ("fat"). This means that if a vertex or edge has multiple properties, those property keys, values, data types, and so on will be stored using multiple rows instead of multiple columns. Such a design is very flexible in the sense that you can add properties dynamically without having to worry about adding too many columns or even reaching the physical maximum limit of number of columns a table may have. However, for some applications you may prefer to have a wide table if the properties are somewhat homogeneous.

Example A-17 Pivot

The following CREATE TABLE ... AS SELECT statement uses PIVOT to add four columns: 'company',' occupation',' name', and 'religion'.



The following DESCRIBE statement shows the definition of the new table, including the four added columns. (The output is reformatted for readability.)

```
SQL> DESCRIBE pg wide;
                                                           Null?
Name
                                                                     Type
 VID
                                                           NOT NULL NUMBER
 Τ
NUMBER (38)
 'company'
NVARCHAR2 (15000)
 'occupation'
NVARCHAR2 (15000)
 'name'
NVARCHAR2 (15000)
 'religion'
NVARCHAR2 (15000)
```

A.6.6 SQL-Based Property Graph Analytics

In addition to the analytical functions offered by the graph server (PGX), the property graph feature in Oracle Spatial and Graph supports several native, SQL-based property graph analytics.

The benefits of SQL-based analytics are:

- Easier analysis of larger graphs that do not fit in physical memory
- Cheaper analysis since no graph data is transferred outside the database
- Better analysis using the current state of a property graph database
- Simpler analysis by eliminating the step of synchronizing an in-memory graph with the latest updates from the graph database

However, when a graph (or a subgraph) fits in memory, then running analytics provided by the graph server (PGX) usually provides better performance than using SQL-based analytics.

Because many of the analytics implementation require using intermediate data structures, most SQL- (and PL/SQL-) based analytics APIs have parameters for working tables (wt). A typical flow has the following steps:

- Prepare the working table or tables.
- 2. Perform analytics (one or multiple calls).
- 3. Perform cleanup

The following subtopics provide SQL-based examples of some popular types of property graph analytics.

- Shortest Path Examples
- Collaborative Filtering Overview and Examples



A.6.6.1 Shortest Path Examples

The following examples demonstrate SQL-based shortest path analytics.

Example A-18 Shortest Path Setup and Computation

Consider shortest path, for example. Internally, Oracle Database uses the bidirectional Dijkstra algorithm. The following code snippet shows an entire prepare, perform, and cleanup workflow.

```
set serveroutput on
DECLARE
  wt1 varchar2(100); -- intermediate working tables
  n number;
        varchar2(1000);
  path
  weights varchar2(1000);
BEGIN
  -- prepare
  opg apis.find sp prep('connectionsGE$', wt1);
  dbms output.put line('working table name ' || wt1);
  -- compute
  opg apis.find sp(
     'connectionsGE$',
      1,
                                    -- start vertex ID
      53,
                                    -- destination vertex ID
      wt1,
                                  -- working table (for Dijkstra expansion)
      dop \Rightarrow 1,
                                   -- degree of parallelism
      stats_freq=>1000, -- frequency to collect statistics
path_output => path, -- shortest path (a sequence of vertices)
      weights_output => weights, -- edge weights
      options => null
      );
  dbms output.put line('path ' || path);
  dbms output.put line('weights ' || weights);
  -- cleanup (commented out here; see text after the example)
  -- opg apis.find sp cleanup('connectionsGE$', wt1);
END;
```

This example may produce the following output. Note that if *no* working table name is provided, the preparation step will automatically generate a temporary table name and create it. Because the temporary working table name uses the session ID, your output will probably be different.

```
working table name    "CONNECTIONSGE$$TWFS12"
path    1  3    52  53
weights 4  3  1    1  1
PL/SQL procedure successfully completed.
```



If you want to know the definition of the working table or tables, then skip the cleanup phase (as shown in the preceding example that comments out the call to find_sp_cleanup). After the computation is done, you can describe the working table or tables.

SQL> describe "CO			"CONNEC	NNECTIONSGE\$\$TWFS12"			
	Name)		Null?	Туре		
	NID				NUMBER		
	D2S				NUMBER		
	P2S				NUMBER		
	D2T				NUMBER		
	P2T				NUMBER		
	F				NUMBER (38)		
	В				NUMBER (38)		

For advanced users who want to try different table creation options, such as using inmemory or advanced compression, you can pre-create the preceding working table and pass the name in.

Example A-19 Shortest Path: Create Working Table and Perform Analytics

The following statements show some advanced options, first creating a working table with the same column structure and basic compression enabled, then passing it to the SQL-based computation. The code optimizes the intermediate table for computations with CREATE TABLE compression and in-memory options.

```
create table connections$MY EXP(
NID
                                  NUMBER,
 D2S
                                  NUMBER,
 P2S
                                  NUMBER,
 D2T
                                  NUMBER,
 P2T
                                  NUMBER,
 F
                                NUMBER (38),
                                NUMBER (38)
) compress nologging;
DECLARE
 wt1 varchar2(100) := 'connections$MY EXP';
 n number;
 path
          varchar2(1000);
 weights varchar2(1000);
  dbms output.put line('working table name ' || wt1);
  -- compute
  opg apis.find sp(
     'connectionsGE$',
      1,
      53.
      wt1,
      dop \Rightarrow 1,
      stats freq=>1000,
      path output => path,
```



```
weights_output => weights,
    options => null
    );
dbms_output.put_line('path ' || path);
dbms_output.put_line('weights ' || weights);
-- cleanup
-- opg_apis.find_sp_cleanup('connectionsGE$', wt1);
END;
//
```

At the end of the computation, if the working table has not been dropped or truncated, you can check the content of the working table, as follows. Note that the working table structure may vary between releases.

SQL> select * from connections\$MY EXP;						
NID	D2S	P2S	D2T	P2T	F	В
1	0		1.000E+100		1	-1
53	1.000E+100		0		-1	1
54	1.000E+100		1	53	-1	1
52	1.000E+100		1	53	-1	1
5	1	1	1.000E+100		0	-1
26	1	1	1.000E+100		0	-1
8	1000	1	1.000E+100		0	-1
3	1	1	2	52	0	0
15	1	1	1.000E+100		0	-1
21	1	1	1.000E+100		0	-1
19	1	1	1.000E+100		0	-1

Example A-20 Shortest Path: Perform Multiple Calls to Same Graph

To perform multiple calls to the same graph, only a *single call* to the preparation step is needed. The following shows an example of computing shortest path for multiple pairs of vertices in the same graph.

```
DECLARE
 wt1 varchar2(100); -- intermediate working tables
 n number;
 path
       varchar2(1000);
 weights varchar2(1000);
BEGIN
 -- prepare
 opg apis.find sp prep('connectionsGE$', wt1);
 dbms output.put line('working table name ' || wt1);
  -- find shortest path from vertex 1 to vertex 53
  opg apis.find sp( 'connectionsGE$', 1, 53,
      wt1, dop => 1, stats freq=>1000, path output => path, weights output
=> weights, options => null);
 dbms output.put line('path
                             ' || path);
  dbms output.put line('weights ' || weights);
```



```
-- find shortest path from vertex 2 to vertex 36
  opg apis.find sp( 'connectionsGE$', 2, 36,
      wt1, dop => 1, stats freq=>1000, path output => path,
weights output => weights, options => null);
  dbms output.put line('path
                              ' || path);
  dbms output.put line('weights ' || weights);
  -- find shortest path from vertex 30 to vertex 4
  opg apis.find sp( 'connectionsGE$', 30, 4,
      wt1, dop => 1, stats freq=>1000, path output => path,
weights output => weights, options => null);
  dbms output.put line('path
                              ' || path);
  dbms output.put line('weights ' || weights);
  -- cleanup
 opg apis.find sp cleanup('connectionsGE$', wt1);
END;
```

The example's output may be as follows: three shortest paths have been found for the multiple pairs of vertices provided.

```
working table name "CONNECTIONSGE$$TWFS12"
path 1 3 52 53
weights 4 3 1 1 1
path 2 36
weights 2 1 1
path 30 21 1 4
weights 4 3 1 1 1

PL/SQL procedure successfully completed.
```

A.6.6.2 Collaborative Filtering Overview and Examples

Collaborative filtering, also referred to as social filtering, filters information by using the recommendations of other people. Collaborative filtering is widely used in systems that recommend purchases based on purchases by others with similar preferences.

The following examples demonstrate SQL-based collaborative filtering analytics.

Example A-21 Collaborative Filtering Setup and Computation

This example shows how to use SQL-based collaborative filtering, specifically using matrix factorization to recommend telephone brands to customers. This example assumes there exists a graph called "PHONES" in the database. This example graph contains customer and item vertices, and edges with a 'rating' label linking some customer vertices to other some item vertices. The rating labels have a numeric value corresponding to the rating that a specific customer (edge OUT vertex) assigned to the specified product (edge IN vertex).

The following figure shows this graph.



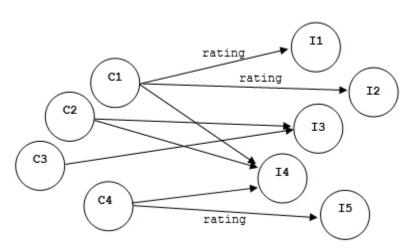


Figure A-1 Phones Graph for Collaborative Filtering

```
set serveroutput on
DECLARE
 wt l varchar2(32); -- working tables
 wt r varchar2(32);
 wt 11 varchar2(32);
 wt r1 varchar2(32);
 wt i varchar2(32);
 wt ld varchar2(32);
 wt rd varchar2(32);
  edge tab name varchar2(32) := 'phonesge$';
  edge label varchar2(32) := 'rating';
  rating property varchar2(32) := '';
  iterations integer
                            := 100;
                              := 0.001;
 min error
                 number
                 integer
                             := 5;
 k
 learning_rate     number
                              := 0.001;
  decrease rate number
                              := 0.95;
  regularization number
                              := 0.02;
                number
number
 dop
                              := 2;
 dop
tablespace
                varchar2(32) := null;
  options
                  varchar2(32) := null;
BEGIN
  -- prepare
  opg_apis.cf_prep(edge_tab_name,wt_l,wt_r,wt_l1,wt_r1,wt_i,wt_ld,wt_rd);
  dbms output.put line('working table wt l ' || wt l);
  dbms_output.put_line('working table wt_r ' || wt r);
  dbms output.put line('working table wt l1 ' || wt l1);
  dbms output.put line('working table wt r1 ' || wt r1);
  dbms output.put line('working table wt i ' | | wt i);
  dbms output.put line('working table wt ld ' || wt ld);
  dbms output.put line('working table wt rd ' || wt rd);
  -- compute
  opg apis.cf(edge tab name, edge label, rating property, iterations,
```

Example A-22 Collaborative Filtering: Validating the Intermediate Error

At the end of every computation, you can check the current error of the algorithm with the following query as long as the data in the working tables has not been already deleted. The following SQL query illustrates how to get the intermediate error of a current run of the collaborative filtering algorithm.

Note that the regularization parameter and the working table name (parameter wt_i) should be replaced according to the values used when running the OPG_APIS.CF algorithm. In the preceding previous example, replace <regularization> with 0.02 and <wt i> with "PHONESGE\$\$CFI149" as follows:

```
SELECT /*+ parallel(48) */ SQRT(SUM((w1-w2)*(w1-w2) + 0.02/2 *
(err_reg_l+err_reg_r))) AS err
FROM "PHONESGE$$CFI149";
```

This query may produce the following output.

PL/SQL procedure successfully completed.

```
ERI
-----4.82163662
```

f the value of the current error is too high or if the predictions obtained from the matrix factorization results of the collaborative filtering are not yet useful, you can run more iterations of the algorithm, by reusing the working tables and the progress made so far. The following example shows how to make predictions using the SQL-based collaborative filtering.

Example A-23 Collaborative Filtering: Making Predictions

The result of the collaborative filtering algorithm is stored in the tables wt_l and wt_r , which are the two factors of a matrix product. These matrix factors should be used when making the predictions of the collaborative filtering.



In a typical flow of the algorithm, the two matrix factors can be used to make the predictions before calling the OPG_APIS.CF_CLEANUP procedure, or they can be copied and persisted into other tables for later use. The following example demonstrates the latter case:

```
DECLARE
  wt 1 varchar2(32); -- working tables
  wt r varchar2(32);
  wt 11 varchar2(32);
  wt r1 varchar2(32);
  wt i varchar2(32);
  wt ld varchar2(32);
  wt rd varchar2(32);
  edge_tab_name varchar2(32) := 'phonesge$';
  edge label varchar2(32) := 'rating';
  rating_property varchar2(32) := '';
 iterations integer := 100;
min_error number := 0.001;
k integer := 5;
learning_rate number := 0.001;
decrease_rate number := 0.95;
regularization number := 0.02;
dop number := 2;
  tablespace varchar2(32) := null;
options varchar2(32) := null;
BEGIN
  -- prepare
  opg apis.cf prep(edge tab name, wt 1, wt r, wt 11, wt r1, wt i, wt 1d, wt rd);
  -- compute
  opg apis.cf(edge tab name, edge label, rating property, iterations,
                min error, k, learning rate, decrease rate, regularization, dop,
                wt l,wt r,wt ll,wt rl,wt i,wt ld,wt rd,tablespace,options);
  -- save only these two tables for later predictions
  EXECUTE IMMEDIATE 'CREATE TABLE customer mat AS SELECT * FROM ' || wt 1;
  EXECUTE IMMEDIATE 'CREATE TABLE item mat AS SELECT * FROM ' || wt r;
  -- cleanup
  opg apis.cf cleanup('phonesge$',wt l,wt r,wt l1,wt r1,wt i,wt ld,wt rd);
END;
```

This example will produce the only the following output.

```
PL/SQL procedure successfully completed.
```

Now that the matrix factors are saved in the tables customer_mat and item_mat, you can use the following query to check the "error" (difference) between the real values (those values that previously existed in the graph as 'ratings') and the estimated predictions (the result of the matrix multiplication in a certain customer row and item column).

Note that the following query is customized with a join on the vertex table in order return an NVARCHAR property of the vertices (for example, the name property) instead of a numeric

ID. This query will return all the predictions for every single customer vertex to every item vertex in the graph.

```
SELECT /*+ parallel(48) */ MIN(vertex1.v) AS customer,
                           MIN(vertex2.v) AS item,
                           MIN(edges.vn) AS real,
                           SUM(1.v * r.v) AS predicted
FROM PHONESGE$ edges,
     CUSTOMER MAT 1,
     ITEM MAT r,
     PHONESVT$ vertex1,
     PHONESVT$ vertex2
WHERE l.k = r.k
 AND l.c = edges.svid(+)
 AND r.p = edges.dvid(+)
 AND l.c = vertex1.vid
 AND r.p = vertex2.vid
GROUP BY 1.c, r.p
ORDER BY l.c, r.p -- This order by clause is optional
;
```

This query may produce an output similar to the following (some rows are omitted for brevity).

CUSTOMER	ITEM	REAL	PREDICTED
Adam	Apple	5	3.67375703
Adam	Blackberry		3.66079652
Adam	Danger		2.77049596
Adam	Ericsson		4.21764858
Adam	Figo		3.10631337
Adam	Google	4	4.42429022
Adam	Huawei	3	3.4289115
Ben	Apple		2.82127589
Ben	Blackberry	2	2.81132282
Ben	Danger	3	2.12761307
Ben	Ericsson	3	3.2389595
Ben	Figo		2.38550534
Ben	Google		3.39765075
Ben	Huawei		2.63324582
Don	Apple		1.3777496
Don	Blackberry	1	1.37288909
Don	Danger	1	1.03900439
Don	Ericsson		1.58172236
Don	Figo	1	1.16494421
Don	Google		1.65921807
Don	Huawei	1	1.28592648
Erik	Apple	3	2.80809351
Erik	Blackberry	3	2.79818695
Erik	Danger		2.11767182
Erik	Ericsson	3	3.2238255
Erik	Figo		2.3743591
Erik	Google	3	3.38177526
Erik	Huawei	3	2.62094201



If you want to check only some rows to decide whether the prediction results are ready or more iterations of the algorithm should be run, the previous query can be wrapped in an outer query. The following example will select only the first 11 results.

```
SELECT /*+ parallel(48) */ * FROM (
SELECT /*+ parallel(48) */ MIN(vertex1.v) AS customer,
                           MIN(vertex2.v) AS item,
                           MIN(edges.vn) AS real,
                           SUM(1.v * r.v) AS predicted
FROM PHONESGE$ edges,
     CUSTOMER MAT 1,
     ITEM MAT r,
     PHONESVT$ vertex1,
     PHONESVT$ vertex2
WHERE l.k = r.k
  AND l.c = edges.svid(+)
  AND r.p = edges.dvid(+)
  AND l.c = vertex1.vid
  AND r.p = vertex2.vid
GROUP BY 1.c, r.p
ORDER BY 1.c, r.p
) WHERE rownum <= 11;
```

This query may produce an output similar to the following.

CUSTOMER	ITEM	REAL	PREDICTED
Adam	Apple	5	3.67375703
Adam	Blackberry		3.66079652
Adam	Danger		2.77049596
Adam	Ericsson		4.21764858
Adam	Figo		3.10631337
Adam	Google	4	4.42429022
Adam	Huawei	3	3.4289115
Ben	Apple		2.82127589
Ben	Blackberry	2	2.81132282
Ben	Danger	3	2.12761307
Ben	Ericsson	3	3.2389595

To get a prediction for a specific vertex (customer, item, or both) the query can be restricted with the desired ID values. For example, to get the predicted value of vertex 1 (customer) and vertex 105 (item), you can use the following guery.



This query may produce an output similar to the following.

will be created based on this parameter. */,

A.7 Creating Property Graph Views on an RDF Graph

With Oracle Graph, you can view RDF data as a property graph to execute graph analytics operations by creating property graph views over an RDF graph stored in Oracle Database.

Given an RDF model (or a virtual model), the property graph feature creates two views, a <graph_name>VT\$ view for vertices and a <graph_name>GE\$ view for edges.

The PGUtils.createPropertyGraphViewOnRDF method lets you customize a property graph view over RDF data:

RDFPredicate[] predListForEdges /* an array of RDFPredicate specifying how to create edge view using these predicates; each RDFPredicate includes two (or three) fields: an URL of the RDF predicate, the edge label in the Property Graph, the weight of the edge (optional). The mapping from RDF predicates to edges will be created based on this parameter. */)

This operation requires the name of the property graph, the name of the RDF Model used to generate the Property Graph view, and a set of mappings determining how triples will be parsed into vertices or edges. The <code>createPropertyGraphViewOnRDF</code> method requires a *key/value mapping* array specifying how RDF predicates are mapped to Key/Value properties for vertices, and an *edge mapping* array specifying how RDF predicates are mapped to edges. The <code>PGUtils.RDFPredicate</code> API lets you create a map from RDF assertions to vertices/edges.

Vertices are created based on the triples matching at least one of the RDF predicates in the key/value mappings. Each triple satisfying one of the RDF predicates defined in the mapping array is parsed into a vertex with ID based on the internal RDF resource

ID of the subject of the triple, and a key/value pair whose key is defined by the mapping itself and whose value is obtained from the object of the triple.

The following example defines a key/value mapping of the RDF predicate URI http://purl.org/dc/elements/1.1/title to the key/value property with property name title.

Edges are created based on the triples matching at least one of the RDF predicates in the edge mapping array. Each triple satisfying the RDF predicate defined in the mapping array is parsed into an edge with ID based on the row number, an edge label defined by the mapping itself, a source vertex obtained from the RDF Resource ID of the subject of the triple, and a destination vertex obtained from the RDF Resource ID of the object of the triple. For each triple parsed here, two vertices will be created if they were not generated from the key/value mapping.

The following example defines an edge mapping of the RDF predicate URI http://purl.org/dc/elements/1.1/reference to an edge with a label references and a weight of 0.5d.

The following example creates a property graph view over the RDF model articles describing different publications, their authors, and references. The generated property graph will include vertices with some key/value properties that may include title and creator. The edges in the property graph will be determined by the references among publications.

```
Oracle oracle = null;
Connection conn = null;
OraclePropertyGraph pggraph = null;
  // create the connection instance to Oracle database
  OracleDataSource ds = new oracle.jdbc.pool.OracleDataSource();
  ds.setURL(jdbcUrl);
  conn = (OracleConnection) ds.getConnection(user, password);
  // define some string variables for RDF predicates
  String titleURL = "http://purl.org/dc/elements/1.1/title";
  String creatorURL = "http://purl.org/dc/elements/1.1/creator";
  String serialnumberURL = "http://purl.org/dc/elements/1.1/serialnumber";
  String widthURL = "http://purl.org/dc/elements/1.1/width";
  String weightURL = "http://purl.org/dc/elements/1.1/weight";
  String onsaleURL = "http://purl.org/dc/elements/1.1/onsale";
  String publicationDateURL = "http://purl.org/dc/elements/1.1/publicationDate";
  String publicationTimeURL = "http://purl.org/dc/elements/1.1/publicationTime";
  String referencesURL = "http://purl.org/dc/terms/references";
  // create RDFPredicate[] predsForVertexAttrs to specify how to map
  // RDF predicate to vertex keys
  RDFPredicate[] predsForVertexAttrs = new RDFPredicate[8];
  predsForVertexAttrs[0] = RDFPredicate.getInstance(titleURL, "title");
  predsForVertexAttrs[1] = RDFPredicate.getInstance(creatorURL, "creator");
  predsForVertexAttrs[2] = RDFPredicate.getInstance(serialnumberURL,
```



```
"serialnumber");
 predsForVertexAttrs[3] = RDFPredicate.getInstance(widthURL, "width");
 predsForVertexAttrs[4] = RDFPredicate.getInstance(weightURL, "weight");
 predsForVertexAttrs[5] = RDFPredicate.getInstance(onsaleURL, "onsale");
 predsForVertexAttrs[6] = RDFPredicate.getInstance(publicationDateURL,
                                                    "publicationDate");
 predsForVertexAttrs[7] = RDFPredicate.getInstance(publicationTimeURL,
                                                    "publicationTime");
 // create RDFPredicate[] predsForEdges to specify how to map RDF predicates to
 // edges
 RDFPredicate[] predsForEdges = new RDFPredicate[1];
 predsForEdges[0] = RDFPredicate.getInstance(referencesURL, "references", 0.5d);
 // create PG view on RDF model
 PGUtils.createPropertyGraphViewOnRDF(conn, "articles", "articles", false,
                                       predsForVertexAttrs, predsForEdges);
 // get the Property Graph instance
 oracle = new Oracle(jdbcUrl, user, password);
 pggraph = OraclePropertyGraph.getInstance(oracle, "articles", 24);
 System.err.println("----- Vertices from property graph view -----");
 pggraph.getVertices();
 System.err.println("----- Edges from property graph view -----");
 pggraph.getEdges();
finally {
 pggraph.shutdown();
 oracle.dispose();
 conn.close();
```

Given the following triples in the articles RDF model (11 triples), the output property graph will include two vertices, one for http://nature.example.com/Article1 (v1) and another one for http://nature.example.com/Article2 (v2). For vertex v1, it has eight properties, whose values are the same as their RDF predicates. For example, v1's title is "All about XYZ". Similarly for vertex v2, it has two properties: title and creator. The output property graph will include a single edge (eid:1) from vertex v1 to vertex v2 with an edge label "references" and a weight of 0.5d.

```
<http://nature.example.com/Article1> <http://purl.org/dc/elements/1.1/title>
"All about XYZ"^^xsd:string.
<http://nature.example.com/Article1> <http://purl.org/dc/elements/1.1/creator>
"Jane Smith"^^xsd:string.
<http://nature.example.com/Article1> <http://purl.org/dc/elements/1.1/</pre>
serialnumber> "123456"^^xsd:integer.
<http://nature.example.com/Article1> <http://purl.org/dc/elements/1.1/width>
"10.5"^^xsd:float.
<http://nature.example.com/Article1> <http://purl.org/dc/elements/1.1/weight>
"1.08"^^xsd:double.
<http://nature.example.com/Article1> <http://purl.org/dc/elements/1.1/onsale>
"false"^^xsd:boolean.
<http://nature.example.com/Article1> <http://purl.org/dc/elements/1.1/</pre>
publicationDate> "2016-03-08"^^xsd:date)
<http://nature.example.com/Article1> <http://purl.org/dc/elements/1.1/</pre>
publicationTime> "2016-03-08T10:10:10"^^xsd:dateTime)
<http://nature.example.com/Article2> <http://purl.org/dc/elements/1.1/title> "A
review of ABC"^^xsd:string.
<http://nature.example.com/Article2> <http://purl.org/dc/elements/1.1/creator>
```



```
"Joe Bloggs"^^xsd:string.
<a href="http://nature.example.com/Article1">http://purl.org/dc/terms/references</a> <a href="http://purl.org/dc/terms/references">http://purl.org/dc/terms/references</a> <a href="http://purl.org/dc/terms/reference
```

The preceding code will produce an output similar as the following. Note that the internal RDF resource ID values may vary across different Oracle databases.

```
----- Vertices from property graph view -----
Vertex ID 7299961478807817799 {creator:str:Jane Smith, onsale:bol:false,
publicationDate:dat:Mon Mar 07 16:00:00 PST 2016, publicationTime:dat:Tue Mar 08
02:10:10 PST 2016, serialnumber:dbl:123456.0, title:str:All about XYZ,
weight:dbl:1.08, width:flo:10.5}
Vertex ID 7074365724528867041 {creator:str:Joe Bloggs, title:str:A review of ABC}
----- Edges from property graph view -----
Edge ID 1 from Vertex ID 7299961478807817799 {creator:str:Jane Smith,
onsale:bol:false, publicationDate:dat:Mon Mar 07 16:00:00 PST 2016,
publicationTime:dat:Tue Mar 08 02:10:10 PST 2016, serialnumber:dbl:123456.0,
title:str:All about XYZ, weight:dbl:1.08, width:flo:10.5} = [references] => Vertex ID
7074365724528867041 {creator:str:Joe Bloggs, title:str:A review of ABC}
edgeKV[{weight:dbl:0.5}]
```

A.8 Quick Start: Interactively Analyze Graph Data Stored in Property Graph Schema Objects

This tutorial shows how you can quickly get started using property graph data and learn to execute PGQL queries and run graph algorithms on the data and display results.

The tutorials in this section are:

- Quick Start: Create and Query a Graph in the Database, Load into Graph Server (PGX) for Analytics
 - This tutorial shows how you can get started using property graph data when you create a graph and persist it in the database. The graph can be queried in the database. This tutorial uses the JShell client.
- Quick Start: Create, Query, and Analyze a Graph in Graph Server (PGX)
 This tutorial shows how you can quickly get started using property graph data when using the graph server (PGX).

A.8.1 Quick Start: Create and Query a Graph in the Database, Load into Graph Server (PGX) for Analytics

This tutorial shows how you can get started using property graph data when you create a graph and persist it in the database. The graph can be queried in the database. This tutorial uses the JShell client.

See Create and Query a Graph in the Database for more information on creating and storing graphs in database.

- Convert existing relational data into a graph in the database.
- Query this graph using PGQL.

In Load the Graph into Memory and Run Graph Analytics, you will run graph algorithms after loading the graph into the graph server (PGX).



• Load the graph into the graph server (PGX), run graph algorithms on this graph, and visualize results.

Prerequisites for the following quickstart are:

An installation of Oracle Graph server.

See Oracle Graph Server and Client Installation for information to download Oracle Graph Server and Client.

- An installation of Oracle Graph client
- Java 11
 - The graph server can work with Java 8 or Java 11.
 - The JShell client used in this example requires Java 11.

For Java downloads, see https://www.oracle.com/technetwork/java/javase/overview/index.html.

- Connection details for your Oracle Database. See Verifying Database
 Compatibility to identify any limitations. The Property Graph feature is supported
 for Oracle Database versions 12.2 and later.
- Basic knowledge about how to run commands on Oracle Database (for example, using SQL*Plus or SQL Developer).

Set up the example data

This example uses the HR (human resources) sample dataset.

- For instructions how to import that data into a user managed database, see: https://github.com/oracle/db-sample-schemas
- If you are using Autonomous Database, see: https://www.thatjeffsmith.com/archive/2019/07/creating-hr-in-oracle-autonomous-database-w-sql-developer-web/

Note that the database schema storing the graph must have the privileges listed in Required Privileges for Database Users.

- Create and Query a Graph in the Database
 In this section, you will use the Oracle Graph client to create a graph from relational tables and store it in the property graph schema in the database.
- Load the Graph into Memory and Run Graph Analytics

A.8.1.1 Create and Query a Graph in the Database

In this section, you will use the Oracle Graph client to create a graph from relational tables and store it in the property graph schema in the database.

Major tasks for this tutorial:

- Start the shell
- Open a JDBC database connection
- · Create a PGQL connection
- Write and execute the graph creation statement
- Run a few PGQL queries



Start the shell

On the system where Oracle Graph client is installed, start the shell by as follows:

```
cd <client-install-dir>
./bin/opg4j --no_connect
```

The --no_connect option indicates that you are not connecting to the graph server (PGX). You will only be connecting to the database in this example.

Note that JAVA HOME should be set to Java 11 before you start the shell. For example:

```
export JAVA HOME=/usr/lib/jvm/java-11-oracle
```

See Interactive Graph Shell CLIs for details about the shell.

Open a JDBC database connection

Inside the shell prompt, use the standard JDBC Java API to obtain a database connection object. For example:

```
opg4j> var jdbcUrl = "<jdbc-url>" // for example:
jdbc:oracle:thin:@myhost:1521/myservice
opg4j> var user = "<db-user>" // for example: hr
opg4j> var pass = "<db-pass>"
opg4j> var conn = DriverManager.getConnection(jdbcUrl, user, pass)
conn ==> oracle.jdbc.driver.T4CConnection@57e6cb01
```

Connecting to an Autonomous Database works the same way: provide a JDBC URL that points to the local wallet. See Using Oracle Graph with the Autonomous Database for an example.

Create a PGQL connection

Convert the JDBC connection into a PGQL connection object. For example:

```
opg4j> conn.setAutoCommit(false)
opg4j> var pgql = PgqlConnection.getConnection(conn)
pgql ==> oracle.pg.rdbms.pgql.PgqlConnection@6fb3d3bb
```

Write and execute the graph creation statement

Using a text editor, write a CREATE PROPERTY GRAPH statement that describes how the HR sample data should be converted into a graph. Save this file as create.pgql at a location of your choice. For example:

```
CREATE PROPERTY GRAPH hr

VERTEX TABLES (

employees LABEL employee

PROPERTIES ARE ALL COLUMNS EXCEPT ( job_id, manager_id, department_id ),

departments LABEL department

PROPERTIES ( department id, department name ),
```



```
jobs LABEL job
    PROPERTIES ARE ALL COLUMNS,
  job history
   PROPERTIES ( start date, end date ),
  locations LABEL location
    PROPERTIES ARE ALL COLUMNS EXCEPT ( country id ),
  countries LABEL country
    PROPERTIES ARE ALL COLUMNS EXCEPT ( region id ),
  regions LABEL region
EDGE TABLES (
  employees AS works for
    SOURCE employees
    DESTINATION KEY ( manager id ) REFERENCES employees (employee id)
   NO PROPERTIES,
  employees AS works at
    SOURCE employees
    DESTINATION departments
    NO PROPERTIES,
  employees AS works as
    SOURCE employees
    DESTINATION jobs
   NO PROPERTIES,
  departments AS managed by
    SOURCE departments
    DESTINATION employees
    NO PROPERTIES,
  job history AS for employee
    SOURCE job history
    DESTINATION employees
    LABEL for
    NO PROPERTIES,
  job history AS for department
    SOURCE job history
    DESTINATION departments
    LABEL for
    NO PROPERTIES,
  job history AS for job
    SOURCE job history
    DESTINATION jobs
    LABEL for
    NO PROPERTIES,
  departments AS department located in
    SOURCE departments
    DESTINATION locations
    LABEL located in
    NO PROPERTIES,
  locations AS location located in
    SOURCE locations
    DESTINATION countries
   LABEL located in
   NO PROPERTIES,
  countries AS country located in
    SOURCE countries
    DESTINATION regions
```



```
LABEL located_in NO PROPERTIES )
```

Then, back in your graph shell, execute the CREATE PROPERTY GRAPH statement by sending it to your PGQL connection. Replace <path> with the path to the directory containing the create.pgql file:

```
opg4j> pgql.prepareStatement(Files.readString(Paths.get("<path>/
create.pgql"))).execute()
$16 ==> false
```

Run a few PGQL queries

Now that you have a graph named hr, you can use PGQL to run a few queries against it directly on the database. For example:

```
// define a little helper function that executes the query, prints the
results and properly closes the statement
opg4j > Consumer < String > query = q -> { try(var s = pgql.prepareStatement(q))}
{ s.execute(); s.getResultSet().print(); } catch(Exception e) { throw new
RuntimeException(e); } }
query ==> $Lambda$605/0x000000100ae6440@6c9e7af2
// print the number of vertices in the graph
opg4j> query.accept("SELECT COUNT(v) FROM MATCH (v) ON hr")
+----+
| COUNT(v) |
+----+
| 215
+----+
// print the number of edges in the graph
opq4j> query.accept("SELECT COUNT(e) FROM MATCH ()-[e]->() ON hr")
+----+
| count(e) |
+----+
I 433
+----+
// find the highest earning managers
opg4j> query.accept("SELECT DISTINCT m.FIRST NAME, m.LAST NAME, m.SALARY
FROM MATCH (v:EMPLOYEE)-[:WORKS FOR]->(m:EMPLOYEE) ON hr ORDER BY m.SALARY
DESC")
| m.FIRST NAME | m.LAST NAME | m.SALARY |
+----+
| Lex
           | De Haan | 17000.0 |
         | Kochhar | 17000.0 |
Neena
| John
           | Russell
                       | 14000.0 |
```



A.8.1.2 Load the Graph into Memory and Run Graph Analytics

Major tasks for this tutorial:

- Load the graph from the property graph schema into memory
- Execute algorithms and query the algorithm results
- Share the Graph with Other Sessions

Load the graph from the property graph schema into memory

In this section of the quickstart, you will load the graph stored in the Property Graphs schema in the database into the graph server (PGX). This will enable you to run a variety of different built-in algorithms on the graph and will also improve query performance for larger graphs.

First, start the JShell client and connect to the graph server (PGX):

```
./bin/opg4j --base_url https://<graph server host>:7007 --username
<graphuser>
```

<graphuser> is the database user you will use to for the PGX server authentication.
You will be prompted for the database password.





For demo purposes only, if you have set <code>enable_tls</code> to <code>false</code> in the <code>/etc/oracle/graph/server.conf</code> file you can use an <code>http</code> instead of <code>https</code> connection.

```
./bin/opg4j --base url http://<graph server host>:7007 --username <graphuser>
```

This starts the shell and makes a connection to the graph server.



Always use low-privilege read-only database user accounts for PGX, as explained in Security Best Practices with Graph Data.

Next load the graph into memory in this server.

To load the graph into memory, create a PGX graph config object, using the PGX graph config builder API to do this directly in the shell.

The following example creates a PGX graph config object. It lists the properties to load into memory so that you can exclude other properties, thus reducing memory consumption.

```
Supplier<GraphConfig> pgxConfig = () -> { return
GraphConfigBuilder.forPropertyGraphRdbms()
.setName("hr")
 .addVertexProperty("COUNTRY NAME", PropertyType.STRING)
 .addVertexProperty("DEPARTMENT NAME", PropertyType.STRING)
 .addVertexProperty("FIRST_NAME", PropertyType.STRING)
 .addVertexProperty("LAST NAME", PropertyType.STRING)
 .addVertexProperty("EMAIL", PropertyType.STRING)
 .addVertexProperty("PHONE NUMBER", PropertyType.STRING)
 .addVertexProperty("SALARY", PropertyType.DOUBLE)
 .addVertexProperty("MIN SALARY", PropertyType.DOUBLE)
 .addVertexProperty("MAX SALARY", PropertyType.DOUBLE)
 .addVertexProperty("STREET ADDRESS", PropertyType.STRING)
 .addVertexProperty("POSTAL_CODE", PropertyType.STRING)
 .addVertexProperty("CITY", PropertyType.STRING)
 .addVertexProperty("STATE PROVINCE", PropertyType.STRING)
 .addVertexProperty("REGION NAME", PropertyType.STRING)
 .setPartitionWhileLoading(PartitionWhileLoading.BY LABEL)
 .setLoadVertexLabels(true)
 .setLoadEdgeLabel(true)
 .build(); }
```

Now that you have a graph config object, use the following API to read the graph into PGX:

```
opg4j> var graph = session.readGraphWithProperties(pgxConfig.get())
graph ==> PgxGraph[name=hr,N=215,E=433,created=1586996113457]
```



The session object is created for you automatically.

Execute algorithms and query the algorithm results

Now that you have the graph in memory, you can run any built-in algorithm using a single API invocation. For example, for pagerank:

```
opg4j> analyst.pagerank(graph)
$31==> VertexProperty[name=pagerank,type=double,graph=hr]
```

As you can see from the preceding outputs, each algorithm created a new vertex property on the graph holding the output of the algorithm. To print the most important people in the graph (according to pagerank), you can run the following query:

Share the Graph with Other Sessions

After you load the graph into the graph server, you can use the publish() API to make the graph visible to other sessions, such as the graph visualization session. For example:

```
opg4j> graph.publish(VertexProperty.ALL, EdgeProperty.ALL)
```

The published graph will include any new properties you add to the graph by calling functions, such as pagerank.

You can use the Graph Visualization Application by navigating to <my-server-name>:7007/ui/ in your browser.



You can connect to a particular client session by providing the session ID when you log into the Graph Visualization Application. You will then be able to visualize all graphs in the session, even if they have not been published.

```
opg4j> session
session ==> PgxSession[ID=5adf83ab-31b1-4a0e-8c08-
d6a95ba63ee0,source=pgxShell]
```

The session id is 5adf83ab-31b1-4a0e-8c08-d6a95ba63ee0.



You must create a server certificate to connect to the graph server (PGX) from the Graph Visualization Application. See Setting Up Transport Layer Security for more details

A.8.2 Quick Start: Create, Query, and Analyze a Graph in Graph Server (PGX)

This tutorial shows how you can quickly get started using property graph data when using the graph server (PGX).

This is for use cases where the graph is available as long as the graph server (PGX) session is active. The graph is not persisted in the database.

- Create a graph in the graph server (PGX), directly from existing relational data
- Query this graph using PGQL in the graph server (PGX)
- Run graph algorithms in the graph server (PGX) on this graph and display results

Prerequisites for the following quickstart are:

- An installation of Oracle Graph server.
 - See Installing Oracle Graph Server for information to download Oracle Graph Server.
- An installation of Oracle Graph client.

See Installing the Java Client From the Graph Server and Client Downloads for information to download Oracle Graph Client.

You will authenticate yourself as the database user to the graph server, and these database credentials are used to access the database tables and create a graph.

- Java 11
 - The graph server can work with Java 8 or Java 11.
 - The JShell client used in this example requires Java 11.

For Java downloads, see https://www.oracle.com/technetwork/java/javase/overview/index.html.

Major tasks for this tutorial:

- · Set up the example data
- Start the shell



- · Write and execute the graph creation statement
- Run a few PGQL gueries
- Execute algorithms and query the algorithm results
- Share the Graph with Other Sessions

Set up the example data

This example uses the HR (human resources) sample dataset.

- For instructions how to import that data into a user managed database, see: https://github.com/oracle/db-sample-schemas
- If you are using Autonomous Database, see: https://www.thatjeffsmith.com/archive/2019/07/creating-hr-in-oracle-autonomous-database-w-sql-developer-web/

Note that the database schema storing the graph must have the privileges listed in Required Privileges for Database Users.

Start the shell

On the system where Oracle Graph Client is installed, start the shell as follows. This is an example of starting a shell in remote mode and connecting to the graph server (PGX):

```
./bin/opg4j --base_url https://<graph server host>:7007 --username
<graphuser>
```

<graphuser> is the database user you will use to for the PGX server authentication.
You will be prompted for the database password.



For demo purposes only, if you have set <code>enable_tls</code> to <code>false</code> in the <code>/etc/oracle/graph/server.conf</code> file you can use an <code>http</code> instead of <code>https</code> connection.

```
./bin/opg4j --base_url http://<graph server host>:7007 --username
<graphuser>
```

This starts the shell and makes a connection to the graph server.

Note that, JAVA HOME should be set to Java 11 before you start the shell. For example:

```
export JAVA HOME=/usr/lib/jvm/java-11-oracle
```

See Interactive Graph Shell CLIs for details about the shell.



Write and execute the graph creation statement

Create a graph with employees, departments, and "employee works at department", by executing a CREATE PROPERTY GRAPH statement. The following statement creates a graph in the graph server (PGX):

```
opg4j> String statement =
      "CREATE PROPERTY GRAPH hr simplified "
    + " VERTEX TABLES ( "
         hr.employees LABEL employee "
    + "
           PROPERTIES ARE ALL COLUMNS EXCEPT ( job id, manager id,
department id ), "
    + " hr.departments LABEL department "
           PROPERTIES ( department id, department name ) "
    + " ) "
    + " EDGE TABLES ( "
    + " hr.employees AS works_at "
    + "
           SOURCE KEY ( employee id ) REFERENCES employees (employee id) "
   + " DESTINATION departments " + " PROPERTIES ( employee_id ) "
    + " ) "
opg-jshell> session.executePgql(statement)
```

To get a handle to the graph, execute:

```
opg4j> var g = session.getGraph("HR_SIMPLIFIED")
```

Run a few PGQL queries

You can use this handle to run PGQL queries on this graph. For example, to find the department that "Nandita Sarchand" works for, execute:

To get an overview of the types of vertices and their frequencies, execute:

```
opg4j> String query =
    "SELECT label(n), COUNT(*) "
    + "FROM MATCH (n) "
    + "GROUP BY label(n) "
    + "ORDER BY COUNT(*) DESC"
```



To get an overview of the types of edges and their frequencies, execute:

Execute algorithms and query the algorithm results

Now that you have the graph in memory, you can run each built-in algorithms using a single API invocation. For example, for pagerank:

```
opg4j> analyst.pagerank(g)
$31==> VertexProperty[name=pagerank,type=double,graph=hr]
```

As you can see from the preceding outputs, each algorithm created a new vertex property on the graph holding the output of the algorithm. To print the most important people in the graph (according to pagerank), you can run the following query:



In the following example, we order departments by their pagerank value. Departments with higher pagerank values have more employees.

opg4j> session.queryPgql("SELECT m.DEPARTMENT_NAME, m.pagerank FROM MATCH
(m:DEPARTMENT) ON hr simplified ORDER BY m.pagerank").print().close()

+ DEPARTMENT_NAME	· 	+ pagerank
+ Manufacturing		+ 0.001119402985074627
Construction	1	0.001119402985074627
Contracting	1	0.001119402985074627
Operations	1	0.001119402985074627
IT Support	1	0.001119402985074627
NOC	1	0.001119402985074627
IT Helpdesk	ı	0.001119402985074627
Government Sales		0.001119402985074627
Retail Sales	i	0.001119402985074627
Recruiting	i	0.001119402985074627
Payroll	i	0.001119402985074627
Treasury	i	0.001119402985074627
Corporate Tax	i	0.001119402985074627
Control And Credit	i	0.001119402985074627
Shareholder Services	İ	0.001119402985074627
Benefits	İ	0.001119402985074627
Human Resources	Ì	0.0020708955223880596
Administration		0.0020708955223880596
Public Relations		0.0020708955223880596
Marketing		0.003022388059701493
Accounting		0.003022388059701493
Executive		0.003973880597014925
IT		0.005876865671641792
Purchasing		0.006828358208955224
Finance		0.006828358208955224
Sales		0.03347014925373134
Shipping		0.043936567164179076

Share the Graph with Other Sessions

After you load the graph into the server, you can use the <code>publish()</code> API to make the graph visible to other sessions, such as the graph visualization session. For example:

```
opg4j> graph.publish(VertexProperty.ALL, EdgeProperty.ALL)
```

The published graph will include any new properties you add to the graph by calling functions, such as pagerank.

Ensure that the logged-in user has the privilege to publish graphs. You can do this by adding the privilege PGX_SESSION_ADD_PUBLISHED_GRAPH to the GRAPH_DEVELOPER role as explained in Adding Permissions to Publish the Graph. We had given the GRAPH_DEVELOPER role to the database user in Installing PL/SQL Packages in Oracle Database.



You can use the Graph Visualization Application by navigating to <my-server-name>:7007/ui/ in your browser.

You can connect to a particular client session by providing the session ID when you log into the Graph Visualization Application. You will then be able to visualize all graphs in the session, even if they have not been published.

```
opg4j> session
session ==> PgxSession[ID=5adf83ab-31b1-4a0e-8c08-
d6a95ba63ee0,source=pgxShell]
```

The session id is 5adf83ab-31b1-4a0e-8c08-d6a95ba63ee0.



You must create a server certificate to connect to the graph server (PGX) from the Graph Visualization Application. See Setting Up Transport Layer Security for more details.

A.9 Working with Property Graph Objects in SQL Developer

You can use Oracle SQL Developer to execute PGQL statements and queries directly on property graph schema graphs in the database.

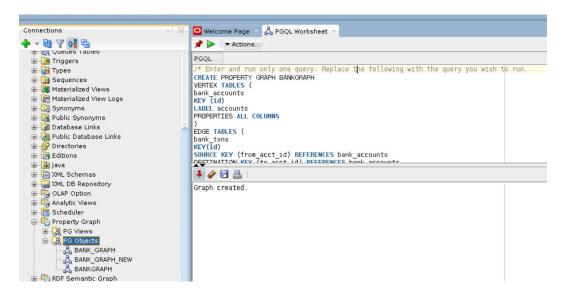
You can view all the property graph objects existing in your database schema by expanding **PG Objects** under the **Property Graph** node in the **Connections** navigator.

You can run PGQL queries for a property graph object in a **PGQL Worksheet**. The following steps show a few examples for creating, updating and dropping a property graph object using SQL Developer.

- Right-click the Property Graph node and select Open PGQL Worksheet.
 PGQL Worksheet opens in a new tab and it contains the Run Query icon for executing PGQL queries.
- 2. Create a property graph object by running a CREATE PROPERTY GRAPH statement in the PGQL Worksheet. For example:



Figure A-2 Creating a Property Graph Object



The result of the query execution is displayed in the bottom pane of the Editor. On successful query execution, you can right-click and refresh the **PG Objects** node to view the newly created graph under **PG Objects**.

3. Click on the newly created graph.

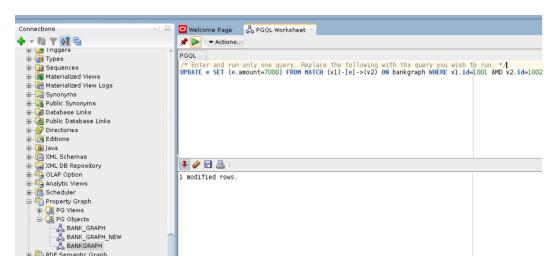
This opens a **PGQL Worksheet** in a new tab with the following default query:

```
SELECT id(e), id(v), id(n) FROM MATCH (v)-[e]-(n) ON \langle graph\_name \rangle LIMIT 100
```

4. Run any PGQL update query like performing an INSERT or an UPDATE operation against a property graph object.

For example, the following shows the execution of a PGQL UPDATE query:

Figure A-3 Updating a Property Graph Object

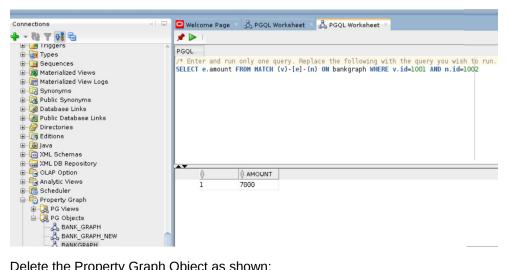




The related edge is updated in the graph.

Run a PGQL SELECT query to view the newly updated edge as shown:

Figure A-4 Running a PGQL SELECT Query



6. Delete the Property Graph Object as shown:



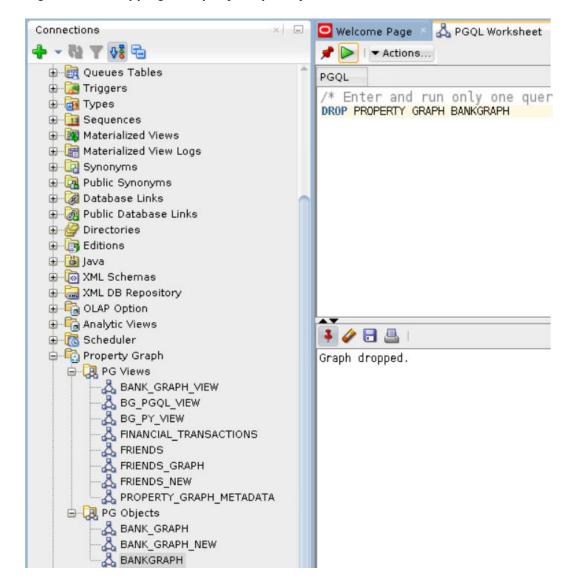


Figure A-5 Dropping a Property Graph Object

The graph is dropped.

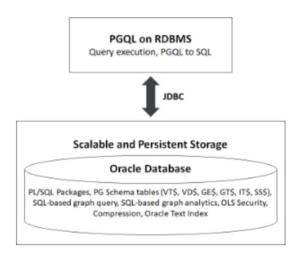
A.10 Executing PGQL Queries Against Property Graph Schema Tables

This topic explains how you can execute PGQL queries directly against the graph stored in property graph schema tables.

The PGQL query execution flow is shown in the following figure.



Figure A-6 PGQL on Property Graph Schema Tables in Oracle Database (RDBMS)



The basic execution flow is:

- The PGQL query is submitted to PGQL on RDBMS through a Java API.
- 2. The PGQL query is translated to SQL.
- 3. The translated SQL is submitted to Oracle Database by JDBC.
- 4. The SQL result set is wrapped as a PGQL result set and returned to the caller.

The ability to execute PGQL queries directly against property graph data stored in Oracle Database provides several benefits.

- PGQL provides a more natural way to express graph queries than SQL manually written to guery schema tables, including VT\$, VD\$, GE\$, and GT\$.
- PGQL queries can be executed without the need to load a snapshot of your graph data into PGX, so there is no need to worry about staleness of frequently updated graph data.
- PGQL queries can be executed against graph data that is too large to fit in memory.
- The robust and scalable Oracle SQL engine can be used to execute PGQL queries.
- Mature tools for management, monitoring and tuning of Oracle Database can be used to tune and monitor PGQL queries.
- PGQL Features Supported in Property Graph Schema
- Creating Property Graphs through CREATE PROPERTY GRAPH Statements
- Dropping Property Graphs through DROP PROPERTY GRAPH Statements
- Using the oracle.pg.rdbms.pggl Java Package to Execute PGQL Queries
- Using the Python Client to Execute PGQL Queries
- Performance Considerations for PGQL Queries



A.10.1 PGQL Features Supported in Property Graph Schema

PGQL is a SQL-like query language for querying property graph data. It is based on the concept of graph pattern matching and allows you to specify, among other things, topology constraints, paths, filters, sorting and aggregation.

The Java API for PGQL defined in the oracle.pg.rdbms.pgql package supports the PGQL specification with a few exceptions. (Refer to the PGQL Specification).

The following table describes the list of supported and unsupported PGQL features:

Table A-1 Supported PGQL Features and Limitations for PG Schema Graphs

Feature	PG Schema
CREATE PROPERTY GRAPH	Supported
DROP PROPERTY GRAPH	Supported
Fixed-length pattern matching	Supported
Variable-length pattern matching goals	Supported: • Reachability Limitations:
	• ANY
	• ANY SHORTEST
	• TOP k SHORTEST
	• ALL SHORTEST
	• ANY CHEAPEST
	• TOP k CHEAPEST
	• ALL
Variable-length pattern matching quantifiers	Supported: * +
	• ?
	• {n}
	• { n, }
	• { n, m }
	• { , m }
Variable-length path unnesting	Not supported
GROUP BY	Supported
HAVING	Supported
Aggregations	Supported: • COUNT
	• MIN, MAX, AVG, SUM
	Limitations:
	• LISTAGG
	• ARRAY_AGG
DISTINCT • SELECT DISTINCT	Supported
 Aggregation with DISTINCT (such as, COUNT(DISTINCT e.prop)) 	
SELECT v.*	Not Supported
•	I F T T T T



Table A-1 (Cont.) Supported PGQL Features and Limitations for PG Schema Graphs

Feature	PG Schema
ORDER BY (+ASC/DESC), LIMIT, OFFSET	Supported
Data Types	Supported: NVARCHAR2 (15000) NUMBER BOOLEAN (stored like NVARCHAR) TIMESTAMP (6) WITH TIME ZONE
JSON	No built-in JSON support. However, JSON strings (VARCHAR2) can be mapped into NVARCHAR2 (15000) data type.
Operators	 Supported: Relational: +, -, *, /, %, - (unary minus) Arithmetic: =, <>, <, >, <=, >= Logical: AND, OR, NOT Limitations: String: (concat)
Functions and predicates	Supported: IS NULL, IS NOT NULL JAVA_REGEXP_LIKE (based on CONTAINS) ABS, CEIL/CEILING, FLOOR, ROUND EXTRACT ID LABEL, HAS_LABEL ALL_DIFFERENT IN_DEGREE, OUT_DEGREE CAST CASE IN and NOT IN Limitations: LOWER, UPPER SUBSTRING LABELS
User-defined functions	Not supported
Subqueries: Scalar subqueries EXISTS and NOT EXISTS subqueries	Supported
INSERT/UPDATE/DELETE	Supported for Oracle Database 19c and later
INTERVAL literals and operations	Not supported

The following explains a few PGQL features that require special consideration.

- Temporal Types
- Type Casting
- CONTAINS Built-in Function



A.10.1.1 Temporal Types

The temporal types DATE, TIMESTAMP and TIMESTAMP WITH TIMEZONE are supported in PGQL queries.

All of these value types are represented internally using the Oracle SQL TIMESTAMP WITH TIME ZONE type. DATE values are automatically converted to TIMESTAMP WITH TIME ZONE by assuming the earliest time in UTC+0 timezone (for example, 2000-01-01 becomes 2000-01-01 00:00:00.00+00:00). TIMESTAMP values are automatically converted to TIMESTAMP WITH TIME ZONE by assuming UTC+0 timezone (for example, 2000-01-01 12:00:00.00 becomes 2000-01-01 12:00:00.00+00:00).

Temporal constants are written in PGQL queries as follows.

- DATE 'YYYY-MM-DD'
- TIMESTAMP 'YYYY-MM-DD HH24:MI:SS.FF'
- TIMESTAMP WITH TIMEZONE 'YYYY-MM-DD HH24:MI:SS.FFTZH:TZM'

Some examples are DATE '2000-01-01', TIMESTAMP '2000-01-01 14:01:45.23', TIMESTAMP WITH TIMEZONE '2000-01-01 13:00:00.00-05:00', and TIMESTAMP WITH TIMEZONE '2000-01-01 13:00:00.00+01:00'.

In addition, temporal values can be obtained by casting string values to a temporal type. The supported string formats are:

- DATE 'YYYY-MM-DD'
- TIMESTAMP 'YYYY-MM-DD HH24:MI:SS.FF' and 'YYYY-MM-DD"T"HH24:MI:SS.FF'
- TIMESTAMP WITH TIMEZONE 'YYYY-MM-DD HH24:MI:SS.FFTZH:TZM' and 'YYYY-MM-DD"T"HH24:MI:SS.FFTZH:TZM'.

Some examples are CAST ('2005-02-04' AS DATE), CAST ('1990-01-01 12:00:00.00' AS TIMESTAMP), CAST ('1985-01-01T14:05:05.00-08:00' AS TIMESTAMP WITH TIMEZONE).

When consuming results from a PgqlResultSet object, getObject returns a java.sql.Timestamp object for temporal types.

Bind variables can only be used for the TIMESTAMP WITH TIMEZONE temporal type in PGQL, and a setTimestamp method that takes a java.sql.Timestamp object as input is used to set the bind value. As a simpler alternative, you can use a string bind variable in a CAST statement to bind temporal values (for example, CAST (? AS TIMESTAMP WITH TIMEZONE) followed by setString (1, "1985-01-01T14:05:05.00-08:00")). See also Using Bind Variables in PGQL Queries for more information about bind variables.

A.10.1.2 Type Casting

Type casting is supported in PGQL with a SQL-style CAST (VALUE AS DATATYPE) syntax, for example CAST('25' AS INT), CAST (10 AS STRING), CAST ('2005-02-04' AS DATE), CAST(e.weight AS STRING). Supported casting operations are summarized in the following table. Y indicates that the conversion is supported, and N indicates that it is not supported. Casting operations on invalid values (for example, CAST('xyz' AS INT)) or unsupported conversions (for example, CAST (10 AS TIMESTAMP)) return NULL instead of raising a SQL exception.



"to" type from from from from from from from from from STRIN INT LON FLOA **DOUB BOOLE** DAT **TIMESTA TIMESTA** LE ΑN MP MP WITH G Ε **TIMEZON** Ε to STRING Y Υ Υ Υ Υ Υ Υ Υ Υ to INT Υ Υ Υ Υ Υ Υ Ν Ν Ν to LONG Υ Υ Υ Υ Υ Υ Ν Ν Ν to FLOAT Υ Υ Υ Υ Υ Υ Ν Ν Ν Υ Υ Υ Υ Υ to Υ Ν Ν Ν **DOUBLE** Υ Υ Υ Υ Υ Υ Ν Ν Ν **BOOLEAN** to DATE Υ Ν Υ Υ Υ Ν Ν Ν Ν Υ Υ Ν Ν Ν Ν Ν to **TIMESTA** MP Υ to Υ Ν Ν Ν Ν Ν Υ **TIMESTA** MP WITH **TIMEZON** Ε

Table A-2 Type Casting Support in PGQL (From and To Types)

An example query that uses type casting is:

```
SELECT e.name, CAST (e.birthDate AS STRING) AS dob
FROM MATCH (e)
WHERE e.birthDate < CAST ('1980-01-01' AS DATE)
```

A.10.1.3 CONTAINS Built-in Function

A CONTAINS built-in function is supported. It is used in conjunction with an Oracle Text index on vertex and edge properties. CONTAINS returns true if a value matches an Oracle Text search string and false if it does not match.

An example query is:

```
SELECT v.name
FROM MATCH (v)
WHERE CONTAINS(v.abstract, 'Oracle')
```

See also Using a Text Index with PGQL Queries for more information about using full text indexes with PGQL.



A.10.2 Creating Property Graphs through CREATE PROPERTY GRAPH Statements

You can use PGQL to create property graphs from relational database tables. A CREATE PROPERTY GRAPH statement defines a set of vertex tables that are transformed into vertices and a set of edge tables that are transformed into edges. For each table a key, a label and a set of column properties can be specified. The column types CHAR, NCHAR, VARCHAR2, NVARCHAR2, NUMBER, LONG, FLOAT, DATE, TIMESTAMP and TIMESTAMP WITH TIMEZONE are supported for CREATE PROPERTY GRAPH column properties.

When a CREATE PROPERTY GRAPH statement is called, a property graph schema for the graph is created, and the data is copied from the source tables into the property graph schema tables. The graph is created as a one-time copy and is not automatically kept in sync with the source data.

Example A-24 PgglCreateExample1.java

This example shows how to create a property graph from a set of relational tables. Notice that the example creates tables Person, Hobby, and Hobbies, so they should not exist before running the example. The example also shows how to execute a query against a property graph.

```
import java.sql.Connection;
import java.sql.Statement;
import oracle.pg.rdbms.pgql.PgqlConnection;
import oracle.pg.rdbms.pgql.PgqlResultSet;
import oracle.pg.rdbms.pgql.PgqlStatement;
import oracle.ucp.jdbc.PoolDataSourceFactory;
import oracle.ucp.jdbc.PoolDataSource;
 * This example shows how to create a Property Graph from relational
 * data stored in Oracle Database executing a PGQL create statement.
public class PgqlCreateExample1
 public static void main(String[] args) throws Exception
    int idx=0;
    String host
                             = args[idx++];
    String port
                             = args[idx++];
    String sid
                             = args[idx++];
   String user
                             = args[idx++];
    String password
                            = args[idx++];
    String graph
                             = args[idx++];
    Connection conn = null;
    Statement stmt = null;
    PgqlStatement pgqlStmt = null;
    PgglResultSet rs = null;
```



```
try {
      //Get a jdbc connection
      PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
      pds.setURL("jdbc:oracle:thin:@"+host+":"+port +":"+sid);
     pds.setUser(user);
     pds.setPassword(password);
      conn = pds.getConnection();
      conn.setAutoCommit(false);
      // Create relational data
      stmt = conn.createStatement();
      //Table Person
      stmt.executeUpdate(
       "create table Person( " +
       " id NUMBER,
       " name VARCHAR2(20), " +
        " dob TIMESTAMP
       ")");
      // Insert some data
      stmt.executeUpdate("insert into Person values(1,'Alan', DATE
'1995-05-26')");
      stmt.executeUpdate("insert into Person values(2, 'Ben', DATE
'2007-02-15')");
      stmt.executeUpdate("insert into Person values(3,'Claire', DATE
'1967-11-30')");
      // Table Hobby
      stmt.executeUpdate(
        "create table Hobby(
        " id NUMBER,
                              " +
        " name VARCHAR2(20) " +
       ")");
      // Insert some data
      stmt.executeUpdate("insert into Hobby values(1, 'Sports')");
      stmt.executeUpdate("insert into Hobby values(2, 'Music')");
      // Table Hobbies
      stmt.executeUpdate(
        "create table Hobbies ("+
       " person NUMBER, "+
       " hobby NUMBER, "+
       " strength NUMBER "+
        ")");
      // Insert some data
      stmt.executeUpdate("insert into Hobbies values(1, 1, 20)");
      stmt.executeUpdate("insert into Hobbies values(1, 2, 30)");
      stmt.executeUpdate("insert into Hobbies values(2, 1, 10)");
```

```
stmt.executeUpdate("insert into Hobbies values(3, 2, 20)");
  //Commit changes
  conn.commit();
  // Get a PGQL connection
  PgqlConnection pgqlConn = PgqlConnection.getConnection(conn);
  // Create a PgqlStatement
 pgqlStmt = pgqlConn.createStatement();
  // Execute PGQL to create property graph
  String pgql =
   "Create Property Graph " + graph + " " +
   "VERTEX TABLES ( " +
   " Person " +
        Key(id) " +
       Label \"people\"" +
       PROPERTIES (name AS \"first name\", dob AS \"birthday\")," +
   " Hobby " +
        Key(id) Label \"hobby\" PROPERTIES(name AS \"name\")" +
   ")" +
   "EDGE TABLES (" +
   " Hobbies" +
        SOURCE KEY(person) REFERENCES Person (id) " +
        DESTINATION KEY(hobby) REFERENCES Hobby (id) " +
        LABEL \"likes\" PROPERTIES (strength AS \"score\")" +
   ")":
 pgqlStmt.execute(pgql);
 // Execute a PGQL query to verify Graph creation
 paal =
   "SELECT p.\"first name\", p.\"birthday\", h.\"name\", e.\"score\" " +
   "FROM MATCH (p:\"people\")-[e:\"likes\"]->(h:\"hobby\") ON " + graph;
 rs = pgqlStmt.executeQuery(pgql, "");
 // Print the results
 rs.print();
finally {
 // close the sql statment
 if (stmt != null) {
   stmt.close();
  // close the result set
 if (rs != null) {
   rs.close();
 // close the statement
 if (pgqlStmt != null) {
   pgqlStmt.close();
  // close the connection
 if (conn != null) {
   conn.close();
```

```
} }
```

The output for PgqlCreateExample1.java is:

Example A-25 PgqlCreateExample2.java

This example shows how a create property graph statement without specifying any keys. Notice that the example creates tables Person, Hobby, and Hobbies, so they should not exist before running the example.

```
import java.sql.Connection;
import java.sql.Statement;
import oracle.pg.rdbms.pgql.PgqlConnection;
import oracle.pg.rdbms.pgql.PgqlResultSet;
import oracle.pg.rdbms.pgql.PgqlStatement;
import oracle.ucp.jdbc.PoolDataSourceFactory;
import oracle.ucp.jdbc.PoolDataSource;
/**
* This example shows how to create a Property Graph from relational
* data stored in Oracle Database executing a PGQL create statement.
public class PgqlCreateExample2
 public static void main(String[] args) throws Exception
   int idx=0;
   String host
                             = args[idx++];
   String port
                           = args[idx++];
   String sid
                           = args[idx++];
                           = args[idx++];
   String user
   String password
                           = args[idx++];
   String graph
                             = args[idx++];
   Connection conn = null;
   Statement stmt = null;
   PgqlStatement pgqlStmt = null;
   PgglResultSet rs = null;
   try {
```



```
//Get a jdbc connection
      PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
      pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
     pds.setURL("jdbc:oracle:thin:@"+host+":"+port +":"+sid);
     pds.setUser(user);
     pds.setPassword(password);
     conn = pds.getConnection();
      conn.setAutoCommit(false);
      // Create relational data
     stmt = conn.createStatement();
     //Table Person
      stmt.executeUpdate(
       "create table Person( " +
       " id NUMBER,
                             " +
       " name VARCHAR2(20), " +
       " dob TIMESTAMP,
                              " +
       " CONSTRAINT pk person PRIMARY KEY(id)" +
       ")");
      // Insert some data
      stmt.executeUpdate("insert into Person values(1,'Alan', DATE
'1995-05-26')");
      stmt.executeUpdate("insert into Person values(2, 'Ben', DATE
'2007-02-15')");
      stmt.executeUpdate("insert into Person values(3,'Claire', DATE
'1967-11-30')");
      // Table Hobby
      stmt.executeUpdate(
       "create table Hobby( " +
       " id NUMBER,
       " name VARCHAR2(20), " +
       " CONSTRAINT pk hobby PRIMARY KEY(id)" +
      // Insert some data
      stmt.executeUpdate("insert into Hobby values(1, 'Sports')");
     stmt.executeUpdate("insert into Hobby values(2, 'Music')");
     // Table Hobbies
      stmt.executeUpdate(
       "create table Hobbies ( "+
       " person NUMBER, "+
       " hobby
                   NUMBER, "+
       " strength NUMBER, "+
       " CONSTRAINT fk hobbies1 FOREIGN KEY (person) REFERENCES
Person(id), "+
       " CONSTRAINT fk hobbies2 FOREIGN KEY (hobby) REFERENCES Hobby(id)"+
       ")");
      // Insert some data
      stmt.executeUpdate("insert into Hobbies values(1, 1, 20)");
     stmt.executeUpdate("insert into Hobbies values(1, 2, 30)");
```

```
stmt.executeUpdate("insert into Hobbies values(2, 1, 10)");
  stmt.executeUpdate("insert into Hobbies values(3, 2, 20)");
  //Commit changes
  conn.commit();
  // Get a PGQL connection
  PgqlConnection pgqlConn = PgqlConnection.getConnection(conn);
  // Create a PgqlStatement
  pgqlStmt = pgqlConn.createStatement();
  // Execute PGQL to create property graph
  String pggl =
    "Create Property Graph " + graph + " " +
    "VERTEX TABLES ( " +
    " Person " +
       Label people +
       PROPERTIES ALL COLUMNS," +
    " Hobby " +
        Label hobby PROPERTIES ALL COLUMNS EXCEPT (id) " +
    ")" +
    "EDGE TABLES (" +
    " Hobbies" +
       SOURCE Person DESTINATION Hobby " +
        LABEL likes NO PROPERTIES" +
    ")";
  pgqlStmt.execute(pgql);
  // Execute a PGQL query to verify Graph creation
 pgql =
   "SELECT p.NAME AS person, p.DOB, h.NAME AS hobby " +
   "FROM MATCH (p:people) - [e:likes] -> (h:hobby) ON " + graph;
  rs = pgqlStmt.executeQuery(pgql, "");
  // Print the results
 rs.print();
finally {
  // close the sql statment
 if (stmt != null) {
   stmt.close();
  // close the result set
 if (rs != null) {
   rs.close();
  // close the statement
  if (pgqlStmt != null) {
   pgqlStmt.close();
  // close the connection
 if (conn != null) {
    conn.close();
  }
```



```
}
```

The output for PgqlCreateExample2.java is:

A.10.3 Dropping Property Graphs through DROP PROPERTY GRAPH Statements

You can use PGQL to drop property graphs. When a DROP PROPERTY GRAPH statement is called, all the property graph schema tables of the graph are dropped.

Example A-26 PgqlDropExample1.java

This example shows how to drop a property graph.

```
import java.sql.Connection;
import oracle.pg.rdbms.pggl.PgglConnection;
import oracle.pg.rdbms.pgql.PgqlStatement;
import oracle.ucp.jdbc.PoolDataSourceFactory;
import oracle.ucp.jdbc.PoolDataSource;
/**
 * This example shows how to drop a Property executing a PGQL drop statement.
public class PgqlDropExample1
 public static void main(String[] args) throws Exception
   int idx=0;
   String host
                           = args[idx++];
   String port
                           = args[idx++];
                           = args[idx++];
   String sid
   String user
                            = args[idx++];
   String password
String graph
                           = args[idx++];
                            = args[idx++];
    String graph
   Connection conn = null;
    PgqlStatement pgqlStmt = null;
    try {
      //Get a jdbc connection
```



```
PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
      pds.setURL("jdbc:oracle:thin:@"+host+":"+port +":"+sid);
      pds.setUser(user);
      pds.setPassword(password);
      conn = pds.getConnection();
      conn.setAutoCommit(false);
      // Get a PGQL connection
      PgqlConnection pgqlConn = PgqlConnection.getConnection(conn);
      // Create a PgqlStatement
      pgqlStmt = pgqlConn.createStatement();
      // Execute PGQL to drop property graph
      String pgql = "Drop Property Graph " + graph;
      pgqlStmt.execute(pgql);
    finally {
      // close the statement
      if (pgqlStmt != null) {
        pgqlStmt.close();
      // close the connection
      if (conn != null) {
        conn.close();
```

A.10.4 Using the oracle.pg.rdbms.pgql Java Package to Execute PGQL Queries

The Java API in the <code>oracle.pg.rdbms.pgql</code> package provides support for executing PGQL queries against Oracle Database. This topic explains how to use the Java API through a series of examples.



Note:

Effective with Release 21c, the following classes in the oracle.pg.rdbms package are deprecated:

```
oracle.pg.rdbms.OraclePgglColumnDescriptorImpl
oracle.pg.rdbms.OraclePgglColumnDescriptor
oracle.pg.rdbms.OraclePgglExecutionFactory
oracle.pg.rdbms.OraclePgglExecution
oracle.pg.rdbms.PgglPreparedStatement
oracle.pg.rdbms.OraclePgqlResultElementImpl
oracle.pg.rdbms.OraclePgglResultElement
oracle.pg.rdbms.OraclePgqlResultImpl
oracle.pg.rdbms.OraclePgqlResultIterable
oracle.pg.rdbms.OraclePgqlResultIteratorImpl
oracle.pg.rdbms.OraclePgqlResult
oracle.pg.rdbms.OraclePgqlResultSetImpl
oracle.pg.rdbms.OraclePgglResultSet
oracle.pg.rdbms.OraclePgglResultSetMetaDataImpl
oracle.pg.rdbms.OraclePgglResultSetMetaData
oracle.pg.rdbms.PgqlSqlQueryTransImpl
oracle.pg.rdbms.PgqlSqlQueryTrans
oracle.pg.rdbms.PgqlStatement
```

You should instead use equivalent classes in oracle.pg.rdbms.pgql:

```
oracle.pg.rdbms.pgql.PgqlColumnDescriptorImpl
oracle.pg.rdbms.pgql.PgqlColumnDescriptor
oracle.pg.rdbms.pgql.PgqlConnection
oracle.pg.rdbms.pgql.PgqlExecution
oracle.pg.rdbms.pgql.PgqlPreparedStatement
oracle.pg.rdbms.pgql.PgqlResultElementImpl
oracle.pg.rdbms.pgql.PgqlResultElement
oracle.pg.rdbms.pgql.PgqlResultSetImpl
oracle.pg.rdbms.pgql.PgqlResultSet
oracle.pg.rdbms.pgql.PgqlResultSet
oracle.pg.rdbms.pgql.PgqlResultSetMetaDataImpl
oracle.pg.rdbms.pgql.PgqlSqlTransImpl
oracle.pg.rdbms.pgql.PgqlSqlTrans
oracle.pg.rdbms.pgql.PgqlSqlTrans
```

One difference between <code>oracle.pg.rdbms.OraclePgqlResultSet</code> and <code>oracle.pg.rdbms.pgql.PgqlResultSet</code> is that <code>oracle.pg.rdbms.pgql.PgqlResultSet</code> does not provide APIs to retrieve vertex and edge objects. Existing code using those interfaces should be changed to project IDs rather than <code>OracleVertex</code> and <code>OracleEdge</code> objects. You can obtain an <code>OracleVertex</code> or <code>OracleEdge</code> object from the projected ID values by calling <code>OracleVertex.getInstance()</code> or <code>OracleEdge.getInstance()</code>. (For an example, see <code>Example A-41.)</code>

See Oracle Graph Property Graph Java APIs for more details on setting the classpath for compiling and executing your Java applications.

The following test_graph data set in Oracle flat file format will be used in the examples in subtopics that follow. The data set includes a vertex file (test_graph.opv) and an edge file (test_graph.ope).

```
test graph.opv:
```



```
2, fname, 1, Ray, , , person
2, lname, 1, Green, , , person
2, mval, 5, , , 1985-01-01T12:00:00.000Z, person
2, age, 2,, 41,, person
0, bval, 6, Y, , , person
0, fname, 1, Bill, , , person
0, lname, 1, Brown, , , person
0, mval, 1, y, , , person
0, age, 2,, 40,, person
1, bval, 6, Y, , , person
1, fname, 1, John, , , person
1, lname, 1, Black, , , person
1, mval, 2, , 27, , person
1, age, 2,, 30,, person
3, bval, 6, N, , , person
3, fname, 1, Susan, , , person
3, lname, 1, Blue, , , person
3, mval, 6, N, , , person
3, age, 2, , 35, , person
test graph.ope:
4,0,1,knows,mval,1,Y,,
4,0,1,knows,firstMetIn,1,MI,,
4,0,1,knows,since,5,,,1990-01-01T12:00:00.000Z
16,0,1,friendOf,strength,2,,6,
7,1,0,knows,mval,5,,,2003-01-01T12:00:00.000Z
7,1,0,knows,firstMetIn,1,GA,,
7,1,0,knows,since,5,,,2000-01-01T12:00:00.000Z
17,1,0,friendOf,strength,2,,7,
9,1,3,knows,mval,6,N,,
9,1,3,knows,firstMetIn,1,SC,,
9,1,3,knows,since,5,,,2005-01-01T12:00:00.000Z
10,2,0,knows,mval,1,N,,
10,2,0,knows,firstMetIn,1,TX,,
10,2,0,knows,since,5,,,1997-01-01T12:00:00.000Z
12,2,3,knows,mval,3,,342.5,
12,2,3,knows,firstMetIn,1,TX,,
12,2,3,knows,since,5,,,2011-01-01T12:00:00.000Z
19,2,3,friendOf,strength,2,,4,
14,3,1, knows, mval, 1, a,,
14,3,1,knows,firstMetIn,1,CA,,
14,3,1,knows,since,5,,,2010-01-01T12:00:00.000Z
15,3,2,knows,mval,1,z,,
15,3,2,knows,firstMetIn,1,CA,,
15,3,2,knows,since,5,,,2004-01-01T12:00:00.000Z
5,0,2,knows,mval,2,,23,
5,0,2,knows,firstMetIn,1,OH,,
5,0,2,knows,since,5,,,2002-01-01T12:00:00.000Z
6,0,3,knows,mval,3,,159.7,
6,0,3,knows,firstMetIn,1,IN,,
6,0,3,knows,since,5,,,1994-01-01T12:00:00.000Z
8,1,2,knows,mval,6,Y,,
8,1,2,knows,firstMetIn,1,FL,,
8,1,2,knows,since,5,,,1999-01-01T12:00:00.000Z
18,1,3,friendOf,strength,2,,5,
11,2,1,knows,mval,2,,1001,
11, 2, 1, knows, firstMetIn, 1, OK, ,
11,2,1,knows,since,5,,,2003-01-01T12:00:00.000Z
13,3,0,knows,mval,5,,,2001-01-01T12:00:00.000Z
13,3,0,knows,firstMetIn,1,CA,,
```



```
13,3,0,knows,since,5,,,2006-01-01T12:00:00.000Z 20,3,1,friendOf,strength,2,,3,
```

- Basic Query Execution
- Security Techniques for PGQL Queries
- Using a Text Index with PGQL Queries
- Obtaining the SQL Translation for a PGQL Query
- Additional Options for PGQL Translation and Execution
- · Querying Another User's Property Graph
- Using Query Optimizer Hints with PGQL
- Modifying Property Graphs through INSERT, UPDATE, and DELETE Statements

A.10.4.1 Basic Query Execution

Two main Java Interfaces, PgqlStatement and PgqlResultSet, are used for PGQL execution. This topic includes several examples of basic query execution.

Example A-27 GraphLoaderExample.java

GraphLoaderExample.java loads some Oracle property graph data that will be used in subsequent examples in this topic.

```
import oracle.pq.rdbms.Oracle;
import oracle.pg.rdbms.OraclePropertyGraph;
import oracle.pg.rdbms.OraclePropertyGraphDataLoader;
 * This example shows how to create an Oracle Property Graph
 * and load data into it from vertex and edge flat files.
public class GraphLoaderExample
 public static void main(String[] args) throws Exception
    int idx=0;
   String host
                             = args[idx++];
                         = args[idx++];
   String port
                             = args[idx++];
   String user
String password
String graph
    String sid
                           = args[idx++];
= args[idx++];
                             = args[idx++];
    String vertexFile = args[idx++];
String edgeFile = args[idx++];
    String edgeFile
                               = args[idx++];
    Oracle oracle = null;
    OraclePropertyGraph opg = null;
    trv {
      // Create a connection to Oracle
      oracle = new Oracle("jdbc:oracle:thin:@"+host+":"+port +":"+sid, user,
password);
```



```
// Create a property graph
      opg = OraclePropertyGraph.getInstance(oracle, graph);
      // Clear any existing data
      opg.clearRepository();
      // Load data from opv and ope files
      OraclePropertyGraphDataLoader opgLoader =
OraclePropertyGraphDataLoader.getInstance();
      opgLoader.loadData(opg, vertexFile, edgeFile, 1);
      System.out.println("Vertices loaded:" + opg.countVertices());
      System.out.println("Edges loaded:" + opg.countEdges());
    finally {
      // close the property graph
      if (opg != null) {
        opg.close();
      // close oracle
      if (oracle != null) {
        oracle.dispose();
    }
```

GraphLoaderExample.java gives the following output for test graph.

```
Vertices loaded:4
Edges loaded:17
```

Example A-28 PgqlExample1.java

PgqlExample1.java executes a PGQL query and prints the query result. PgqlConnection is used to obtain a PgqlStatement. Next, it calls the executeQuery method of PgqlStatement, which returns a PgqlResultSet object. PgqlResultSet provides a print() method, which shows results in a tabular mode.

The PgqlResultSet and PgqlStatement objects should be closed after consuming the query result.

```
import java.sql.Connection;
import oracle.pg.rdbms.pgql.PgqlConnection;
import oracle.pg.rdbms.pgql.PgqlResultSet;
import oracle.pg.rdbms.pgql.PgqlStatement;
import oracle.ucp.jdbc.PoolDataSourceFactory;
import oracle.ucp.jdbc.PoolDataSource;

/**
    * This example shows how to execute a basic PGQL query against disk-
```

```
resident
 ^{\star} PG data stored in Oracle Database and iterate through the result.
public class PgqlExample1
  public static void main(String[] args) throws Exception
    int idx=0;
    String host
                              = args[idx++];
    String port
                             = args[idx++];
    String sid
                            = args[idx++];
    String user
                            = args[idx++];
    String password
                            = args[idx++];
    String graph
                            = args[idx++];
    Connection conn = null;
    PgqlStatement ps = null;
    PgglResultSet rs = null;
    try {
      //Get a jdbc connection
      PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
      pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
      pds.setURL("jdbc:oracle:thin:@"+host+":"+port +":"+sid);
      pds.setUser(user);
      pds.setPassword(password);
      conn = pds.getConnection();
      // Get a PGQL connection
      PgqlConnection pgqlConn = PgqlConnection.getConnection(conn);
      pgqlConn.setGraph(graph);
      // Create a PgqlStatement
      ps = pgqlConn.createStatement();
      // Execute query to get a PgqlResultSet object
      String pgql =
        "SELECT v.\"fname\" AS fname, v.\"lname\" AS lname, v.\"mval\" AS
mval "+
        "FROM MATCH (v)";
      rs = ps.executeQuery(pgql, /* query string */
                               /* options */);
      // Print the results
      rs.print();
    }
    finally {
      // close the result set
      if (rs != null) {
       rs.close();
      // close the statement
      if (ps != null) {
```

```
ps.close();
}
// close the connection
if (conn != null) {
    conn.close();
}
}
}
```

PgqlExample1.java gives the following output for test_graph (which can be loaded using GraphLoaderExample.java code).

Example A-29 PgqlExample2.java

PgqlExample2.java shows a PGQL query with a temporal filter on an edge property.

- PgqlResultSet provides an interface for consuming the query result that is very similar to the java.sql.ResultSet interface.
- A next() method allows moving through the query result, and a close() method allows releasing resources after the application is fiished reading the query result.
- In addition, PgqlResultSet provides getters for String, Integer, Long, Float, Double, Boolean, LocalDateTime, and OffsetDateTime, and it provides a generic getObject() method for values of any type.

```
import java.sql.Connection;
import java.text.SimpleDateFormat;
import java.util.Date;
import oracle.pg.rdbms.pgql.PgqlConnection;
import oracle.pg.rdbms.pgql.PgqlStatement;
import oracle.pgql.lang.ResultSet;
import oracle.ucp.jdbc.PoolDataSourceFactory;
import oracle.ucp.jdbc.PoolDataSource;

/**
    * This example shows how to execute a PGQL query with a temporal edge
    * property filter against disk-resident PG data stored in Oracle
Database
    * and iterate through the result.
    */
public class PgqlExample2
```



```
{
 public static void main(String[] args) throws Exception
   int idx=0;
   String host
                             = args[idx++];
   String port
                            = args[idx++];
   String sid
                            = args[idx++];
    String user
                            = args[idx++];
    String password
                            = args[idx++];
    String graph
                             = args[idx++];
    Connection conn = null;
    PgglStatement ps = null;
    ResultSet rs = null;
    try {
      //Get a jdbc connection
      PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
     pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
     pds.setURL("jdbc:oracle:thin:@"+host+":"+port +":"+sid);
     pds.setUser(user);
     pds.setPassword(password);
     conn = pds.getConnection();
      // Create a Pgql connection
      PgqlConnection pgqlConn = PgqlConnection.getConnection(conn);
      pgqlConn.setGraph(graph);
      // Create a PgqlStatement
     ps = pgqlConn.createStatement();
      // Execute query to get a ResultSet object
      String pgql =
        "SELECT v.\"fname\" AS n1, v2.\"fname\" AS n2, e.\"firstMetIn\" AS
loc "+
        "FROM MATCH (v) - [e:\ \ \ \ ) - [v2)"+
        "WHERE e.\"since\" > TIMESTAMP '2000-01-01 00:00:00.00+00:00'";
      rs = ps.executeQuery(pgql, "");
      // Print results
     printResults(rs);
    finally {
      // close the result set
     if (rs != null) {
       rs.close();
      // close the statement
      if (ps != null) {
       ps.close();
      // close the connection
      if (conn != null) {
```

```
conn.close();
      }
    }
  }
  /**
   * Prints a PGQL ResultSet
  */
 static void printResults(ResultSet rs) throws Exception
    StringBuffer buff = new StringBuffer("");
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-
dd'T'HH:mm:ss.SSSXXX");
    while (rs.next()) {
      buff.append("[");
      for (int i = 1; i <= rs.getMetaData().getColumnCount(); i++) {</pre>
        // use generic getObject to handle all types
        Object mval = rs.getObject(i);
        String mStr = "";
        if (mval instanceof java.lang.String) {
          mStr = "STRING: "+mval.toString();
        else if (mval instanceof java.lang.Integer) {
         mStr = "INTEGER: "+mval.toString();
        else if (mval instanceof java.lang.Long) {
         mStr = "LONG: "+mval.toString();
        else if (mval instanceof java.lang.Float) {
         mStr = "FLOAT: "+mval.toString();
        else if (mval instanceof java.lang.Double) {
         mStr = "DOUBLE: "+mval.toString();
        else if (mval instanceof java.sql.Timestamp) {
         mStr = "DATE: "+sdf.format((Date)mval);
        else if (mval instanceof java.lang.Boolean) {
         mStr = "BOOLEAN: "+mval.toString();
        if (i > 1) {
         buff.append(",\t");
        buff.append(mStr);
      buff.append("]\n");
    System.out.println(buff.toString());
}
```

PgqlExample2.java gives the following output for test_graph (which can be loaded using GraphLoaderExample.java code).

```
[STRING: Susan, STRING: Bill, STRING: CA]
[STRING: Susan, STRING: John, STRING: CA]
[STRING: Susan, STRING: Ray, STRING: CA]
[STRING: Bill, STRING: Ray, STRING: OH]
[STRING: Ray, STRING: John, STRING: OK]
[STRING: Ray, STRING: Susan, STRING: TX]
[STRING: John, STRING: Susan, STRING: SC]
[STRING: John, STRING: Bill, STRING: GA]
```

Example A-30 PgqlExample3.java

PgqlExample3.java shows a PGQL query with grouping and aggregation.

```
import java.sql.Connection;
import oracle.pg.rdbms.pgql.PgqlConnection;
import oracle.pg.rdbms.pgql.PgqlResultSet;
import oracle.pg.rdbms.pgql.PgqlStatement;
import oracle.ucp.jdbc.PoolDataSourceFactory;
import oracle.ucp.jdbc.PoolDataSource;
 * This example shows how to execute a PGQL query with aggregation
 * against disk-resident PG data stored in Oracle Database and iterate
 * through the result.
 * /
public class PgqlExample3
 public static void main(String[] args) throws Exception
    int idx=0;
   String host
                             = args[idx++];
   String port
                           = args[idx++];
                            = args[idx++];
   String sid
   String user
                            = args[idx++];
    String password
                           = args[idx++];
    String graph
                            = args[idx++];
    Connection conn = null;
    PgglStatement ps = null;
    PgqlResultSet rs = null;
    try {
      //Get a jdbc connection
      PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
     pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
     pds.setURL("jdbc:oracle:thin:@"+host+":"+port +":"+sid);
     pds.setUser(user);
     pds.setPassword(password);
      conn = pds.getConnection();
      // Create a Pgql connection
      PgqlConnection pgqlConn = PgqlConnection.getConnection(conn);
      pgqlConn.setGraph(graph);
```



```
// Create a PgqlStatement
     ps = pgqlConn.createStatement();
      // Execute query to get a ResultSet object
      String pgql =
        "SELECT v.\"fname\" AS \"fname\", COUNT(v2) AS \"friendCnt\" "+
       "FROM MATCH (v) - [e: \friendOf\] -> (v2) "+
       "GROUP BY v "+
        "ORDER BY \"friendCnt\" DESC";
      rs = ps.executeQuery(pgql, "");
      // Print results
      rs.print();
   finally {
      // close the result set
     if (rs != null) {
       rs.close();
      // close the statement
      if (ps != null) {
       ps.close();
      // close the connection
     if (conn != null) {
        conn.close();
   }
  }
}
```

PgqlExample3.java gives the following output for test_graph (which can be loaded using GraphLoaderExample.java code).

Example A-31 PgqlExample4.java

PgqlExample4.java shows a PGQL path query.

```
import java.sql.Connection;
import oracle.pg.rdbms.pgql.PgqlConnection;
import oracle.pg.rdbms.pgql.PgqlResultSet;
import oracle.pg.rdbms.pgql.PgqlStatement;
import oracle.ucp.jdbc.PoolDataSourceFactory;
import oracle.ucp.jdbc.PoolDataSource;
```



```
* This example shows how to execute a path query in PGQL against
 * disk-resident PG data stored in Oracle Database and iterate
 * through the result.
public class PgqlExample4
  public static void main(String[] args) throws Exception
    int idx=0;
    String host
                            = args[idx++];
    String port
                            = args[idx++];
    String sid
                            = args[idx++];
    String user
                            = args[idx++];
                            = args[idx++];
    String password
    String graph
                             = args[idx++];
    Connection conn = null;
    PgglStatement ps = null;
    PgqlResultSet rs = null;
    try {
      //Get a jdbc connection
      PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
      pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
      pds.setURL("jdbc:oracle:thin:@"+host+":"+port +":"+sid);
      pds.setUser(user);
     pds.setPassword(password);
      conn = pds.getConnection();
      // Create a Pgql connection
      PgqlConnection pgqlConn = PgqlConnection.getConnection(conn);
      pgqlConn.setGraph(graph);
      // Create a PgqlStatement
      ps = pgqlConn.createStatement();
  // Execute query to get a ResultSet object
      String pggl =
        "PATH fof AS ()-[:\"friendOf\"|\"knows\"]->() "+
        "SELECT v2.\"fname\" AS friend "+
       "FROM MATCH (v) -/: fof*/-> (v2)"+
        "WHERE v.\"fname\" = 'John' AND v != v2";
      rs = ps.executeQuery(pgql, "");
      // Print results
      rs.print();
    finally {
      // close the result set
      if (rs != null) {
       rs.close();
```

```
}
// close the statement
if (ps != null) {
    ps.close();
}
// close the connection
if (conn != null) {
    conn.close();
}
}
}
```

PgqlExample4.java gives the following output for test_graph(which can be loaded using GraphLoaderExample.java code).

```
+----+
| FRIEND |
+----+
| Susan |
| Bill |
| Ray |
```

A.10.4.2 Security Techniques for PGQL Queries

Programs executing dynamic queries might be subject to injection attacks that could compromise integrity and functioning of the applications.

This topic presents some techniques that can be used to prevent injection attacks when building PGQL queries using string concatenation.

- Using Bind Variables in PGQL Queries
- · Verifying PGQL Identifiers

A.10.4.2.1 Using Bind Variables in PGQL Queries

Bind variables can be used in PGQL queries for better performance and increased security. Constant scalar values in PGQL queries can be replaced with bind variables. Bind variables are denoted by a '?' (question mark). Consider the following two queries that select people who are older than a constant age value.

```
// people older than 30
SELECT v.fname AS fname, v.lname AS lname, v.age AS age
FROM MATCH (v)
WHERE v.age > 30

// people older than 40
SELECT v.fname AS fname, v.lname AS lname, v.age AS age
FROM MATCH (v)
WHERE v.age > 40
```

The SQL translations for these queries would use the constants 30 and 40 in a similar way for the age filter. The database would perform a hard parse for each of these queries. This hard parse time can often exceed the execution time for simple queries.

You could replace the constant in each query with a bind variable as follows.

```
SELECT v.fname AS fname, v.lname AS lname, v.age AS age FROM MATCH (v) WHERE v.age > ?
```

This will allow the SQL engine to create a generic cursor for this query, which can be reused for different age values. As a result, a hard parse is no longer required to execute this query for different age values, and the parse time for each query will be drastically reduced.

In addition, applications that use bind variables in PGQL queries are less vulnerable to injection attacks than those that use string concatenation to embed constant values in PGQL queries.

See also *Oracle Database SQL Tuning Guide* for more information on cursor sharing and bind variables.

The PgqlPreparedStatement interface can be used to execute queries with bind variables as shown in PgqlExample5.java. PgqlPreparedStatement provides several set methods for different value types that can be used to set values for query execution.

There are a few limitations with bind variables in PGQL. Bind variables can only be used for constant property values. That is, vertices and edges cannot be replaced with bind variables. Also, once a particular bind variable has been set to a type, it cannot be set to a different type. For example, if setInt(1, 30) is executed for an PgqlPreparedStatement, you cannot call setString(1, "abc") on that same PgqlPreparedStatement.

Example A-32 PgqlExample5.java

 ${\tt PgqlExample5.java} \ \ \textbf{shows how to use bind variables with a PGQL query}.$

```
import java.sql.Connection;
import oracle.pg.rdbms.pgql.PgqlConnection;
import oracle.pg.rdbms.pggl.PgglPreparedStatement;
import oracle.pg.rdbms.pggl.PgglResultSet;
import oracle.ucp.jdbc.PoolDataSourceFactory;
import oracle.ucp.jdbc.PoolDataSource;
/**
* This example shows how to use bind variables with a PGQL query.
public class PgqlExample5
 public static void main(String[] args) throws Exception
   int idx=0;
   String host
                            = args[idx++];
   String port
                           = args[idx++];
   String sid
                           = args[idx++];
   String user
                           = args[idx++];
   String password
                           = args[idx++];
   String graph
                             = args[idx++];
```



```
Connection conn = null;
    PgqlPreparedStatement pps = null;
    PgglResultSet rs = null;
    try {
      //Get a jdbc connection
      PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
      pds.setURL("jdbc:oracle:thin:@"+host+":"+port +":"+sid);
      pds.setUser(user);
      pds.setPassword(password);
      conn = pds.getConnection();
      // Create a Pgql connection
      PgqlConnection pgqlConn = PgqlConnection.getConnection(conn);
      pgqlConn.setGraph(graph);
      // Query string with a bind variable (denoted by ?)
      String pggl =
        "SELECT v.\"fname\" AS fname, v.\"lname\" AS lname, v.\"age\"
AS age "+
        "FROM MATCH (v) "+
        "WHERE v.\"age\" > ?";
      // Create a PgqlPreparedStatement
      pps = pgqlConn.prepareStatement(pgql);
      // Set filter value to 30
     pps.setInt(1, 30);
      // execute query
      rs = pps.executeQuery();
      // Print query results
      System.out.println("-- Values for v.\"age\" > 30 --");
      rs.print();
      // close result set
      rs.close();
      // set filter value to 40
      pps.setInt(1, 40);
      // execute query
      rs = pps.executeQuery();
      // Print query results
      System.out.println("-- Values for v.\"age\" > 40 --");
      rs.print();
      // close result set
      rs.close();
    finally {
      // close the result set
```

```
if (rs != null) {
    rs.close();
}
// close the statement
if (pps != null) {
    pps.close();
}
// close the connection
if (conn != null) {
    conn.close();
}
}
}
```

PgqlExample5.java has the following output for test_graph (which can be loaded using GraphLoaderExample.java code).

Example A-33 PgqlExample6.java

PgqlExample6.java shows a query with two bind variables: one String variable and one Timestamp variable.

```
import java.sql.Connection;
import java.sql.Timestamp;

import java.time.OffsetDateTime;
import java.time.ZoneOffset;

import oracle.pg.rdbms.pgql.PgqlConnection;
import oracle.pg.rdbms.pgql.PgqlPreparedStatement;
import oracle.pg.rdbms.pgql.PgqlResultSet;

import oracle.ucp.jdbc.PoolDataSourceFactory;
import oracle.ucp.jdbc.PoolDataSource;

/**
    * This example shows how to use multiple bind variables with a PGQL query.
    */
public class PgqlExample6
{
```



```
public static void main(String[] args) throws Exception
    int idx=0;
    String host
                             = args[idx++];
    String port
                            = args[idx++];
    String sid
                             = args[idx++];
    String user
                            = args[idx++];
    String password
                            = args[idx++];
    String graph
                             = args[idx++];
    Connection conn = null;
    PgqlPreparedStatement pps = null;
    PgglResultSet rs = null;
    try {
      //Get a jdbc connection
      PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
      pds.setURL("jdbc:oracle:thin:@"+host+":"+port +":"+sid);
      pds.setUser(user);
      pds.setPassword(password);
      conn = pds.getConnection();
      // Create a Pggl connection
      PgqlConnection pgqlConn = PgqlConnection.getConnection(conn);
      pgqlConn.setGraph(graph);
      // Query string with multiple bind variables
      String pggl =
        "SELECT v1.\"fname\" AS fname1, v2.\"fname\" AS fname2 "+
        "FROM MATCH (v1)-[e:\"knows\"]->(v2) "+
        "WHERE e.\"since\" < ? AND e.\"firstMetIn\" = ?";
      // Create a PgglPreparedStatement
      pps = pgqlConn.prepareStatement(pgql);
      // Set e.since < 2006-01-01T12:00:00.00Z
      Timestamp t =
Timestamp.valueOf(OffsetDateTime.parse("2006-01-01T12:00:01.002").atZon
eSameInstant(ZoneOffset.UTC).toLocalDateTime());
      pps.setTimestamp(1, t);
      // Set e.firstMetIn = 'CA'
      pps.setString(2, "CA");
      // execute query
      rs = pps.executeQuery();
      // Print query results
      System.out.println("-- Values for e.\"since\" <</pre>
2006-01-01T12:00:01.00Z AND e.\"firstMetIn\" = 'CA' --");
      rs.print();
      // close result set
```



```
rs.close();
      // Set e.since < 2000-01-01T12:00:00.00Z
Timestamp.valueOf(OffsetDateTime.parse("2000-01-01T12:00:00.00Z").atZoneSameI
nstant(ZoneOffset.UTC).toLocalDateTime());
      pps.setTimestamp(1, t);
      // Set e.firstMetIn = 'TX'
      pps.setString(2, "TX");
      // execute query
      rs = pps.executeQuery();
      // Print query results
      System.out.println("-- Values for e.\"since\" <</pre>
2000-01-01T12:00:00.00Z AND e.\"firstMetIn\" = 'TX' --");
      rs.print();
      // close result set
      rs.close();
    }
    finally {
      // close the result set
      if (rs != null) {
       rs.close();
      // close the statement
      if (pps != null) {
        pps.close();
      // close the connection
      if (conn != null) {
        conn.close();
    }
}
```

PgqlExample6.java gives the following output for test_graph (which can be loaded using GraphLoaderExample.java code).



A.10.4.2.2 Verifying PGQL Identifiers

For some parts of a PGQL query the parser does not allow use of bind variables. In such cases, the input can be verified using the printIdentifier method in package oracle.pgql.lang.ir.PgqlUtils.

Consider the following query execution that concatenates the graph against which the graph pattern will be matched:

```
stmt.executeQuery("SELECT n.name FROM MATCH (n) ON " + graphName, "");
```

In order to avoid injection, the identifier graphName should be verified as follows:

```
stmt.executeQuery("SELECT n.name FROM MATCH (n) ON " +
PgqlUtils.printIdentifier(graphName), "");
```

A.10.4.3 Using a Text Index with PGQL Queries

PGQL queries executed against Oracle Database can use Oracle Text indexes created for vertex and edge properties. After creating a text index, you can use the CONTAINS operator to perform a full text search. CONTAINS has two arguments: a vertex or edge property, and an Oracle Text search string. Any valid Oracle Text search string can be used, including advanced features such as wildcards, stemming, and soundex.

Example A-34 PgqlExample7.java

PgqlExample7.java shows how to execute a CONTAINS query.

```
import java.sql.CallableStatement;
import java.sql.Connection;
import oracle.pg.rdbms.pggl.PgglConnection;
import oracle.pg.rdbms.pgql.PgqlResultSet;
import oracle.pq.rdbms.pqql.PqqlStatement;
import oracle.ucp.jdbc.PoolDataSourceFactory;
import oracle.ucp.jdbc.PoolDataSource;
/**
 * This example shows how to use an Oracle Text index with a PGQL
query.
 * /
public class PgglExample7
 public static void main(String[] args) throws Exception
    int idx=0;
    String host
                            = args[idx++];
                            = args[idx++];
    String port
    String sid
                            = args[idx++];
    String user
                             = args[idx++];
```



```
String password
                          = args[idx++];
String graph
                          = args[idx++];
Connection conn = null;
PgglStatement ps = null;
PgqlResultSet rs = null;
try {
  //Get a jdbc connection
  PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
 pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
 pds.setURL("jdbc:oracle:thin:@"+host+":"+port +":"+sid);
 pds.setUser(user);
 pds.setPassword(password);
  conn = pds.getConnection();
  // Create text index with SQL API
 CallableStatement cs = null;
  // text index on vertices
  cs = conn.prepareCall(
    "begin opg apis.create vertices text idx(:1,:2); end;"
  cs.setString(1,user);
  cs.setString(2,graph);
  cs.execute();
  cs.close();
  // text index on edges
  cs = conn.prepareCall(
    "begin opg apis.create edges text idx(:1,:2); end;"
  cs.setString(1,user);
  cs.setString(2,graph);
  cs.execute();
  cs.close();
  // Get a PGQL connection
  PgqlConnection pgqlConn = PgqlConnection.getConnection(conn);
  pgqlConn.setGraph(graph);
  // Create a PgglStatement
 ps = pgqlConn.createStatement();
  // Query using CONTAINS text search operator on vertex property
  // Find all vertices with an lname property value that starts with 'B'
  String pgql =
    "SELECT v.\"fname\" AS fname, v.\"lname\" AS lname "+
    "FROM MATCH (v) "+
    "WHERE CONTAINS (v.\"lname\", 'B%')";
  // execute query
  rs = ps.executeQuery(pggl, "");
  // print results
  System.out.println("-- Vertex Property Query --");
```

```
rs.print();
      // close result set
      rs.close();
      // Query using CONTAINS text search operator on edge property
      // Find all knows edges with a firstMetIn property value that
ends with 'A'
      pgql =
        "SELECT v1.\"fname\" AS fname1, v2.\"fname\" AS fname2,
e.\"firstMetIn\" AS loc "+
        "FROM MATCH (v1) - [e:\ \ \ \ \ \ \ ] -> (v2) "+
        "WHERE CONTAINS (e.\"firstMetIn\",'%A')";
      // execute query
      rs = ps.executeQuery(pgql, "");
      // print results
      System.out.println("-- Edge Property Query --");
      rs.print();
    }
    finally {
      // close the result set
      if (rs != null) {
       rs.close();
      // close the statement
      if (ps != null) {
       ps.close();
      // close the connection
      if (conn != null) {
        conn.close();
    }
  }
}
```

PgqlExample7.java has the following output for test_graph (which can be loaded using GraphLoaderExample.java code).



```
| Susan | John | CA |
| Susan | Ray | CA |
```

A.10.4.4 Obtaining the SQL Translation for a PGQL Query

You can obtain the SQL translation for a PGQL query through methods in PgqlStatement and PgqlPreparedStatement. The raw SQL for a PGQL query can be useful for several reasons:

- You can execute the SQL directly against the database with other SQL-based tools or interfaces (for example, SQL*Plus or SQL Developer).
- You can customize and tune the generated SQL to optimize performance or to satisfy a particular requirement of your application.
- You can build a larger SQL query that joins a PGQL subquery with other data stored in Oracle Database (such as relational tables, spatial data, and JSON data).

Example A-35 PgqlExample8.java

PgqlExample8.java shows how to obtain the raw SQL translation for a PGQL query. The translateQuery method of PgqlStatement returns an PgqlSqlQueryTrans object that contains information about return columns from the query and the SQL translation itself.

The translated SQL returns different columns depending on the type of "logical" object or value projected from the PGQL query. A vertex or edge projected in PGQL has two corresponding columns projected in the translated SQL:

- \$IT: id type NVARCHAR(1): 'V' for vertex or 'E' for edge
- \$ID: vertex or edge identifier NUMBER: same content as VID or EID columns in VT\$ and GE\$ tables

A property value or constant scalar value projected in PGQL has four corresponding columns projected in the translated SQL:

- \$T : value type NUMBER: same content as T column in VT\$ and GE\$ tables
- \$V: value NVARCHAR2(15000): same content as V column in VT\$ and GE\$ tables
- \$VN: number value NUMBER: same content as VN column in VT\$ and GE\$ tables
- \$VT: temporal value TIMESTAMP WITH TIME ZONE: same content as VT column in VT\$ and GE\$ tables

```
import java.sql.Connection;
import oracle.pg.rdbms.pgql.PgqlColumnDescriptor;
import oracle.pg.rdbms.pgql.PgqlConnection;
import oracle.pg.rdbms.pgql.PgqlStatement;
import oracle.pg.rdbms.pgql.PgqlSqlQueryTrans;

import oracle.ucp.jdbc.PoolDataSourceFactory;
import oracle.ucp.jdbc.PoolDataSource;

/**
    * This example shows how to obtain the SQL translation for a PGQL query.
    */
public class PgqlExample8
{
```



```
public static void main(String[] args) throws Exception
   int idx=0;
   String host
                           = args[idx++];
   String port
                           = args[idx++];
   String sid
                           = args[idx++];
   String user
                           = args[idx++];
   String password
                           = args[idx++];
    String graph
                            = args[idx++];
    Connection conn = null;
    PgqlStatement ps = null;
    try {
      //Get a jdbc connection
      PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
      pds.setURL("jdbc:oracle:thin:@"+host+":"+port +":"+sid);
     pds.setUser(user);
     pds.setPassword(password);
      conn = pds.getConnection();
      // Create a Pgql connection
      PgqlConnection pgqlConn = PgqlConnection.getConnection(conn);
      pgqlConn.setGraph(graph);
      // PGQL query to be translated
      String pggl =
        "SELECT v1, v1.\"fname\" AS fname1, e, e.\"since\" AS since "+
        "FROM MATCH (v1) - [e:\ \ \ \ \ \ \ ] -> (v2)";
      // Create a PgqlStatement
     ps = pgqlConn.createStatement();
      // Get the SQL translation
      PgqlSqlQueryTrans sqlTrans = ps.translateQuery(pgql,"");
      // Get the return column descriptions
      PgqlColumnDescriptor[] cols = sqlTrans.getReturnTypes();
      // Print column descriptions
      System.out.println("-- Return Columns -----");
      printReturnCols(cols);
      // Print SQL translation
      System.out.println("-- SQL Translation -----");
      System.out.println(sqlTrans.getSqlTranslation());
    finally {
      // close the statement
      if (ps != null) {
       ps.close();
```

```
}
      // close the connection
      if (conn != null) {
        conn.close();
  }
   * Prints return columns for a SQL translation
  static void printReturnCols(PgqlColumnDescriptor[] cols) throws Exception
    StringBuffer buff = new StringBuffer("");
    for (int i = 0; i < cols.length; i++) {
      String colName = cols[i].getColName();
      PgglColumnDescriptor.Type colType = cols[i].getColType();
      int offset = cols[i].getSqlOffset();
      String readableType = "";
      switch(colType) {
        case VERTEX:
          readableType = "VERTEX";
          break;
        case EDGE:
          readableType = "EDGE";
          break;
        case VALUE:
          readableType = "VALUE";
          break;
      }
      buff.append("colName=["+colName+"] colType=["+readableType+"]
offset=["+offset+"]\n");
    System.out.println(buff.toString());
}
PgqlExample8.java has the following output for test graph (which can be loaded using
GraphLoaderExample.java code).
-- Return Columns -----
colName=[v1] colType=[VERTEX] offset=[1]
colName=[fname1] colType=[VALUE] offset=[3]
colName=[e] colType=[EDGE] offset=[7]
colName=[since] colType=[VALUE] offset=[9]
-- SQL Translation -----
SELECT n'V' AS "V1$IT",
TO$0.SVID AS "V1$ID",
TO$1.T AS "FNAME1$T",
TO$1.V AS "FNAME1$V",
TO$1.VN AS "FNAME1$VN",
TO$1.VT AS "FNAME1$VT",
```



```
n'E' AS "E$IT",
TO$0.EID AS "E$ID",
TO$0.T AS "SINCE$T",
TO$0.V AS "SINCE$V",
TO$0.VN AS "SINCE$VN",
T0$0.VT AS "SINCE$VT"
FROM ( SELECT L.EID, L.SVID, L.DVID, L.EL, R.K, R.T, R.V, R.VN, R.VT
  FROM "SCOTT".TEST GRAPHGT$ L,
      (SELECT * FROM "SCOTT".TEST GRAPHGE$ WHERE K=n'since' ) R
 WHERE L.EID = R.EID(+)
) TO$0,
( SELECT L.VID, L.VL, R.K, R.T, R.V, R.VN, R.VT
  FROM "SCOTT". TEST GRAPHVD$ L,
      (SELECT * FROM "SCOTT".TEST GRAPHVT$ WHERE K=n'fname' ) R
 WHERE L.VID = R.VID(+)
) T0$1
WHERE TO$0.SVID=TO$1.VID AND
(T0$0.EL = n'knows' AND T0$0.EL IS NOT NULL)
```

Example A-36 PgqlExample9.java

You can also obtain the SQL translation for PGQL queries with bind variables. In this case, the corresponding SQL translation will also contain bind variables. The PgqlSqlQueryTrans interface has a getSqlBvList method that returns an ordered List of Java Objects that should be bound to the SQL query (the first Object on the list should be set at position 1, and the second should be set at position 2, and so on).

PgqlExample9.java shows how to get and execute the SQL for a PGQL query with bind variables.

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Timestamp;
import java.util.List;
import oracle.pg.rdbms.pgql.PgqlColumnDescriptor;
import oracle.pg.rdbms.pgql.PgqlConnection;
import oracle.pg.rdbms.pgql.PgqlPreparedStatement;
import oracle.pg.rdbms.pgql.PgqlSqlQueryTrans;
import oracle.ucp.jdbc.PoolDataSourceFactory;
import oracle.ucp.jdbc.PoolDataSource;
/**
 ^{\star} This example shows how to obtain and execute the SQL translation
for a
 * PGQL query that uses bind variables.
public class PgqlExample9
{
  public static void main(String[] args) throws Exception
    int idx=0;
    String host
                              = args[idx++];
```



```
String port
                            = args[idx++];
   String sid
                            = args[idx++];
   String user
                           = args[idx++];
   String password
                           = args[idx++];
                             = args[idx++];
   String graph
   Connection conn = null;
   PgglPreparedStatement pgglPs = null;
   PreparedStatement sqlPs = null;
   try {
     //Get a jdbc connection
     PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
     pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
     pds.setURL("jdbc:oracle:thin:@"+host+":"+port +":"+sid);
     pds.setUser(user);
     pds.setPassword(password);
     conn = pds.getConnection();
     // Create a Pgql connection
      PgqlConnection pgqlConn = PgqlConnection.getConnection(conn);
     pgqlConn.setGraph(graph);
      // Execute query to get a ResultSet object
      String pggl =
       "SELECT v1, v1.\"fname\" AS fname1, v1.\"age\" AS age, ? as constVal
"+
       "FROM MATCH (v1) "+
       "WHERE v1.\"fname\" = ? OR v1.\"age\" < ?";
      // Create a PgglStatement
     pgqlPs = pgqlConn.prepareStatement(pgql);
     // set bind values
     pgqlPs.setDouble(1, 2.05d);
     pgqlPs.setString(2, "Bill");
     pgqlPs.setInt(3, 35);
      // Get the SQL translation
      PgqlSqlQueryTrans sqlTrans = pgqlPs.translateQuery("");
      // Get the SQL String
      String sqlStr = sqlTrans.getSqlTranslation();
      // Get the return column descriptions
      PgqlColumnDescriptor[] cols = sqlTrans.getReturnTypes();
      // Get the bind values
     List<Object> bindVals = sqlTrans.getSqlBvList();
      // Print column descriptions
      System.out.println("-- Return Columns -----");
```

```
printReturnCols(cols);
     // Print SQL translation
     System.out.println("-- SQL Translation -----");
     System.out.println(sqlStr);
     // Print Bind Values
     System.out.println("\n-- Bind Values
-----");
     for (Object obj : bindVals) {
       System.out.println(obj.toString());
     // Execute Query
     // Get PreparedStatement
     sqlPs = conn.prepareStatement("SELECT COUNT(*) FROM
("+sqlStr+")");
     // Set bind values and execute the PreparedStatement
     executePs(sqlPs, bindVals);
     // Set new bind values in the PGQL PreparedStatement
     pgqlPs.setDouble(1, 3.02d);
     pgqlPs.setString(2, "Ray");
     pgqlPs.setInt(3, 30);
     // Print Bind Values
     bindVals = sqlTrans.getSqlBvList();
     System.out.println("\n-- Bind Values
----");
     for (Object obj : bindVals) {
       System.out.println(obj.toString());
     // Execute the PreparedStatement with new bind values
     executePs(sqlPs, bindVals);
   finally {
     // close the SQL statement
     if (sqlPs != null) {
       sqlPs.close();
     // close the statement
     if (pgqlPs != null) {
       pgqlPs.close();
     // close the connection
     if (conn != null) {
       conn.close();
 }
  * Executes a SQL PreparedStatement with the input bind values
  */
```

```
static void executePs(PreparedStatement ps, List<Object> bindVals) throws
Exception
  {
    ResultSet rs = null;
    try {
      // Set bind values
      for (int idx = 0; idx < bindVals.size(); idx++) {</pre>
        Object o = bindVals.get(idx);
        // String
        if (o instanceof java.lang.String) {
          ps.setNString(idx + 1, (String)o);
        // Int
        else if (o instanceof java.lang.Integer) {
          ps.setInt(idx + 1, ((Integer)o).intValue());
        // Long
        else if (o instanceof java.lang.Long) {
          ps.setLong(idx + 1, ((Long)o).longValue());
        // Float
        else if (o instanceof java.lang.Float) {
          ps.setFloat(idx + 1, ((Float)o).floatValue());
        // Double
        else if (o instanceof java.lang.Double) {
          ps.setDouble(idx + 1, ((Double)o).doubleValue());
       // Timestamp
       else if (o instanceof java.sql.Timestamp) {
         ps.setTimestamp(idx + 1, (Timestamp)o);
       else {
         ps.setString(idx + 1, bindVals.get(idx).toString());
     }
      // Execute query
      rs = ps.executeQuery();
      if (rs.next()) {
        System.out.println("\n-- Execute Query: Result has "+rs.getInt(1)+"
rows --");
    finally {
      // close the SQL ResultSet
      if (rs != null) {
       rs.close();
  }
   * Prints return columns for a SQL translation
   * /
```

```
static void printReturnCols(PgqlColumnDescriptor[] cols) throws
Exception
  {
    StringBuffer buff = new StringBuffer("");
    for (int i = 0; i < cols.length; i++) {
      String colName = cols[i].getColName();
      PgqlColumnDescriptor.Type colType = cols[i].getColType();
      int offset = cols[i].getSqlOffset();
      String readableType = "";
      switch(colType) {
        case VERTEX:
          readableType = "VERTEX";
          break;
        case EDGE:
          readableType = "EDGE";
          break;
        case VALUE:
          readableType = "VALUE";
          break;
      buff.append("colName=["+colName+"] colType=["+readableType+"]
offset=["+offset+"]\n");
    }
    System.out.println(buff.toString());
}
PgqlExample9.java has the following output for test graph (which can be loaded
using GraphLoaderExample.java code).
--- Return Columns -----
colName=[v1] colType=[VERTEX] offset=[1]
colName=[fname1] colType=[VALUE] offset=[3]
colName=[age] colType=[VALUE] offset=[7]
colName=[constVal] colType=[VALUE] offset=[11]
-- SOL Translation -----
SELECT n'V' AS "V1$IT",
T0$0.VID AS "V1$ID",
TO$0.T AS "FNAME1$T",
TO$0.V AS "FNAME1$V",
TO$0.VN AS "FNAME1$VN",
TO$0.VT AS "FNAME1$VT",
T0$1.T AS "AGE$T",
TO$1.V AS "AGE$V",
TO$1.VN AS "AGE$VN",
TO$1.VT AS "AGE$VT",
4 AS "CONSTVAL$T",
to_nchar(?,'TM9','NLS_Numeric_Characters=''.,''') AS "CONSTVAL$V",
? AS "CONSTVAL$VN",
to_timestamp_tz(null) AS "CONSTVAL$VT"
FROM ( SELECT L.VID, L.VL, R.K, R.T, R.V, R.VN, R.VT
 FROM "SCOTT".TEST GRAPHVD$ L,
      (SELECT * FROM "SCOTT".TEST GRAPHVT$ WHERE K=n'fname') R
```



```
WHERE L.VID = R.VID(+)
) TO$0,
( SELECT L.VID, L.VL, R.K, R.T, R.V, R.VN, R.VT
 FROM "SCOTT". TEST GRAPHVD$ L,
      (SELECT * FROM "SCOTT".TEST GRAPHVT$ WHERE K=n'age' ) R
 WHERE L.VID = R.VID(+)
) T0$1
WHERE T0$0.VID=T0$1.VID AND
((T0\$0.T = 1 AND T0\$0.V = ?) OR T0\$1.VN < ?)
-- Bind Values -----
2.05
2.05
Bill
-- Execute Query: Result has 2 rows --
-- Bind Values -----
3.02
3.02
Ray
-- Execute Query: Result has 1 rows --
```

A.10.4.5 Additional Options for PGQL Translation and Execution

Several options are available to influence PGQL query translation and execution. The following are the main ways to set query options:

- Through explicit arguments to executeQuery and translateQuery
- Through flags in the options string argument of executeQuery and translateQuery
- Through Java JVM arguments.

The following table summarizes the available query arguments for PGQL translation and execution.

Table A-3 PGQL Translation and Execution Options

Option	Default	Explict Argument	Options Flag	JVM Argument
Degree of parallelism		parallel	none	none
Timeout	unlimited	timeout	none	none
Dynamic sampling	2	dynamicSamplin g	none	none
Maximum number of results	unlimited	maxResults	none	none
GT\$ table usage	on	none	USE_GT_TAB=F	- Doracle.pg.rdbms.pgql.useGtTab=false
CONNEC T BY usage	off	none	USE_RW=F	-Doracle.pg.rdbms.pgql.useRW=false



Option	Default	Explict Argument	Options Flag	JVM Argument
Distinct recursive WITH usage	off	none	USE_DIST_RW=T	- Doracle.pg.rdbms.pgql.useDistRW=tru e
Maximum path length	unlimited	none	MAX_PATH_LEN=n	-Doracle.pg.rdbms.pgql.maxPathLen=n
Project null properties	true	none	PROJ_NULL_PROP S=F	- Doracle.pg.rdbms.pgql.projNullProps=f alse
VT\$ VL column usage	on	none	USE_VL_COL=F	- Doracle.pg.rdbms.pgql.useVLCol=false

Table A-3 (Cont.) PGQL Translation and Execution Options

- Query Options Controlled by Explicit Arguments
- Using the GT\$ Skeleton Table
- Path Query Options
- Options for Partial Object Construction

A.10.4.5.1 Query Options Controlled by Explicit Arguments

Some query options are controlled by explicit arguments to methods in the Java API.

- The executeQuery method of PgqlStatement has explicit arguments for timeout in seconds, degree of parallelism, optimizer dynamic sampling, and maximum number of results.
- The translateQuery method has explicit arguments for degree of parallelism, optimizer dynamic sampling, and maximum number of results.
 PgqlPreparedStatement also provides those same additional arguments for executeQuery and translateQuery.

Example A-37 PgqlExample10.java

PgqlExample10.java shows PGQL query execution with additional options controlled by explicit arguments.

```
import java.sql.Connection;
import oracle.pg.rdbms.pgql.PgqlConnection;
import oracle.pg.rdbms.pgql.PgqlResultSet;
import oracle.pg.rdbms.pgql.PgqlStatement;

import oracle.ucp.jdbc.PoolDataSourceFactory;
import oracle.ucp.jdbc.PoolDataSource;

/**
    * This example shows how to execute a PGQL query with various options.
    */
```



```
public class PgqlExample10
  public static void main(String[] args) throws Exception
    int idx=0;
    String host
                              = args[idx++];
    String port
                            = args[idx++];
    String sid
                            = args[idx++];
                            = args[idx++];
    String user
    String password
                            = args[idx++];
    String graph
                              = args[idx++];
    Connection conn = null;
    PgqlStatement ps = null;
    PgqlResultSet rs = null;
    try {
      //Get a jdbc connection
      PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
      pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
      pds.setURL("jdbc:oracle:thin:@"+host+":"+port +":"+sid);
      pds.setUser(user);
      pds.setPassword(password);
      conn = pds.getConnection();
      // Get a PGQL connection
      PgqlConnection pgqlConn = PgqlConnection.getConnection(conn);
      pgqlConn.setGraph(graph);
      // Create a PgglStatement
      ps = pgqlConn.createStatement();
      // Execute query to get a ResultSet object
      String pggl =
        "SELECT v1.\"fname\" AS fname1, v2.\"fname\" AS fname2 "+
        "FROM MATCH (v1) - [:\"friendOf\"] -> (v2)";
      rs = ps.executeQuery(pgql /* query string */,
                           100 /* timeout (sec): 0 is default and implies
no timeout */,
                           2
                                /* parallel: 1 is default */,
                                /* dynamic sampling: 2 is default */,
                               /* max results: -1 is default and implies no
limit */,
                               /* options */);
      // Print query results
      rs.print();
    finally {
      // close the result set
      if (rs != null) {
        rs.close();
      }
```

```
// close the statement
if (ps != null) {
    ps.close();
}
// close the connection
if (conn != null) {
    conn.close();
}
}
}
```

PgqlExample10.java gives the following output for test_graph (which can be loaded using GraphLoaderExample.java code).

A.10.4.5.2 Using the GT\$ Skeleton Table

The property graph relational schema defines a GT\$ skeleton table that stores a single row for each edge in the graph, no matter how many properties an edge has. This skeleton table is populated by default so that PGQL query execution can take advantage of the GT\$ table and avoid sorting operations on the GE\$ table in many cases, which gives a significant performance improvement.

You can add "USE_GT_TAB=F" to the options argument of executeQuery and translateQuery or use -Doracle.pg.rdbms.pgql.useGtTab=false in the Java command line to turn off GT\$ table usage.

Example A-38 PgglExample11.java

PgqlExample11.java shows a query that uses the GT\$ skeleton table.

```
import java.sql.Connection;
import oracle.pg.rdbms.pgql.PgqlConnection;
import oracle.pg.rdbms.pgql.PgqlSqlQueryTrans;
import oracle.pg.rdbms.pgql.PgqlStatement;

import oracle.ucp.jdbc.PoolDataSourceFactory;
import oracle.ucp.jdbc.PoolDataSource;

/**
  * This example shows how to avoid using the GT$ skeleton table for
  * PGQL query execution.
  */
public class PgqlExample11
{
```



```
public static void main(String[] args) throws Exception
   int idx=0;
                           = args[idx++];
   String host
   String port
                           = args[idx++];
   String sid
                           = args[idx++];
   String user
                           = args[idx++];
   String password
                           = args[idx++];
   String graph
                           = args[idx++];
   Connection conn = null;
   PgqlStatement ps = null;
   try {
     //Get a jdbc connection
     PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
     pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
     pds.setURL("jdbc:oracle:thin:@"+host+":"+port +":"+sid);
     pds.setUser(user);
     pds.setPassword(password);
     conn = pds.getConnection();
     // Get a PGOL connection
     PgqlConnection pgqlConn = PgqlConnection.getConnection(conn);
     pgqlConn.setGraph(graph);
     // Create a PgqlStatement
     ps = pgqlConn.createStatement();
     // Execute query to get a ResultSet object
     String pgql =
       "SELECT id(v1), id(v2) "+
       "FROM MATCH (v1)-[knows]->(v2)";
     // Get the SQL translation with GT table
     PgqlSqlQueryTrans sqlTrans = ps.translateQuery(pgql,"");
     // Print SQL translation
     System.out.println("-- SQL Translation with GT Table
----");
     System.out.println(sqlTrans.getSqlTranslation());
     // Get the SQL translation without GT table
     sqlTrans = ps.translateQuery(pgql,"USE GT TAB=F");
     // Print SQL translation
     System.out.println("-- SQL Translation without GT Table
----");
     System.out.println(sqlTrans.getSqlTranslation());
   finally {
     // close the statement
```

```
if (ps != null) {
    ps.close();
}
// close the connection
if (conn != null) {
    conn.close();
}
}
}
```

PgqlExample11.java gives the following output for test_graph (which can be loaded using GraphLoaderExample.java code).

```
-- SQL Translation with GT Table -----
SELECT 7 AS "id(v1)$T",
to nchar(T0$0.SVID,'TM9','NLS Numeric Characters=''.,''') AS "id(v1)$V",
T0$0.SVID AS "id(v1)$VN",
to_timestamp_tz(null) AS "id(v1)$VT",
7 AS "id(v2)$T",
to nchar(T0$0.DVID,'TM9','NLS Numeric Characters=''.,''') AS "id(v2)$V",
T0$0.DVID AS "id(v2)$VN",
to timestamp tz(null) AS "id(v2)$VT"
FROM "SCOTT".TEST GRAPHGT$ T0$0
-- SQL Translation without GT Table -----
SELECT 7 AS "id(v1)$T",
to nchar(T0$0.SVID, 'TM9', 'NLS Numeric Characters=''.,''') AS "id(v1)$V",
T0$0.SVID AS "id(v1)$VN",
to timestamp tz(null) AS "id(v1) $VT",
7 AS "id(v2)$T",
to nchar(T0$0.DVID, 'TM9', 'NLS Numeric Characters=''.,''') AS "id(v2)$V",
\overline{\text{T0$0.DVID}} AS "id(v2)$VN",
to timestamp tz(null) AS "id(v2)$VT"
FROM (SELECT DISTINCT EID, SVID, DVID, EL FROM "SCOTT". TEST GRAPHGE$) T0$0
```

A.10.4.5.3 Path Query Options

A few options are available for executing path queries in PGQL. There are two basic evaluation methods available in Oracle SQL: CONNECT BY or recursive WITH clauses. Recursive WITH is the default evaluation method. In addition, you can further modify the recursive WITH evaluation method to include a DISTINCT modifier during the recursive step of query evaluation. Computing distinct vertices at each step helps prevent a combinatorial explosion in highly connected graphs. The DISTINCT modifier is not added by default because it requires a specific parameter setting in the database ("_recursive_with_control"=8).

You can also control the maximum length of paths searched. Path length in this case is defined as the number of repetitions allowed when evaluating the * and + operators. The default maximum length is unlimited.

Path evaluation options are summarized as follows.

- CONNECT BY: To use CONNECT BY, specify 'USE_RW=F' in the options argument or specify -Doracle.pg.rdbms.pgql.useRW=false in the Java command line
- Distinct Modifier in Recursive WITH: To use the DISTINCT modifier in the recursive step, first set "_recursive_with_control"=8 in your database session,

then specify 'USE_DIST_RW=T' in the options argument or specify - Doracle.pg.rdbms.pgql.useDistRW=true in the Java command line. You will encounter ORA-32486: unsupported operation in recursive branch of recursive WITH clause if "recursive with control" has not been set to 8 in your session.

Path Length Restriction: To limit maximum number of repetitions when evaluating * and + to n, specify 'MAX_PATH_LEN=n' in the query options argument or specify - Doracle.pg.rdbms.pgql.maxPathLen=n in the Java command line.

Example A-39 PgqlExample12.java

PgqlExample12.java shows path query translations under various options.

```
import java.sql.Connection;
import java.sql.Statement;
import oracle.pg.rdbms.pgql.PgqlConnection;
import oracle.pq.rdbms.pqql.PqqlSqlQueryTrans;
import oracle.pg.rdbms.pgql.PgqlStatement;
import oracle.ucp.jdbc.PoolDataSourceFactory;
import oracle.ucp.jdbc.PoolDataSource;
* This example shows how to use various options with PGQL path queries.
* /
public class PgqlExample12
{
 public static void main(String[] args) throws Exception
   int idx=0;
   String host
                           = args[idx++];
   String port
                           = args[idx++];
                            = args[idx++];
   String sid
   String user
                            = args[idx++];
   String password
                           = args[idx++];
   String graph
                            = args[idx++];
   Connection conn = null;
   PgglStatement ps = null;
   try {
      //Get a jdbc connection
     PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
     pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
     pds.setURL("jdbc:oracle:thin:@"+host+":"+port +":"+sid);
     pds.setUser(user);
     pds.setPassword(password);
     conn = pds.getConnection();
      // Get a PGQL connection
      PgqlConnection pgqlConn = PgqlConnection.getConnection(conn);
     pgqlConn.setGraph(graph);
```



```
// Create a PgqlStatement
      ps = pgqlConn.createStatement();
      // Set " recursive with control"=8 to enable distinct
optimization
      // optimization for recursive with
      Statement stmt = conn.createStatement();
      stmt.executeUpdate("alter session set
\" recursive with control\"=8");
      stmt.close();
      // Path Query to illustrate options
      String pgql =
        "PATH fof AS ()-[:\"friendOf\"]->() "+
        "SELECT id(v1), id(v2) "+
        "FROM MATCH (v1) - / : fof*/-> (v2) "+
        "WHERE id(v1) = 2";
      // get SQL translation with defaults - Non-distinct Recursive
WITH
      PgqlSqlQueryTrans sqlTrans =
        ps.translateQuery(pgql /* query string */,
                             /* parallel: default is 1 */,
                             /* dynamic sampling: default is 2 */,
                         -1 /* max results: -1 implies no limit */,
                             /* options */);
      System.out.println("-- Default Path Translation
----");
      System.out.println(sqlTrans.getSqlTranslation()+"\n");
      // get SQL translation with DISTINCT reachability optimization
      sqlTrans =
       ps.translateQuery(pgql /* query string */,
                             /* parallel: default is 1 */,
                              /* dynamic sampling: default is 2 */,
                         -1 /* max results: -1 implies no limit */,
                         " USE DIST RW=T " /* options */);
      System.out.println("-- DISTINCT RW Path Translation
----");
      System.out.println(sqlTrans.qetSqlTranslation()+"\n");
      // get SQL translation with CONNECT BY
      sqlTrans =
       ps.translateQuery(pgql /* query string */,
                         2 /* parallel: default is 1 */,
                             /* dynamic sampling: default is 2 */,
                         -1 /* max results: -1 implies no limit */,
                         " USE RW=F " /* options */);
     System.out.println("-- CONNECT BY Path Translation
  ----");
     System.out.println(sqlTrans.getSqlTranslation()+"\n");
    finally {
      // close the statement
```

```
if (ps != null) {
        ps.close();
      // close the connection
      if (conn != null) {
         conn.close();
    }
  }
}
PgqlExample12.java gives the following output for test graph (which can be loaded using
GraphLoaderExample.java code).
-- Default Path Translation -----
SELECT /*+ parallel(2) */ * FROM(SELECT 7 AS "id(v1)$T",
to nchar(TO$0.SVID, 'TM9', 'NLS Numeric Characters=''.,''') AS "id(v1)$V",
T0$0.SVID AS "id(v1)$VN",
to timestamp tz(null) AS "id(v1)$VT",
7 AS "id(v2)$T",
to nchar(T0$0.DVID,'TM9','NLS Numeric Characters=''.,''') AS "id(v2)$V",
T0$0.DVID AS "id(v2)$VN",
to_timestamp_tz(null) AS "id(v2)$VT"
FROM (/*Path[*/SELECT DISTINCT SVID, DVID
FROM (
SELECT 2 AS SVID, 2 AS DVID
FROM SYS.DUAL
WHERE EXISTS (
SELECT 1
FROM "SCOTT".TEST GRAPHVT$
WHERE VID = 2)
UNION ALL
SELECT SVID, DVID FROM
(WITH RW (ROOT, DVID) AS
( SELECT ROOT, DVID FROM
(SELECT SVID ROOT, DVID
FROM (SELECT TO$0.SVID AS SVID,
T0$0.DVID AS DVID
FROM "SCOTT".TEST GRAPHGT$ T0$0
WHERE T0\$0.SVID = 2 AND
(T0$0.EL = n'friendOf' AND T0$0.EL IS NOT NULL))
) UNION ALL
SELECT RW.ROOT, R.DVID
FROM (SELECT TO$0.SVID AS SVID,
T0$0.DVID AS DVID
FROM "SCOTT".TEST GRAPHGT$ T0$0
WHERE (T0$0.EL = n'friendOf' AND T0$0.EL IS NOT NULL)) R, RW
WHERE RW.DVID = R.SVID)
CYCLE DVID SET cycle col TO 1 DEFAULT 0
SELECT ROOT SVID, DVID FROM RW))/*]Path*/) T0$0
WHERE T0\$0.SVID = 2)
-- DISTINCT RW Path Translation -----
SELECT /*+ parallel(2) */ * FROM(SELECT 7 AS "id(v1)$T",
to nchar(T0$0.SVID,'TM9','NLS Numeric Characters=''.,''') AS "id(v1)$V",
T0$0.SVID AS "id(v1)$VN",
to timestamp tz(null) AS "id(v1)$VT",
7 AS "id(v2)$T",
to nchar(T0$0.DVID, 'TM9', 'NLS Numeric Characters=''.,''') AS "id(v2)$V",
```



```
T0$0.DVID AS "id(v2)$VN",
to timestamp tz(null) AS "id(v2)$VT"
FROM (/*Path[*/SELECT DISTINCT SVID, DVID
FROM (
SELECT 2 AS SVID, 2 AS DVID
FROM SYS.DUAL
WHERE EXISTS (
SELECT 1
FROM "SCOTT".TEST GRAPHVT$
WHERE VID = 2)
UNION ALL
SELECT SVID, DVID FROM
(WITH RW (ROOT, DVID) AS
( SELECT ROOT, DVID FROM
(SELECT SVID ROOT, DVID
FROM (SELECT TO$0.SVID AS SVID,
TO$0.DVID AS DVID
FROM "SCOTT".TEST GRAPHGT$ T0$0
WHERE T0\$0.SVID = 2 AND
(T0$0.EL = n'friendOf' AND T0$0.EL IS NOT NULL))
) UNION ALL
SELECT DISTINCT RW.ROOT, R.DVID
FROM (SELECT TO$0.SVID AS SVID,
TO$0.DVID AS DVID
FROM "SCOTT".TEST GRAPHGT$ T0$0
WHERE (T0$0.EL = n'friendOf' AND T0$0.EL IS NOT NULL)) R, RW
WHERE RW.DVID = R.SVID )
CYCLE DVID SET cycle col TO 1 DEFAULT 0
SELECT ROOT SVID, DVID FROM RW))/*]Path*/) T0$0
WHERE T0$0.SVID = 2)
-- CONNECT BY Path Translation -----
SELECT /*+ parallel(2) */ * FROM(SELECT 7 AS "id(v1)$T",
to nchar(T0$0.SVID, 'TM9', 'NLS Numeric Characters=''.,''') AS "id(v1)$V",
T0$0.SVID AS "id(v1)$VN",
to timestamp tz(null) AS "id(v1)$VT",
7 AS "id(v2)$T",
to nchar(T0$0.DVID, 'TM9', 'NLS Numeric Characters=''.,''') AS "id(v2)$V",
T0$0.DVID AS "id(v2)$VN",
to_timestamp_tz(null) AS "id(v2)$VT"
FROM (/*Path[*/SELECT DISTINCT SVID, DVID
FROM (
SELECT 2 AS SVID, 2 AS DVID
FROM SYS.DUAL
WHERE EXISTS (
SELECT 1
FROM "SCOTT".TEST GRAPHVT$
WHERE VID = 2)
UNION ALL
SELECT SVID, DVID
(SELECT CONNECT_BY_ROOT TO$0.SVID AS SVID, TO$0.DVID AS DVID
FROM (
SELECT TO$0.SVID AS SVID,
T0$0.DVID AS DVID
FROM "SCOTT".TEST GRAPHGT$ T0$0
WHERE (T0$0.EL = n'friendOf' AND T0$0.EL IS NOT NULL)) T0$0
START WITH T0\$0.SVID = 2
CONNECT BY NOCYCLE PRIOR DVID = SVID))/*]Path*/) T0$0
WHERE T0$0.SVID = 2)
```

The query plan for the first query with the default recursive WITH strategy should look similar to the following.

```
-- default RW
| Id | Operation
                                                       | Name
 0 | SELECT STATEMENT
   1 | TEMP TABLE TRANSFORMATION
   2 | LOAD AS SELECT (CURSOR DURATION MEMORY)
                                                      | SYS TEMP 0FD9D6662 37AA44
        UNION ALL (RECURSIVE WITH) BREADTH FIRST
   3 |
         PX COORDINATOR
   4 |
           PX SEND QC (RANDOM)
   5 |
                                                       | :TQ20000
           LOAD AS SELECT (CURSOR DURATION MEMORY)
                                                       | SYS TEMP OFD9D6662 37AA44
   7 |
            PX PARTITION HASH ALL
       TABLE ACCESS BY LOCAL INDEX ROWID BATCHED
| * 8 |
                                                       | TEST GRAPHGT$
|* 9 |
           INDEX RANGE SCAN
                                                       | TEST GRAPHXSG$
| 10 | PX COORDINATOR
          PX SEND QC (RANDOM)
| 11 |
                                                       | :TQ10000
           LOAD AS SELECT (CURSOR DURATION MEMORY)
| 12 |
                                                       | SYS TEMP OFD9D6662 37AA44
| 13 |
            NESTED LOOPS
| 14 | PX BLOCK ITERATOR
|* 15 |
                                                       | SYS TEMP 0FD9D6662 37AA44
          TABLE ACCESS FULL
| 16 | PARTITION HASH ALL
|* 17 | TABLE ACCESS BY LOCAL INDEX ROWID BATCHED |* 18 | INDEX RANGE SCAN
                                                        | TEST GRAPHGT$
                                                         TEST GRAPHXSG$
| 19 | PX COORDINATOR
| 20 | PX SEND QC (RANDOM)
                                                       | :TQ30001
| 21 |
        VIEW
1 22 1
         HASH UNIQUE
           PX RECEIVE
1 23 1
            PX SEND HASH
| 24 |
                                                       | :TO30000
        HASH UNIQUE
| 25 |
         VIEW
  26 |
          UNION-ALL
PX SELECTOR
  27 |
  28 |
            FILTER
FAST DUAL
|* 29 |
           PARTITION HASH SINGLE
INDEX SKIP SCAN
VIEW
  30 I
| 31 |
| * 32 |
                                                         TEST GRAPHXQV$
| 33 |
            VIEW
| * 34 |
             PX BLOCK ITERATOR
| 35 |
               TABLE ACCESS FULL
                                                       | SYS TEMP 0FD9D6662 37AA44 |
```

The query plan for the second query that adds a DISTINCT modifier in the recursive step should look similar to the following.



```
3 | UNION ALL (RECURSIVE WITH) BREADTH FIRST
 4 | PX COORDINATOR
. | 5 | PX SEND QC (RANDOM)
| :TO20000
         LOAD AS SELECT (CURSOR DURATION MEMORY)
1 6 1
                                                SYS TEMP OFD9D6669 37AA44 |
  7 | PX PARTITION HASH ALL
|* 8 | TABLE ACCESS BY LOCAL INDEX ROWID BATCHED
TEST GRAPHGT$
|* 9 | INDEX RANGE SCAN
TEST GRAPHXSG$
| 10 | PX COORDINATOR
| 11 | PX SEND QC (RANDOM)
| :TQ10001
           | 12 | LOAD AS SELECT (CURSOR DURATION MEMORY)
SYS TEMP OFD9D6669 37AA44 |
| 13 | SORT GROUP BY
| 14 | PX RECEIVE
| 15 | PX SEND HASH
| :TQ10000
| 16 | SORT GROUP BY
| 17 |
         NESTED LOOPS
| 18 |
          PX BLOCK ITERATOR
             1
|* 19 | TABLE ACCESS FULL
                                                 SYS TEMP 0FD9D6669 37AA44 |
| 20 | PARTITION HASH ALL
            .
|* 21 |
             TABLE ACCESS BY LOCAL INDEX ROWID BATCHED
             1
TEST GRAPHGT$
1* 22 1
             INDEX RANGE SCAN
TEST GRAPHXSG$
| 23 | PX COORDINATOR
| 24 | PX SEND QC (RANDOM)
| :TQ30001
| 25 | VIEW
| 26 | HASH UNIQUE
| 27 |
         PX RECEIVE
 28 |
          PX SEND HASH
| :TQ30000
 29 |
         HASH UNIQUE
  30 |
         VIEW
  31 |
         UNION-ALL
  32 |
          PX SELECTOR
l* 33 l
          FILTER
```



The query plan for the third query that uses CONNECTY BY should look similar to the following.

| Id | Operation | 0 | SELECT STATEMENT | 1 | VIEW | 2 | HASH UNIQUE | 3 | VIEW | 4 | UNION-ALL |* 5 | FILTER 6 1 FAST DUAL 7 | PARTITION HASH SINGLE INDEX SKIP SCAN VIEW | * 8 | | TEST GRAPHXQV\$ |* 9 | CONNECT BY WITH FILTERING PX COORDINATOR |* 10 | | 11 | PX SEND QC (RANDOM) 12 | | :TQ10000 | 13 | PX PARTITION HASH ALL |* 14 | TABLE ACCESS BY LOCAL INDEX ROWID BATCHED | TEST_GRAPHGT\$ |* 15 | INDEX RANGE SCAN | TEST_GRAPHXSG\$
NESTED LOOPS | | 16 | | 17 | CONNECT BY PUMP | 18 | PARTITION HASH ALL |* 19 | TABLE ACCESS BY LOCAL INDEX ROWID BATCHED | TEST GRAPHGT\$ INDEX RANGE SCAN | TEST GRAPHXSG\$ |* 20 |

Example A-40 PgqlExample13.java

PgqlExample13.java shows how to set length restrictions during path query evaluation.

```
import java.sql.Connection;
import oracle.pg.rdbms.pgql.PgqlConnection;
import oracle.pg.rdbms.pgql.PgqlResultSet;
import oracle.pg.rdbms.pgql.PgqlStatement;
import oracle.ucp.jdbc.PoolDataSourceFactory;
import oracle.ucp.jdbc.PoolDataSource;
```



```
/**
 ^{\star} This example shows how to use the maximum path length option for
 * PGQL path queries.
 */
public class PgqlExample13
  public static void main(String[] args) throws Exception
    int idx=0;
    String host
                             = args[idx++];
    String port
                            = args[idx++];
    String sid
                            = args[idx++];
    String user
                            = args[idx++];
    String password
                            = args[idx++];
    String graph
                            = args[idx++];
    Connection conn = null;
    PgglStatement ps = null;
    PgqlResultSet rs = null;
    try {
      //Get a jdbc connection
      PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
      pds.setURL("jdbc:oracle:thin:@"+host+":"+port +":"+sid);
      pds.setUser(user);
      pds.setPassword(password);
      conn = pds.getConnection();
      // Get a PGQL connection
      PgqlConnection pgqlConn = PgqlConnection.getConnection(conn);
      pgqlConn.setGraph(graph);
      // Create a PgqlStatement
      ps = pgqlConn.createStatement();
      // Path Query to illustrate options
      String pggl =
        "PATH fof AS ()-[:\"friendOf\"]->() "+
        "SELECT v1.\"fname\" AS fname1, v2.\"fname\" AS fname2 "+
        "FROM MATCH (v1) - / : fof*/-> (v2) "+
        "WHERE v1.\"fname\" = 'Ray'";
      // execute query for 1-hop
      rs = ps.executeQuery(pgql, " MAX PATH LEN=1 ");
      // print results
      System.out.println("-- Results for 1-hop -----");
      rs.print();
      // close result set
```

```
rs.close();
     // execute query for 2-hop
     rs = ps.executeQuery(pgql, " MAX PATH LEN=2 ");
     // print results
     System.out.println("-- Results for 2-hop -----");
     rs.print();
     // close result set
     rs.close();
     // execute query for 3-hop
     rs = ps.executeQuery(pgql, " MAX PATH LEN=3 ");
     // print results
     System.out.println("-- Results for 3-hop -----");
     rs.print();
     // close result set
     rs.close();
   finally {
     // close the result set
     if (rs != null) {
       rs.close();
     // close the statement
     if (ps != null) {
       ps.close();
     // close the connection
     if (conn != null) {
       conn.close();
   }
 }
}
```

PgqlExample13.java has the following output for test_graph (which can be loaded using GraphLoaderExample.java code).



A.10.4.5.4 Options for Partial Object Construction

When reading edges from a query result, there are two possible behaviors when adding the start and end vertex to any local caches:

- Add only the vertex ID, which is available from the edge itself. This option is the default, for efficiency.
- Add the vertex ID, and retrieve all properties for the start and end vertex. For this behavior, you can call setPartial(true) on each OracleVertex object constructed from your PGQL query result set.

Example A-41 PgqlExample14.java

PgqlExample14.java illustrates this difference in behavior. This program first executes a query to retrieve all edges, which causes the incident vertices to be added to a local cache. The second query retrieves all vertices. The program then prints each OracleVertex object to show which properties have been loaded.

```
import java.sql.Connection;
import oracle.pg.rdbms.Oracle;
import oracle.pg.rdbms.OraclePropertyGraph;
import oracle.pg.rdbms.OracleVertex;
import oracle.pg.rdbms.pgql.PgqlConnection;
import oracle.pq.rdbms.pqql.PqqlResultSet;
import oracle.pg.rdbms.pgql.PgqlStatement;
import oracle.ucp.jdbc.PoolDataSourceFactory;
import oracle.ucp.jdbc.PoolDataSource;
 * This example shows the behavior of setPartial(true) for
OracleVertex objects
 * created from PGQL query results.
public class PgqlExample14
  public static void main(String[] args) throws Exception
    int idx=0;
    String host
                            = args[idx++];
    String port
                            = args[idx++];
```



```
String sid
                        = args[idx++];
String user
                        = args[idx++];
String password
                        = args[idx++];
String graph
                         = args[idx++];
Connection conn = null;
Oracle oracle = null;
OraclePropertyGraph opg = null;
PgqlStatement ps = null;
PgglResultSet rs = null;
try {
  //Get a jdbc connection
  PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
  pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
  pds.setURL("jdbc:oracle:thin:@"+host+":"+port +":"+sid);
  pds.setUser(user);
  pds.setPassword(password);
  conn = pds.getConnection();
  // Get a PGQL connection
  PgqlConnection pgqlConn = PgqlConnection.getConnection(conn);
  pgqlConn.setGraph(graph);
  // Create a PgqlStatement
  ps = pgqlConn.createStatement();
  // Query to illustrate set partial
  String pggl =
    "SELECT id(e), label(e) "+
    "FROM MATCH (v1)-[e:\"knows\"]->(v2)";
  // execute query
  rs = ps.executeQuery(pgql, " ");
  // print results
  System.out.println("-- Results for edge query -----");
  rs.print();
  // close result set
  rs.close();
  // Create an Oracle Property Graph instance
  oracle = new Oracle(conn);
  opg = OraclePropertyGraph.getInstance(oracle,graph);
  // Query to retrieve vertices
  paal =
    "SELECT id(v) "+
    "FROM MATCH (v)";
  // Get each vertex object in result and print with toString()
  rs = ps.executeQuery(pgql, " ");
```

```
// iterate through result
      System.out.println("-- Vertex objects retrieved from vertex
query --");
      while (rs.next()) {
        Long vid = rs.getLong(1);
        OracleVertex v = OracleVertex.getInstance(opg, vid);
        System.out.println(v.toString());
      // close result set
      rs.close();
      // Execute the same query but call setPartial(true) for each
vertex
      rs = ps.executeQuery(pgql, " ");
      System.out.println("-- Vertex objects retrieved from vertex
query with setPartial(true) --");
      while (rs.next()) {
        Long vid = rs.getLong(1);
        OracleVertex v = OracleVertex.getInstance(opg, vid);
        v.setPartial(true);
        System.out.println(v.toString());
      // close result set
      rs.close();
    finally {
      // close the result set
      if (rs != null) {
       rs.close();
      // close the statement
      if (ps != null) {
       ps.close();
      // close the connection
      if (conn != null) {
        conn.close();
      // close the property graph
      if (opg != null) {
        opg.close();
      // close oracle
      if (oracle != null) {
        oracle.dispose();
    }
}
```

The output for PgqlExample14.java (which can be loaded using GraphLoaderExample.java code) is:

```
-- Results for edge query ------+
```



```
| id(e) | label(e) |
       | knows
16
| 11 | knows
| 10 | knows
1 5
      | knows
      | knows
1 4
| 13 | knows
| 9
       | knows
| 12
       | knows
1 8
       | knows
| 7
       | knows
| 14
       knows
| 15
       knows
-- Vertex objects retrieved from vertex query --
Vertex ID 3 [NULL] {}
Vertex ID 0 [NULL] {}
Vertex ID 2 [NULL] {}
Vertex ID 1 [NULL] {}
-- Vertex objects retrieved from vertex query with setPartial(true) --
Vertex ID 3 [NULL] {bval:bol:false, fname:str:Susan, lname:str:Blue, mval:bol:false,
Vertex ID 0 [NULL] {bval:bol:true, fname:str:Bill, lname:str:Brown, mval:str:y,
age:int:40}
Vertex ID 2 [NULL] {fname:str:Ray, lname:str:Green, mval:dat:1985-01-01 04:00:00.0,
age:int:41}
Vertex ID 1 [NULL] {bval:bol:true, fname:str:John, lname:str:Black, mval:int:27,
age:int:30}
```

A.10.4.6 Querying Another User's Property Graph

You can query another user's property graph data if you have been granted the appropriate privileges in the database. For example, to query GRAPH1 in SCOTT's schema, you must have READ privilege on SCOTT.GRAPH1GE\$, SCOTT.GRAPH1VT\$, SCOTT.GRAPH1GT\$, and SCOTT.GRAPH1VD\$.

Example A-42 PgglExample15.java

PgqlExample15.java shows how another user can query a graph in SCOTT's schema.

```
import java.sql.Connection;
import oracle.pg.rdbms.pgql.PgqlConnection;
import oracle.pg.rdbms.pgql.PgqlResultSet;
import oracle.pg.rdbms.pgql.PgqlStatement;

import oracle.ucp.jdbc.PoolDataSourceFactory;
import oracle.ucp.jdbc.PoolDataSource;

/**
    * This example shows how to query a property graph located in another user's
    * schema. READ privilege on GE$, VT$, GT$ and VD$ tables for the other
user's
    * property graph are required to avoid ORA-00942: table or view does not
exist.
    */
public class PgqlExample15
```



```
{
  public static void main(String[] args) throws Exception
    int idx=0;
   String host
                            = args[idx++];
    String port
                            = args[idx++];
   String sid
                            = args[idx++];
    String user
                            = args[idx++];
                            = args[idx++];
    String password
    String graph
                             = args[idx++];
    Connection conn = null;
    PgglStatement ps = null;
    PgqlResultSet rs = null;
    try {
      //Get a jdbc connection
      PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
      pds.setURL("jdbc:oracle:thin:@"+host+":"+port +":"+sid);
      pds.setUser(user);
      pds.setPassword(password);
      conn = pds.getConnection();
      // Get a PGQL connection
      PgqlConnection pgqlConn = PgqlConnection.getConnection(conn);
      pgqlConn.setGraph(graph);
      // Set schema so that we can query Scott's graph
      pgqlConn.setSchema("SCOTT");
      // Create a PgqlStatement
      ps = pgqlConn.createStatement();
      // Execute query to get a ResultSet object
      String pgql =
        "SELECT v.\"fname\" AS fname, v.\"lname\" AS lname "+
        "FROM MATCH (v)";
      rs = ps.executeQuery(pgql, "");
      // Print query results
      rs.print();
    finally {
      // close the result set
      if (rs != null) {
       rs.close();
      // close the statement
      if (ps != null) {
       ps.close();
```



```
}
    // close the connection
    if (conn != null) {
        conn.close();
    }
    }
}
```

The following SQL statements create database user USER2 and grant the necessary privileges. You can also use the <code>OraclePropertyGraph.grantAccess</code> Java API to achieve the same effect.

```
SQL> grant connect, resource, unlimited tablespace to user2 identified by user2;
Grant succeeded.

SQL> grant read on scott.test_graphvt$ to user2;
Grant succeeded.

SQL> grant read on scott.test_graphge$ to user2;
Grant succeeded.

SQL> grant read on scott.test_graphgt$ to user2;
Grant succeeded.

SQL> grant read on scott.test_graphyt$ to user2;
Grant succeeded.

SQL> grant read on scott.test_graphvd$ to user2;
Grant succeeded.
```

The output for PgqlExample15.java for the test_graph data set when connected to the database as USER2 is as follows. Note that test_graph should have already been loaded (using GraphLoaderExample.java code) as GRAPH1 by user SCOTT before running PgqlExample15.

A.10.4.7 Using Query Optimizer Hints with PGQL

The Java API allows query optimizer hints that influence the join type when executing PGQL queries. The <code>executeQuery</code> and <code>translateQuery</code> methods in <code>PgqlStatement</code> and <code>PgqlPreparedStatement</code> accept the following strings in the options argument to influence the query plan for the corresponding SQL query.

- ALL EDGE NL Use Nested Loop join for all joins that involve the \$GE and \$GT tables.
- ALL_EDGE_HASH Use HASH join for all joins that involve the \$GE and \$GT tables.
- ALL_VERTEX_NL Use Nested Loop join for all joins that involve the \$VT table.



• ALL_VERTEX_HASH – Use HASH join for all joins that involve the \$VT table.

Example A-43 PgglExample16.java

PgqlExample16.java shows how to use optimizer hints to influence the joins used for a graph traversal.

```
import java.sql.Connection;
import oracle.pg.rdbms.pggl.PgglConnection;
import oracle.pg.rdbms.pgql.PgqlSqlQueryTrans;
import oracle.pg.rdbms.pgql.PgqlStatement;
import oracle.ucp.jdbc.PoolDataSourceFactory;
import oracle.ucp.jdbc.PoolDataSource;
 * This example shows how to use query optimizer hints with PGQL
queries.
 */
public class PgglExample16
  public static void main(String[] args) throws Exception
    int idx=0;
    String host
                             = args[idx++];
                            = args[idx++];
    String port
    String sid
                            = args[idx++];
                            = args[idx++];
    String user
    String password
                            = args[idx++];
    String graph
                             = args[idx++];
    Connection conn = null;
    PgqlStatement ps = null;
    try {
      //Get a jdbc connection
      PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
      pds.setURL("jdbc:oracle:thin:@"+host+":"+port +":"+sid);
      pds.setUser(user);
      pds.setPassword(password);
      conn = pds.getConnection();
      // Get a PGQL connection
      PgqlConnection pgqlConn = PgqlConnection.getConnection(conn);
      pgqlConn.setGraph(graph);
      // Create a PgqlStatement
      ps = pgqlConn.createStatement();
      // Query to illustrate join hints
      String pggl =
        "SELECT id(v1), id(v4) "+
```



```
"FROM MATCH (v1)-[:\"friendOf\"]->(v2)-[:\"friendOf\"]-
>(v3)-[:\"friendOf\"]->(v4)";
      // get SQL translation with hash join hint
      PgqlSqlQueryTrans sqlTrans =
        ps.translateQuery(pgql /* query string */,
                         " ALL EDGE HASH " /* options */);
      // print SQL translation
      System.out.println("-- Query with ALL EDGE HASH -----");
      System.out.println(sqlTrans.qetSqlTranslation()+"\n");
      // get SQL translation with nested loop join hint
      sqlTrans =
       ps.translateQuery(pgql /* query string */,
                         " ALL EDGE NL " /* options */);
      // print SQL translation
      System.out.println("-- Query with ALL EDGE NL -----");
      System.out.println(sqlTrans.getSqlTranslation()+"\n");
    finally {
      // close the statement
      if (ps != null) {
       ps.close();
      // close the connection
     if (conn != null) {
        conn.close();
  }
}
```

The output for PgqlExample16.java for test_graph (which can be loaded using GraphLoaderExample.java code) is:

```
-- Query with ALL EDGE HASH -----
SELECT /*+ USE HASH(T0$0 T0$1 T0$2) */ 7 AS "id(v1)$T",
to nchar(T0$0.SVID, 'TM9', 'NLS Numeric Characters=''.,''') AS "id(v1)$V",
T0$0.SVID AS "id(v1)$VN",
to_timestamp_tz(null) AS "id(v1)$VT",
7 AS "id(v4)$T",
to nchar(T0$2.DVID, 'TM9', 'NLS Numeric Characters=''.,''') AS "id(v4)$V",
T0$2.DVID AS "id(v4)$VN",
to timestamp tz(null) AS "id(v4)$VT"
FROM "SCOTT". TEST GRAPHGT$ T0$0,
"SCOTT".TEST GRAPHGT$ T0$1,
"SCOTT".TEST GRAPHGT$ T0$2
WHERE TO$0.DVID=TO$1.SVID AND
T0$1.DVID=T0$2.SVID AND
(TO$0.EL = n'friendOf' AND TO$0.EL IS NOT NULL) AND
(TO$1.EL = n'friendOf' AND TO$1.EL IS NOT NULL) AND
(T0$2.EL = n'friendOf' AND T0$2.EL IS NOT NULL)
-- Query with ALL EDGE NL -----
SELECT /*+ USE NL(T0\$0 T0\$1 T0\$2) */ 7 AS "id(v1)\$T",
to_nchar(T0$0.SVID,'TM9','NLS_Numeric_Characters=''.,''') AS "id(v1)$V",
```



```
T0$0.SVID AS "id(v1)$VN",

to_timestamp_tz(null) AS "id(v1)$VT",

7 AS "id(v4)$T",

to_nchar(T0$2.DVID,'TM9','NLS_Numeric_Characters=''.,''') AS "id(v4)$V",

T0$2.DVID AS "id(v4)$VN",

to_timestamp_tz(null) AS "id(v4)$VT"

FROM "SCOTT".TEST_GRAPHGT$ T0$0,

"SCOTT".TEST_GRAPHGT$ T0$1,

"SCOTT".TEST_GRAPHGT$ T0$2

WHERE T0$0.DVID=T0$1.SVID AND

T0$1.DVID=T0$2.SVID AND

(T0$0.EL = n'friendOf' AND T0$0.EL IS NOT NULL) AND

(T0$1.EL = n'friendOf' AND T0$2.EL IS NOT NULL) AND

(T0$2.EL = n'friendOf' AND T0$2.EL IS NOT NULL)
```

The query plan for the first query that uses ALL_EDGE_HASH should look similar to the following.

						-
I	d		Operation		Name	
						-
	0		SELECT STATEMENT			
*	1		HASH JOIN			
*	2		HASH JOIN			
	3		PARTITION HASH ALL			
*	4		TABLE ACCESS FULL		TEST_GRAPHGT\$	
	5		PARTITION HASH ALL			
*	6		TABLE ACCESS FULL		TEST_GRAPHGT\$	
	7		PARTITION HASH ALL			
*	8		TABLE ACCESS FULL		TEST_GRAPHGT\$	
						_

The query plan for the second query that uses ALL_EDGE_NL should look similar to the following.

					-
	Id Operation			Name	
					-
	0		SELECT STATEMENT	1	
	1		NESTED LOOPS	1	
	2		NESTED LOOPS	1	
	3		PARTITION HASH ALL		
*	4		TABLE ACCESS FULL	TEST_GRAPHGT\$	
	5		PARTITION HASH ALL		
*	6		TABLE ACCESS BY LOCAL INDEX ROWID BATCHED	TEST_GRAPHGT\$	
*	7		INDEX RANGE SCAN	TEST_GRAPHXSG\$	
	8		PARTITION HASH ALL		
*	9		TABLE ACCESS BY LOCAL INDEX ROWID BATCHED	TEST_GRAPHGT\$	
*	10		INDEX RANGE SCAN	TEST_GRAPHXSG\$	
					-

A.10.4.8 Modifying Property Graphs through INSERT, UPDATE, and DELETE Statements

PGQL supports INSERT, UPDATE, and DELETE operations on Property Graphs. The method execute in PgqlStatement lets you execute such DML operations. This topic provides several examples of such operations.



JDBC connection auto commit must be off in order to be able to execute INSERT, UPDATE, and DELETE statements.

Example A-44 PgqlExample17.java (Insert)

PgqlExample17.java inserts several vertices and edges into a graph. Notice that the special property _ora_id is used to define ID values of vertices and edges. If the property _ora_id is omitted, a unique ID is generated for each new vertex or edge that is inserted into the graph.

```
import java.sql.Connection;
import oracle.pg.rdbms.pgql.PgqlConnection;
import oracle.pg.rdbms.pgql.PgqlResultSet;
import oracle.pg.rdbms.pgql.PgqlStatement;
import oracle.ucp.jdbc.PoolDataSourceFactory;
import oracle.ucp.jdbc.PoolDataSource;
/**
 * This example shows how to execute a PGQL INSERT operation.
public class PgqlExample17
 public static void main(String[] args) throws Exception
   int idx=0;
   String host
                           = args[idx++];
   String port
                            = args[idx++];
   String sid
                            = args[idx++];
    String user
                            = args[idx++];
    String password
                            = args[idx++];
                            = args[idx++];
    String graph
    Connection conn = null;
    PgqlStatement ps = null;
    PgqlResultSet rs = null;
    try {
      //Get a jdbc connection
      PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
      pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
     pds.setURL("jdbc:oracle:thin:@"+host+":"+port +":"+sid);
     pds.setUser(user);
     pds.setPassword(password);
      conn = pds.getConnection();
      conn.setAutoCommit(false);
      // Get a PGQL connection
      PgqlConnection pgqlConn = PgqlConnection.getConnection(conn);
```



```
pgqlConn.setGraph(graph);
      // Create a PgglStatement
      ps = pgqlConn.createStatement();
      // Execute insert statement
      String pgql =
        "INSERT VERTEX p1 LABELS (person) PROPERTIES (p1.\" ora id\" =
1, p1.fname = 'Jake') "+
            , VERTEX p2 LABELS (person) PROPERTIES (p2.\" ora id\" =
2, p2.fname = 'Amy') "+
              , VERTEX p3 LABELS (person) PROPERTIES (p3.\" ora id\" =
3, p3.fname = 'Erik') "+
            , VERTEX p4 LABELS (person) PROPERTIES (p4.\" ora id\" =
4, p4.fname = 'Jane') "+
       " , EDGE e1 BETWEEN p1 AND p2 LABELS (knows) PROPERTIES
(e1.\" ora id\" = 1, e1.since = DATE '2003-04-21') "+
           , EDGE e2 BETWEEN p1 AND p3 LABELS (knows) PROPERTIES
(e2.)" ora id\" = 2, e2.since = DATE '2010-02-10') "+
       " , EDGE e3 BETWEEN p3 AND p4 LABELS (knows) PROPERTIES
(e3.\" ora id\" = 3, e3.since = DATE '1999-01-03') ";
      ps.execute(pgql, /* query string */
                   "", /* query options */
                   "" /* modify options */);
      // Execute a query to verify insertion
      pgql =
         " SELECT id(p1) AS id1, p1.fname AS person1, id(p2) as id2,
p2.fname AS person2, id(e) as e, e.since "+
             FROM MATCH (p1) - [e:knows] -> (p2) "+
          "ORDER BY id1, id2";
      rs = ps.executeQuery(pgql, "");
      // Print the results
      rs.print();
    finally {
      // close the result set
      if (rs != null) {
       rs.close();
      // close the statement
      if (ps != null) {
       ps.close();
      // close the connection
      if (conn != null) {
       conn.close();
  }
}
```

The output for PgqlExample17.java is:

For more examples of INSERT statement, see the INSERT section in the PGQL specification.

Example A-45 PgqlExample18.java (Update)

PgqlExample18.java updates several properties of vertices and edges that are matched in the FROM clause of an UPDATE statement.

```
import java.sql.Connection;
import oracle.pg.rdbms.pgql.PgqlConnection;
import oracle.pg.rdbms.pgql.PgqlResultSet;
import oracle.pg.rdbms.pgql.PgqlStatement;
import oracle.ucp.jdbc.PoolDataSourceFactory;
import oracle.ucp.jdbc.PoolDataSource;
/**
 * This example shows how to execute a PGQL UPDATE operation.
public class PgqlExample18
 public static void main(String[] args) throws Exception
    int idx=0;
   String host
                            = args[idx++];
   String port
                           = args[idx++];
   String sid
                            = args[idx++];
    String user
                            = args[idx++];
    String password
                            = args[idx++];
    String graph
                             = args[idx++];
    Connection conn = null;
    PgqlStatement ps = null;
    PgqlResultSet rs = null;
    try {
      //Get a jdbc connection
      PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
     pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
     pds.setURL("jdbc:oracle:thin:@"+host+":"+port +":"+sid);
     pds.setUser(user);
     pds.setPassword(password);
      conn = pds.getConnection();
      conn.setAutoCommit(false);
      // Get a PGQL connection
```



```
PgqlConnection pgqlConn = PgqlConnection.getConnection(conn);
      pgqlConn.setGraph(graph);
      // Create a PgqlStatement
     ps = pgqlConn.createStatement();
      // Execute update statement
      String pggl =
        "UPDATE p1 SET (p1.age = 47, p1.lname = 'Red'), "+
               p2 SET (p2.age = 29, p2.lname = 'White'), "+
                e SET (e.strength = 100) "+
       "FROM MATCH (p1) -[e:knows]-> (p2) "+
        "WHERE pl.fname = 'Jake' AND p2.fname = 'Amy'";
      ps.execute(pgql, /* query string */
                   "", /* query options */
                   "" /* modify options */);
      // Execute a query to verify update
     paal =
          "SELECT pl.fname AS fnamel, pl.lname AS lnamel, pl.age AS
age1, "+
                 p2.fname AS fname2, p2.lname AS lname2, p2.age AS
age2, e.strength "+
         "FROM MATCH (p1) -[e:knows]-> (p2)";
      rs = ps.executeQuery(pgql, "");
      // Print the results
     rs.print();
    finally {
     // close the result set
     if (rs != null) {
       rs.close();
     // close the statement
     if (ps != null) {
       ps.close();
      // close the connection
     if (conn != null) {
       conn.close();
  }
}
```

The output for PgqlExample18.java applied on a graph where PgqlExample17.java has been previously executed is:



```
| Erik | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null> | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null | <null
```

For more examples of UPDATE statement, see the UPDATE section in the PGQL specification.

Example A-46 PgqlExample19.java (Delete)

PgqlExample19.java deletes edges that are matched in the FROM clause of a DELETE statement.

```
import java.sql.Connection;
import oracle.pg.rdbms.pgql.PgqlConnection;
import oracle.pg.rdbms.pgql.PgqlResultSet;
import oracle.pg.rdbms.pgql.PgqlStatement;
import oracle.ucp.jdbc.PoolDataSourceFactory;
import oracle.ucp.jdbc.PoolDataSource;
 * This example shows how to execute a PGQL DELETE operation.
public class PgglExample19
 public static void main(String[] args) throws Exception
   int idx=0;
   String host
                            = args[idx++];
                           = args[idx++];
   String port
   String sid
                           = args[idx++];
   String user
                            = args[idx++];
   String password
                           = args[idx++];
    String graph
                             = args[idx++];
   Connection conn = null;
    PgqlStatement ps = null;
   PgglResultSet rs = null;
    try {
      //Get a jdbc connection
      PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
     pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
     pds.setURL("jdbc:oracle:thin:@"+host+":"+port +":"+sid);
     pds.setUser(user);
     pds.setPassword(password);
      conn = pds.getConnection();
     conn.setAutoCommit(false);
      // Get a PGQL connection
      PgqlConnection pgqlConn = PgqlConnection.getConnection(conn);
     pgqlConn.setGraph(graph);
      // Create a PgqlStatement
```



```
ps = pgqlConn.createStatement();
      // Execute delete statement
      String pgql =
        "DELETE e "+
        " FROM MATCH (p1) -[e:knows]-> (p2) "+
        " WHERE pl.fname = 'Jake'";
      ps.execute(pgql, /* query string */
                   "", /* query options */
                   "" /* modify options */);
      // Execute a query to verify delete
      pgql =
          "SELECT pl.fname AS fname1, p2.fname AS fname2 "+
          " FROM MATCH (p1) -[e:knows]-> (p2)";
      rs = ps.executeQuery(pgql, "");
      // Print the results
      rs.print();
    }
    finally {
      \ensuremath{//} close the result set
     if (rs != null) {
       rs.close();
      // close the statement
      if (ps != null) {
        ps.close();
      // close the connection
     if (conn != null) {
        conn.close();
     }
    }
  }
}
```

The output for PgqlExample19.java applied on a graph where PgqlExample18.java has been previously executed is:

```
+-----+
| FNAME1 | FNAME2 |
+------+
| Erik | Jane |
+-------+
```

For more examples of DELETE statement, see the DELETE section in the PGQL specification.

Example A-47 PgqlExample20.java (Multiple Modifications)

PgqlExample20.java executes multiple modifications in the same statement: an edge is inserted, vertex properties are updated, and another edge is deleted.

```
import java.sql.Connection;
```



```
import oracle.pg.rdbms.pgql.PgqlConnection;
import oracle.pg.rdbms.pgql.PgqlResultSet;
import oracle.pg.rdbms.pggl.PgglStatement;
import oracle.ucp.jdbc.PoolDataSourceFactory;
import oracle.ucp.jdbc.PoolDataSource;
/**
 * This example shows how to execute a PGQL
 * INSERT/UPDATE/DELETE operation.
public class PgqlExample20
{
  public static void main(String[] args) throws Exception
    int idx=0;
    String host
                             = args[idx++];
                            = args[idx++];
    String port
                            = args[idx++];
    String sid
                            = args[idx++];
    String user
    String password
                            = args[idx++];
    String graph
                             = args[idx++];
    Connection conn = null;
    PgqlStatement ps = null;
    PqqlResultSet rs = null;
    try {
      //Get a jdbc connection
      PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
      pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
      pds.setURL("jdbc:oracle:thin:@"+host+":"+port +":"+sid);
      pds.setUser(user);
      pds.setPassword(password);
      conn = pds.getConnection();
      conn.setAutoCommit(false);
      // Get a PGQL connection
      PgglConnection pgglConn = PgglConnection.getConnection(conn);
      pgqlConn.setGraph(graph);
      // Create a PgqlStatement
      ps = pgqlConn.createStatement();
      // Execute INSERT/UPDATE/DELETE statement
      String pgql =
        "INSERT EDGE f BETWEEN p2 AND p1 LABELS (knows) PROPERTIES (f.since
= e.since) "+
        "UPDATE p1 SET (p1.age = 30) "+
        " , p2 SET (p2.age = 25) "+
        "DELETE e "+
        " FROM MATCH (p1) -[e:knows]-> (p2) "+
        " WHERE pl.fname = 'Erik'";
```

```
ps.execute(pgql, /* query string */
                "", /* query options */
                "" /* modify options */);
    // Execute a query to verify INSERT/UPDATE/DELETE
   pgql =
        "SELECT pl.fname AS fnamel, pl.age AS agel, "+
       " p2.fname AS fname2, p2.age AS age2, e.since "+
       " FROM MATCH (p1) -[e:knows]-> (p2)";
    rs = ps.executeQuery(pggl, "");
    // Print the results
   rs.print();
 finally {
   // close the result set
   if (rs != null) {
     rs.close();
   // close the statement
   if (ps != null) {
     ps.close();
   // close the connection
   if (conn != null) {
     conn.close();
}
```

The output for PgqlExample20.java applied on a graph where PgqlExample19.java has been previously executed is:

For more examples of INSERT/UPDATE/DELETE statements, see the Combining INSERT, UPDATE and DELETE section in the PGQL specification.

Additional Options for PGQL Statement Execution

A.10.4.8.1 Additional Options for PGQL Statement Execution

Several options are available to influence PGQL statement execution. The following are the main ways to set query options:

- Through flags in the modify options string argument of execute
- Through Java JVM arguments.

The following table summarizes the main options for modifying PGQL statement execution.



Table A-4 PGQL Statement Modification Options

Option	Default	Options Flag	JVM Argument
Auto commit	true if JDBC auto commit is off, false if JDBC auto commit is on	AUTO_COMMIT=F	- Doracle.pg.rdbms.pgql.auto Commit=false
Delete cascade	true	DELETE_CASCADE=F	- Doracle.pg.rdbms.pgql.dele teCascade=false

- Turning Off PGQL Auto Commit
- Turning Off Cascading Deletion

A.10.4.8.1.1 Turning Off PGQL Auto Commit

When an INSERT, UPDATE, or DELETE operation is executed, a commit is performed automatically at the end of the PGQL execution so that changes are persisted on the RDBMS side.

The flag AUTO_COMMIT=F can be added to the options argument of execute or the flag Doracle.pg.rdbms.pgql.autoCommit=false can be set in the Java command line to turn off auto commit. Notice that when auto commit is off, you must perform any necessary commits or rollbacks on the JDBC connection in order to persist or cancel graph modifications.

Example A-48 Turn Off Auto Commit and Roll Back Changes

PgqlExample21.java turns off auto commit and performs a rollback of the changes.

```
import java.sql.Connection;
import oracle.pg.rdbms.pgql.PgqlConnection;
import oracle.pg.rdbms.pgql.PgqlResultSet;
import oracle.pg.rdbms.pgql.PgqlStatement;
import oracle.ucp.jdbc.PoolDataSourceFactory;
import oracle.ucp.jdbc.PoolDataSource;
/**
 * This example shows how to modify a PGQL graph
 * with auto commit off.
public class PgqlExample21
{
 public static void main(String[] args) throws Exception
   int idx=0;
   String host
                            = args[idx++];
                            = args[idx++];
    String port
    String sid
                           = args[idx++];
    String user
                           = args[idx++];
    String password
                           = args[idx++];
    String graph
                             = args[idx++];
```



```
Connection conn = null;
    PgqlStatement ps = null;
    PgglResultSet rs = null;
    try {
      //Get a jdbc connection
      PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
      pds.setURL("jdbc:oracle:thin:@"+host+":"+port +":"+sid);
     pds.setUser(user);
     pds.setPassword(password);
      conn = pds.getConnection();
      conn.setAutoCommit(false);
      // Get a PGQL connection
      PgqlConnection pgqlConn = PgqlConnection.getConnection(conn);
     pgglConn.setGraph(graph);
      // Create a PgqlStatement
     ps = pgqlConn.createStatement();
      // Delete all the edges in the graph
      String pggl =
        "DELETE e "+
       " FROM MATCH () -[e]-> ()";
      ps.execute(pgql, /* query string */
                 "AUTO COMMIT=F" /* modify options */);
      // Execute a query to verify deletion
     pgql =
          "SELECT COUNT(e) "+
          " FROM MATCH () -[e]-> ()";
      rs = ps.executeQuery(pggl, "");
      // Print the results
      System.out.println("Number of edges after deletion:");
      rs.print();
     rs.close();
      // Rollback the changes. This is possible because
      // AUTO COMMIT=F flag was used in execute
      conn.rollback();
      // Execute a query to verify rollback
     pgql =
          "SELECT COUNT(e) "+
          " FROM MATCH () -[e]-> ()";
      rs = ps.executeQuery(pgql, "");
      // Print the results
      System.out.println("Number of edges after rollback:");
      rs.print();
```

```
finally {
    // close the result set
    if (rs != null) {
        rs.close();
    }
    // close the statement
    if (ps != null) {
        ps.close();
    }
    // close the connection
    if (conn != null) {
        conn.close();
    }
}
```

PgqlExample21.java gives the following output for a graph with one edge:

A.10.4.8.1.2 Turning Off Cascading Deletion

When a vertex is deleted from a graph, all its input and output edges are also deleted automatically.

Using the flag <code>DELETE_CASCADE=F</code> in the <code>options</code> argument of <code>execute</code> of setting the flag or setting the flag <code>Doracle.pg.rdbms.pgql.autoCommit=false</code> in the Java command line lets you turn off cascading deletion. When a vertex with input or output edges is deleted and cascading deletion is off, an error is thrown to warn about the unsafe operation that you are trying to perform.

Example A-49 Turn Off Cascading Deletion

PgqlExample22.java attempts to delete a vertex with an output edge when cascading deletion is off.

```
import java.sql.Connection;
import oracle.pg.rdbms.pgql.PgqlConnection;
import oracle.pg.rdbms.pgql.PgqlStatement;
import oracle.pg.rdbms.pgql.PgqlToSqlException;
import oracle.ucp.jdbc.PoolDataSourceFactory;
import oracle.ucp.jdbc.PoolDataSource;
```



```
* This example shows the use of DELETE CASCADE flag.
public class PgqlExample22
 public static void main(String[] args) throws Exception
    int idx=0;
   String host
                            = args[idx++];
   String port
                           = args[idx++];
   String sid
                           = args[idx++];
    String user
                           = args[idx++];
    String password
                           = args[idx++];
    String graph
                           = args[idx++];
    Connection conn = null;
    PgglStatement ps = null;
    try {
      //Get a jdbc connection
      PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
      pds.setURL("jdbc:oracle:thin:@"+host+":"+port +":"+sid);
     pds.setUser(user);
     pds.setPassword(password);
      conn = pds.getConnection();
      conn.setAutoCommit(false);
      // Get a PGQL connection
      PgqlConnection pgqlConn = PgqlConnection.getConnection(conn);
     pgqlConn.setGraph(graph);
      // Create a PgqlStatement
     ps = pgqlConn.createStatement();
      // Delete all the vertices with output edges
      // This will throw an error
      String pggl =
       "DELETE v "+
       " FROM MATCH (v) -[e]-> ()";
     ps.execute(pgql, /* query string */
                "DELETE CASCADE=F" /* modify options */);
    catch (PgqlToSqlException ex) {
      System.out.println("Error in execution: " + ex.getMessage());
    finally {
      // close the statement
      if (ps != null) {
       ps.close();
```

```
}
// close the connection
if (conn != null) {
    conn.close();
}
}
}
```

PgqlExample22.java gives the following output for a graph with at least one edge:

Error in execution: Attempting to delete vertices with incoming/outgoing edges. Drop edges first or turn on $\texttt{DELETE_CASCADE}$ option

A.10.5 Using the Python Client to Execute PGQL Queries

You can use the new Python package OPG4Py for executing PGQL queries against Oracle Database. This new package contains a sub-package Pgq1 with one or more modules that wraps around the Java API in the oracle.pg.rdbms.pgq1 package.

See Python API Reference for more information.

- Creating a Property Graph Using the Python Client
- Dropping a Property Graph Using the Python Client
- Basic Query Execution
- Iterating a Query Result Set

A.10.5.1 Creating a Property Graph Using the Python Client

You can create a property graph using the CREATE PROPERTY GRAPH statement in Python.

Creating a Property Graph Using the Python Client

Launch the Python client as shown:

```
./bin/opg4py --no connect
```

Create a PGQL connection to connect to the database as shown:

```
>>> pgql_conn = opg4py.pgql.get_connection(<user>, <password>, <jdbc_url>)
PgqlConnection(schema: GRAPHUSER, graph: None)
```

Create a PGQL statement as shown:

```
>>> pgql_statement = pgql_conn.create_statement()
PgqlStatement(java pgql statement: oracle.pg.rdbms.pgql.PgqlStatement)
```

Define and execute the CREATE PROPERTY GRAPH statement as shown:



```
LABEL accounts
PROPERTIES ALL COLUMNS
)

EDGE TABLES (
bank_txns
SOURCE KEY (from_acct_id) REFERENCES bank_accounts
(acct_id)

DESTINATION KEY (to_acct_id) REFERENCES bank_accounts
(acct_id)

LABEL transfers PROPERTIES ALL COLUMNS
)
"""
```

where *<graph_name>* is the name of the graph.

```
pgql statement.execute(pgql)
```

The graph gets created.

A.10.5.2 Dropping a Property Graph Using the Python Client

You can drop a property graph using the DROP PROPERTY GRAPH statement in Python.

Dropping a Property Graph Using the Python Client

Define and execute the DROP PROPERTY GRAPH statement as shown:

```
>>> pgql = "DROP PROPERTY GRAPH <graph name>"
```

where *<graph_name>* is the name of the graph.

```
>>> pgql statement.execute(pgql)
```

The graph gets dropped.

A.10.5.3 Basic Query Execution

You can execute PGQL queries using the <code>opg4py.pgql</code> Python wrapper.

Executing PGQL Queries Using the Python Client

Set the graph for querying as shown:

```
>>> pgql_conn.set_graph("<graph name>")
```

where *<graph_name>* is the name of the graph.

• Define and execute the PGQL SELECT query. For example,

```
>>> pgql = "SELECT e.from_acct_id, e.to_acct_id, e.amount FROM
MATCH (n:accounts) -[e:transfers]-> (m:accounts) on bank_graph
limit 10"
```



Execute and print the result set as shown:

PgqlResultSet(java_pgql_result_set: oracle.pg.rdbms.pgql.PgqlResultSet, #
of results: 0)

Also, you can convert the PGQL result set obtained in the preceding code to a Pandas dataframe using the to pandas() method.

Note:

The pandas package must be installed in your system to successfully execute the call to to_pandas(). This package is automatically installed at the time of the Python client installation for versions Python 3.8 and Python 3.9. However, if your call to to_pandas() fails, verify if the pandas module is installed in your system. In case the module is found missing or your Python version differs from the earlier mentioned versions, then install the pandas package manually.

A.10.5.4 Iterating a Query Result Set

You can iterate your query result set using the methods in PgglResultSet.

You can position the cursor for iterating your query result set using the following methods:

- first() : boolean
- next() : boolean
- previous(): boolean
- last() : boolean
- before first()
- after last()
- absolute(target row value) : boolean
- relative(offset value) : boolean



```
Once the cursor is positioned at the desired row, you can use the following getters to
obtain values:
   get(column idx) : Object
   get(column name) : Object
   get boolean(column idx) : boolean
   get boolean(column name) : boolean
   get date(column idx) : datetime.date
   get date(column name) : datetime.date
   get float(column idx) : Float
   get float(column name) : Float
   get integer(column idx) : Integer
   get integer(column name) : Integer
   get list(column idx) : List
   get list(column name) : List
   get_string(column_idx) : String
   get string(column name) : String
   get time(column idx) : datetime.time
   get time(column name) : datetime.time
   get time with timezone(column idx) : datetime.time
   get time with timezone(column name) : datetime.time
   get timestamp(column idx) : datetime.datetime
   get timestamp(column name) : datetime.datetime
   get timestamp with timezone(column idx) : datetime.datetime
   get timestamp with timezone(column name) : datetime.datetime
   get value type(column idx) : Integer
   get value type (column name) : Integer
   get vertex labels(column idx) : List
   get vertex labels(column name) : List
```

See Retrieving PGQL-on-RDBMS results documentation for more information.

fetchone(): Tuple

fetchall(): List of tuples

fetchmany (no of rows): List of tuples



The following code samples illustrate cursor operations for iterating a result set using few of the cursor position and getter methods. These examples reference the query result set obtained in the example in the previous section.

```
# Call first() and retrieve value for "FROM ACCT ID"
>>> pgql result set.first()
>>> pgql result set.get("FROM ACCT ID")
781.0
# Call next() and retrieve value for "FROM ACCT ID"
>>> pgql result set.next()
True
>>> pgql result set.get("FROM ACCT ID")
978.0
# Call last() and retrieve value for "FROM ACCT ID"
>>> pgql result set.last()
True
>>> pgql result set.get("FROM ACCT ID")
842.0
# Call previous() and retrieve value for "FROM ACCT ID"
>>> pgql result set.previous()
>>> pgql result set.get("FROM ACCT ID")
838.0
# Reset the result set and offset by 6. Then retrieve value for
"FROM ACCT ID"
>>> pgql result set.before first()
>>> pgql result set.relative(6)
>>> pgql result set.get("FROM ACCT ID")
925.0
# Reach the end of the result set and offset by -2. Then retrieve value for
"FROM ACCT ID"
>>> pgql result set.after_last()
>>> pgql result set.relative(-2)
True
>>> pgql result set.get("FROM ACCT ID")
838.0
# Call absolute() and provide an absolute row value. Then retrieve value for
"FROM ACCT ID"
>>> pggl result set.absolute(3)
True
>>> pgql result set.get float("FROM ACCT ID")
900.0
# Get a specific row or a set of rows
>>> pgql result set.get slice(0,1)
[781.0, 712.0, 1000.0]
```



```
>>> pgql_result_set.get_row(0)
[781.0, 712.0, 1000.0]
```

Iterating a Result Set Using the Python Index Operator

You can also iterate through the query result set using the Python index operator as shown:

```
# Retrieving a value from a tuple
>>> pgql_result_set[4, "double", "FROM_ACCT_ID"]
907.0

# Retrieving a value using index value
>>> pgql_result_set[4].get("FROM_ACCT_ID")
907.0

# Fetch a row or a set of rows
>>> pgql_result_set[0:1]
[781.0, 712.0, 1000.0]
```

Iterating a Result Set Using a Python loop

Optionally, you can also iterate through the query result set using a Python loop. For example:

Iterating a Result Set Using Fetch Methods

You can iterate through the query result set and fetch rows using the fetch methods. For example:

```
# Using the fetch methods to fetch rows from the result set
>>> pgql_result_set.fetchone()
(781.0, 712.0, 1000.0)
>>> pgql_result_set.fetchmany(4)
[(190.0, 555.0, 1000.0), (191.0, 329.0, 1000.0), (198.8. 57.0,
1000.0), (220.0, 441.0, 1000.0)]
>>> pgql_result_set.fetchall()
[(251.0, 387.0, 1000.0), (254.0, 188.0, 1000.0), (259.0, 305.0,
1000.0), (261.0, 145.0, 1000.0), (263.0, 40.0, 1000.0)]
```



A.10.6 Performance Considerations for PGQL Queries

Many factors affect the performance of PGQL queries in Oracle Database. The following are some recommended practices for query performance.

- Query Optimizer Statistics
- Parallel Query Execution
- Optimizer Dynamic Sampling
- Bind Variables
- Path Queries

Query Optimizer Statistics

Good, up-to-date query optimizer statistics are critical for query performance. Ensure that you run OPG_APIS.ANALYZE_PG after any significant updates to your property graph data.

Parallel Query Execution

Use parallel query execution to take advantage of Oracle's parallel SQL engine. Parallel execution often gives a significant speedup versus serial execution. Parallel execution is especially critical for path queries evaluated using the recursive WITH strategy.

See also the *Oracle Database VLDB and Partitioning Guide* for more information about parallel query execution.

Optimizer Dynamic Sampling

Due to the inherent flexibility of the graph data model, static information may not always produce optimal query plans. In such cases, dynamic sampling can be used by the query optimizer to sample data at run time for better query plans. The amount of data sampled is controlled by the dynamic sampling level used. Dynamic sampling levels range from 0 to 11. The best level to use depends on a particular dataset and workload, but levels of 2 (default), 6, or 11 often give good results.

See also Supplemental Dynamic Statistics in the Oracle Database SQL Tuning Guide.

Bind Variables

Use bind variables for constants whenever possible. The use of bind variables gives a very large reduction in query compilation time, which dramatically increases throughput for query workloads with queries that differ only in the constant values used. In addition, queries with bind variables are less vulnerable to injection attacks.

Path Queries

Path queries in PGQL that use the + (plus sign) or * (asterisk) operator to search for arbitrary length paths require special consideration because of their high computational complexity. You should use parallel execution and use the DISTINCT option for Recursive WITH (USE_DIST_RW=T) for the best performance. Also, for large, highly connected graphs, it is a good idea to use MAX_PATH_LEN=n to limit the number of repetitions of the recursive step to a reasonable number. A good strategy can be to start with a small repetition limit, and iteratively increase the limit to find more and more results.



A.11 OPG_APIS Package Subprograms

The OPG_APIS package contains subprograms (functions and procedures) for working with property graphs in an Oracle database.

To use the subprograms in this chapter, you must understand the conceptual and usage information in earlier chapters of this book.

This chapter provides reference information about the subprograms, in alphabetical order.

- OPG APIS.ANALYZE PG
- OPG_APIS.CF
- OPG APIS.CF CLEANUP
- OPG APIS.CF PREP
- OPG_APIS.CLEAR_PG
- OPG_APIS.CLEAR_PG_INDICES
- OPG_APIS.CLONE_GRAPH
- OPG_APIS.COUNT_TRIANGLE
- OPG_APIS.COUNT_TRIANGLE_CLEANUP
- OPG_APIS.COUNT_TRIANGLE_PREP
- OPG APIS.COUNT TRIANGLE RENUM
- OPG_APIS.CREATE_EDGES_TEXT_IDX
- OPG APIS.CREATE PG
- OPG APIS.CREATE PG SNAPSHOT TAB
- OPG APIS.CREATE PG TEXTIDX TAB
- OPG APIS.CREATE STAT TABLE
- OPG_APIS.CREATE_SUB_GRAPH
- OPG APIS.CREATE VERTICES TEXT IDX
- OPG_APIS.DROP_EDGES_TEXT_IDX
- OPG APIS.DROP PG
- OPG APIS.DROP PG VIEW
- OPG_APIS.DROP_VERTICES_TEXT_IDX
- OPG_APIS.ESTIMATE_TRIANGLE_RENUM
- OPG_APIS.EXP_EDGE_TAB_STATS
- OPG_APIS.EXP_VERTEX_TAB_STATS
- OPG APIS.FIND CC MAPPING BASED
- OPG APIS.FIND CLUSTERS CLEANUP
- OPG_APIS.FIND_CLUSTERS_PREP
- OPG_APIS.FIND_SP



- OPG_APIS.FIND_SP_CLEANUP
- OPG_APIS.FIND_SP_PREP
- OPG_APIS.GET_BUILD_ID
- OPG_APIS.GET_GEOMETRY_FROM_V_COL
- OPG_APIS.GET_GEOMETRY_FROM_V_T_COLS
- OPG_APIS.GET_LATLONG_FROM_V_COL
- OPG_APIS.GET_LATLONG_FROM_V_T_COLS
- OPG_APIS.GET_LONG_LAT_GEOMETRY
- OPG_APIS.GET_LATLONG_FROM_V_COL
- OPG_APIS.GET_LONGLAT_FROM_V_T_COLS
- OPG_APIS.GET_OPG_VERSION
- OPG_APIS.GET_SCN
- OPG_APIS.GET_VERSION
- OPG_APIS.GET_WKTGEOMETRY_FROM_V_COL
- OPG_APIS.GET_WKTGEOMETRY_FROM_V_T_COLS
- OPG_APIS.GRANT_ACCESS
- OPG_APIS.IMP_EDGE_TAB_STATS
- OPG_APIS.IMP_VERTEX_TAB_STATS
- OPG APIS.PR
- OPG APIS.PR CLEANUP
- OPG_APIS.PR_PREP
- OPG_APIS.PREPARE_TEXT_INDEX
- OPG_APIS.RENAME_PG
- OPG APIS.SPARSIFY GRAPH
- OPG_APIS.SPARSIFY_GRAPH_CLEANUP
- OPG_APIS.SPARSIFY_GRAPH_PREP

A.11.1 OPG APIS.ANALYZE PG

```
OPG_APIS.ANALYZE_PG(
graph_name IN VARCHAR2,
estimate_percent IN NUMBER,
method_opt IN VARCHAR2,
degree IN NUMBER,
cascade IN BOOLEAN,
no_invalidate IN BOOLEAN,
force IN BOOLEAN DEFAULT FALSE,
options IN VARCHAR2 DEFAULT NULL);
```



Description

Hathers, for a given property graph, statistics for the VT\$, GE\$, IT\$, and GT\$ tables.

Parameters

graph_name

Name of the property graph.

estimate_percent

Percentage of rows to estimate in the schema tables (NULL means compute). The valid range is [0.000001,100]. Use the constant <code>DBMS_STATS.AUTO_SAMPLE_SIZE</code> to have Oracle Database determine the appropriate sample size for good statistics. This is the usual default.

mrthod_opt

Accepts either of the following options, or both in combination, for the internal property graph schema tables:

- FOR ALL [INDEXED | HIDDEN] COLUMNS [size clause]
- FOR COLUMNS [size clause] column|attribute [size_clause] [,column|attribute [size clause]...]

size_clause is defined as size_clause := SIZE {integer | REPEAT | AUTO |
SKEWONLY}

- integer: Number of histogram buckets. Must be in the range [1,254].
- REPEAT: Collects histograms only on the columns that already have histograms.
- AUTO: Oracle Database determines the columns to collect histograms based on data distribution and the workload of the columns.
- SKEWONLY: Oracle Database determines the columns to collect histograms based on the data distribution of the columns

column is defined as column := column name | (extension)

- column name: name of a column
- extension: Can be either a column group in the format of (column name, column name [, ...]) or an expression.

The usual default is: FOR ALL COLUMNS SIZE AUTO

degree

Degree of parallelism for the property graph schema tables. The usual default for degree is NULL, which means use the table default value specified by the DEGREE clause in the CREATE TABLE or ALTER TABLE statement. Use the constant <code>DBMS_STATS.DEFAULT_DEGREE</code> to specify the default value based on the initialization parameters. The <code>AUTO_DEGREE</code> value determines the degree of parallelism automatically. This is either 1 (serial execution) or <code>DEFAULT_DEGREE</code> (the system default value based on number of CPUs and initialization parameters) according to size of the object.



cascade

Gathers statistics on the indexes for the property graph schema tables. Use the constant <code>DBMS_STATS.AUTO_CASCADE</code> to have Oracle Database determine whether index statistics are to be collected or not. This is the usual default.

no invalidate

If TRUE, does not invalidate the dependent cursors. If FALSE, invalidates the dependent cursors immediately. If DBMS_STATS.AUTO_INVALIDATE (the usual default) is in effect, Oracle Database decides when to invalidate dependent cursors.

force

If TRUE, performs the operation even if one or more underlying tables are locked.

options

(Reserved for future use.)

Usage Notes

Only the owner of the property graph can call this procedure.

Examples

The following example gather statistics for property graph mypg.

```
EXECUTE OPG_APIS.ANALYZE_PG('mypg', estimate_percent=> 0.001, method_opt=>'FOR ALL
COLUMNS SIZE AUTO', degree=>4, cascade=>true, no_invalidate=>false, force=>true,
options=>NULL);
```

A.11.2 OPG_APIS.CF

Format

```
OPG_APIS.CF(

edge_tab_name IN VARCHAR2,
edge_label IN VARCHAR2,
rating_property IN VARCHAR2,
iterations IN NUMBER DEFAULT 10,
min_error IN NUMBER DEFAULT 0.001,
k IN NUMBER DEFAULT 5,
learning_rate IN NUMBER DEFAULT 0.0002,
decrease_rate IN NUMBER DEFAULT 0.95,
regularization IN NUMBER DEFAULT 0.095,
regularization IN NUMBER DEFAULT 0.02,
dop IN NUMBER DEFAULT 0.02,
wt_l IN/OUT VARCHAR2,
wt_r IN/OUT VARCHAR2,
wt_rI IN/OUT VARCHAR2,
wt_r1 IN/OUT VARCHAR2,
wt_i IN/OUT VARCHAR2,
wt_i IN/OUT VARCHAR2,
wt_i IN/OUT VARCHAR2,
wt_rd IN/OUT VARCHAR2,
wt_rd IN/OUT VARCHAR2,
tablespace IN VARCHAR2 DEFAULT NULL,
options IN VARCHAR2 DEFAULT NULL);
```

Description

Runs collaborative filtering using matrix factorization on the given graph. The resulting factors of the matrix product will be stored on the left and right tables.

Parameters

edge tab name

Name of the property graph edge table (GE\$).

edge_label

Label of the edges that hold the rating property.

rating_property

(Reserved for future use: Name of the rating property.)

iterations

Maximum number of iterations that should be performed. Default = 10.

min error

Minimal error to reach. If at some iteration the error value is lower than this value, the procedure finishes.. Default = 0.001.

k

Number of features for the left and right side products. Default = 5.

learning_rate

Learning rate for the gradient descent. Default = 0.0002.

decrease rate

(Reserved for future use: Decrease rate if the learning rate is too large for an effective gradient descent. Default = 0.95.)

regularization

An additional parameter to avoid overfitting. Default = 0.02

dop

Degree of parallelism. Default = 8.

wt I

Name of the working table that holds the left side of the matrix factorization.

wt_r

Name of the working table that holds the right side of the matrix factorization.

wt I1

Name of the working table that holds the left side intermediate step in the gradient descent.

wt r1

Name of the working table that holds the right side intermediate step in the gradient descent.

wt I

Name of the working table that holds intermediate matrix product.

wt Id

Name of the working table that holds intermediate left side delta in gradient descent.

wt rd

Name of the working table that holds intermediate right side delta in gradient descent.



tablespace

Name of the tablespace to use for storing intermediate data.

options

Additional settings for operation. An optional string with one or more (comma-separated) of the following values:

- 'INMEMORY=T' is an option for creating the schema tables with an 'inmemory' clause.
- 'IMC_MC_B=T' creates the schema tables with an INMEMORY MEMCOMPRESS BASIC clause.

Usage Notes

For information about collaborative filtering with RDF data, see SQL-Based Property Graph Analytics, especially Collaborative Filtering Overview and Examples.

If the working tables already exist, you can specify their names for the working table-related parameters. In this case, the algorithm can continue the progress of the previous iterations without recreating the tables.

If the working tables do not exist, or if you do not want to use existing working tables, you must first call the OPG_APIS.CF_PREP procedure, which creates the necessary working tables.

The final result of the collaborative filtering algorithm are the working tables wt_l and wt_r , which are the two factors of a matrix product. These matrix factors should be used when making predictions for collaborative filtering.

If (and only if) you have no interest in keeping the output matrix factors and the current progress of the algorithm for future use, you can call the OPG_APIS.CF_CLEANUP procedure to drop all the working tables that hold intermediate tables and the output matrix factors.

Examples

The following example calls the OPG_APIS.CF_PREP procedure to create the working tables, and then the OPG_APIS.CF procedures to run collaborative filtering on the phones graph using the edges with the rating label.

```
DECLARE
  wt 1 varchar2(32);
  wt r varchar2(32);
  wt 11 varchar2(32);
  wt r1 varchar2(32);
  wt i varchar2(32);
   wt ld varchar2(32);
   wt rd varchar2(32);
  edge_tab_name varchar2(32) := 'phonesge$';
edge_label varchar2(32) := 'rating';
   rating_property varchar2(32) := '';
  iterations integer := 100;
min_error number := 0.00
  min_error
                                             := 0.001;
                          integer
                                             := 5;
  learning_rate number := 0.001;
decrease_rate number := 0.95;
regularization number := 0.02;
dop number := 2;
tablespace varchar2(32) := null;
options varchar2(32) := null;
```



The following example assumes that OPG_APIS.CF_PREP had been run previously, and it specifies the various working tables that were created during that run. In this case, the preceding example automatically assigned suffixes like '\$\$CFL57' to the names of the working tables. (The output names can be printed when they are generated or be user-defined in the call to OPG_APIS.CF_PREP.) Thus, the following example can run more iterations of the algorithm using OPG_APIS.CF without needing to call OPG_APIS.CF_PREP first, thereby continuing the progress of the previous run.

```
DECLARE
 wt l varchar2(32) = 'phonesge$$CFL57';
 wt r varchar2(32) = 'phonesge$$CFR57';
 wt l1 varchar2(32) = 'phonesge$$CFL157';
 wt_r1 varchar2(32) = 'phonesge$$CFR157';
 wt i varchar2(32) = 'phonesge$$CFI57';
  wt ld varchar2(32) = 'phonesge$$CFLD57';
 wt rd varchar2(32) = 'phonesge$$CFRD57';
  edge tab name varchar2(32) := 'phonesge$';
  edge label varchar2(32) := 'rating';
  rating_property varchar2(32) := '';
 iterations inco,
min_error number
integer
 iterations integer := 100;
                              := 0.001;
                              := 5;
 learning_rate number
                              := 0.001;
  decrease rate number
                              := 0.95;
  regularization number
                              := 0.02;
           number
 tablespace
                  varchar2(32) := null;
                  varchar2(32) := null;
 options
BEGIN
opg apis.cf(edge tab name,edge label,rating property,iterations,min error,k,
            learning rate, decrease rate, regularization, dop,
            wt l, wt r, wt l1, wt r1, wt i, wt ld, wt rd, tablespace, options);
END:
```

A.11.3 OPG APIS.CF CLEANUP



Description

Preforms cleanup work after graph collaborative filtering has been done. All the working tables that hold intermediate tables and the output matrix factors are dropped.

Parameters

edge_tab_name

Name of the property graph edge table (GE\$).

wt I

Name of the working table that holds the left side of the matrix factorization.

wt i

Name of the working table that holds the right side of the matrix factorization.

wt I1

Name of the working table that holds the left side intermediate step in the gradient descent.

wt r1

Name of the working table that holds the right side intermediate step in the gradient descent.

wt

Name of the working table that holds intermediate matrix product.

wt Id

Name of the working table that holds intermediate left side delta in gradient descent.

wt rd

Name of the working table that holds intermediate right side delta in gradient descent.

options

(Reserved for future use.)

Usage Notes

Call this procedure only when you have no interest in keeping the output matrix factors and the current progress of the algorithm for future use.

Do **not** call this procedure if more predictions will be made using the resulting product factors ($wt\ 1$ and $wt\ r$ tables), unless you have previous made backup copies of these two tables.

See also the information about the OPG APIS.CF procedure.

Examples

The following example drops the working tables that were created in the example for the OPG_APIS.CF_PREP procedure.

```
DECLARE

wt_1 varchar2(32) = 'phonesge$$CFL57';

wt_r varchar2(32) = 'phonesge$$CFL57';

wt_11 varchar2(32) = 'phonesge$$CFL157';

wt_r1 varchar2(32) = 'phonesge$$CFR157';

wt_i varchar2(32) = 'phonesge$$CFI57';

wt_ld varchar2(32) = 'phonesge$$CFLD57';

wt_rd varchar2(32) = 'phonesge$$CFRD57';

BEGIN

opg_apis.cf_cleanup('phonesge$',wt_1,wt_r,wt_11,wt_r1,wt_i,wt_ld,wt_rd);
```



```
END;
```

A.11.4 OPG APIS.CF PREP

Format

Description

Preforms preparation work, including creating the necessary intermediate tables, for a later call to the OPG APIS.CF procedure that will perform collaborative filtering.

Parameters

edge_tab_name

Name of the property graph edge table (GE\$).

wt

Name of the working table that holds the left side of the matrix factorization.

wt r

Name of the working table that holds the right side of the matrix factorization.

wt I1

Name of the working table that holds the left side intermediate step in the gradient descent.

wt r1

Name of the working table that holds the right side intermediate step in the gradient descent.

wt I

Name of the working table that holds intermediate matrix product.

wt ld

Name of the working table that holds intermediate left side delta in gradient descent.

wt rd

Name of the working table that holds intermediate right side delta in gradient descent.

options

Additional settings for operation. An optional string with one or more (commaseparated) of the following values:

 'INMEMORY=T' is an option for creating the schema tables with an 'inmemory' clause.



 'IMC_MC_B=T' creates the schema tables with an INMEMORY MEMCOMPRESS BASIC clause.

Usage Notes

The names of the working tables can be specified or left as null parameters, If the name of any working table parameter is not specified, a name is automatically genenerated and is returned as an OUT parameter. The working table names can be used when you call the OPG APIS.CF procedure to run the collaborative filtering algorithm.

See also the Usage Notes and Examples for OPG APIS.CF.

Examples

The following example creates the working tables for a graph named phones, and it prints the names that were automatically generated for the working tables.

```
DECLARE
  wt 1 varchar2(32);
  wt_r varchar2(32);
  wt 11 varchar2(32);
  wt r1 varchar2(32);
  wt i varchar2(32);
  wt ld varchar2(32);
  wt rd varchar2(32);
BEGIN
  opg apis.cf prep('phonesge$',wt 1,wt r,wt 11,wt r1,wt i,wt ld,wt rd);
  dbms output.put line(' wt l ' || wt l);
  dbms output.put line(' wt r ' || wt r);
  dbms output.put line(' wt 11 ' || wt 11);
  dbms_output.put_line(' wt_r1 ' || wt_r1);
  dbms_output.put_line(' wt_i ' || wt_i);
  dbms output.put line(' wt ld ' || wt ld);
  dbms output.put line(' wt rd ' || wt rd);
END:
```

A.11.5 OPG_APIS.CLEAR_PG

Format

```
OPG_APIS.CLEAR_PG(
          graph_name IN VARCHAR2);
```

Description

Clears all data from a property graph.

Parameters

graph name

Name of the property graph.

Usage Notes

This procedure removes all data in the property graph by deleting data in the graph tables (VT\$, GE\$, and so on).

Examples

The following example removes all data from the property graph named mypg.

```
EXECUTE OPG_APIS.CLEAR_PG('mypg');
```

A.11.6 OPG_APIS.CLEAR_PG_INDICES

Format

```
OPG_APIS.CLEAR_PG(
          graph_name IN VARCHAR2);
```

Description

Removes all text index metadata in the IT\$ table of the property graph.

Parameters

graph_name

Name of the property graph.

Usage Notes

This procedure does not actually remove text index data

Examples

The following example removes all index metadata of the property graph named mypg.

```
EXECUTE OPG_APIS.CLEAR_PG_INDICES('mypg');
```

A.11.7 OPG_APIS.CLONE_GRAPH

Format

Description

Makes a clone of the original graph, giving the new graph a new name.

Parameters

orgGraph

Name of the original property graph.

newGraph

Name of the new (clone) property graph.



dop

Degree of parallelism for the operation.

num_hash_ptns

Number of hash partitions used to partition the vertices and edges tables. It is recommended to use a power of 2 (2, 4, 8, 16, and so on).

tbs

Name of the tablespace to hold all the graph data and index data.

Usage Notes

The original property graph must aleady exist in the database.

Examples

The following example creates a clone graph named mypgclone from the property graph mypg in the tablespace my ts using a degree of parallelism of 4 and 8 partitions.

```
EXECUTE OPG_APIS.CLONE_GRAPH('mypg', 'mypgclone', 4, 8, 'my_ts');
```

A.11.8 OPG_APIS.COUNT_TRIANGLE

Format

```
OPG_APIS.COUNT_TRIANGLE(

edge_tab_name IN VARCHAR2,

wt_und IN OUT VARCHAR2,

num_sub_ptns IN NUMBER DEFAULT 1,

dop IN INTEGER DEFAULT 1,

tbs IN VARCHAR2 DEFAULT NULL,

options IN VARCHAR2 DEFAULT NULL
) RETURN NUMBER;
```

Description

Performs triangle counting in property graph.

Parameters

edge tab name

Name of the property graph edge table.

wt und

A working table holding an undirected version of the graph.

num_sub_ptns

Number of logical subpartitions used in calculating triangles . Must be a positive integer, power of 2 (1, 2, 4, 8, ...). For a graph with a relatively small maximum degree, use the value 1 (the default).

dop

Degree of parallelism for the operation. The default is 1.

tbs

Name of the tablespace to hold the data stored in working tables.



options

Additional settings for the operation:

'PDML=T' enables parallel DML.

Usage Notes

The property graph edge table must exist in the database, and the OPG_APIS.COUNT_TRIANGLE_PREP. procedure must already have been executed.

Examples

The following example performs triangle counting in the property graph named connections

```
set serveroutput on
DECLARE
 wt1 varchar2(100);
                     -- intermediate working table
 wt2 varchar2(100);
 wt3 varchar2(100);
 n number;
  opg_apis.count_triangle_prep('connectionsGE$', wt1, wt2, wt3);
  n := opg_apis.count_triangle(
     'connectionsGE$',
      wt1,
      num sub ptns=>1,
      dop=>2,
      tbs => 'MYPG TS',
      options=>'PDML=T'
  dbms output.put line('total number of triangles ' || n);
END;
```

A.11.9 OPG_APIS.COUNT_TRIANGLE_CLEANUP

Format

```
COUNT_TRIANGLE_CLEANUP(
edge_tab_name IN VARCHAR2,
wt_undBM IN VARCHAR2,
wt_rnmap IN VARCHAR2,
wt_undAM IN VARCHAR2,
options IN VARCHAR2 DEFAULT NULL);
```

Description

Cleans up and drops the temporary working tables used for triangle counting.

Parameters

edge tab name

Name of the property graph edge table.

wt undBM

A working table holding an undirected version of the original graph (before renumbering optimization).

wt_rnmap

A working table that is a mapping table for renumbering optimization.

wt undAM

A working table holding the undirected version of the graph data after applying the renumbering optimization.

options

Additional settings for operation. An optional string with one or more (comma-separated) of the following values:

PDML=T enables parallel DML.

Usage Notes

You should use this procedure to clean up after triangle counting.

The working tables must exist in the database.

Examples

The following example performs triangle counting in the property graph named connections, and drops the working table after it has finished.

```
set serveroutput on
DECLARE
 wt1 varchar2(100); -- intermediate working table
 wt2 varchar2(100);
 wt3 varchar2(100);
 n number;
BEGIN
  opg apis.count triangle prep('connectionsGE$', wt1, wt2, wt3);
  n := opg apis.count triangle renum(
     'connectionsGE$',
      wt1,
      wt2,
      wt3,
      num sub ptns=>1,
      dop = >2,
      tbs => 'MYPG TS',
      options=>'PDML=T'
  dbms_output.put_line('total number of triangles ' || n);
  opg apis.count triangle cleanup('connectionsGE$', wt1, wt2, wt3);
```

A.11.10 OPG_APIS.COUNT_TRIANGLE_PREP



Description

Prepares for running triangle counting.

Parameters

edge_tab_name

Name of the property graph edge table.

wt undBM

A working table holding an undirected version of the original graph (before renumbering optimization).

wt rnmap

A working table that is a mapping table for renumbering optimization.

wt undAM

A working table holding the undirected version of the graph data after applying the renumbering optimization.

options

Additional settings for operation. An optional string with one or more (commaseparated) of the following values:

- CREATE_UNDIRECTED=T
- REUSE UNDIRECTED TAB=T

Usage Notes

The property graph edge table must exist in the database.

Examples

The following example prepares for triangle counting in a property graph named connections.

```
set serveroutput on
DECLARE
 wt1 varchar2(100); -- intermediate working table
 wt2 varchar2(100);
 wt3 varchar2(100);
 n number;
  opg apis.count triangle prep('connectionsGE$', wt1, wt2, wt3);
 n := opg apis.count triangle renum(
     'connectionsGE$',
     wt1,
     wt2,
     wt3,
     num sub ptns=>1,
      dop=>2,
      tbs => 'MYPG TS',
      options=>'CREATE UNDIRECTED=T, REUSE UNDIREC TAB=T'
  dbms output.put line('total number of triangles ' || n);
```



```
END;
```

A.11.11 OPG_APIS.COUNT_TRIANGLE_RENUM

Format

```
COUNT_TRIANGLE_RENUM(
edge_tab_name IN VARCHAR2,
wt_undBM IN VARCHAR2,
wt_rnmap IN VARCHAR2,
wt_undAM IN VARCHAR2,
num_sub_ptns IN INTEGER DEFAULT 1,
dop IN INTEGER DEFAULT 1,
tbs IN VARCHAR2 DEFAULT NULL,
options IN VARCHAR2 DEFAULT NULL
) RETURN NUMBER;
```

Description

Performs triangle counting in property graph, with the optimization of renumbering the vertices of the graph by their degree.

Parameters

edge_tab_name

Name of the property graph edge table.

wt undBM

A working table holding an undirected version of the original graph (before renumbering optimization).

wt_rnmap

A working table that is a mapping table for renumbering optimization.

wt undAM

A working table holding the undirected version of the graph data after applying the renumbering optimization.

num_sub_ptns

Number of logical subpartitions used in calculating triangles . Must be a positive integer, power of 2 (1, 2, 4, 8, ...). For a graph with a relatively small maximum degree, use the value 1 (the default).

aop

Degree of parallelism for the operation. The default is 1 (no parallelism).

tbs

Name of the tablespace to hold the data stored in working tables.

options

Additional settings for operation. An optional string with one or more (comma-separated) of the following values:

PDML=T enables parallel DML.



Usage Notes

This function makes the algorithm run faster, but requires more space.

The property graph edge table must exist in the database, and the OPG_APIS.COUNT_TRIANGLE_PREP procedure must already have been executed.

Examples

The following example performs triangle counting in the property graph named connections. It does not perform the cleanup after it finishes, so you can count triangles again on the same graph without calling the preparation procedure.

```
set serveroutput on
DECLARE
 wt1 varchar2(100); -- intermediate working table
 wt2 varchar2(100);
 wt3 varchar2(100);
 n number;
  opg apis.count triangle prep('connectionsGE$', wt1, wt2, wt3);
  n := opg apis.count triangle renum(
     'connectionsGE$',
      wt1,
      wt2,
      wt3,
      num_sub_ptns=>1,
      dop = >2,
      tbs => 'MYPG TS',
      options=>'PDML=T'
  dbms output.put line('total number of triangles ' || n);
END;
```

A.11.12 OPG_APIS.CREATE_EDGES_TEXT_IDX

Format

```
OPG_APIS.CREATE_EDGES_TEXT_IDX(
    graph_owner IN VARCHAR2,
    pref_owner IN VARCHAR2,
    pref_owner IN VARCHAR2 DEFAULT NULL,
    datastore IN VARCHAR2 DEFAULT NULL,
    filter IN VARCHAR2 DEFAULT NULL,
    storage IN VARCHAR2 DEFAULT NULL,
    wordlist IN VARCHAR2 DEFAULT NULL,
    stoplist IN VARCHAR2 DEFAULT NULL,
    lexer IN VARCHAR2 DEFAULT NULL,
    dop IN INTEGER DEFAULT NULL,
    options IN VARCHAR2 DEFAULT NULL,);
```

Description

Creates a text index on a property graph edge table.

Parameters

graph_owner

Owner of the property graph.

graph_name

Name of the property graph.

pref_owner

Owner of the preference.

datastore

The way that documents are stored.

filter

The way that documents can be converted to plain text.

storage

The way that the index data is stored.

wordlist

The way that stem and fuzzy queries should be expanded

stoplist

The words or themes that are not to be indexed.

lexe

The language used for indexing.

dop

The degree of parallelism used for index creation.

options

Additional settings for index creation.

Usage Notes

The property graph must exist in the database.

You must have the ALTER SESSION privilege to run this procedure.

Examples

The following example creates a text index on the edge table of property graph mypg, which is owned by user SCOTT, using the lexer OPG_AUTO_LEXER and a degree of parallelism of 4.

```
EXECUTE OPG_APIS.CREATE_EDGES_TEXT_IDX('SCOTT', 'mypg', 'MDSYS', null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, n
```

A.11.13 OPG_APIS.CREATE_PG



```
tbs IN VARCHAR2 DEFAULT NULL, options IN VARCHAR2 DEFAULT NULL);
```

Description

Creates, for a given property graph name, the necessary property graph schema tables that are necessary to store data about vertices, edges, text indexes, and snapshots.

Parameters

graph name

Name of the property graph.

qob

Degree of parallelism for the operation.

num_hash_ptns

Number of hash partitions used to partition the vertices and edges tables. It is recommended to use a power of 2 (2, 4, 8, 16, and so on).

tbs

Name of the tablespace to hold all the graph data and index data.

options

Options that can be used to customize the creation of indexes on schema tables. (One or more, comma separated.)

- 'SKIP_INDEX=T' skips the default index creation.
- 'SKIP ERROR=T 'ignores errors encountered during table/index creation.
- 'INMEMORY=T' creqtes the schema tables with an INMEMORYclause.
- 'IMC_MC_B=T' creates the schema tables with an INMEMORY BASIC clause.

Usage Notes

You must have the CREATE TABLE and CREATE INDEX privileges to call this procedure.

By default, all the schema tables will be created with basic compression enabled.

Examples

The following example creates a property graph named mypg in the tablespace my_ts using eight partitions.

```
EXECUTE OPG_APIS.CREATE_PG('mypg', 4, 8, 'my_ts');
```

A.11.14 OPG_APIS.CREATE_PG_SNAPSHOT_TAB

```
OPG_APIS.CREATE_PG_SNAPSHOT_TAB(
    graph_owner IN VARCHAR2,
    graph_name IN VARCHAR2,
    dop IN INTEGER DEFAULT NULL,
    tbs IN VARCHAR2 DEFAULT NULL,
    options IN VARCHAR2 DEFAULT NULL);
```



or

Description

Creates, for a given property graph name, the necessary property graph schema table (<graph_name>SS\$) that stores data about snapshots for the graph.

Parameters

graph_owner

Name of the owner of the property graph.

graph_name

Name of the property graph.

dop

Degree of parallelism for the operation.

tbs

Name of the tablespace to hold all the graph snapshot data and associated index.

options

Additional settings for the operation:

- 'INMEMORY=T' is an option for creating the schema tables with an 'inmemory' clause.
- 'IMC_MC_B=T' creates the schema tables with an INMEMORY MEMCOMPRESS BASIC clause.

Usage Notes

You must have the CREATE TABLE privilege to call this procedure.

The created snapshot table has the following structure, which may change between releases.

Name	Null? Type
SSID	NOT NULL NUMBER
CONTENTS	BLOB
SS_FILE	BINARY FILE LOB
TS	TIMESTAMP(6) WITH TIME ZONE
SS COMMENT	VARCHAR2 (512)

By default, all schema tables will be created with basic compression enabled.

Examples

The following example creates a snapshot table for property graph mypg in the current schema, with a degree of parallelism of 4 and using the MY TS tablespace.

```
EXECUTE OPG_APIS.CREATE_PG_SNAPSHOT_TAB('mypg', 4, 'my_ts');
```



A.11.15 OPG_APIS.CREATE_PG_TEXTIDX_TAB

Format

```
OPG_APIS.CREATE_PG_TEXTIDX_TAB(
    graph_owner IN VARCHAR2,
    graph_name IN VARCHAR2,
    dop IN INTEGER DEFAULT NULL,
    tbs IN VARCHAR2 DEFAULT NULL);

Or

OPG_APIS.CREATE_PG_TEXTIDX_TAB(
    graph_name IN VARCHAR2,
    dop IN INTEGER DEFAULT NULL,
    tbs IN VARCHAR2 DEFAULT NULL,
    options IN VARCHAR2 DEFAULT NULL,
    options IN VARCHAR2 DEFAULT NULL);
```

Description

Creates, for a given property graph name, the necessary property graph text index schema table (<graph_name>IT\$) that stores data for managing text index metadata for the graph.

Parameters

graph owner

Name of the owner of the property graph.

graph_name

Name of the property graph.

qob

Degree of parallelism for the operation.

tbs

Name of the tablespace to hold all the graph index metadata and associated index.

options

Additional settings for the operation:

- 'INMEMORY=T' is an option for creating the schema tables with an 'inmemory' clause.
- 'IMC_MC_B=T' creates the schema tables with an INMEMORY MEMCOMPRESS BASIC clause.

Usage Notes

You must have the CREATE TABLE privilege to call this procedure.

The created index metadata table has the following structure, which may change between releases.

```
EIN nvarchar2(80) not null, -- index name
ET number, -- entity type 1 - vertex, 2 -edge
IT number, -- index type 1 - auto 0 - manual
```



```
SE
                                           number,
                                                                                                                        -- search engine 1 -solr, 0 - lucene
                                                 nvarchar2(3100),
                          V
DT number,

LOC nvarchar2(3100),
NUMDIRS number,
VERSION nvarchar2(100),
USEDT number,
STOREF number,
CF nvarchar2(3100),
SS nvarchar2(3100),
Number,
Number,
Number,
Number,
Number,
Number,
Number,
Number,
Number,
Number,
Number,
Number,
Number,
Number,
Number,
Number,
Number,
Number,
Number,
Number,
Number,
Number,
Number of shards
Number of shards
Number of shards
Number of shards
Number of shards
Number of shards
Number of shards
Number of shards
Number of shards
Number of shards
Number of shards
Number of shards
Number of shards
Number of shards
Number of shards
Number of shards
Number of shards
Number of shards
Number of shards
Number of shards
Number of shards
Number of shards
Number of shards
Number of shards
Number of shards
Number of shards
Number of shards
Number of shards
Number of shards
Number of shards
Number of shards
Number of shards
Number of shards
Number of shards
Number of shards
Number of shards
Number of shards
Number of shards
Number of shards
                                 K
                                                                                                                     -- property key use an empty space when
there is no {\rm K}/{\rm V}
                                                                                                                       -- directory type 1 - MMAP, 2 - FS, 3 -
JDBC
                                                                                                                    -- directory location (1, 2)
                                                                                                                    -- property key used to index CAN BE NULL
                                                                                                                  -- user data type (1 or 0)
                                                                                                                   -- store fields into lucene
                                                                                                                      -- solr server admin url
                                MS number,
PO nvarchar2(3100),
DS nvarchar2(3100),
FIL nvarchar2(3100),
STR nvarchar2(3100),
WL nvarchar2(3100),
nvarchar2(3100),
                                                                                                                      -- maximum shards per node
                                                                                                                     -- preferred owner oracle text
                                                                                                                      -- storage
                                                                                                                      -- word list
                                                                                                                     -- stop list
                                 LXR nvarchar2(3100),
                                                                                                                      -- lexer
                                 OPTS nvarchar2(3100),
                                                                                                                     -- options
                                 primary key (EIN, K, ET)
```

By default, all schema tables will be created with basic compression enabled.

Examples

The following example creates a property graph text index metadata table for property graph mypg in the current schema, with a degree of parallelism of 4 and using the MY_TS tablespace.

```
EXECUTE OPG_APIS.CREATE_PG_TEXTIDX_TAB('mypg', 4, 'my_ts');
```

A.11.16 OPG_APIS.CREATE_STAT_TABLE

Format

Description

Creates a table that can hold property graph statistics.

Parameters

stattab

Name of the table to hold statistics

tblapace

Name of the tablespace to hold the statistics table. If none is specified, then the statistics table will be created in the user's default tablespace.

Usage Notes

You must have the CREATE TABLE privilege to call this procedure.

The statistics table has the following columns. Note that the columns and their types may vary between releases.

STATID VARCHAR2 (128) TYPE CHAR (1) VERSION NUMBER FLAGS NUMBER C1 VARCHAR2 (128) C2 VARCHAR2 (128) C3 VARCHAR2 (128) C4 VARCHAR2 (128) C5 VARCHAR2 (128) N1 NUMBER N2 NUMBER N3 NUMBER N4 NUMBER N5 NUMBER N6 NUMBER N7 NUMBER N8 NUMBER N9 NUMBER N10 NUMBER N10 NUMBER N11 NUMBER			
VERSION NUMBER FLAGS NUMBER C1 VARCHAR2 (128) C2 VARCHAR2 (128) C3 VARCHAR2 (128) C4 VARCHAR2 (128) C5 VARCHAR2 (128) N1 NUMBER N2 NUMBER N3 NUMBER N4 NUMBER N5 NUMBER N6 NUMBER N7 NUMBER N8 NUMBER N9 NUMBER N10 NUMBER			
FLAGS NUMBER C1 VARCHAR2 (128) C2 VARCHAR2 (128) C3 VARCHAR2 (128) C4 VARCHAR2 (128) C5 VARCHAR2 (128) N1 NUMBER N2 NUMBER N3 NUMBER N4 NUMBER N5 NUMBER N6 NUMBER N7 NUMBER N8 NUMBER N9 NUMBER N10 NUMBER			
C1 VARCHAR2 (128) C2 VARCHAR2 (128) C3 VARCHAR2 (128) C4 VARCHAR2 (128) C5 VARCHAR2 (128) C6 VARCHAR2 (128) N1 NUMBER N2 NUMBER N3 NUMBER N4 NUMBER N5 NUMBER N6 NUMBER N7 NUMBER N8 NUMBER N9 NUMBER N10 NUMBER			
C2 VARCHAR2 (128) C3 VARCHAR2 (128) C4 VARCHAR2 (128) C5 VARCHAR2 (128) C6 VARCHAR2 (128) N1 NUMBER N2 NUMBER N3 NUMBER N4 NUMBER N5 NUMBER N6 NUMBER N7 NUMBER N8 NUMBER N9 NUMBER N10 NUMBER			
C3 VARCHAR2 (128) C4 VARCHAR2 (128) C5 VARCHAR2 (128) C6 VARCHAR2 (128) N1 NUMBER N2 NUMBER N3 NUMBER N4 NUMBER N5 NUMBER N6 NUMBER N7 NUMBER N8 NUMBER N9 NUMBER N10 NUMBER			
C4 VARCHAR2 (128) C5 VARCHAR2 (128) C6 VARCHAR2 (128) N1 NUMBER N2 NUMBER N3 NUMBER N4 NUMBER N5 NUMBER N6 NUMBER N7 NUMBER N8 NUMBER N9 NUMBER N10 NUMBER	VARCHAR2 (128)		
C5 VARCHAR2 (128) C6 VARCHAR2 (128) N1 NUMBER N2 NUMBER N3 NUMBER N4 NUMBER N5 NUMBER N6 NUMBER N7 NUMBER N8 NUMBER N9 NUMBER N10 NUMBER	VARCHAR2 (128)		
C6 VARCHAR2 (128) N1 NUMBER N2 NUMBER N3 NUMBER N4 NUMBER N5 NUMBER N6 NUMBER N7 NUMBER N8 NUMBER N9 NUMBER N10 NUMBER	VARCHAR2 (128)		
N1 NUMBER N2 NUMBER N3 NUMBER N4 NUMBER N5 NUMBER N6 NUMBER N7 NUMBER N8 NUMBER N8 NUMBER N9 NUMBER N10 NUMBER			
N2 NUMBER N3 NUMBER N4 NUMBER N5 NUMBER N6 NUMBER N7 NUMBER N8 NUMBER N9 NUMBER N10 NUMBER	VARCHAR2 (128)		
N3 NUMBER N4 NUMBER N5 NUMBER N6 NUMBER N7 NUMBER N8 NUMBER N9 NUMBER N10 NUMBER			
N4 NUMBER N5 NUMBER N6 NUMBER N7 NUMBER N8 NUMBER N9 NUMBER N10 NUMBER			
N5 NUMBER N6 NUMBER N7 NUMBER N8 NUMBER N9 NUMBER N10 NUMBER			
N6 NUMBER N7 NUMBER N8 NUMBER N9 NUMBER N10 NUMBER			
N7 NUMBER N8 NUMBER N9 NUMBER N10 NUMBER			
N8 NUMBER N9 NUMBER N10 NUMBER			
N9 NUMBER N10 NUMBER	NUMBER		
N10 NUMBER	NUMBER		
N11 NUMBER			
N12 NUMBER			
N13 NUMBER			
D1 DATE			
T1 TIMESTAMP(6) WITH TIME ZONE			
R1 RAW (1000)			
R2 RAW (1000)			
R3 RAW (1000)	RAW (1000)		
CH1 VARCHAR2 (1000)	VARCHAR2 (1000)		
CL1 CLOB			

Examples

The following example creates a statistics table namedmystat .

```
EXECUTE OPG_APIS.CREATE_STAT_TABLE('mystat', null);
```

A.11.17 OPG_APIS.CREATE_SUB_GRAPH

```
OPG_APIS.CREATE_SUB_GRAPH(
graph_owner IN VARCHAR2,
orgGraph IN VARCHAR2,
newGraph IN VARCHAR2,
nSrc IN NUMBER,
depth IN NUMBER);
```



Description

Creates a subgraph, which is an expansion from a given vertex. The depth of expansion is customizable.

Parameters

graph owner

Owner of the property graph.

orgGraph

Name of the original property graph.

newGraph

Name of the subgraph to be created from the original graph.

nSrc

Vertex ID: the subgraph will be created by expansion from this vertex. For example, nSrc = 1 starts the expansion from the vertex with ID 1.

depth

Depth of expansion: the expansion, following outgoing edges, will include all vertices that are within depth hops away from vertex nSrc. For example, depth = 2 causes the to should include all vertices that are within 2 hops away from vertex nSrc (vertex ID 1 in the preceding example).

Usage Notes

The original property graph must exist in the database.

Examples

The following example creates a subgraph mypgsub from the property graph mypg whose owner is SCOTT. The subgraph includes vertex 1 and all vertices that are reachable from the vertex with ID 1 in 2 hops.

```
EXECUTE OPG_APIS.CREATE_SUB_GRAPH('SCOTT', 'mypg', 'mypgsub', 1, 2);
```

A.11.18 OPG_APIS.CREATE_VERTICES_TEXT_IDX

```
OPG_APIS.CREATE_VERTICES_TEXT_IDX(
    graph_owner IN VARCHAR2,
    graph_name IN VARCHAR2,
    pref_owner IN VARCHAR2 DEFAULT NULL,
    datastore IN VARCHAR2 DEFAULT NULL,
    filter IN VARCHAR2 DEFAULT NULL,
    storage IN VARCHAR2 DEFAULT NULL,
    wordlist IN VARCHAR2 DEFAULT NULL,
    stoplist IN VARCHAR2 DEFAULT NULL,
    lexer IN VARCHAR2 DEFAULT NULL,
    dop IN INTEGER DEFAULT NULL,
    options IN VARCHAR2 DEFAULT NULL,);
```



Description

Creates a text index on a property graph vertex table.

Parameters

graph_owner

Owner of the property graph.

graph_name

Name of the property graph.

pref_owner

Owner of the preference.

datastore

The way that documents are stored.

filter

The way that documents can be converted to plain text.

storage

The way that the index data is stored.

wordlist

The way that stem and fuzzy queries should be expanded

stoplist

The words or themes that are not to be indexed.

lexer

The language used for indexing.

dop

The degree of parallelism used for index creation.

options

Additional settings for index creation.

Usage Notes

The original property graph must exist in the database.

You must have the ALTER SESSION privilege to run this procedure.

Examples

The following example creates a text index on the vertex table of property graph mypg, which is owned by user SCOTT, using the lexer OPG_AUTO_LEXER and a degree of parallelism of 4.

```
EXECUTE OPG_APIS.CREATE_VERTICES_TEXT_IDX('SCOTT', 'mypg', null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, n
```



A.11.19 OPG_APIS.DROP_EDGES_TEXT_IDX

Format

```
OPG_APIS.DROP_EDGES_TEXT_IDX(
    graph_owner IN VARCHAR2,
    graph_name IN VARCHAR2,
    options IN VARCHAR2 DEFAULT NULL);
```

Description

Drops a text index on a property graph edge table.

Parameters

graph_owner

Owner of the property graph.

graph_name

Name of the property graph.

options

Additional settings for the operation.

Usage Notes

A text index must already exist on the property graph edge table.

Examples

The following example drops the text index on the edge table of property graph mypg that is owned by user SCOTT.

```
EXECUTE OPG_APIS.DROP_EDGES_TEXT_IDX('SCOTT', 'mypg', null);
```

A.11.20 OPG_APIS.DROP_PG

Format

```
OPG_APIS.DROP_PG(
          graph_name IN VARCHAR2);
```

Description

Drops (deletes) a property graph.

Parameters

graph_name

Name of the property graph.

Usage Notes

All the graph tables (VT\$, GE\$, and so on) will be dropped from the database.

Examples

The following example drops the property graph named mypg.

```
EXECUTE OPG APIS.DROP PG('mypg');
```

A.11.21 OPG_APIS.DROP_PG_VIEW

Format

```
OPG_APIS.DROP_PG_VIEW(
          graph_name           IN VARCHAR2);
          options                 IN VARCHAR2);
```

Description

Drops (deletes) the view definition of a property graph.

Parameters

graph name

Name of the property graph.

options

(Reserved for future use.)

Usage Notes

Oracle supports creating physical property graphs and property graph views. For example, given an RDF model, it supports creating property graph views over the RDF model, so that you can run property graph analytics on top of the RDF graph.

This procedure cannot be undone.

Examples

The following example drops the view definition of the property graph named mypq.

```
EXECUTE OPG_APIS.DROP_PG_VIEW('mypg');
```

A.11.22 OPG_APIS.DROP_VERTICES_TEXT_IDX

Format

```
OPG_APIS.DROP_VERTICES_TEXT_IDX(
    graph_owner IN VARCHAR2,
    graph_name IN VARCHAR2,
    options IN VARCHAR2 DEFAULT NULL);
```

Description

Drops a text index on a property graph vertex table.



Parameters

graph owner

Owner of the property graph.

graph name

Name of the property graph.

options

Additional settings for the operation.

Usage Notes

A text index must already exist on the property graph vertex table.

Examples

The following example drops the text index on the vertex table of property graph mypg that is owned by user SCOTT.

```
EXECUTE OPG_APIS.DROP_VERTICES_TEXT_IDX('SCOTT', 'mypg', null);
```

A.11.23 OPG_APIS.ESTIMATE_TRIANGLE_RENUM

Format

```
COUNT_TRIANGLE_ESTIMATE(
edge_tab_name IN VARCHAR2,
wt_undBM IN VARCHAR2,
wt_rnmap IN VARCHAR2,
wt_undAM IN VARCHAR2,
num_sub_ptns IN INTEGER DEFAULT 1,
chunk_id IN INTEGER DEFAULT 1,
dop IN INTEGER DEFAULT 1,
tbs IN VARCHAR2 DEFAULT NULL,
options IN VARCHAR2 DEFAULT NULL
) RETURN NUMBER;
```

Description

Estimates the number of triangles in a property graph.

Parameters

edge_tab_name

Name of the property graph edge table.

wt_undBM

A working table holding an undirected version of the original graph (before renumbering optimization).

wt rnmap

A working table that is a mapping table for renumbering optimization.



wt undAM

A working table holding the undirected version of the graph data after applying the renumbering optimization.

num_sub_ptns

Number of logical subpartitions used in calculating triangles . Must be a positive integer, power of 2 (1, 2, 4, 8, ...). For a graph with a relatively small maximum degree, use the value 1 (the default).

chunk_id

The logical subpartition to be used in triangle estimation (Only this partition will be counted). It must be an integer between 0 and num sub ptns*num sub ptns-1.

dop

Degree of parallelism for the operation. The default is 1 (no parallelism).

tbs

Name of the tablespace to hold the data stored in working tables.

options

Additional settings for operation. An optional string with one or more (commaseparated) of the following values:

PDML=T enables parallel DML.

Usage Notes

This function counts the total triangles in a portion of size 1/ (num_sub_ptns*num_sub_ptns) of the graph; so to estimate the total number of triangles in the graph, you can multiply the result by num sub_ptns*num sub_ptns.

The property graph edge table must exist in the database, and the OPG_APIS.COUNT_TRIANGLE_PREP procedure must already have been executed.

Examples

The following example estimates the number of triangle in the property graph named connections. It does not perform the cleanup after it finishes, so you can count triangles again on the same graph without calling the preparation procedure.

```
set serveroutput on
DECLARE
  wt1 varchar2(100); -- intermediate working table
  wt2 varchar2(100);
 wt3 varchar2(100);
  n number;
BEGIN
  opg apis.count triangle prep('connectionsGE$', wt1, wt2, wt3);
  n := opg apis.estimate triangle renum(
     'connectionsGE$',
      wt1,
      wt2,
      wt3,
      num_sub_ptns=>64,
      chunk id=>2048,
      dop = >2,
      tbs => 'MYPG TS',
      options=>'PDML=T'
```



```
);
dbms_output.put_line('estimated number of triangles ' || (n * 64 * 64));
END;
/
```

A.11.24 OPG_APIS.EXP_EDGE_TAB_STATS

Format

Description

Retrieves statistics for the edge table of a given property graph and stores them in the user-created statistics table.

Parameters

graph_name

Name of the property graph.

stattab

Name of the statistics table.

statid

Optional identifier to associate with these statistics within stattab.

cascade

If TRUE, column and index statistics are exported.

statown

Schema containing stattab.

stat category

Specifies what statistics to export, using a comma to separate values. The supported values are 'OBJECT_STATS' (the default: table statistics, column statistics, and index statistics) and 'SYNOPSES' (auxiliary statistics created when statistics are incrementally maintained).

Usage Notes

(None.)

Examples

The following example creates a statistics table, exports into this table the property graph edge table statistics, and issues a query to count the relevant rows for the newly created statistics.

```
EXECUTE OPG_APIS.CREATE_STAT_TABLE('mystat', null);

EXECUTE OPG_APIS.EXP_EDGE_TAB_STATS('mypg', 'mystat', 'edge_stats_id_1', true, null, 'OBJECT_STATS');
```

```
SELECT count(1) FROM mystat WHERE statid='EDGE_STATS_ID_1';
153
```

A.11.25 OPG_APIS.EXP_VERTEX_TAB_STATS

Format

Description

Retrieves statistics for the vertex table of a given property graph and stores them in the user-created statistics table.

Parameters

graph_name

Name of the property graph.

stattab

Name of the statistics table.

statid

Optional identifier to associate with these statistics within stattab.

cascade

If TRUE, column and index statistics are exported.

statown

Schema containing stattab.

stat category

Specifies what statistics to export, using a comma to separate values. The supported values are 'OBJECT_STATS' (the default: table statistics, column statistics, and index statistics) and 'SYNOPSES' (auxiliary statistics created when statistics are incrementally maintained).

Usage Notes

(None.)

Examples

The following example creates a statistics table, exports into this table the property graph vertex table statistics, and issues a query to count the relevant rows for the newly created statistics.

```
EXECUTE OPG_APIS.CREATE_STAT_TABLE('mystat',null);
EXECUTE OPG_APIS.EXP_VERTEX_TAB_STATS('mypg', 'mystat', 'vertex_stats_id_1',
```

```
true, null, 'OBJECT_STATS');
SELECT count(1) FROM mystat WHERE statid='VERTEX_STATS_ID_1';
108
```

A.11.26 OPG_APIS.FIND_CC_MAPPING_BASED

Format

```
OPG_APIS.FIND_CC_MAPPING_BASED(
   edge_tab_name IN VARCHAR2,
   wt_clusters IN OUT VARCHAR2,
   wt_undir IN OUT VARCHAR2,
   wt_cluas IN OUT VARCHAR2,
   wt_newas IN OUT VARCHAR2,
   wt_delta IN OUT VARCHAR2,
   dop IN INTEGER DEFAULT 4,
   rounds IN INTEGER DEFAULT 0,
   tbs IN VARCHAR2 DEFAULT NULL,
   options IN VARCHAR2 DEFAULT NULL);
```

Description

Finds connected components in a property graph. All connected components will be stored in the wt_clusters table. The original graph is treated as undirected.

Parameters

edge tab name

Name of the property graph edge table.

wt clusters

A working table holding the final vertex cluster mappings. This table has two columns (VID NUMBER, CLUSTER_ID NUMBER). Column VID stores the vertex ID values, and column CLUSTER_ID stores the corresponding cluster ID values. Cluster ID values are long integers that can have gaps between them.

If an empty name is specified, a new table will be generated, and its name will be returned.

wt_undir

A working table holding an undirected version of the graph.

wt cluas

A working table holding current cluster assignments.

wt_newas

A working table holding updated cluster assignments.

wt_delta

A working table holding changes ("delta") in cluster assignments.

don

Degree of parallelism for the operation. The default is 4.



rounds

Maximum umber of iterations to perform in searching for connected components. The default value of 0 (zero) means that computation will continue until all connected components are found.

tbs

Name of the tablespace to hold the data stored in working tables.

ontions

Additional settings for the operation.

'PDML=T' enables parallel DML.

Usage Notes

The property graph edge table must exist in the database, and the OPG_APIS.FIND_CLUSTERS_PREP. procedure must already have been executed.

Examples

The following example finds the connected components in a property graph named mypg.

```
DECLARE
  wtClusters varchar2(200) := 'mypg_clusters';
 wtUnDir varchar2(200);
wtCluas varchar2(200);
wtNewas varchar2(200);
wtDelta varchar2(200);
BEGIN
  opg apis.find clusters prep('mypgGE$', wtClusters, wtUnDir,
      wtCluas, wtNewas, wtDelta, '');
  dbms output.put line('working tables names ' || wtClusters || ' '
|| wtUnDir || ' ' || wtCluas || ' ' || wtNewas
                                                       11 ' '
|| wtDelta );
opg apis.find cc mapping based(''mypgGE$', wtClusters, wtUnDir,
      wtCluas, wtNewas, wtDelta, 8, 0, 'MYTBS', 'PDML=T');
-- logic to consume results in wtClusters
-- select /*+ parallel(8) */ count(distinct cluster id)
-- from mypg clusters;
-- cleanup all the working tables
  opg apis.find clusters cleanup('mypgGE$', wtClusters, wtUnDir,
      wtCluas, wtNewas, wtDelta, '');
END;
```

A.11.27 OPG_APIS.FIND_CLUSTERS_CLEANUP

Format

```
OPG_APIS.FIND_CLUSTERS_CLEANUP(
edge_tab_name IN VARCHAR2,
wt_clusters IN OUT VARCHAR2,
wt undir IN OUT VARCHAR2,
```



Description

Cleans up after running weakly connected components (WCC) cluster detection.

Parameters

edge_tab_name

Name of the property graph edge table.

wt clusters

A working table holding the final vertex cluster mappings. This table has two columns (VID NUMBER, CLUSTER_ID NUMBER). Column VID stores the vertex ID values, and column CLUSTER_ID stores the corresponding cluster ID values. Cluster ID values are long integers that can have gaps between them.

If an empty name is specified, a new table will be generated, and its name will be returned.

wt undir

A working table holding an undirected version of the graph.

wt_cluas

A working table holding current cluster assignments.

wt newas

A working table holding updated cluster assignments.

wt delta

A working table holding changes ("delta") in cluster assignments.

options

(Reserved for future use.)

Usage Notes

The property graph edge table must exist in the database.

Examples

The following example cleans up after performing doing cluster detection in a property graph named mypq.

```
EXECUTE OPG_APIS.FIND_CLUSTERS_CLEANUP('mypgGE$', wtClusters, wtUnDir, wtCluas, wtNewas, wtDelta, null);
```

A.11.28 OPG_APIS.FIND_CLUSTERS_PREP

Format

```
OPG_APIS.FIND_CLUSTERS_PREP(
edge_tab_name IN VARCHAR2,
wt_clusters IN OUT VARCHAR2,
wt_undir IN OUT VARCHAR2,
wt_cluas IN OUT VARCHAR2,
wt_newas IN OUT VARCHAR2,
```



Description

Prepares for running weakly connected components (WCC) cluster detection.

Parameters

edge_tab_name

Name of the property graph edge table.

wt clusters

A working table holding the final vertex cluster mappings. This table has two columns (VID NUMBER, CLUSTER_ID NUMBER). Column VID stores the vertex ID values, and column CLUSTER_ID stores the corresponding cluster ID values. Cluster ID values are long integers that can have gaps between them.

If an empty name is specified, a new table will be generated, and its name will be returned.

wt undir

A working table holding an undirected version of the graph.

wt cluas

A working table holding current cluster assignments.

wt newas

A working table holding updated cluster assignments.

wt delta

A working table holding changes ("delta") in cluster assignments.

options

Additional settings for index creation.

Usage Notes

The property graph edge table must exist in the database.

Examples

The following example prepares for doing cluster detection in a property graph named mypg.



A.11.29 OPG_APIS.FIND_SP

Format

```
OPG_APIS.FIND_SP(

edge_tab_name IN VARCHAR2,
source IN NUMBER,
dest IN NUMBER,
exp_tab IN OUT VARCHAR2,
dop IN INTEGER,
stats_freq IN INTEGER DEFAULT 20000,
path_output OUT VARCHAR2,
weights_output OUT VARCHAR2,
edge_tab_name IN VARCHAR2,
options IN VARCHAR2 DEFAULT NULL,
scn IN NUMBER DEFAULT NULL);
```

Description

Finds the shortest path between given source vertex and destination vertex in the property graph. It assumes each edge has a numeric weight property. (The actual edge property name is not significant.)

Parameters

edge_tab_name

Name of the property graph edge table.

source

Source (start) vertex ID.

dest

Destination (end) vertex ID.

exp_tab

Name of the expansion table to be used for shortest path calculations.

dop

Degree of parallelism for the operation.

stats_freq

Frequency for collecting statistics on the table.

path_output

The output shortest path. It consists of IDs of vertices on the shortest path, which are separated by the space character.

weights_output

The output shortest path weights. It consists of weights of edges on the shortest path, which are separated by the space character.

options

Additional settings for the operation. An optional string with one or more (comma-separated) of the following values:



- CREATE UNDIRECTED=T
- REUSE UNDIRECTED TAB=T

scn

SCN for the edge table. It can be null.

Usage Notes

The property graph edge table must exist in the database, and the OPG APIS.FIND SP PREP procedure must have already been called.

Examples

The following example prepares for shortest-path calculation, and then finds the shortest path from vertex 1 to vertex 35 in a property graph named mypg.

```
set serveroutput on
DECLARE
    w    varchar2(2000);
    wtExp varchar2(2000);
    vPath varchar2(2000);

BEGIN
    opg_apis.find_sp_prep('mypgGE$', wtExp, null);
    opg_apis.find_sp('mypgGE$', 1, 35, wtExp, 1, 200000, vPath, w, null, null);
    dbms_output.put_line('Shortest path ' || vPath);
    dbms_output.put_line('Path weights ' || w);
END;
//
```

The output will be similar to the following. It shows one shortest path starting from vertex 1, to vertex 2, and finally to the destination vertex (35).

```
Shortest path 1 2 35
Path weights 3 2 1 1
```

A.11.30 OPG APIS.FIND SP CLEANUP

Format

Description

Cleans up after running one or more shortest path calculations.

Parameters

edge_tab_name

Name of the property graph edge table.

exp_tab

Name of the expansion table used for shortest path calculations.



options

(Reserved for future use.)

Usage Notes

There is no need to call this procedure after each OPG_APIS.FIND_SP call. You can run multiple shortest path calculations before calling OPG_APIS.FIND_SP_CLEANUP.

Examples

The following example does cleanup work after doing shortest path calculations in a property graph named mypg.

```
EXECUTE OPG_APIS.FIND_SP_CLEANUP('mypgGE$', wtExpTab, null);
```

A.11.31 OPG APIS.FIND SP PREP

Format

Description

Prepares for shortest path calculations.

Parameters

edge tab name

Name of the property graph edge table.

exp tab

Name of the expansion table to be used for shortest path calculations. If it is empty, an intermediate working table will be created and the table name will be returned in exp tab.

options

Additional settings for the operation. An optional string with one or more (comma-separated) of the following values:

- CREATE UNDIRECTED=T
- REUSE UNDIRECTED TAB=T

Usage Notes

The property graph edge table must exist in the database.

Examples

The following example does preparation work before doing shortest path calculations in a property graph named mypg

```
set serveroutput on
DECLARE
    wtExp varchar2(2000); -- name of working table for shortest path calculation
BEGIN
    opg_apis.find_sp_prep('mypgGE$', wtExp, null);
```



```
dbms_output.put_line('Working table name ' || wtExp);
END;
//
```

The output will be similar to the following. (Your output may be different depending on the SQL session ID.)

Working table name "MYPGGE\$\$TWFS277"

A.11.32 OPG_APIS.GET_BUILD_ID

Format

```
OPG APIS.GET BUILD ID() RETURN VARCHAR2;
```

Description

Returns the current build ID of the Oracle Spatial and Graph property graph support, in YYYYMMDD format.

Parameters

(None.)

Usage Notes

(None.)

Examples

The following example returns the current build ID of the Oracle Spatial and Graph property graph support.

A.11.33 OPG_APIS.GET_GEOMETRY_FROM_V_COL

Format

Description

Returns an SDO_GEOMETRY object constructed using spatial data and optionally an SRID value.

Parameters

V

A String containing spatial data in serialized form.

srid

SRID (coordinate system identifier) to be used in the resulting SDO_GEOMETRY object. The default value is 8307, the Oracle Spatial SRID for the WGS 84 longitude/latitude coordinate system.

Usage Notes

If there is incorrect syntax or a parsing error, this function returns NULL instead of generating an exception.

Examples

The following examples show point, line, and polygon geometries.

```
SQL> select opg apis.get geometry from v col('10.0 5.0',8307) from dual;
OPG APIS.GET GEOMETRY FROM V COL('10.05.0',8307) (SDO GTYPE, SDO SRID,
SDO POINT (
           ______
SDO GEOMETRY (2001, 8307, SDO POINT TYPE (10, 5, NULL), NULL, NULL)
SQL> select opg apis.get geometry from v col('LINESTRING(30 10, 10 30, 40
40)',8307) from dual;
OPG APIS.GET GEOMETRY FROM V COL('LINESTRING(3010,1030,4040)',8307)
(SDO GTYPE, S
SDO GEOMETRY (2002, 8307, NULL, SDO ELEM INFO ARRAY (1, 2, 1),
SDO ORDINATE ARRAY(
30, 10, 10, 30, 40, 40))
SQL> select opg apis.get geometry from v col('POLYGON((-83.6 34.1, -83.6
34.3, -83.4 34.3, -83.4 34.1, -83.6 34.1))', 8307) from dual;
OPG APIS.GET GEOMETRY FROM V COL('POLYGON((-83.634.1,-83.634.3,-83.434.3,-83.
434
SDO GEOMETRY (2003, 8307, NULL, SDO ELEM INFO ARRAY (1, 1003, 1),
SDO ORDINATE ARR
AY(-83.6, 34.1, -83.6, 34.3, -83.4, 34.3, -83.4, 34.1, -83.6, 34.1))
```

A.11.34 OPG_APIS.GET_GEOMETRY_FROM_V_T_COLS

Format



Description

Returns an SDO_GEOMETRY object constructed using spatial data, a type value, and optionally an SRID value.

Parameters

ν

A String containing spatial data in serialized form,

t

Value indicating the type of value represented by the \forall parameter. Must be 20. (A null value or any other value besides 20 returns a null SDO_GEOMETRY object.)

srid

SRID (coordinate system identifier) to be used in the resulting SDO_GEOMETRY object. The default value is 8307, the Oracle Spatial SRID for the WGS 84 longitude/ latitude coordinate system.

Usage Notes

If there is incorrect syntax or a parsing error, this function returns NULL instead of generating an exception.

Examples

The following examples show point, line, and polygon geometries.

```
SQL> select opg apis.get geometry from v t cols('10.0 5.0', 20, 8307)
from dual;
OPG APIS.GET GEOMETRY FROM V T COLS('10.05.0', 20, 8307) (SDO GTYPE,
SDO SRID, SDO
SDO GEOMETRY (2001, 8307, SDO POINT TYPE (10, 5, NULL), NULL, NULL)
SQL> select opg apis.get geometry from v t cols('LINESTRING(30 10, 10
30, 40 40)', 20, 8307) from dual;
OPG APIS.GET GEOMETRY FROM V T COLS('LINESTRING(3010,1030,4040)',20,830
7) (SDO GT
SDO GEOMETRY (2002, 8307, NULL, SDO ELEM INFO ARRAY (1, 2, 1),
SDO ORDINATE ARRAY(
30, 10, 10, 30, 40, 40))
SQL> select opg apis.get geometry from v t cols('POLYGON((-83.6
34.1, -83.6 34.3, -83.4 34.3, -83.4 34.1, -83.6 34.1))', 20, 8307)
from dual;
OPG APIS.GET GEOMETRY FROM V T COLS('POLYGON((-83.634.1,-83.634.3,-83.4
```



```
34.3,-83.
---
SDO_GEOMETRY(2003, 8307, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1),
SDO_ORDINATE_ARR
AY(-83.6, 34.1, -83.6, 34.3, -83.4, 34.3, -83.4, 34.1, -83.6, 34.1))
```

A.11.35 OPG_APIS.GET_LATLONG_FROM_V_COL

Format

Description

Returns an SDO_GEOMETRY object constructed using spatial data and optionally an SRID value.

Parameters

ν

A String containing spatial data in serialized form.

srid

SRID (coordinate system identifier) to be used in the resulting SDO_GEOMETRY object. The default value is 8307, the Oracle Spatial SRID for the WGS 84 longitude/latitude coordinate system.

Usage Notes

This function assumes that for each vertex in the geometry in the v parameter, the *first* number is the *latitude* value and the second number is the longitude value. (This is the reverse of the order in an SDO_GEOMETRY object definition, where longitude is first and latitude is second).

If there is incorrect syntax or a parsing error, this function returns NULL instead of generating an exception.

Examples

The following example returns a point SDO_GEOMETRY object. Notice that the coordinate values of the input point are "swapped" in the returned SDO_GEOMETRY object.

```
SQL> select opg_apis.get_latlong_from_v_col('5.1 10.0', 8307) from dual;

OPG_APIS.GET_LATLONG_FROM_V_COL('5.110.0',8307) (SDO_GTYPE, SDO_SRID, SDO_POINT(X
---
SDO_GEOMETRY(2001, 8307, SDO_POINT_TYPE(10, 5.1, NULL), NULL, NULL)
```



A.11.36 OPG_APIS.GET_LATLONG_FROM_V_T_COLS

Format

Description

Returns an SDO_GEOMETRY object constructed using spatial data, a type value, and optionally an SRID value.

Parameters

v

A String containing spatial data in serialized form.

t

Value indicating the type of value represented by the \forall parameter. Must be 20. (A null value or any other value besides 20 returns a null SDO_GEOMETRY object.)

srid

SRID (coordinate system identifier) to be used in the resulting SDO_GEOMETRY object. The default value is 8307, the Oracle Spatial SRID for the WGS 84 longitude/ latitude coordinate system.

Usage Notes

This function assumes that for each vertex in the geometry in the v parameter, the *first* number is the *latitude* value and the second number is the longitude value. (This is the reverse of the order in an SDO_GEOMETRY object definition, where longitude is first and latitude is second).

If there is incorrect syntax or a parsing error, this function returns NULL instead of generating an exception.

Examples

The following example returns a point SDO_GEOMETRY object. Notice that the coordinate values of the input point are "swapped" in the returned SDO_GEOMETRY object.



A.11.37 OPG_APIS.GET_LONG_LAT_GEOMETRY

Format

Description

Returns an SDO_GEOMETRY object constructed using X and Y point coordinate values, and optionally an SRID value.

Parameters

X

The X (first coordinate) value in the SDO_POINT_TYPE element of the geometry definition.

У

The Y (second coordinate) value in the SDO_POINT_TYPE element of the geometry definition.

srid

SRID (coordinate system identifier) to be used in the resulting SDO_GEOMETRY object. The default value is 8307, the Oracle Spatial SRID for the WGS 84 longitude/latitude coordinate system.

Usage Notes

If there is incorrect syntax or a parsing error, this function returns NULL instead of generating an exception.

Examples

The following example returns the geometry object for a point with X, Y coordinates 10.5, 5.0, and it uses 8307 as the SRID in the resulting geometry object.

```
SQL> select opg_apis.get_long_lat_geometry(10.0, 5.0, 8307) from dual;

OPG_APIS.GET_LONG_LAT_GEOMETRY(10.0, 5.0, 8307) (SDO_GTYPE, SDO_SRID, SDO_POINT(X,
----
SDO_GEOMETRY(2001, 8307, SDO_POINT_TYPE(10, 5, NULL), NULL, NULL)
```

A.11.38 OPG_APIS.GET_LATLONG_FROM_V_COL

Format



Description

Returns an SDO_GEOMETRY object constructed using spatial data and optionally an SRID value.

Parameters

ν

A String containing spatial data in serialized form.

srid

SRID (coordinate system identifier) to be used in the resulting SDO_GEOMETRY object. The default value is 8307, the Oracle Spatial SRID for the WGS 84 longitude/ latitude coordinate system.

Usage Notes

This function assumes that for each vertex in the geometry in the v parameter, the *first* number is the *latitude* value and the second number is the longitude value. (This is the reverse of the order in an SDO_GEOMETRY object definition, where longitude is first and latitude is second).

If there is incorrect syntax or a parsing error, this function returns NULL instead of generating an exception.

Examples

The following example returns a point SDO_GEOMETRY object. Notice that the coordinate values of the input point are "swapped" in the returned SDO_GEOMETRY object.

A.11.39 OPG_APIS.GET_LONGLAT_FROM_V_T_COLS

Format

Description

Returns an SDO_GEOMETRY object constructed using spatial data, a type value, and optionally an SRID value.



Parameters

V

A String containing spatial data in serialized form.

t

Value indicating the type of value represented by the v parameter. Must be 20. (A null value or any other value besides 20 returns a null SDO GEOMETRY object.)

srid

SRID (coordinate system identifier) to be used in the resulting SDO_GEOMETRY object. The default value is 8307, the Oracle Spatial SRID for the WGS 84 longitude/latitude coordinate system.

Usage Notes

If there is incorrect syntax or a parsing error, this function returns NULL instead of generating an exception.

Examples

This function assumes that for each vertex in the geometry in the \forall parameter, the first number is the longitude value and the second number is the latitude value (which is the order in an SDO_GEOMETRY object definition).

The following example returns a point SDO GEOMETRY object.

```
SQL> select opg_apis.get_longlat_from_v_t_cols('5.1 10.0',20,8307) from dual;

OPG_APIS.GET_LATLONG_FROM_V_T_COLS('5.110.0',20,8307) (SDO_GTYPE, SDO_SRID,
SDO_P
---
SDO GEOMETRY(2001, 8307, SDO POINT TYPE(5.1, 10, NULL), NULL, NULL)
```

A.11.40 OPG APIS.GET OPG VERSION

Format

```
OPG APIS.GET OPG VERSION() RETURN VARCHAR2;
```

Description

Returns the Graph Server and Client version from which the PL/SQL packages were installed.

Parameters

(None.)

Usage Notes

(None.)



Examples

The following example returns the Graph Server and Client version from which the PL/SQL packages were installed.

A.11.41 OPG_APIS.GET_SCN

Format

```
OPG APIS.GET SCN() RETURN NUMBER;
```

Description

Returns the SCN (system change number) of the Oracle Spatial and Graph property graph support, in YYYYMMDD format.



Effective with Release 20.3, the OPG_APIS.GET_SCN function is **deprecated**. Instead, to retrieve the current SCN (system change number), use the DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER function:

```
SELECT dbms flashback.get system change number FROM DUAL;
```

Parameters

(None.)

Usage Notes

The SCN value is incremented after each commit.

Examples

The following example returns the current build ID of the Oracle Spatial and Graph property graph support.



A.11.42 OPG_APIS.GET_VERSION



The <code>OPG_APIS.GET_VERSION()</code> function is deprecated and will be desupported in a future release. Instead, use <code>OPG_APIS.GET_OPG_VERSION</code>.

Format

```
OPG APIS.GET VERSION() RETURN VARCHAR2;
```

Description

Returns the current version of the Oracle Spatial and Graph property graph support.

Parameters

(None.)

Usage Notes

(None.)

Examples

The following example returns the current version of the Oracle Spatial and Graph property graph support.

A.11.43 OPG_APIS.GET_WKTGEOMETRY_FROM_V_COL

Format

Description

Returns an SDO_GEOMETRY object based on a geometry in WKT (well known text) form and optionally an SRID.

Parameters

v

A String containing spatial data in serialized form.

srid

SRID (coordinate system identifier) to be used in the resulting SDO_GEOMETRY object. The default value is 8307, the Oracle Spatial SRID for the WGS 84 longitude/ latitude coordinate system.

Usage Notes

If there is incorrect syntax or a parsing error, this function returns NULL instead of generating an exception.

Examples

The following statements return a point geometry and a line string geometry

```
SQL> select opg_apis.get_wktgeometry_from_v_col('POINT(10.0 5.1)', 8307) from dual;

OPG_APIS.GET_WKTGEOMETRY_FROM_V_COL('POINT(10.05.1)', 8307) (SDO_GTYPE, SDO_SRID,

SDO_GEOMETRY(2001, 8307, SDO_POINT_TYPE(10, 5.1, NULL), NULL, NULL)

SQL> select opg_apis.get_wktgeometry_from_v_col('LINESTRING(30 10, 10 30, 40 40)', 8307) from dual;

OPG_APIS.GET_WKTGEOMETRY_FROM_V_COL('LINESTRING(3010,1030,4040)', 8307) (SDO_GTYPE

SDO_GEOMETRY(2002, 8307, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1), SDO_ORDINATE_ARRAY(30, 10, 10, 30, 40, 40))
```

A.11.44 OPG_APIS.GET_WKTGEOMETRY_FROM_V_T_COLS

Format

Description

Returns an SDO_GEOMETRY object based on a geometry in WKT (well known text) form, a type value, and optionally an SRID.

Parameters

ν

A String containing spatial data in serialized form.

t

Value indicating the type of value represented by the v parameter. Must be 20. (A null value or any other value besides 20 returns a null SDO GEOMETRY object.)

srid

SRID (coordinate system identifier) to be used in the resulting SDO_GEOMETRY object. The default value is 8307, the Oracle Spatial SRID for the WGS 84 longitude/latitude coordinate system.

Usage Notes

If there is incorrect syntax or a parsing error, this function returns NULL instead of generating an exception.

Examples

The following statements return a point geometry and a polygon geometry

A.11.45 OPG_APIS.GRANT_ACCESS

Format

```
OPG_APIS.GRANT_ACCESS(
graph_owner IN VARCHAR2,
graph_name IN VARCHAR2,
other_user IN VARCHAR2,
privilege IN VARCHAR2);
```

Description

Grants access privileges on a property graph to another database user.



Parameters

graph_owner

Owner of the property graph.

graph_name

Name of the property graph.

other_user

Name of the database user to which on e or more access privileges will be granted.

privilege

A string of characters indicating privileges: R for read, S for select, U for update, D for delete, D for insert, D for all. Do not use commas or any other delimiter. If you specify D, do not specify any other values because D includes all access privileges.

Usage Notes

(None.)

Examples

The following example grants read and select (RS) privileges on the mypg property graph owned by database user SCOTT to database user PGUSR. It then connects as PGUSR and queries the mypg vertex table in the SCOTT schema.

```
CONNECT scott/<password>

EXECUTE OPG_APIS.GRANT_ACCESS('scott', 'mypg', 'pgusr', 'RS');

CONNECT pgusr/<password>

SELECT count(1) from scott.mypgVT$;
```

A.11.46 OPG_APIS.IMP_EDGE_TAB_STATS

Format

Description

Retrieves statistics for the given property graph edge table (GE\$) from the user statistics table identified by stattab and stores them in the dictionary. If cascade is TRUE, all index statistics associated with the specified table are also imported.



Parameters

graph_name

Name of the property graph.

stattab

Name of the statistics table.

statid

Optional identifier to associate with these statistics within stattab.

cascade

If TRUE, column and index statistics are exported.

statown

Schema containing stattab.

no invalidate

If TRUE, does not invalidate the dependent cursors. If FALSE, invalidates the dependent cursors immediately. If DBMS_STATS.AUTO_INVALIDATE (the usual default) is in effect, Oracle Database decides when to invalidate dependent cursors.

force

If TRUE, performs the operation even if the statistics are locked.

stat category

Specifies what statistics to export, using a comma to separate values. The supported values are 'OBJECT_STATS' (the default: table statistics, column statistics, and index statistics) and 'SYNOPSES' (auxiliary statistics created when statistics are incrementally maintained).

Usage Notes

(None.)

Examples

The following example creates a statistics table, exports into this table the edge table statistics, issues a query to count the relevant rows for the newly created statistics, and finally imports the statistics back.

```
EXECUTE OPG_APIS.CREATE_STAT_TABLE('mystat', null);

EXECUTE OPG_APIS.EXP_EDGE_TAB_STATS('mypg', 'mystat', 'edge_stats_id_1', true, null, 'OBJECT_STATS');

SELECT count(1) FROM mystat WHERE statid='EDGE_STATS_ID_1';

153

EXECUTE OPG_APIS.IMP_EDGE_TAB_STATS('mypg', 'mystat', 'edge_stats_id_1', true, null, false, true, 'OBJECT_STATS');
```



A.11.47 OPG_APIS.IMP_VERTEX_TAB_STATS

Format

Description

Retrieves statistics for the given property graph vertex table (VT\$) from the user statistics table identified by stattab and stores them in the dictionary. If cascade is TRUE, all index statistics associated with the specified table are also imported.

Parameters

graph name

Name of the property graph.

stattab

Name of the statistics table.

statid

Optional identifier to associate with these statistics within stattab.

cascade

If TRUE, column and index statistics are exported.

statown

Schema containing stattab.

no invalidate

If TRUE, does not invalidate the dependent cursors. If FALSE, invalidates the dependent cursors immediately. If DBMS_STATS.AUTO_INVALIDATE (the usual default) is in effect, Oracle Database decides when to invalidate dependent cursors.

force

If TRUE, performs the operation even if the statistics are locked.

stat_category

Specifies what statistics to export, using a comma to separate values. The supported values are 'OBJECT_STATS' (the default: table statistics, column statistics, and index statistics) and 'SYNOPSES' (auxiliary statistics created when statistics are incrementally maintained).

Usage Notes

(None.)



Examples

The following example creates a statistics table, exports into this table the vertex table statistics, issues a query to count the relevant rows for the newly created statistics, and finally imports the statistics back.

```
EXECUTE OPG_APIS.CREATE_STAT_TABLE('mystat', null);

EXECUTE OPG_APIS.EXP_VERTEX_TAB_STATS('mypg', 'mystat', 'vertex_stats_id_1', true, null, 'OBJECT_STATS');

SELECT count(1) FROM mystat WHERE statid='VERTEX_STATS_ID_1';

108

EXECUTE OPG_APIS.IMP_VERTEX_TAB_STATS('mypg', 'mystat', 'vertex_stats_id_1', true, null, false, true, 'OBJECT STATS');
```

A.11.48 OPG_APIS.PR

Format

```
OPG_APIS.PR(

edge_tab_name IN VARCHAR2,

d IN NUMBER DEFAULT 0.85,

num_iterations IN NUMBER DEFAULT 10,

convergence IN NUMBER DEFAULT 0.1,

dop IN INTEGER DEFAULT 4,

wt_node_pr IN OUT VARCHAR2,

wt_node_nextpr IN OUT VARCHAR2,

wt_edge_tab_deg IN OUT VARCHAR2,

wt_delta IN OUT VARCHAR2,

tablespace IN VARCHAR2 DEFAULT NULL,

options IN VARCHAR2 DEFAULT NULL,

num vertices OUT NUMBER);
```

Description

Prepares for page rank calculations.

Parameters

edge tab name

Name of the property graph edge table.

d

Damping factor.

num_iterations

Number of iterations for calculating the page rank values.

convergence

A threshold. If the difference between the page rank value of the current iteration and next iteration is lower than this threshold, then computation stops.

dop

Degree od parallelism for the operation.



wt_node_pr

Name of the working table to hold the page rank values of the vertices.

wt_node_pr

Name of the working table to hold the page rank values of the vertices.

wt_node_next_pr

Name of the working table to hold the page rank values of the vertices in the next iteration.

wt_edge_tab_deg

Name of the working table to hold edges and node degree information.

wt delta

Name of the working table to hold information about some special vertices.

tablespace

Name of the tablespace to hold all the graph data and index data.

options

Additional settings for the operation. An optional string with one or more (commaseparated) of the following values:

- CREATE UNDIRECTED=T
- REUSE UNDIRECTED TAB=T

num vertices

Number of vertices processed by the page rank calculation.

Usage Notes

The property graph edge table must exist in the database, and the OPG APIS.PR PREP procedure must have been called.

Examples

The following example performs preparation, and then calculates the page rank value of vertices in a property graph named mypg.

```
set serveroutput on
DECLARE
   wt pr varchar2(2000); -- name of the table to hold PR value of the current
iteration
   wt npr varchar2(2000); -- name of the table to hold PR value for the next
iteration
   wt3 varchar2(2000);
   wt4 varchar2(2000);
        varchar2(2000);
   wt.5
   n vertices number;
BEGIN
   wt pr := 'mypgPR';
    opg_apis.pr_prep('mypgGE$', wt_pr, wt_npr, wt3, wt4, null);
    dbms output.put line('Working table names ' || wt pr
      || ', wt_npr ' || wt_npr || ', wt3 ' || wt3 || ', wt4 '|| wt4);
   opg_apis.pr('mypgGE$', 0.85, 10, 0.01, 4, wt_pr, wt_npr, wt3, wt4, 'SYSAUX',
null, n vertices)
END;
```



The output will be similar to the following.

```
Working table names "MYPGPR", wt_npr "MYPGGE$$TWPRX277", wt3 "MYPGGE$$TWPRE277", wt4 "MYPGGE$$TWPRD277"
```

The calculated page rank value is stored in the mypgpr table which has the following definition and data.

```
SQL> desc mypgpr;
                       Null? Type
                       NOT NULL NUMBER
PR
                                NUMBER
С
                                 NUMBER
SQL> select node, pr from mypgpr;
    NODE
            PR
______
     101
          .1925
          .2775
     201
     102
           .1925
     104 .74383125
     105 .313625
     103
          .1925
          .15
     100
```

A.11.49 OPG_APIS.PR_CLEANUP

200 .15

Format

```
OPG_APIS.PR_CLEANUP(

edge_tab_name IN VARCHAR2,

wt_node_pr IN OUT VARCHAR2,

wt_node_nextpr IN OUT VARCHAR2,

wt_edge_tab_deg IN OUT VARCHAR2,

wt_delta IN OUT VARCHAR2,

options IN VARCHAR2 DEFAULT NULL);
```

Description

Performs cleanup after performing page rank calculations.

Parameters

edge tab name

Name of the property graph edge table.

wt_node_pr

Name of the working table to hold the page rank values of the vertices.

wt_node_next_pr

Name of the working table to hold the page rank values of the vertices in the next iteration.

wt_edge_tab_deg

Name of the working table to hold edges and node degree information.



wt delta

Name of the working table to hold information about some special vertices.

options

Additional settings for the operation. An optional string with one or more (commaseparated) of the following values:

- CREATE UNDIRECTED=T
- REUSE UNDIRECTED TAB=T

Usage Notes

You do not need to do cleanup after each call to the OPG_APIS.PR procedure. You can run several page rank calculations before calling the OPG_APIS.PR_CLEANUP procedure.

Examples

The following example does the cleanup work after running page rank calculations in a property graph named mypg.

```
EXECUTE OPG APIS.PR CLEANUP('mypgGE$', wt pr, wt npr, wt3, wt4, null);
```

A.11.50 OPG_APIS.PR_PREP

Format

Description

Prepares for page rank calculations.

Parameters

edge_tab_name

Name of the property graph edge table.

wt_node_pr

Name of the working table to hold the page rank values of the vertices.

wt_node_next_pr

Name of the working table to hold the page rank values of the vertices in the next iteration.

wt_edge_tab_deg

Name of the working table to hold edges and node degree information.

wt delta

Name of the working table to hold information about some special vertices.



options

Additional settings for the operation. An optional string with one or more (comma-separated) of the following values:

- CREATE_UNDIRECTED=T
- REUSE UNDIRECTED TAB=T

Usage Notes

The property graph edge table must exist in the database.

Examples

The following example does the preparation work before running page rank calculations in a property graph named mypg.

The output will be similar to the following.

```
Working table names "MYPGPR", wt_npr "MYPGGE$$TWPRX277", wt3 "MYPGGE$$TWPRE277", wt4 "MYPGGE$$TWPRD277"
```

A.11.51 OPG_APIS.PREPARE_TEXT_INDEX

Format

```
OPG APIS.PREPARE TEXT INDEX();
```

Description

Performs preparatory work needed before a text index can be created on any NVARCHAR2 columns.

Parameters

None.

Usage Notes

You must have the ALTER SESSION to run this procedure.



Examples

The following example performs preparatory work needed before a text index can be created on any NVARCHAR2 columns.

```
EXECUTE OPG APIS.PREPARE TEXT INDEX();
```

A.11.52 OPG_APIS.RENAME_PG

Format

Description

Renames a property graph.

Parameters

graph_name

Name of the property graph.

new_graph_name

New name for the property graph.

Usage Notes

The graph name property graph must exist in the database.

Examples

The following example changes the name of a property graph named mypg to mynewpg.

```
EXECUTE OPG APIS.RENAME PG('mypg', 'mynewpg');
```

A.11.53 OPG_APIS.SPARSIFY_GRAPH

Format

```
OPG_APIS.SPARSIFY_GRAPH(

edge_tab_name IN VARCHAR2,
threshold IN NUMBER DEFAULT 0.5,
min_keep IN INTEGER DEFAULT 1,
dop IN INTEGER DEFAULT 4,
wt_out_tab IN OUT VARCHAR2,
wt_und_tab IN OUT VARCHAR2,
wt_hsh_tab IN OUT VARCHAR2,
wt_mch_tab IN OUT VARCHAR2,
tbs IN VARCHAR2 DEFAULT NULL,
options IN VARCHAR2 DEFAULT NULL);
```

Description

Performs sparsification (edge trimming) for a property graph edge table.



Parameters

edge tab name

Name of the property graph edge table (GE\$).

threshold

A numeric value controlling how much sparsification needs to be performed. The lower the value, the more edges will be removed. Some typical values are: 0.1, 0.2, ..., 0.5

min keep

A positive integer indicating at least how many adjacent edges should be kept for each vertex. A recommended value is 1.

dop

Degree of parallelism for the operation.

wt_out_tab

A working table to hold the output, a sparsified graph.

wt und tab

A working table to hold the undirected version of the original graph.

wt hsh tab

A working table to hold the min hash values of the graph.

wt mch tab

A working table to hold matching count of min hash values.

ths

A working table to hold the working table data.

options

Additional settings for operation. An optional string with one or more (comma-separated) of the following values:

- 'INMEMORY=T' is an option for creating the schema tables with an 'inmemory' clause.
- 'IMC_MC_B=T' creates the schema tables with an INMEMORY MEMCOMPRESS BASIC clause.

Usage Notes

The CREATE TABLE privilege is required to call this procedure.

The sparsification algorithm used is a min hash based local sparsification. See "Local graph sparsification for scalable clustering", Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data: https://cs.uwaterloo.ca/~tozsu/courses/CS848/W15/presentations/ElbagouryPresentation-2.pdf

Sparsification only involves the topology of a graph. None of the properties (K/V) are relevant.

Examples

The following example does the preparation work for the edges table of mypg, prints out the working table names, and runs sparsification. The output, a sparsified graph, is stored in a table named LEAN_PG, which has two columns, SVID and DVID.

SQL> set serveroutput on DECLARE



```
my lean pg varchar2(100) := 'lean pg'; -- output table
 wt2 varchar2(100);
 wt3 varchar2(100);
 wt4 varchar2(100);
BEGIN
 opg apis.sparsify graph prep('mypgGE$', my lean pg, wt2, wt3, wt4, null);
 dbms_output.put_line('wt2 ' || wt2 || ', wt3 ' || wt3 || ', wt4 '|| wt4);
 opg_apis.sparsify_graph('mypgGE$', 0.5, 1, 4, my_lean_pg, wt2, wt3, wt4,
'SEMTS', null);
END;
wt2 "MYPGGE$$TWSPAU275", wt3 "MYPGGE$$TWSPAH275", wt4 "MYPGGE$$TWSPAM275"
SQL> describe lean_pg;
Name
                         Null? Type
 SVID
                              NUMBER
 DVID
                               NUMBER
```

A.11.54 OPG_APIS.SPARSIFY_GRAPH_CLEANUP

Format

```
OPG_APIS.SPARSIFY_GRAPH_CLEANUP(
    edge_tab_name IN VARCHAR2,
    wt_out_tab IN OUT VARCHAR2,
    wt_und_tab IN OUT VARCHAR2,
    wt_hsh_tab IN OUT VARCHAR2,
    wt_mch_tab IN OUT VARCHAR2,
    options IN VARCHAR2 DEFAULT NULL);
```

Description

Cleans up after sparsification (edge trimming) for a property graph edge table.

Parameters

edge_tab_name

Name of the property graph edge table (GE\$).

wt_out_tab

A working table to hold the output, a sparsified graph.

wt_und_tab

A working table to hold the undirected version of the original graph.

wt_hsh_tab

A working table to hold the min hash values of the graph.

wt mch tab

A working table to hold matching count of min hash values.

tbs

A working table to hold the working table data

options

(Reserved for future use.)

Usage Notes

The working tables will be dropped after the operation completes.

Examples

The following example does the preparation work for the edges table of mypg, prints out the working table names, runs sparsification, and then performs cleanup.

```
SQL> set serveroutput on
DECLARE
    my_lean_pg    varchar2(100) := 'lean_pg';
    wt2 varchar2(100);
    wt3 varchar2(100);
    wt4 varchar2(100);

BEGIN
    opg_apis.sparsify_graph_prep('mypgGE$', my_lean_pg, wt2, wt3, wt4, null);
    dbms_output.put_line('wt2 ' || wt2 || ', wt3 ' || wt3 || ', wt4 '|| wt4);

    opg_apis.sparsify_graph('mypgGE$', 0.5, 1, 4, my_lean_pg, wt2, wt3, wt4, 'SEMTS', null);

-- Add logic here to consume SVID, DVID in LEAN_PG table
-- cleanup
    opg_apis.sparsify_graph_cleanup('mypgGE$', my_lean_pg, wt2, wt3, wt4, null);
END;
//
```

A.11.55 OPG APIS.SPARSIFY GRAPH PREP

Format

```
OPG_APIS.SPARSIFY_GRAPH_PREP(
   edge_tab_name IN VARCHAR2,
   wt_out_tab IN OUT VARCHAR2,
   wt_und_tab IN OUT VARCHAR2,
   wt_hsh_tab IN OUT VARCHAR2,
   wt_mch_tab IN OUT VARCHAR2,
   options IN VARCHAR2 DEFAULT NULL);
```

Description

Prepares working table names that are necessary to run sparsification for a property graph edge table.

Parameters

edge_tab_name

Name of the property graph edge table (GE\$).

wt out tab

A working table to hold the output, a sparsified graph.



wt und tab

A working table to hold the undirected version of the original graph.

wt hsh tab

A working table to hold the min hash values of the graph.

wt mch tab

A working table to hold the matching count of min hash values.

options

Additional settings for operation. An optional string with one or more (commaseparated) of the following values:

- 'INMEMORY=T' is an option for creating the schema tables with an 'inmemory' clause.
- 'IMC_MC_B=T' creates the schema tables with an INMEMORY MEMCOMPRESS BASIC clause.

Usage Notes

The sparsification algorithm used is a min hash based local sparsification. See "Local graph sparsification for scalable clustering", Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data: https://cs.uwaterloo.ca/~tozsu/courses/CS848/W15/presentations/ElbagouryPresentation-2.pdf

Examples

The following example does the preparation work for the edges table of mypg and prints out the working table names.

```
set serveroutput on

DECLARE
    my_lean_pg    varchar2(100) := 'lean_pg';
    wt2 varchar2(100);
    wt3 varchar2(100);
    wt4 varchar2(100);

BEGIN
    opg_apis.sparsify_graph_prep('mypgGE$', my_lean_pg, wt2, wt3, wt4, null);
    dbms_output.put_line('wt2 ' || wt2 || ', wt3 ' || wt3 || ', wt4 '|| wt4);
END;
//
```

The output may be similar to the following.

```
wt2 "MYPGGE$$TWSPAU275", wt3 "MYPGGE$$TWSPAH275", wt4 "MYPGGE$$TWSPAM275"
```

A.12 OPG_GRAPHOP Package Subprograms

The OPG_GRAPHOP package contains subprograms for various operations on property graphs in an Oracle database.

To use the subprograms in this chapter, you must understand the conceptual and usage information in earlier chapters of this book.

This chapter provides reference information about the subprograms, in alphabetical order.



OPG_GRAPHOP.POPULATE_SKELETON_TAB

A.12.1 OPG_GRAPHOP.POPULATE_SKELETON_TAB

Format

```
OPG_GRAPHOP.POPULATE_SKELETON_TAB(
graph IN VARCHAR2,
dop IN INTEGER DEFAULT 4,
tbs IN VARCHAR2 DEFAULT NULL,
options IN VARCHAR2 DEFAULT NULL);
```

Description

Populates the skeleton table (<graph-name>GT\$). By default, any existing content in the skeleton table is truncated (removed) before the table is populated.

Parameters

graph

Name of the property graph.

dop

Degree of parallelism for the operation.

tbs

Name of the tablespace to hold the index data for the skeleton table.

options

Options that can be used to customize the populating of the skeleton table. (One or more, comma separated.)

- 'KEEP_DATA=T' causes any existing table not to be removed before the table is populated. New rows are added after the existing ones.
- 'PDML=T' skips the default index creation.

Usage Notes

You must have the CREATE TABLE and CREATE INDEX privileges to call this procedure.

There is a unique index constraint on EID column of the skeleton table (GE\$). So if you specify the KEEP_DATA=T option and if the new data overlaps with existing one, then the unique key constraint will be violated, resulting in an error.

Examples

The following example populates the skeleton table of the property graph named mypg.

```
EXECUTE OPG_GRAPHOP.POPULATE_SKELETON_TAB('mypg',4, 'pgts', 'PDML=T');
```



B

Mapping Graph Server Roles to Default Privileges

Installing the PL/SQL packages of the Oracle Graph Server and Client distribution on the target Oracle Database, automatically creates the following roles and assigns the default permissions as shown in the following table:

Table B-1 Mapping Graph Server Roles to Default Privileges

Roles	Description	Permission
GRAPH_ADMINISTR ATOR	User who performs operations on the graph server (PGX) using the Java API. (As compared to running start and stop operations as an OS user.)	PGX_SESSION_CREATE PGX_SERVER_GET_INFO PGX_SERVER_MANAGE
GRAPH_DEVELOPER	User who creates graphs, publishes graphs, modifies graphs, queries graphs, and views graphs using the Java API or SQLcI or the graph visualization application.	PGX_SESSION_CREATE PGX_SESSION_NEW_GRAPH PGX_SESSION_GET_PUBLISHED_GR APH PGX_SESSION_MODIFY_MODEL PGX_SESSION_READ_MODEL
GRAPH_USER	User who queries graphs and views graphs Java API or SQLcl or the graph visualization application.	PGX_SESSION_CREATE PGX_SESSION_GET_PUBLISHED_GR APH



C

Disabling Transport Layer Security (TLS) in Graph Server

For demonstration or evaluation purposes, it is possible to turn off transport layer security (TLS) of the graph server.



Caution:

This is **not** recommended for production. In a secure configuration, the server must always have TLS enabled.

The following instructions only apply if you installed the graph server via the RPM package.



If you deployed the graph server into your own web server (e.g Weblogic or Apache Tomcat), please refer to the manual of your web server for TLS configuration.

- 1. Edit /etc/oracle/graph/server.conf to change enable tls to false.
- 2. Edit the WEB-INF/web.xml file inside the WAR file in /opt/oracle/graph/graphviz and configure cookies to be sent over non-secure connections by setting <secure>false</secure> as follows:

3. Additionally, replace `https` with `http` in the `pgx.base_url` property in the same WEB-INF/web.xml file. For example:

4. Restart the server.

```
sudo systemctl restart pgx
```

The graph server now accepts connections over HTTP instead of HTTPS.

On Oracle Linux 7, you can execute the following script to perform the preceding four steps all at once:

```
echo "$(jq '.enable_tls = false' /etc/oracle/graph/server.conf)"
> /etc/oracle/graph/server.conf
WAR=$(find /opt/oracle/graph/graphviz -name '*.war')
TMP=$(mktemp -d)
cd $TMP
unzip $WAR WEB-INF/web.xml
sed -i 's|<secure>true</secure>|<secure>false</secure>|' WEB-INF/
web.xml
sed -i 's|https://|http://|' WEB-INF/web.xml
sudo zip $WAR WEB-INF/web.xml
rm -r $TMP
sudo systemctl restart pgx
```



D

Migrating Property Graph Applications from Before Release 21c

If you are migrating from a previous version of Oracle Spatial and Graph to Release 21c, you may need to make some changes to existing property graph-related applications.

Also note that Oracle Graph Server and Client is required for property graph applications. This can be downloaded from Oracle Software Delivery Cloud or from Oracle Downloads page.

Security-Related Changes

The Property Graph feature contains a series of enhancements to further strengthen the security of the property graph component of product. The following enhancements may require manual changes to existing graph applications so that they continue to work properly.

 Graph configuration files now require sensitive information such as passwords to be stored in Java Keystore files

If you use graph configuration files you are required to use Java Keystore files to store sensitive information such as passwords. (See Store the Database Password in a Keystore for how to create and reference such a keystore.)

All existing graph configuration files with secrets in them must be migrated to the keystore-based approach.

 In a three-tier deployment, access to the PGX server file system requires a directories allowlist

By default, the PGX server does not allow remote access to the local file system. This can be explicitly allowed, though, in /etc/oracle/graph/pgx.conf by setting allow_local_filesystem to true. If you set allow_local_filesystem to true, you must also specify a list of directories that are allowed to be accessed, by setting datasource dir whitelist. For example:

```
"allow_local_filesystem": true,
"datasource dir whitelist": ["/scratch/data1", "/scratch/data2"]
```

This will allow remote users to read and write data on the server's file-system from and into /scratch/data1 and /scratch/data2.

 In a three-tier deployment, reading from remote locations into PGX is no longer allowed by default

Previously, PGX allowed graph data to be read from remote locations over FTP or HTTP. This is no longer allowed by default and requires explicit opt-in by the server administrator. To opt-in, specify the <code>allowed_remote_loading_locations</code> configuration option in <code>/etc/oracle/graph/pgx.conf</code>. For example:

```
allowed remote loading locations: ["*"]
```

In addition:

- The ftp and http protocols are no longer supported for loading or storing data because they are unencrypted and thus insecure.
- Configuration files can no longer be loaded from remote locations, but must be loaded from the local file system.

Removed shell command line options

The following command line options of the Groovy-based opg shell have been removed and will no longer work:

- --attach the shell no longer supports attaching to existing sessions via command line
- --password the shell will prompt now for the password

Also note that the Groovy-based shell has been deprecated, and you are encourage to use the new JShell-based shell instead (see Interactive Graph Shell CLIs).

Changes to PGX APIs

The following APIs no longer return graph configuration information:

- ServerInstance#getGraphInfo()
- ServerInstance#getGraphInfos()
- ServerInstance#getServerState()

The REST API now identifies collections, graphs, and properties by UUID instead of a name.

The namespaces for graphs and properties are session private by default now. This implies that some operations that would previously throw an exception due to a naming conflict could succeed now.

PgxGraph#publish() throws an exception now if a graph with the given name has been published before.

Migrating Data to a New Database Version

Oracle Graph Server and Client works with older database versions. (See Verifying Database Compatibility for information.) If as part of your upgrade you also upgraded your Oracle Database, you can migrate your existing graph data that was stored using the Oracle Property Graph format by invoking the following helper script in your database after the upgrade:

```
sqlplus> EXECUTE
mdsys.opg.migrate pg to current(graph name=>'mygraph');
```

The preceding example migrates the property graph *mygraph* to the current database version.

Uninstalling Previous Versions of Property Graph Libraries

This is only necessary if you are using Oracle Database versions 12.2, 18c, or 19c.

Use of the Property Graph feature of Oracle Database now requires Oracle Graph Server and Client that is installed separately. After you have completed the Graph Server and Client installation, complete the preceding migration steps (if needed), and confirmed that everything is working well, it is recommended that you remove the



binaries of *older* graph installations from your Oracle Database installation by performing the following un-install steps:

- Make sure the Property Graph mid-tier components are not in use on the target database host. For example, ensure that there is no application running which uses any files under \$ORACLE_HOME/md/property_graph. Examples of such an application are a running PGX server on the same host as the database or a client application that references the JAR files under \$ORACLE_HOME/md/property_graph/lib.
 - It is **not** necessary to shut down the database to perform the uninstall. The Oracle database itself does not reference or use any files under $\protect\prot$
- 2. Remove the files under <code>\$ORACLE_HOME/md/property_graph</code> on your database host. On Linux, you can copy the following helper script to your database host and run it with as the DBA operating system user: <code>/opt/oracle/graph/scripts/patch-opg-oracle-home.sh</code>



Е

Upgrading From Graph Server and Client 20.4.x to 21.x

If you are upgrading from Graph Server and Client 20.4.x to 21.x version, you may need to create new roles in database and migrate authorization rules from pgx.conf file to the database. Also, starting from Graph Server and Client Release 21.1, TLS is enforced at the time of the RPM file installation.

One of the main enhancements of Graph Server and Client Release 21.1 is moving the graph access permissions from the pgx.conf file to the database. A new set of graph roles with default permissions are created automatically in the database, at the time of the PL/SQL packages installation. See Table B-1 in the appendix for more details on the default mappings.

In order to comply with this feature you must perform the database actions explained in the following sections:

Creating additional roles in the database

The roles in the database with additional privileges are created when you install the 21.x PL/SQL packages in your database as part of the upgrade. If you are not able to install the PL/SQL packages, for example if you are using an Autonomous Database, see User Authentication and Authorization for more information on manually creating these roles in the database with the default set of privileges.

Migrating authorization rules

You must execute database $\tt GRANTS$ for user-added mappings contained in the $\tt pgx.conf$ file when upgrading to 21.x.

The following examples explain the various scenarios where migration of authorization rules may or may not apply.

Example E-1 Migrating user-added mappings to database

To migrate the following user-added mappings in pgx.conf file:

```
"authorization": [{
    "pgx_role": "GRAPH_DEVELOPER",
    "pgx_permissions": [{
        "grant": "PGX_SESSION_ADD_PUBLISHED_GRAPH"
    },
```

You must execute the following GRANT statement in the database used by 21.x:

```
GRANT PGX SESSION ADD PUBLISHED GRAPH TO GRAPH DEVELOPER
```

Example E-2 Migrating user-added file system authorization rules to database

To migrate the following user-added file system authorization rules in pgx.conf file:

```
"file_locations": [{
    "name": "my_hdfs_graph_data",
    "location": "hdfs:/data/graphs"
}],
"authorization": [{
    "pgx_role": "GRAPH_DEVELOPER",
    "pgx_permissions": [{
        "file_location": "my_hdfs_graph_data",
        "grant": "read"
    },
...
```

You must execute the following GRANT statement in the database used by 21.x:

CREATE OR REPLACE DIRECTORY my_hdfs_graph_data AS 'hdfs:/data/graphs' GRANT READ ON DIRECTORY my hdfs graph data TO GRAPH DEVELOPER

Example E-3 User-added graph authorization rules for preloaded graphs



No migration required for user-added graph authorization rules for preloaded graphs.

You must not migrate user-added graph authorization rules for preloaded graphs (as shown in the following code) as these rules continue to be configured in pgx.conf file.

```
"preload_graphs": [{
    "path": "/data/my-graph.json",
    "name": "global_graph"
}],
"authorization": [{
    "pgx_role": "GRAPH_DEVELOPER",
    "pgx_permissions": [{
        "preloaded_graph": "global_graph",
        "grant": "read"
    },
...
```

Self-signed TLS certificate now generated upon RPM installation

In Graph Server and Client 21.x the RPM installation generates a self-signed certificate into /etc/oracle/graph, which the server uses to enable TLS by default.

According to security best practices, access to the certificate is restricted to the <code>oraclegraph</code> operating system user. The implication of this is that you no longer can start the graph server via the <code>/opt/oracle/graph/pgx/bin/start-server</code> script, even if your user is part of the <code>oraclegraph</code> group. Instead, manage the lifecycle of the graph server via <code>systemctl</code> commands. For example:

sudo systemctl start pgx

Another possible option is to change the ownership of the certificate as shown:

sudo chown <youruser> /etc/oracle/graph/server key.pem

Turning off TLS is not recommended as it reduces the security of your connection. However, if you must do so, see Disabling Transport Layer Security (TLS) in Graph Server for more details.



F

Third-Party License Information for Oracle Graph Server and Client

This appendix contains licensing information about third-party products included with Oracle Graph Server and Client.

cytoscape.js

Vendor: Cytoscape Consortium

Version: 3.23.0

Cytoscape.js v. 3.23.0 Source URL: https://github.com/cytoscape/cytoscape.js/tree/v3.23.0 MIT License & Copyright https://github.com/cytoscape/cytoscape.js/blob/v3.23.0/LICENSE

Copyright (c) 2016-2022, The Cytoscape Consortium.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

```
Fourth party dependencies***

Heap v. 0.2.6

Source url: https://github.com/qiao/heap.js/tree/0.2.6

License PSF (other)

License and copyright URL:https://github.com/qiao/heap.js/blob/0.2.6/README.md

Ported by Xueqiao Xu

PSF LICENSE AGREEMENT FOR PYTHON 2.7.2
```

This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and the Individual or Organization ("Licensee") accessing and otherwise using Python 2.7.2 software in source or binary form and its associated documentation.

Subject to the terms and conditions of this License Agreement, PSF hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python 2.7.2 alone or in any derivative version, provided, however, that PSF's License Agreement and PSF's notice of copyright, i.e., "Copyright © 2001-2012 Python Software Foundation; All Rights Reserved" are retained in Python 2.7.2 alone or in any derivative version prepared by Licensee.

In the event Licensee prepares a derivative work that is based on or incorporates Python 2.7.2 or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python 2.7.2.

PSF is making Python 2.7.2 available to Licensee on an "AS IS" basis. PSF MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, PSF MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON 2.7.2 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 2.7.2 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 2.7.2, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

This License Agreement will automatically terminate upon a material breach of its terms and conditions.

Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between PSF and Licensee. This License Agreement does not grant permission to use PSF trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.

By copying, installing or otherwise using Python 2.7.2, Licensee agrees to be bound by the terms and conditions of this License Agreement.

lodash v. 4.17.21

Source url: https://github.com/lodash/lodash/tree/4.17.21 License URL: https://github.com/lodash/lodash/blob/4.17.21/LICENSE Copyright JS Foundation and other contributors

Based on Underscore.js, copyright Jeremy Ashkenas, DocumentCloud and Investigative Reporters & Editors

This software consists of voluntary contributions made by many individuals. For exact contribution history, see the revision history available at https://github.com/lodash/lodash

The following license applies to all parts of this software except as documented below:

====

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall bemonaco included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF

MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

====

Copyright and related rights for sample code are waived via CCO. Sample code is defined as all source code displayed within the prose of the documentation.

CCO: http://creativecommons.org/publicdomain/zero/1.0/

====

Files located in the node_modules and vendor directories are externally maintained libraries used by this software which have their own licenses; we recommend you read them, as their terms may differ from the terms above.

Iodash

Vendor: OpenJS Foundation

Version: 4.17.21

The MIT License

Copyright JS Foundation and other contributors

Based on Underscore.js, copyright Jeremy Ashkenas, DocumentCloud and Investigative Reporters & Editors

This software consists of voluntary contributions made by many individuals. For exact contribution history, see the revision history available at https://github.com/lodash/lodash

The following license applies to all parts of this software except as documented below:

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.



====

Copyright and related rights for sample code are waived via CCO. Sample code is defined as all source code displayed within the prose of the documentation.

CCO: http://creativecommons.org/publicdomain/zero/1.0/

====

Files located in the node_modules and vendor directories are externally maintained libraries used by this software which have their own licenses; we recommend you read them, as their terms may differ from the terms above.

Moment.js

Vendor: JS Foundation and other contributors

Version: 2.29.4

Copyright (c) JS Foundation and other contributors

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

three.js

Vendor: three.js authors

Version: 0.126.0

The MIT License

Copyright © 2010-2018 three.js authors

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in



all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Nimbus JOSE+JWT

Vendor: Connect2id Ltd.

Version: 9.14

```
Copyright Info for Nimbus-jose-jwt 9.14 (Apache 2.0 License)
* nimbus-jose-jwt
 * Copyright 2012-2021, Connect2id Ltd.
 * Licensed under the Apache License, Version 2.0 (the "License"); you may not use
 * this file except in compliance with the License. You may obtain a copy of the
 * License at
     http://www.apache.org/licenses/LICENSE-2.0
 * Unless required by applicable law or agreed to in writing, software distributed
 * under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR
 * CONDITIONS OF ANY KIND, either express or implied. See the License for the
 * specific language governing permissions and limitations under the License.
4th Party Dependencies:

    com.github.stephenc.jcip » jcip-annotations (Apache 2.0)

------ Copyright Info ------
\star Copyright 2013 Stephen Connolly.
* Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
      http://www.apache.org/licenses/LICENSE-2.0
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 ^{\star} See the License for the specific language governing permissions and
 * limitations under the License.
2. net.minidev:json-smart (Apache 2.0)
===copyright info for json-smart ======
/*
     Copyright 2011 JSON-SMART authors
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
```



```
* http://www.apache.org/licenses/LICENSE-2.0
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
3.net.minidev:accessories-smart (Apache 2.0)
===copyright info for accessories-smart ======
     Copyright 2011 JSON-SMART authors
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 * http://www.apache.org/licenses/LICENSE-2.0
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
4. org.ow2.asm » asm (BSD 3-clause)
----- Copyright Info -----
/***
* ASM: a very small and fast Java bytecode manipulation framework
 * Copyright (c) 2000-2011 INRIA, France Telecom
 * All rights reserved.
 ^{\star} Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * 1. Redistributions of source code must retain the above copyright
     notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
     notice, this list of conditions and the following disclaimer in the
     documentation and/or other materials provided with the distribution.
 ^{\star} 3. Neither the name of the copyright holders nor the names of its
     contributors may be used to endorse or promote products derived from
     this software without specific prior written permission.
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
 * THE POSSIBILITY OF SUCH DAMAGE.
----- Standard Apache 2.0 License text common to all
components licensed under Apache 2.0-
Apache License
                          Version 2.0, January 2004
```

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document. "Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License. "Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity. "You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License. "Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution." "Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

- 2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.
- 3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work,



where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

- 4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:
 - (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and
 - (b) You must cause any modified files to carry prominent notices stating that You changed the files; and
 - (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
 - (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

- 5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.
- 6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.
- 7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions



- of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.
- 8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
- 9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.
 END OF TERMS AND CONDITIONS

jackson-annotations

Vendor: FasterXML, LLC

Version: 2.13.2

Apache License

Version 2.0, January 2004 http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

- 2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.
- 3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.
- 4. Redistribution. You may reproduce and distribute copies of the



Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

- (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and
- (b) You must cause any modified files to carry prominent notices stating that You changed the files; and
- (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
- (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

- 5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.
- 6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.
- 7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any



risks associated with Your exercise of permissions under this License.

- 8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
- 9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Copyright © 2007-2022 FasterXML. All rights reserved.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0
Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.



See the License for the specific language governing permissions and limitations under the License.

jackson-core

Vendor: FasterXML, LLC

Version: 2.13.2

Copyright © 2008-2022 FasterXML. All rights reserved.

Jackson JSON processor

Jackson is a high-performance, Free/Open Source JSON processing library. It was originally written by Tatu Saloranta (tatu.saloranta@iki.fi), and has been in development since 2007.

It is currently developed by a community of developers.

Licensing

Jackson 2.x core and extension components are licensed under Apache License 2.0 To find the details that apply to this artifact see the accompanying LICENSE file.

Credits

A list of contributors may be found from CREDITS(-2.x) file, which is included in some artifacts (usually source distributions); but is always available from the source code management (SCM) system project uses. Apache License

Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.



"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

- 2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.
- 3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.
- 4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:
 - (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and



- (b) You must cause any modified files to carry prominent notices stating that You changed the files; and
- (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
- (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

- 5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.
- 6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.
- 7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.
- 8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be



liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

jackson-databind

Vendor: FasterXML, LLC

Version: 2.13.4.2

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction,

and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the



Work and such Derivative Works in Source or Object form.

- 3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.
- 4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:
 - (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and
 - (b) You must cause any modified files to carry prominent notices stating that You changed the files; and
 - (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
 - (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work



by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

- 6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.
- 7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.
- 8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
- 9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License.



You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

----- Copyright notices ------

Jackson JSON processor

Jackson is a high-performance, Free/Open Source JSON processing library. It was originally written by Tatu Saloranta (tatu.saloranta@iki.fi), and has been in development since 2007.

It is currently developed by a community of developers.

Licensing

Jackson 2.x core and extension components are licensed under Apache License 2.0 To find the details that apply to this artifact see the accompanying LICENSE file.

Credits

A list of contributors may be found from CREDITS(-2.x) file, which is included in some artifacts (usually source distributions); but is always available from the source code management (SCM) system project uses.

----- Fourth-party information ------

== jackson-core

== License

Apache 2.0

== Copyright Notices

Copyright (c) 2007- Tatu Saloranta, tatu.saloranta@iki.fi

Jackson JSON processor

Jackson is a high-performance, Free/Open Source JSON processing library. It was originally written by Tatu Saloranta (tatu.saloranta@iki.fi), and has been in development since 2007.

It is currently developed by a community of developers.

Licensing

Jackson 2.x core and extension components are licensed under Apache License 2.0 To find the details that apply to this artifact see the accompanying LICENSE file.

Credits

A list of contributors may be found from CREDITS(-2.x) file, which is included in some artifacts (usually source distributions); but is always available from the source code management (SCM) system project uses.

----- (separator)------

== jackson-annotations

== License

Apache 2.0

== Copyright Notices Copyright © 2007-2022 FasterXML. All rights reserved.

Licensed under the Apache License, Version 2.0 (the "License");

you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

----- (separator)

jackson-jaxrs-base

Vendor: FasterXML, LLC

Version: 2.13.2

This copy of Jackson JSON processor databind module is licensed under the Apache (Software) License, version 2.0 ("the License"). See the License for details about distribution rights, and the specific rights regarding derivate works.

You may obtain a copy of the License at:

http://www.apache.org/licenses/LICENSE-2.0

Jackson JSON processor

Jackson is a high-performance, Free/Open Source JSON processing library. It was originally written by Tatu Saloranta (tatu.saloranta@iki.fi), and has been in development since 2007.

It is currently developed by a community of developers, as well as supported commercially by FasterXML.com.

Licensing

Jackson core and extension components may licensed under different licenses. To find the details that apply to this artifact see the accompanying LICENSE file. For more information, including possible other licensing options, contact FasterXML.com (http://fasterxml.com).

Credits

A list of contributors may be found from CREDITS file, which is included in some artifacts (usually source distributions); but is always available from the source code management (SCM) system project uses.

jackson-jaxrs-base v2.13.2

COPYRIGHT: Copyright FasterXML.com

LICENSE: Apache 2.0

Apache License Version 2.0, January 2004 http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction,



and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the



Work and such Derivative Works in Source or Object form.

- 3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.
- 4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:
 - (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and
 - (b) You must cause any modified files to carry prominent notices stating that You changed the files; and
 - (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
 - (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

- 6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.
- 7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.
- 8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
- 9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

4P Dependencies:

jackson-core
jackson-annotations
jackson-databind

COPYRIGHT:

Copyright (c) 2007- Tatu Saloranta, tatu.saloranta@iki.fi

Copyright (c) Fasterxml

LICENSE: Apache 2.0

Apache License Version 2.0, January 2004



http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and



subsequently incorporated within the Work.

- 2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.
- 3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.
- 4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:
 - (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and
 - (b) You must cause any modified files to carry prominent notices stating that You changed the files; and
 - (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
 - (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions



for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

- 5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.
- 6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.
- 7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.
- 8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
- 9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

1_____

-------FOURTH PARTY DEPENDENCY

jackson-core 2.12.5

== License

Apache 2.0

== Copyright Notice

Jackson JSON processor



Jackson is a high-performance, Free/Open Source JSON processing library. It was originally written by Tatu Saloranta (tatu.saloranta@iki.fi), and has been in development since 2007.

It is currently developed by a community of developers.

Licensing

Jackson 2.x core and extension components are licensed under Apache License 2.0 To find the details that apply to this artifact see the accompanying LICENSE file.

Credits

A list of contributors may be found from CREDITS(-2.x) file, which is included in some artifacts (usually source distributions); but is always available from the source code management (SCM) system project uses.

------FOURTH PARTY

DEPENDENCY 2-----

jackson-databind 2.12.5

== License

Apache 2.0

== Copyright Notice

Jackson JSON processor

Jackson is a high-performance, Free/Open Source JSON processing library. It was originally written by Tatu Saloranta (tatu.saloranta@iki.fi), and has been in development since 2007.

It is currently developed by a community of developers.

Licensing

Jackson 2.x core and extension components are licensed under Apache License 2.0 To find the details that apply to this artifact see the accompanying LICENSE file.

Credits

A list of contributors may be found from CREDITS(-2.x) file, which is included in some artifacts (usually source distributions); but is always available from the source code management (SCM) system project uses.

jackson-jaxrs-json-provider

Vendor: FasterXML, LLC

Version: 2.13.2

jackson-jaxrs-json-provider

Copyright (c) Fasterxml 2020

NOTICE file text:

Jackson JSON processor

Jackson is a high-performance, Free/Open Source JSON processing library. It was originally written by Tatu Saloranta (tatu.saloranta@iki.fi), and has been in development since 2007.

It is currently developed by a community of developers, as well as supported commercially by FasterXML.com.

Licensing

Jackson core and extension components may be licensed under different licenses. To find the details that apply to this artifact see the accompanying LICENSE file. For more information, including possible other licensing options, contact FasterXML.com (http://fasterxml.com).

Credits

A list of contributors may be found from CREDITS file, which is included in some artifacts (usually source distributions); but is always available from the source code management (SCM) system project uses.

LICENSE: Apache 2.0

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain



separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

- 2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.
- 3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.
- 4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:
 - (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and
 - (b) You must cause any modified files to carry prominent notices stating that You changed the files; and
 - (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and



(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

- 5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.
- 6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.
- 7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.
- 8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
- 9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer,



and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Dependencies:

jackson-core

Copyright © 2008-2019 FasterXML. All rights reserved.

This copy of Jackson JSON processor streaming parser/generator is licensed under the

Apache (Software) License, version 2.0 ("the License"). See the License for details about distribution rights, and the specific rights regarding derivate works.

You may obtain a copy of the License at:

http://www.apache.org/licenses/LICENSE-2.0

jackson-jaxrs-base
Copyright (c) Fasterxml 2020

This copy of Jackson JSON processor databind module is licensed under the Apache (Software) License, version 2.0 ("the License"). See the License for details about distribution rights, and the specific rights regarding derivate works.



```
You may obtain a copy of the License at:
http://www.apache.org/licenses/LICENSE-2.0
_____
jackson-module-jaxb-annotations
Copyright (c) Fasterxml 2020
This copy of Jackson JSON processor `jackson-module-jaxb-annotations` module is
licensed under the
Apache (Software) License, version 2.0 ("the License").
See the License for details about distribution rights, and the
specific rights regarding derivate works.
You may obtain a copy of the License at:
http://www.apache.org/licenses/LICENSE-2.0
jackson-annotations
Copyright (c) Fasterxml 2020
This copy of Jackson JSON processor annotations is licensed under the
Apache (Software) License, version 2.0 ("the License").
See the License for details about distribution rights, and the
specific rights regarding derivate works.
You may obtain a copy of the License at:
http://www.apache.org/licenses/LICENSE-2.0
_____
jackson-databind
Copyright (c) Fasterxml 2020
This copy of Jackson JSON processor databind module is licensed under the
Apache (Software) License, version 2.0 ("the License").
See the License for details about distribution rights, and the
specific rights regarding derivate works.
You may obtain a copy of the License at:
http://www.apache.org/licenses/LICENSE-2.0
_____
jakarta.xml.bind-api
Eclipse Distribution License 1.0
    Copyright (c) 2017, 2018 Oracle and/or its affiliates. All rights reserved.
   Redistribution and use in source and binary forms, with or without
   modification, are permitted provided that the following conditions
    are met:
      - Redistributions of source code must retain the above copyright
        notice, this list of conditions and the following disclaimer.
      - Redistributions in binary form must reproduce the above copyright
        notice, this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.
```



- Neither the name of the Eclipse Foundation, Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

NOTICE file text for jakarta.xml.bind-api

Notices for Eclipse Project for JAXB

This content is produced and maintained by the Eclipse Project for JAXB project.

- * Project home: https://projects.eclipse.org/projects/ee4j.jaxb
- ## Trademarks

Eclipse Project for JAXB is a trademark of the Eclipse Foundation.

Copyright

All content is the property of the respective authors or their employers. For more information regarding authorship of content, please consult the listed source code repository logs.

Declared Project Licenses

This program and the accompanying materials are made available under the terms of the Eclipse Distribution License v. 1.0 which is available at http://www.eclipse.org/org/documents/edl-v10.php.

SPDX-License-Identifier: BSD-3-Clause

Source Code

The project maintains the following source code repositories:

- * https://github.com/eclipse-ee4j/jaxb-api
- ## Third-party Content

This project leverages the following third party content.

None

Cryptography

Content may contain encryption software. The country in which you are currently may have restrictions on the import, possession, and use, and/or re-export to another country, of encryption software. BEFORE using any encryption software,

please check the country's laws, regulations and policies concerning the import, possession, or use, and re-export of encryption software, to see if this is permitted.

jakarta.activation-api

Copyright (c) 2018 Oracle and/or its affiliates. All rights reserved. Eclipse Distribution License $1.0\,$

NOTICE file text for jakarta.activation-api:

Notices for Eclipse Project for JAF

This content is produced and maintained by the Eclipse Project for JAF project.

- * Project home: https://projects.eclipse.org/projects/ee4j.jaf
- ## Copyright

All content is the property of the respective authors or their employers. For more information regarding authorship of content, please consult the listed source code repository logs.

Declared Project Licenses

This program and the accompanying materials are made available under the terms of the Eclipse Distribution License v. 1.0, which is available at http://www.eclipse.org/org/documents/edl-v10.php.

SPDX-License-Identifier: BSD-3-Clause

Source Code

The project maintains the following source code repositories:

* https://github.com/eclipse-ee4j/jaf

jackson-module-jaxb-annotations

Vendor: FasterXML, LLC

Version: 2.13.2

Apache License

Version 2.0, January 2004 http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or

otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

- 2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.
- 3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s)



with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

- 4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:
 - (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and
 - (b) You must cause any modified files to carry prominent notices stating that You changed the files; and
 - (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
 - (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

- 5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.
- 6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.



- 7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.
- 8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
- 9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.



Jackson JSON processor

Jackson is a high-performance, Free/Open Source JSON processing library. It was originally written by Tatu Saloranta (tatu.saloranta@iki.fi), and has been in development since 2007.

It is currently developed by a community of developers, as well as supported commercially by FasterXML.com.

Licensing

Jackson core and extension components may licensed under different licenses. To find the details that apply to this artifact see the accompanying LICENSE file. For more information, including possible other licensing options, contact FasterXML.com (http://fasterxml.com).

Credits

A list of contributors may be found from CREDITS file, which is included in some artifacts (usually source distributions); but is always available from the source code management (SCM) system project uses.

```
_____
Fourth Party Dependency #1 Name : jackson-core
Fourth Party Dependency #1 License : Apache 2.0
Fourth Party Dependency #1 Copyright : Copyright
_____
ourth Party Dependency #2 Name : jackson-annotations
Fourth Party Dependency #2 License : Apache 2.0
Fourth Party Dependency #2 Copyright : Copyright
_____
Fourth Party Dependency #3 Name : jackson-databind
Fourth Party Dependency #3 License : Apache 2.0
Fourth Party Dependency #3 Copyright : Copyright
_____
Fourth Party Dependency #4 Name : jakarta.activation-api
Fourth Party Dependency #4 License : EDL 1.0
Fourth Party Dependency #4 Copyright : Copyright
```

Copyright (c) 2007, Eclipse Foundation, Inc. and its licensors.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of the Eclipse Foundation, Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT

SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Fourth Party Dependency #5 Name: jakarta.xml.bind-api Fourth Party Dependency #5 License: EDL 1.0 Fourth Party Dependency #5 Copyright: Copyright

Copyright (c) 2007, Eclipse Foundation, Inc. and its licensors.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of the Eclipse Foundation, Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Guava

Vendor: Google

Version: 31.1

Copyright (C) 2020 The Guava Authors

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Apache License Version 2.0

Apache License



Version 2.0, January 2004 http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity



- on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.
- 2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.
- 3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.
- 4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:
 - (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and
 - (b) You must cause any modified files to carry prominent notices stating that You changed the files; and
 - (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
 - (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and

may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

- 5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.
- 6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.
- 7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.
- 8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
- 9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a



file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

+--- 4th party: com.google.guava:failureaccess

Copyright (C) 2018 The Guava Authors

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

< Apache License Version 2.0>

+--- 4th party: com.google.guava:listenablefuture

Copyright (C) 2018 The Guava Authors

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

< Apache License Version 2.0>

+--- 4th party: com.google.code.findbugs:jsr305

The JSR-305 reference implementation (lib/jsr305.jar) is

distributed under the terms of the New BSD license:

http://www.opensource.org/licenses/bsd-license.php

See the JSR-305 home page for more information:

http://code.google.com/p/jsr-305/

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- 3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

+--- 4th party: jcip-annotations

Copyright (c) 2005, Brian Goetz and Tim Peierls

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- 3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.



+--- 4th party: com.google.errorprone:error prone annotations

Copyright 2015 The Error Prone Authors.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

< Apache License Version 2.0>

+--- 4th party: com.google.j2objc:j2objc-annotations

Google Inc.
Daniel Connelly

Copyright 2012 Google Inc. All Rights Reserved.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

< Apache License Version 2.0>

+--- 4th party: org.checkerframework:checker-qual

Copyright 2004-present by the Checker Framework developers

MIT License:

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

+--- 4th party: org.codehaus.mojo:animal-sniffer-annotations

MIT License:

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Copyright (c) 2009 codehaus.org.

fuse.js

Vendor: Kirollos Risk

Version: 6.4.6

fuse.js v. 6.4.6
===== License =====
Apache License

Version 2.0, January 2004 http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the

outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

- 2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.
- 3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You



institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

- 4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:
 - (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and
 - (b) You must cause any modified files to carry prominent notices stating that You changed the files; and
 - (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
 - (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

- 5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.
- 6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.



- 7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.
- 8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
- 9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "{}" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright 2017 Kirollos Risk

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

===== Copyright ======

Copyright 2017 Kirollos Risk

MapLibre GL

Vendor: MapLibre

Version: 1.15.2

----- Top-level license -----

BSD 3-Clause

Copyright (c) 2020, Mapbox

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of Mapbox GL JS nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Contains code from glfx.js

Copyright (C) 2011 by Evan Wallace

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Contains a portion of d3-color https://github.com/d3/d3-color

Copyright 2010-2016 Mike Bostock All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of the author nor the names of contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Fourth-Party Dependencies

```
mapbox/geojson-rewind 0.5.0
ISC License
Copyright (c) 2020, Mapbox
-----(separator)------
mapbox/geojson-types 1.0.2
ISC License
Copyright (c) 2018 Mapbox
-----(separator)------
mapbox/point-geometry 0.1.0
ISC License
Copyright (c) 2015, Mapbox
-----(separator)------
mapbox/whoots-js 3.1.0
ISC License
Copyright (c) 2017, Mapbox
-----(separator)------
earcut 2.2.2
ISC License
Copyright (c) 2016, Mapbox
----- (separator)
geojson-vt 3.2.1
ISC License
Copyright (c) 2015, Mapbox
 -----(separator)------
```



```
grid-index 1.1.0
ISC License
Copyright (c) 2016, Mapbox
-----(separator)-----
potpack 1.0.1
ISC License
Copyright (c) 2018, Mapbox
-----(separator)------
Quickselect 2.0.0
ISC License
-----(separator)-----
supercluster 7.1.0
ISC License
Copyright (c) 2016, Mapbox
-----(separator)------
Tinyqueue 2.0.3
ISC License
Copyright (c) 2017, Vladimir Agafonkin
```

All Rights Reserved.

All Rights Reserved.

Permission to use, copy, modify, and/or distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

```
-----(separator)------
mapbox/jsonlint-lines-primitives
MIT License
Copyright (C) 2012 Zachary Carter
----- (separator)
csscolorparser 1.0.3
MIT License
(c) Dean McNamee dean@gmail.com, 2012
-----(separator)------
gl-matrix 3.3.0
MIT License
Copyright (c) 2015-2020, Brandon Jones, Colin MacKenzie IV
-----(separator)-----
minimist 1.2.5
MIT License
-----(separator)-----
murmurhash-js 1.0.0
MIT License
Copyright (c) 2011 Gary Court
-----(separator)-----
vt-pbf 3.1.1
MIT License
Copyright (c) 2015 Anand Thakker
```

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

-----(separator)------

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

-----(separator)------

mapbox/tiny-sdf 1.1.1
BSD 2-Clause License
Copyright © 2016-2017 Mapbox, Inc.

All Rights Reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

----- (separator)

mapbox/unitbezier 0.0.0

BSD-2-Clause

Copyright (C) 2008 Apple Inc. All Rights Reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY APPLE INC. ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL APPLE INC. OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Ported from Webkit

 $\label{lem:lem:matter} $$ $$ $ \text{http://svn.webkit.org/repository/webkit/trunk/Source/WebCore/platform/graphics/UnitBezier.h} $$$

----: (separator) -----:

rw 1.3.3 BSD 3 Clause Copyright (c) 2014-2016, Michael Bostock All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this

list of conditions and the following disclaimer.

- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * The name Michael Bostock may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL MICHAEL BOSTOCK BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

D3

Vendor: Michael Bostock

Version: 7.1.1

Copyright 2010-2020 Mike Bostock

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of the author nor the names of contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

FOURTH-PARTY DEPENDENCY (of d3): d3-array

Copyright 2010-2016 Mike Bostock

All rights reserved.

(see license under d3)

FOURTH-PARTY DEPENDENCY (of d3): d3-axis

Copyright 2010-2016 Mike Bostock

All rights reserved.

(see license under d3)

FOURTH-PARTY DEPENDENCY (of d3): d3-brush



_____ Copyright 2010-2016 Mike Bostock All rights reserved. (see license under d3) FOURTH-PARTY DEPENDENCY (of d3): d3-chord Copyright 2010-2016 Mike Bostock All rights reserved. (see license under d3) _____ FOURTH-PARTY DEPENDENCY (of d3): d3-color -----Copyright 2010-2016 Mike Bostock All rights reserved. (see license under d3) FOURTH-PARTY DEPENDENCY (of d3): d3-contour Copyright 2012-2017 Mike Bostock All rights reserved. (see license under d3) _____ FOURTH-PARTY DEPENDENCY (of d3): d3-delaunay -----Copyright 2018 Observable, Inc. Permission to use, copy, modify, and/or distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies. THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE. FOURTH-PARTY DEPENDENCY (of d3): d3-dispatch _____ Copyright 2010-2016 Mike Bostock All rights reserved. (see license under d3) _____ FOURTH-PARTY DEPENDENCY (of d3): d3-drag _____ Copyright 2010-2016 Mike Bostock All rights reserved. (see license under d3) FOURTH-PARTY DEPENDENCY (of d3): d3-dsv _____ Copyright 2013-2016 Mike Bostock All rights reserved. (see license under d3) _____ FOURTH-PARTY DEPENDENCY (of d3): d3-ease _____ Copyright 2010-2016 Mike Bostock Copyright 2001 Robert Penner All rights reserved. (see license under d3)



FOURTH-PARTY DEPENDENCY (of d3): d3-fetch

Copyright 2016 Mike Bostock All rights reserved.

FOURTH-PARTY DEPENDENCY (of d3): d3-force

Copyright 2010-2016 Mike Bostock

All rights reserved.

(see license under d3)

(see license under d3)

FOURTH-PARTY DEPENDENCY (of d3): d3-format

Copyright 2010-2015 Mike Bostock

All rights reserved.

(see license under d3)

FOURTH-PARTY DEPENDENCY (of d3): d3-geo

Copyright 2010-2016 Mike Bostock

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of the author nor the names of contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This license applies to GeographicLib, versions 1.12 and later.

Copyright (c) 2008-2012, Charles Karney

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

FOURTH-PARTY DEPENDENCY (of d3): d3-hierarchy



_____ Copyright 2010-2016 Mike Bostock All rights reserved. (see license under d3) FOURTH-PARTY DEPENDENCY (of d3): d3-interpolate Copyright 2010-2016 Mike Bostock All rights reserved. (see license under d3) _____ FOURTH-PARTY DEPENDENCY (of d3): d3-path -----Copyright 2015-2016 Mike Bostock All rights reserved. (see license under d3) FOURTH-PARTY DEPENDENCY (of d3): d3-polygon Copyright 2010-2016 Mike Bostock All rights reserved. (see license under d3) _____ FOURTH-PARTY DEPENDENCY (of d3): d3-quadtree _____ Copyright 2010-2016 Mike Bostock All rights reserved. (see license under d3) FOURTH-PARTY DEPENDENCY (of d3): d3-random _____ Copyright 2010-2016 Mike Bostock All rights reserved. (see license under d3) _____ FOURTH-PARTY DEPENDENCY (of d3): d3-scale _____ Copyright 2010-2015 Mike Bostock All rights reserved. (see license under d3) _____ FOURTH-PARTY DEPENDENCY (of d3): d3-scale-chromatic ______

Copyright 2010-2018 Mike Bostock

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of the author nor the names of contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;



LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Apache-Style Software License for ColorBrewer software and ColorBrewer Color Schemes

Copyright (c) 2002 Cynthia Brewer, Mark Harrower, and The Pennsylvania State University.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License. Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION 1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below). "Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf



- of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.
- 2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.
- 3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.
- 4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

- 5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.
- 6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.
- 7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-



INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

- 8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
- 9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

FOURTH-PARTY DEPENDENCY (of d3): d3-selection

Copyright (c) 2010-2018, Michael Bostock All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * The name Michael Bostock may not be used to endorse or promote products derived from this software without specific prior written permission.

 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL MICHAEL BOSTOCK BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

FOURTH-PARTY DEPENDENCY (of d3): d3-shape

Copyright 2010-2015 Mike Bostock

All rights reserved.

(see license under d3)

FOURTH-PARTY DEPENDENCY (of d3): d3-time

Copyright 2010-2016 Mike Bostock

All rights reserved.

(see license under d3)



FOURTH-PARTY DEPENDENCY (of d3): d3-time-format

Copyright 2010-2017 Mike Bostock

All rights reserved.

(see license under d3)

FOURTH-PARTY DEPENDENCY (of d3): d3-timer

Copyright 2010-2016 Mike Bostock

All rights reserved.

(see license under d3)

FOURTH-PARTY DEPENDENCY (of d3): d3-transition

Copyright (c) 2010-2015, Michael Bostock

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * The name Michael Bostock may not be used to endorse or promote products derived from this software without specific prior written permission.

 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"

 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL MICHAEL BOSTOCK BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

TERMS OF USE - EASING EQUATIONS

Open source under the BSD License.

Copyright 2001 Robert Penner

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the author nor the names of contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

FOURTH-PARTY DEPENDENCY (of d3): d3-zoom

Copyright 2010-2016 Mike Bostock All rights reserved.

(see license under d3)

FOURTH-PARTY DEPENDENCY (of d3): commander

(The MIT License)

Copyright (c) 2011 TJ Holowaychuk

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the 'Software'), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED 'AS IS', WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

FOURTH-PARTY DEPENDENCY (of d3): delaunator

TSC License

Copyright (c) 2017, Mapbox

Permission to use, copy, modify, and/or distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

FOURTH-PARTY DEPENDENCY (of d3): iconv-lite

Copyright (c) 2011 Alexander Shtuchkin

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.



FOURTH-PARTY DEPENDENCY (of d3): internmap

Copyright 2021 Mike Bostock

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of the author nor the names of contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

FOURTH-PARTY DEPENDENCY (of d3): rw

Copyright (c) 2014-2016, Michael Bostock

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * The name Michael Bostock may not be used to endorse or promote products derived from this software without specific prior written permission.

 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL MICHAEL BOSTOCK BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

FOURTH-PARTY DEPENDENCY (of d3): safer-buffer

MIT License

Copyright (c) 2018 Nikita Skovoroda

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.



THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Monaco Editor

Vendor: Microsoft Corporation

Version: 0.34.0

monaco-editor

https://github.com/microsoft/monaco-editor/blob/main/LICENSE.txt

The MIT License (MIT)

Copyright (c) 2016 - present Microsoft Corporation

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

OWASP Java Encoder Project

Vendor: Open Web Application Security Project (OWASP)

Version: 1.2.3

Copyright (c) 2015 Jeff Ichnowski All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.



* Neither the name of the OWASP nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

logback-core

Vendor: QOS.ch

Version: 1.2.10

Logback: the reliable, generic, fast and flexible logging framework.

* Copyright (C) 1999-2015, QOS.ch. All rights reserved.

*

- * This program and the accompanying materials are dual-licensed under
- * either the terms of the Eclipse Public License v1.0 as published by
- * the Eclipse Foundation

*

* or (per the licensee's choosing)

*

- * under the terms of the GNU Lesser General Public License version 2.1
- * as published by the Free Software Foundation.

Oracle elects the EPL v1. With respect to this module as provided to you by Oracle, any terms that may apply which differ from this Agreement are offered by Oracle alone and not by any other party

Eclipse Public License - v 1.0

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS ECLIPSE PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

1. DEFINITIONS

"Contribution" means:

- b) in the case of each subsequent Contributor:
- i) changes to the Program, and
- ii) additions to the Program;

where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution 'originates' from a Contributor

if it was added to the Program by such Contributor itself or anyone acting on such Contributor's behalf. Contributions do not include additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program.

"Contributor" means any person or entity that distributes the Program.

"Licensed Patents" mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

"Program" means the Contributions distributed in accordance with this Agreement.

"Recipient" means anyone who receives the Program under this Agreement, including all Contributors.

2. GRANT OF RIGHTS

a) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source

code and object code form.

b) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents

to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code form. This patent

license shall apply to the combination of the Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder.

c) Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to Recipient for

hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient's responsibility to acquire that license before distributing the Program.

- d) Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.
- 3. REQUIREMENTS

A Contributor may choose to distribute the Program in object code form under its



own license agreement, provided that:

- a) it complies with the terms and conditions of this Agreement; and
- b) its license agreement:
- i) effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;
- ii) effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;
- iii) states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and
- iv) states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

When the Program is made available in source code form:

- a) it must be made available under this Agreement; and
- b) a copy of this Agreement must be included with each copy of the Program.

Contributors may not remove or alter any copyright notices contained within the $\operatorname{Program}$.

Each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

4. COMMERCIAL DISTRIBUTION

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor's responsibility alone. Under this section, the Commercial Contributor would have to



defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

5. NO WARRANTY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED

INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely

responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement,

including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

6. DISCLAIMER OF LIABILITY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,

EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. GENERAL

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against any entity (including a crossclaim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2(b) shall

terminate as of the date such litigation is filed.

All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not

cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate, Recipient

agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this Agreement and any licenses

granted by Recipient relating to the Program shall continue and survive.

Everyone is permitted to copy and distribute copies of this Agreement, but in order to avoid inconsistency the Agreement is copyrighted and may only be modified

in the following manner. The Agreement Steward reserves the right to publish new versions (including revisions) of this Agreement from time to time.



No one other than the Agreement Steward has the right to modify this Agreement. The Eclipse Foundation is the initial Agreement Steward. The Eclipse Foundation may assign the responsibility to serve as the Agreement Steward to a suitable separate entity. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages--typically libraries--of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with



the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and

passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems

that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order

to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of

freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing nonfree programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the

Library

has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run. TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

- 2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:
 - a) The modified work must itself be a software library.
- b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.
- c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

other than as an argument passed when the facility is invoked, then you must



(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right

to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1

and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the $\frac{1}{2}$

same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a



derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work

(which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable

"work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable

containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be

able to recompile the application to use the modified definitions.)

b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library

already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified

version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

- c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for
 - a charge no more than the cost of performing this distribution.
- d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.
 - e) Verify that the user has already received a copy of these materials or that you



have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

- 7. You may place library facilities that are a work based on the Library side-byside in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:
- a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.
- b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.
- 8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.
- 9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.
- 10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.
- 11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions

imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to



refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices.

Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

- 12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.
- 13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

- 15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW.

 EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF
 ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE,
 YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
- 16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY



MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

slf4j-api, 1.7.32

Copyright (c) 2004-2007 QOS.ch All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so,

subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Commons IO

Vendor: The Apache Software Foundation

Version: 2.8.0

Apache Commons IO

Copyright 2002-2017 The Apache Software Foundation



This product includes software developed at
The Apache Software Foundation (http://www.apache.org/).

Apache License

Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the



Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

- 2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.
- 3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.
- 4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:
 - (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and
 - (b) You must cause any modified files to carry prominent notices stating that You changed the files; and
 - (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
 - (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside



or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

- 5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.
- 6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.
- 7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.
- 8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
- 9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.



To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Commons Lang

Vendor: The Apache Software Foundation

Version: 3.12.0

NOTICE:

Apache Commons Lang

Copyright 2001-2021 The Apache Software Foundation

This product includes software developed at The Apache Software Foundation (https://www.apache.org/).

LICENSE: Apache 2.0

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity



exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

- 2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.
- 3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct



or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

- 4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:
 - (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and
 - (b) You must cause any modified files to carry prominent notices stating that You changed the files; and
 - (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
 - (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

- 5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.
- 6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.
- Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS,



WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

- 8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
- 9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Tomcat

Vendor: The Apache Software Foundation



Version: 8.5.84

== NOTICE file

Apache Tomcat

Copyright 1999-2022 The Apache Software Foundation

This product includes software developed at
The Apache Software Foundation (https://www.apache.org/).

This software contains code derived from netty-native developed by the Netty project (https://netty.io, https://github.com/netty/netty-tcnative/) and from finagle-native developed at Twitter (https://github.com/twitter/finagle).

Java compilation software for JSP pages is provided by the Eclipse JDT Core Batch Compiler component, which is open source software. The original software and related information is available at https://www.eclipse.org/jdt/core/.

For portions of the Tomcat JNI OpenSSL API and the OpenSSL JSSE integration The org.apache.tomcat.jni and the org.apache.tomcat.net.openssl packages are derivative work originating from the Netty project and the finagle-native project developed at Twitter

- * Copyright 2014 The Netty Project
- * Copyright 2014 Twitter

== LICENCE File

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.



"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

- 2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.
- 3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.
- 4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without



modifications, and in Source or Object form, provided that You meet the following conditions:

- (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and
- (b) You must cause any modified files to carry prominent notices stating that You changed the files; and
- (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
- (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

- 5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.
- 6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.
- 7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.



- 8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
- 9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

APACHE TOMCAT SUBCOMPONENTS:

Apache Tomcat includes a number of subcomponents with separate copyright notices and license terms. Your use of these subcomponents is subject to the terms and conditions of the following licenses.

For the Eclipse JDT Core Batch Compiler (ecj-x.x.x.jar) component:



Eclipse Public License - v 1.0

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS ECLIPSE PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

1. DEFINITIONS

"Contribution" means:

- a) in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and
- b) in the case of each subsequent Contributor:
- i) changes to the Program, and
- ii) additions to the Program;

where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution 'originates' from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor's behalf. Contributions do not include additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program.

"Contributor" means any person or entity that distributes the Program.

"Licensed Patents" mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

"Program" means the Contributions distributed in accordance with this Agreement.

"Recipient" means anyone who receives the Program under this Agreement, including all Contributors.

2. GRANT OF RIGHTS

- a) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form.
- b) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code form. This patent license shall apply to the combination of the Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder.
- c) Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to



Recipient for claims brought by any other entity based on infringement of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient's responsibility to acquire that license before distributing the Program.

d) Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.

3. REQUIREMENTS

A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that:

- a) it complies with the terms and conditions of this Agreement; and
- b) its license agreement:
- i) effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;
- ii) effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;
- iii) states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and
- iv) states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

When the Program is made available in source code form:

- a) it must be made available under this Agreement; and
- b) a copy of this Agreement must be included with each copy of the Program.

Contributors may not remove or alter any copyright notices contained within the $\operatorname{Program}$.

Each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

4. COMMERCIAL DISTRIBUTION

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal



actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor's responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

5. NO WARRANTY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement , including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

6. DISCLAIMER OF LIABILITY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. GENERAL

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2(b) shall terminate as of the date such litigation is filed.

All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate,



Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

Everyone is permitted to copy and distribute copies of this Agreement, but in order to avoid inconsistency the Agreement is copyrighted and may only be modified in the following manner. The Agreement Steward reserves the right to publish new versions (including revisions) of this Agreement from time to time. No one other than the Agreement Steward has the right to modify this Agreement. The Eclipse Foundation is the initial Agreement Steward. The Eclipse Foundation may assign the responsibility to serve as the Agreement Steward to a suitable separate entity. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.



Index

Α

ANALYZE_PG procedure, A-169 automatic delta refresh, 14-55	F
CF procedure, A-171 CF_CLEANUP procedure, A-174 CF_PREP procedure, A-176 CLEAR_PG procedure, A-177	FIND_CC_MAPPING_BASED procedure, A-199 FIND_CLUSTERS_CLEANUP procedure, A-200 FIND_CLUSTERS_PREP procedure, A-201 FIND_SP procedure, A-203 FIND_SP_CLEANUP procedure, A-204 FIND_SP_PREP procedure, A-205
CLEAR_PG_INDICES procedure, A-178	G
CLONE_GRAPH procedure, A-178 collaborative filtering, A-171, A-174, A-176 connected components finding, A-199 COUNT_TRIANGLE function, A-179 COUNT_TRIANGLE_CLEANUP procedure,	geometries getting, A-206, A-207 getting from longitude and latitude, A-211 WKT, A-215, A-216 GET_BUILD_ID function, A-206 GET_GEOMETRY_FROM_V_COL function, A-206 GET_GEOMETRY_FROM_V_T_COLS function, A-207 GET_LATLONG_FROM_V_COL function, A-209, A-211 GET_LATLONG_FROM_V_T_COLS function, A-210 GET_LONG_LAT_GEOMETRY function, A-211 GET_LONGLAT_FROM_V_T_COLS function, A-212 GET_OPG_VERSION function, A-213
D	GET_SCN function, <i>A-214</i> GET_VERSION function, <i>A-215</i> GET_WKTGEOMETRY_FROM_V_COL function,
DROP_EDGES_TEXT_IDX procedure, A-193 DROP_PG procedure, A-193 DROP_PG_VIEW procedure, A-194 DROP_VERTICES_TEXT_IDX procedure, A-194	GET_WKTGEOMETRY_FROM_V_COLIUNCTION, A-215 GET_WKTGEOMETRY_FROM_V_T_COLS function, A-216 GRANT_ACCESS procedure, A-217 Graph server (PGX), 1
E	
edge table statistics exporting, A-197 importing, A-218 ESTIMATE_TRIANGLE_RENUM function, A-195 EXP_EDGE_TAB_STATS procedure, A-197	I IMP_EDGE_TAB_STATS procedure, A-218 IMP_VERTEX_TAB_STATS procedure, A-220

EXP_VERTEX_TAB_STATS procedure, A-198



0	OPG_APIS package (continued) PR_PREP, A-224
OPG_APIS package	PREPARE TEXT INDEX, A-225
ANALYZE_PG, A-169	reference information, A-168
CF, A-171	RENAME_PG, A-226
CF_CLEANUP, A-174	SPARSIFY_GRAPH, A-226
CF_PREP, <i>A-176</i>	SPARSIFY_GRAPH_CLEANUP, A-228
-	
CLEAR_PG, <i>A-177</i> CLEAR_PG_INDICES, <i>A-178</i>	SPARSIFY_GRAPH_PREP, A-229
	OPG_GRAPHOP package
CLONE_GRAPH, A-178	POPULATE_SKELETON_TAB, A-231
COUNT_TRIANGLE, A-179	reference information, A-230
COUNT_TRIANGLE_CLEANUP, A-180	
COUNT_TRIANGLE_PREP, A-181	P
COUNT_TRIANGLE_RENUM, A-183	
CREATE_EDGES_TEXT_IDX, A-184	page rank
CREATE_PG, A-185	calculating, A-221
CREATE_PG_SNAPSHOT_TAB, A-186	cleanup, A-223
CREATE_PG_TEXTIDX_TAB, A-188	preparing to find, A-224
CREATE_STAT_TABLE, A-189	PGQL (Property Graph Query Language), 11-1
CREATE_SUB_GRAPH, A-190	PGX (in-memory Graph server), 1
CREATE_VERTICES_TEXT_IDX, A-191	PgxML for Graphs, 15-1
DROP_EDGES_TEXT_IDX, A-193	DeepWalk Algorithm, 15-1
DROP_PG, <i>A-193</i>	Pg2vec Algorithm, 15-82
DROP_PG_VIEW, A-194	Supervised GraphWise Algorithm, 15-15
DROP_VERTICES_TEXT_IDX, A-194	Unsupervised GraphWise Algorithm, 15-40
ESTIMATE_TRIANGLE_RENUM, A-195	POPULATE_SKELETON_TAB procedure, A-231
EXP_EDGE_TAB_STATS, A-197	PR procedure, A-221
EXP_VERTEX_TAB_STATS, A-198	PR_CLEANUP procedure, A-223
FIND_CC_MAPPING_BASED, A-199	PR_PREP procedure, A-224
FIND_CLUSTERS_CLEANUP, A-200	PREPARE_TEXT_INDEX procedure, A-225
FIND_CLUSTERS_PREP, A-201	property graph
FIND_SP, <i>A-203</i>	cleanup after sparsifying, A-228
FIND_SP_CLEANUP, A-204	clearing (removing data from), A-177
FIND_SP_PREP, A-205	cloning, A-178
GET_BUILD_ID, A-206	collaborative filtering, A-171, A-174, A-176
GET_GEOMETRY_FROM_V_COL, A-206	creating, A-185
GET_GEOMETRY_FROM_V_T_COLS,	dropping, <i>A-193</i>
A-207	dropping view definition, A-194
GET_LATLONG_FROM_V_COL, A-209,	preparing to sparsify, A-229
A-211	removing text index metadata, A-178
GET_LATLONG_FROM_V_T_COLS, A-210	renaming, <i>A-226</i>
GET_LONG_LAT_GEOMETRY, A-211	sparsifying, A-226
GET_LONGLAT_FROM_V_T_COLS, A-212	property graph access privileges
GET_OPG_VERSION, A-213	grantnig, <i>A-217</i>
GET_SCN, <i>A-214</i>	Property Graph Query Language (PGQL), 11-1
GET_VERSION, A-215	property graph statistics table
GET_WKTGEOMETRY_FROM_V_COL,	creating, A-189
A-215	property graph support
GET_WKTGEOMETRY_FROM_V_T_COLS,	getting build ID, A-206
A-216	getting SCN, A-214
GRANT_ACCESS, A-217	getting version, A-213, A-215
IMP_EDGE_TAB_STATS, A-218	property graph views (PG Views), 1
IMP_VERTEX_TAB_STATS, A-220	
PR, <i>A-221</i> PR CLEANUR 4-223	
PR = PANTP A - 222	

RENAME PG procedure, A-226	subgraph creating, <i>A-190</i>
DELETE: Cancel a Query Execution, 17-7	<u> </u>
GET: Check a Query Completion, 17-7 GET: Get User, 17-6 GET: List Graphs, 17-2 GET: Retrieve a Query Result, 17-8 GET: Run a PGQL Query, 17-3 GET: Run a PGQL Query Asynchronously, 17-7 POST: Login, 17-1	text index on property graph edge table, A-184 on property graph edge table (dropping), A-193 on property graph vertex table, A-191 on property graph vertex table (dropping), A-194
POST: Login, 17-1 POST: Logout, 17-6	preparing, A-225
S S	text index table creating, A-188 triangles
shortest path cleanup, A-204 finding, A-203 preparing to find, A-205 skeleton table populating, A-231 snapshot table creating, A-186 SPARSIFY_GRAPH procedure, A-226 SPARSIFY_GRAPH_CLEANUP procedure, A-228 SPARSIFY_GRAPH_PREP procedure, A-229 SQL property graphs, 1 granting privileges, 4-11 statistics for property graph analyzing, A-169	cleanup after counting, <i>A-180</i> counting, <i>A-179</i> counting and renumbering vertices, <i>A-183</i> estimating the number, <i>A-195</i> preparing to count, <i>A-181</i>
	vertex cluster mappings preparing, A-200, A-201 vertex table statistics exporting, A-198 importing, A-220

