Oracle® Cloud

Using Graph Studio in Oracle Autonomous AI Database

F36880-48
October 2025

Oracle Cloud Using Graph Studio in Oracle Autonomous AI Database,

F36880-48

Primary Author: Lavanya Jayapalan

Contributors: Chuck Murray, Korbi Schmid, Jayant Sharma, Steve Serra, Melliyal Annamalai, Gabriela Montiel, Carol Palmer, Siva Ravada

Contents

6 Work with Graphs in Graph Studio

7 Work with Notebooks in Graph Studio

10 Interactive Graph Visualization in Oracle APEX Applications

11 Work with Jobs in Graph Studio

12 Manage the Compute Environment

Preface

This document describes how to use and manage Graph Studio in Autonomous AI Database and provides references to related documentation.

Audience

This document is intended for Oracle Cloud users who want to use and manage Graph Studio to load and query property graph and RDF graph data.

Related Documents

- Getting Started With Oracle Cloud
- *Oracle AI Database Graph Developer's Guide for Property Graph*
- Property Graph Visualization Developer's Guide and Reference
- *Oracle AI Database Graph Developer's Guide for RDF Graph*

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.

1

What's New in Graph Studio on Oracle Autonomous AI Database

Learn about the latest enhancements and features for the Graph Studio user interface on Oracle Autonomous AI Database. Also, provides information on the deprecated and desupported features.

Feature	Description
Added support for loading a subset of graph properties into memory.	You can load all the graph properties or select specific vertex and edge properties to load into the graph server memory. See Manage Property Graphs for more information.
Added support for querying SQL property graphs in Query Playground page.	You can query a SQL property graph in the SQL tab of the Query Playground page if you are using an Autonomous AI Database instance with Oracle AI Database 26ai. See Manage Property Graphs for more information.
Enhanced Graph Visualization Settings panel.	The Graph Visualization Settings panel provides a great user experience with a new Captions section for better labeling, and an updated Styles tab for enhanced customization. See Settings for Graph Visualization for more information.
Added support for jumping to a specific notebook paragraph.	You can directly jump to a specific paragraph inside a notebook. See Jump to a Paragraph for more information.
Added support for configuring notebook states inside a notebook.	When sharing a notebook, you can control the actions a user can perform in the notebook. See Notebook States for more information.
Added support for visual edge creation when using the property graph wizard for creating graphs.	Graph Studio allows visual creation of edges between two vertex tables through drag and drop action. See Add New Edges During Graph Creation for more information.
Added support for loading graphs into memory inside a notebook.	You can easily load (or unload) a graph into the graph server memory using the **Quickview** button inside a notebook. See Load Graphs Into Memory Using the Quickview Option for more information.
Added support for using OCI Vault secret credentials.	Graph Studio provides a secure way to access secret credentials stored in Oracle Cloud Infrastructure (OCI) Vault, in a Python notebook paragraph. See Use OCI Vault Secret Credentials for more information.
Added support for converting a PGQL property graph to SQL graph.	You can migrate a PGQL property graph to SQL property graph if you are using an Autonomous AI Database instance with Oracle AI Database 26ai. See Convert a PGQL Property Graph to SQL Property Graph for more information.
Added support for visualizing the result of a SQL graph query.	You can visualize the result of a SQL graph query if you are using an Autonomous AI Database instance with Oracle AI Database 26ai. See SQL Interpreter for more information.

Feature	Description
Enhanced and improved graph visualization interface.	The graph visualization panel in the notebook paragraphs is redesigned to provide a new look and feel to enhance user experience in visualizing graphs. However, if you wish to use the previous graph visualization interface, select **Preferences** from the username drop-down menu (on the top right) and disable the **Enable Oracle Graph Visualization Library** option.
Added support for creating RDF graphs with `.ttl` and `.trig` formats.	In addition to `.nt` (N-Triples) and `.nq` (N-Quads) RDF data formats, Graph Studio supports creation of RDF graphs by uploading RDF data files with `.ttl` (Turtle) or `.trig` (TriG) extensions. See Create an RDF Graph in Graph Studio for more information.
Added support for creating SQL Property Graphs.	The option to work with SQL property graphs is available only in Oracle AI Database 26ai. Therefore, you can create and query SQL property graphs in Graph Studio only if you are using an Autonomous AI Database instance with Oracle AI Database 26ai. See Create a Property Graph from Existing Relational Tables and SQL Interpreter for more information.
Added support for estimating the in-memory graph size.	Graph Studio computes the estimated in-memory graph size at the time of creating or editing a PGQL property graph. In addition, when you recompute the graph metadata on the Graphs page, the estimated in-memory graph size gets updated. See Create a Property Graph from Existing Relational Tables and Manage Property Graphs for more information.
Added support for sharing an RDF graph.	Graph Studio supports sharing of RDF graphs and RDF graph collections between different users. See Share an RDF Graph for more information.
Added support for creating a PGQL property graph from an RDF graph.	Graph Studio supports a new **Create PGQL Property Graph** option on an RDF graph. This option guides you through a workflow to create a PGQL property graph from an existing RDF graph. See Create a Property Graph from an RDF Graph for more information.
Added support for visualizing property graphs in APEX applications.	You can use the APEX Graph Visualization plug-in to visualize and interact with property graphs in an APEX application. See Interactive Graph Visualization in Oracle APEX Applications for more information.
Simplified workflow for creating property graphs.	The Graphs page in Graph Studio is enhanced to support the creation of property graphs using a new workflow, without using graph models. To support this new graph creation workflow: • Models page is removed in Graph Studio. Also, note the following: – You can access any existing graph that was created earlier using a model. – You cannot access the model for a graph anymore. • The property graph wizard guides you through the steps to create a property graph. See Create a Property Graph from Existing Relational Tables for more information. • You can also directly edit the graph. See Create a Property Graph by Editing an Existing Graph for more information.
Enhanced Graph Studio user interface	The Graph Studio user interface now supports the Redwood theme. The improved design is user-friendly and makes Graph Studio more intuitive and easier to use.

Desupported Features

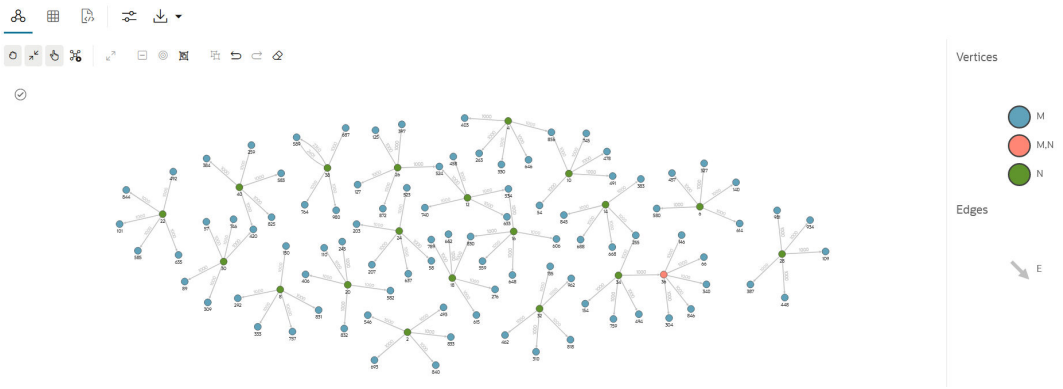
- The **PG Objects** graph type for property graphs is desupported. It is recommended that you create a **PGQL Property Graph** or **SQL Property Graph**. See Move PG Objects to PGQL or SQL Property Graph for more information.

2

Get Started Using Graphs

Graph Studio, a component of Oracle Autonomous AI Database, simplifies the task of developing applications that use graph analysis. The following features, in particular, support the development of high-performing, high-security applications:

- Automatic database administration. Routine database administration tasks such as patching and taking backups are performed automatically, so you can concentrate on developing your application.
- Automatic performance tuning. You spend less time defining and tuning your database.
- Predefined, workload-specific database services.
- Property graph data stored in Autonomous AI Database is fully accessible using Structured Query Language (SQL) and Property Graph Query Language (PGQL), for analytics and interfacing with relational tools.
- Semantic linked data (based on Resource Description Framework (RDF) stored in Autonomous AI Database can be queried using SPARQL Protocol and RDF Query Language (SPARQL).
- Interactive graph visualization. You can visualize the graph query results to find connections, patterns and dependencies within graph data.



Topics

- About Graph Data Support in Autonomous AI Database
- Typical Workflow for Using Graph Studio

About Graph Data Support in Autonomous AI Database

The Property graph and the RDF graph features of Oracle AI Database or earlier Oracle Database versions offer powerful graph support to explore and discover complex relationships in data sets. This applies to all databases in the cloud and on-premises environments.

Property Graph Support

Property graph support provides you a different way to look 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 banking scenario, 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 on dozens of graph analysis algorithms, like PageRank, to measure the relative importance of data entities based on the relationships between them, for example, links between web pages.

For more information about property graph support in your database, see Property Graph Support Overview in *Oracle AI Database Graph Developer's Guide for Property Graph*.

For a quick start on property graph features, see the topic Quick Starts for Using Oracle Property Graph.

RDF Graph Support

RDF graphs conform to a set of W3C (Worldwide Web Consortium) standards. The RDF graph support in database is well suited for knowledge graphs and data integration applications because URIs provide globally unique identifiers and the simple, schemaless triple structure makes it very easy to combine data from several different RDF graphs into a single graph.

You can query and analyze your RDF graph using SPARQL query language.

For more information about RDF graph support in database, see RDF Graph Overview in *Oracle AI Database Graph Developer's Guide for RDF Graph*.

For a quick start with RDF graph features, see the topic Quick Start for Using Semantic Data.

Typical Workflow for Using Graph Studio

A typical workflow with Graph Studio involves several operations.

More Information	Task	Description
Provision an Autonomous AI Database	Create an Autonomous AI Database from the Oracle Cloud Infrastructure Console	Create an Autonomous AI Database Serverless instance for one of the following workload types: • Data Warehouse • Transaction Processing
Create a Graph User	Create Graph Users for Graph Studio	Use Database Actions in Oracle Cloud Infrastructure Console to create and assign Graph users roles
Access the Graph Studio Application	Connect to your Autonomous AI Database using Graph Studio	Start and sign in to Graph Studio

3

Introduction to Graph Data in Autonomous AI Database

Oracle Autonomous AI Database contains features that enable it to function as a scalable graph database.

This chapter outlines the key terms, graph concepts, and the interactive Graph Studio for working with graphs in an Autonomous AI Database.

Topics

Overview of Graph Data in Autonomous AI Database

The graph features of Graph Studio automate the creation of property graphs and RDF graphs in Oracle Autonomous AI Database.

In-memory property graphs are designed using the property graph wizard on the Graphs page in Graph Studio. This feature automates the creation of property graph from relational database tables.

RDF graphs are created by importing RDF data stored in Oracle Autonomous AI Database into Graph Studio.

The features include notebooks and developer APIs for executing property graph queries using PGQL, over 60 built-in property graph algorithms, dozens of visualizations including native graph visualization, and executing RDF graph queries using SPARQL.

Key Terms and Concepts for Working with Graphs

This section briefly explains the key concepts of graphs and other graph features. These may be helpful when working with the interactive Graph Studio available in Autonomous AI Database.

Graph Studio

Graph Studio is a user interface available with Oracle Autonomous AI Database that provides access to all available graph features. You can:

- Create property graphs, execute PGQL queries, graph visualizations, and perform analytics.
- Create RDF graphs, execute SPARQL queries and perform graph visualizations.

Property Graph

A property graph consists of vertices that are linked together by edges. Both vertices and edges can have a set of properties attached to them. Common properties are id and label. The *label* property often identifies what the vertex or edge represent. For example, a vertex representing a bank account may have the label `Account`, while an edge representing a transfer of funds between accounts may have the label `Transfer`.

A property graph is the main data structure used with Graph Studio.

Property Graph Wizard

The property graph wizard in Graph Studio guides you through the steps to easily create a property graph from existing relational database tables.

This graph creation workflow comprises the following steps:

1. **Overview:** Provide the graph name and description.
2. **Select Tables:** Select the input tables.
3. **Define Graph:** View the graph definition and iteratively refine the mappings.
4. **Summary:** View the property graph summary and create the graph for analysis and visualization.

RDF

RDF (Resource Description Framework) is a W3C-standard data model for representing linked data. RDF uses Uniform Resource Identifiers (URIs) as globally-unique identifiers for resources and also uses URIs to identify the type of relationship between two resources. In addition to URIs, RDF uses literals to represent scalar values such as numbers, strings and timestamps.

RDF Graph

RDF models linked data as a directed, labeled RDF graph, where each edge is usually called a triple. The source vertex of the edge is called the subject of the triple. The label or name of the edge is called the predicate of the triple, and the destination vertex of the edge is called the object of the triple.

RDF Graph Collection

An RDF graph collection is an RDF graph that contains all triples from a collection of individual RDF graphs. The collection can also include entailed triples inferred by applying rules and ontologies to the graph collection.

Rule, Rulebase, and Inferencing

A rule is an object that can be applied to draw inferences from semantic data.

A rulebase is an object that contains rules.

Inferencing is the ability to make logical deductions based on rules.

Entailment

An entailment (rules index) is an object containing precomputed triples that can be inferred from applying a specified set of rule bases to a specified set of RDF graphs.

RDF N-Triple Format

N-Triple (`.nt`) is one of the common RDF data formats. Each statement in the file represents a triple:{subject or resource, predicate or property, object or value}.

RDF N-Quad Format

N-Quad (`.nq`) is another popular RDF data format. This format allows both regular triples and extended triples. An extended triple is made up of four components: {subject or resource, predicate or property, object or value, graph name}. The graph name component of an RDF triple must either be null or a URI.

RDF Turtle Format

The Turtle (`.ttl`) format defines a textual syntax for RDF graph.

RDF TriG Format

The TriG (`.trig`) format is a compact textual representation of RDF graph. It is an extension of the Turtle format.

RDF Wizard

The RDF wizard utility in Graph Studio guides you on the steps to create an RDF graph or RDF graph collection.

PGQL Graph Queries

PGQL (Property Graph Query Language) is a graph pattern-matching query language for property graphs. PGQL combines graph pattern matching with familiar constructs from SQL, such as SELECT, FROM, and WHERE. See Property Graph Query Language (PGQL) for more information on PGQL specifications.

SPARQL Queries

SPARQL Protocol and RDF Query Language (SPARQL) is one of the technologies standardized by the W3C for querying RDF data. See the W3C SPARQL 1.1 standard for more information.

Graph Algorithm

A graph algorithm is a pre-packaged set of instructions to traverse or analyze a graph. For example, you can find a shortest path or important vertices in your graph. *PageRank* is a well known graph algorithm, which ranks the importance of vertices. Graph Studio notebooks expose over 60 such algorithms as built-in functions.

Notebooks

Notebooks are interactive browser-based applications that enable data engineers, analysts, and scientists to be more productive by developing, organizing, executing, and sharing code, and by visualizing results without using the command line or needing to install anything. Notebooks enable you to execute code, to work interactively with long workflows, and to collaborate on projects.

In addition to code execution, notebooks support a large set of built-in visualization capabilities.

Job

A job is a potentially long-running asynchronous operation in Graph Studio. An example of a job is loading a graph into memory or creating a graph from tables.

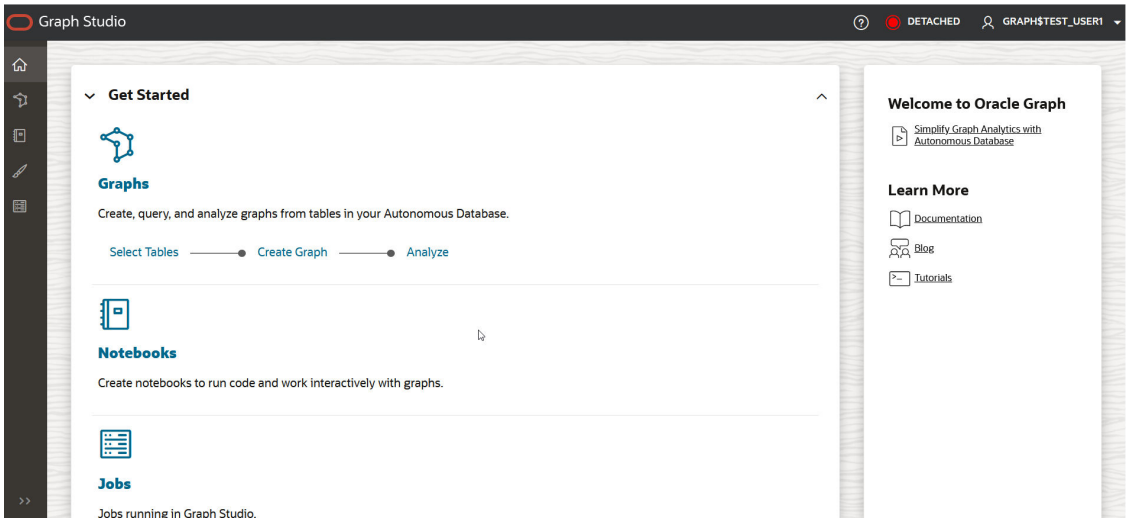
Graph Studio: Interactive, Self-Service User Interface

Graph Studio is the main user interface (UI) for creating, querying, analyzing, and visualizing graphs.

It includes notebooks and developer APIs where you can execute property graph queries using PGQL, RDF graph queries using SPARQL, and over 60 built-in graph algorithms. It also offers dozens of visualizations including native graph visualization.

The overall layout comprises of a left navigation panel that provides quick access to major actions, and the right side that displays content appropriate for the selected option on the left side menu.

The following figure shows the Graph Studio UI.



The navigation menu consists of:

- Overview
- Graphs
- Notebooks
- Templates
- Jobs

Overview

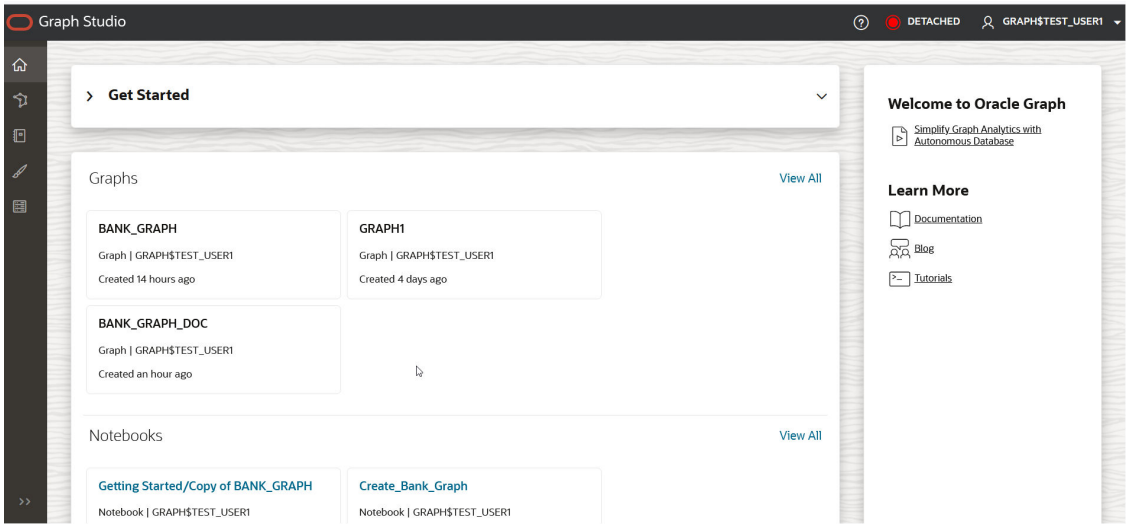
The Overview menu link directs you to the main or landing page. It consists of two sections:

The right section is a welcome panel which shows the following:

- A link to a video which describes how Graph Studio makes it easy to create and work with graphs in Autonomous AI Database.
- A **Learn More** section with links to documentation, blogs, and tutorials.

The middle section shows either of the following:

- A collapsible panel with links to **Graphs**, **Notebooks**, and **Jobs** pages for a first-time user as shown in the preceding figure.
- Cards listing existing graphs, notebooks, and jobs for a returning user with existing content as shown in the following screen .

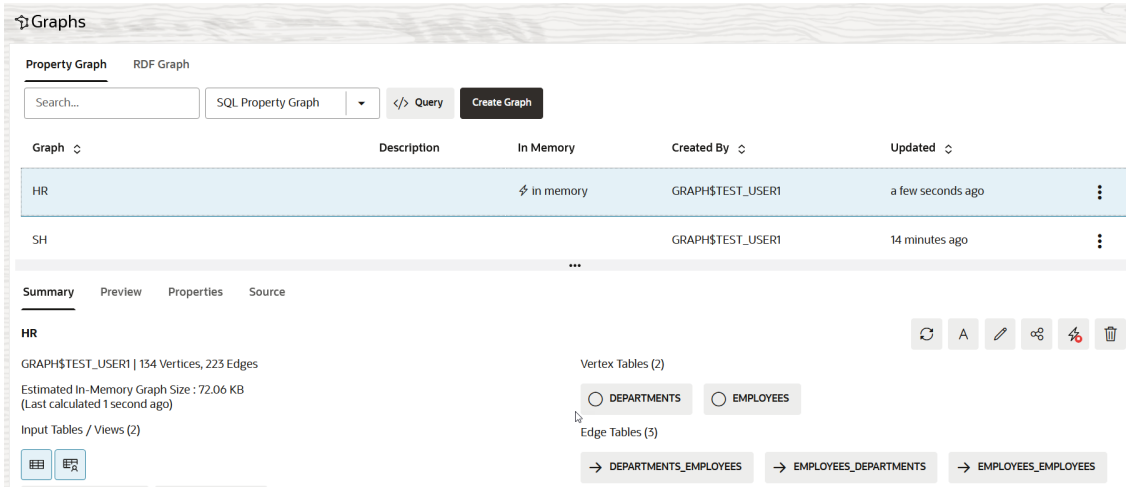


Graphs

The Graphs menu link directs you to the Graphs page which contains the following two tabs:

- Property Graph
- RDF Graph

All existing graphs corresponding to the selected graph type are listed on the Graphs page. Clicking on any graph displays the graph details in the bottom panel.



Depending on the graph, you can perform any of the following actions on this page:

- **Property Graph**
  - **Create** a new property graph.
  - **Query** an existing property graph using PGQL.
  - You can also use any of the following supported options in the graph details section:

    * Explore the **Summary** of the graph and optionally load the graph into memory, share the graph with other users, rename or delete the graph.
    * **Preview** the graph.
    * View the graph **Properties**.
    * View the graph **Source**.
- **RDF Graph**
  - **Create** a new RDF graph.
  - **Query** an existing RDF graph using SPARQL.
  - Explore the RDF graph properties (RDF statements) in the graph details section.

Query Playground

Clicking **</> Query** on the Graphs page will direct you to the Query Playground page. It serves as a notepad for entering and executing simple SQL or PGQL queries on a SQL or PGQL property graph, or SPARQL queries for an RDF graph. It is not meant for testing complex queries or for use in a production environment.

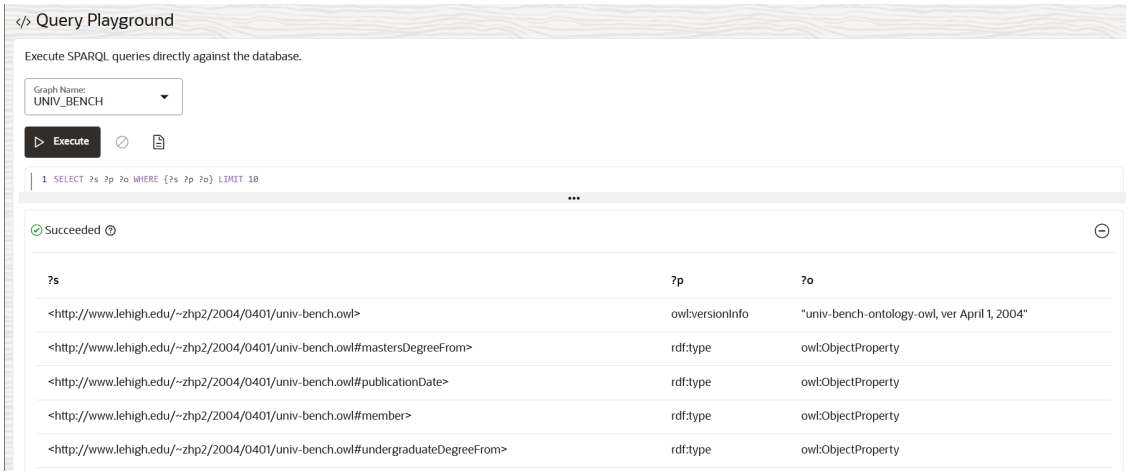
Queries submitted in the playground are executed directly against the graph stored in the Autonomous AI Database as shown:

This means you do not require Graph Studio to be attached to the internal compute environment or initially have the graph loaded into memory in case of property graphs.

The Query Playground page can comprise one or both of the following tabs depending on the database version used in your Autonomous AI Database instance:

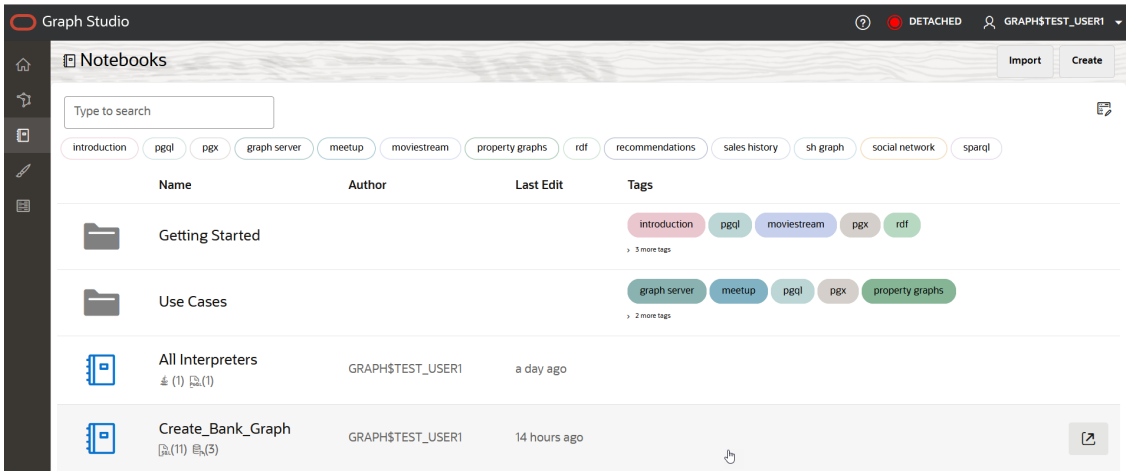
- **SQL**: This default tab is displayed only for Oracle AI Database 26ai. You can run SQL graph queries on SQL property graphs in this tab.
- **PGQL**: You can run PGQL queries on PGQL property graphs in this tab. This is the only tab for Oracle Database 19c.

In case of RDF graphs, the Query Playground interface allows you to select the RDF graph against which the SPARQL query is to be executed as shown:



Notebooks

The Notebooks menu link takes you to the Notebooks page that lists existing notebooks.

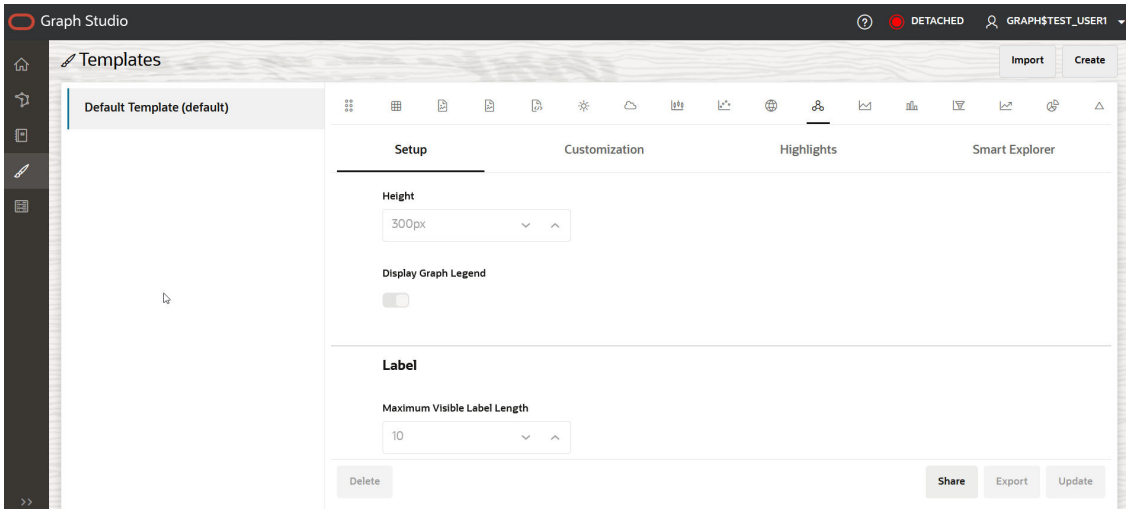


Templates

The Templates menu link directs you to the Templates page. This page consists of a left pane that lists all the existing templates. They are custom built templates with predefined graph visualization and notebook settings. Clicking on an existing template displays the custom data settings on the right pane. These template formats are applied to a notebook.

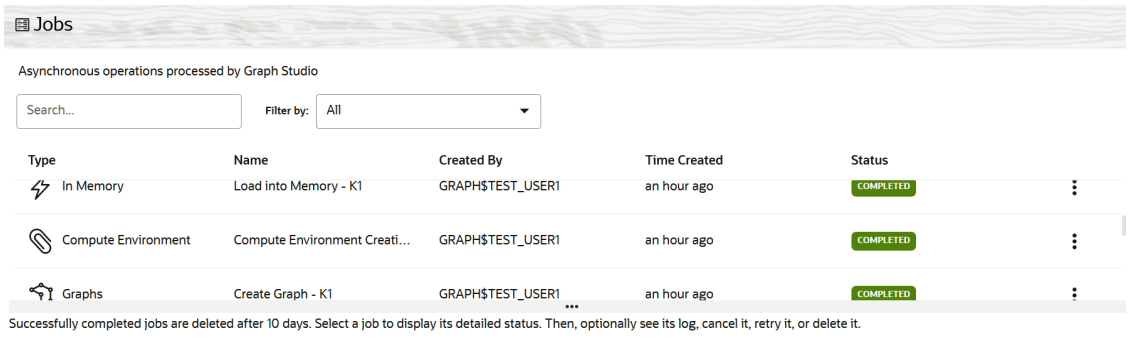
The page also contains the following buttons:

- **Create**: To build a new template
- **Update**: To update a template
- **Delete**: To delete a template
- **Share**: To share a template
- **Import**: To import a template
- **Export**: To export a template



Jobs

The Jobs menu link directs you to the Jobs page that lists previous and current jobs.



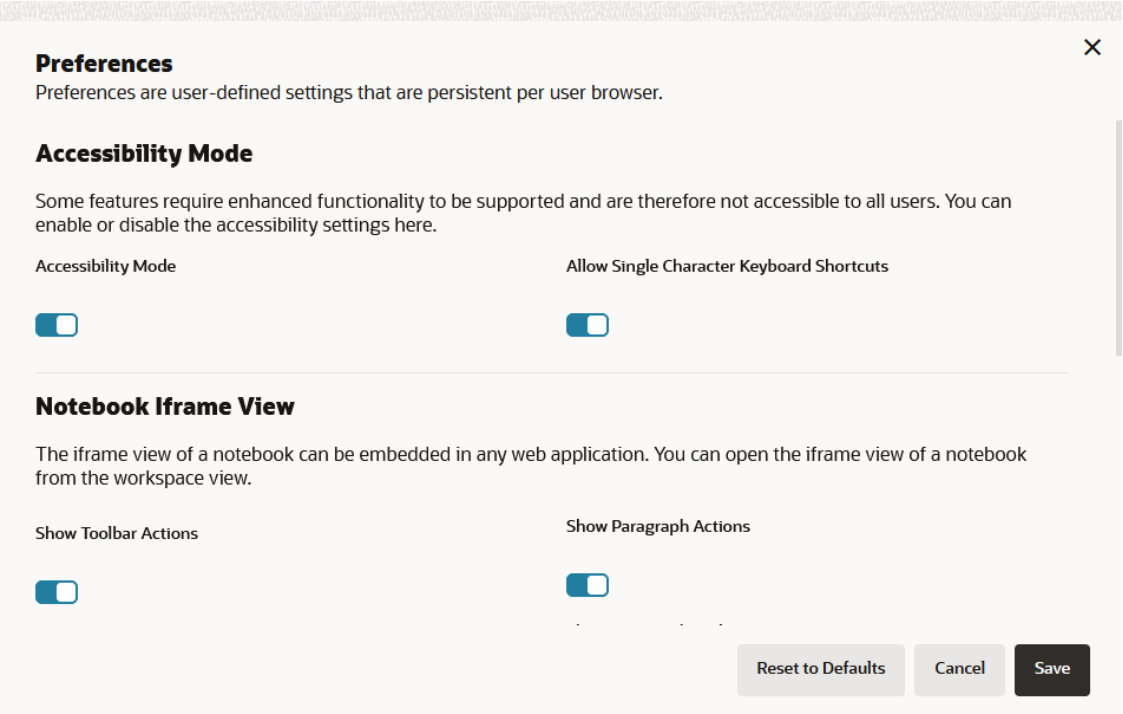
Use Accessibility Mode

You can turn on accessibility mode to allow the use of assistive technology, such as screen readers, to use the Graph Studio interface more effectively.

Some of the features of Graph Studio are not fully accessible. Based on your personal preference, you can turn on **Accessibility Mode** in Graph Studio.

To enable Accessibility Mode, click on your **username** in the top-right drop-down menu of your interface page and then select **Preferences**.

The default setting for Accessibility Mode is Off. To turn on Accessibility Mode, select **On**.



Tutorials and Other Resources

In addition to this user documentation, several tutorials and other resources are available to help you get started with the Graph Studio tool and to become proficient working with graph data.

This user documentation describes the Graph Studio and provides brief descriptions of its main features. It does not list all possible options, and the explanations are often brief. However, the user interface is clear and intuitive, and often provides hover-over context-sensitive help.

You can take two approaches (or a combination) to using this documentation and the available tutorials:

- Continue reading the documentation starting with the major topics such as Work with Jobs in Graph Studio, Work with Notebooks in Graph Studio, and Visualize and Interact with Graph Data in Graph Studio.
  Then try one or more of the tutorials.

or

- Try one or more of the tutorials and use this documentation as needed for explanations and reference.

> **Note**
>
> With both approaches, you are encouraged to first read the following topics for understanding:
>
> - Key Terms and Concepts for Working with Graphs
> - Graph Studio: Interactive, Self-Service User Interface

Tutorials for Working with Graph Data

The tutorials are all available on the Oracle LiveLabs platform. Enter *Graph Studio*, *Property Graph* or *RDF Graph* in the search box.

Other Resources for Working with Graph Data

Other resources include the following technical documentations:

- *Oracle AI Database Graph Developer's Guide for Property Graph*
- *Oracle AI Database Graph Developer's Guide for RDF Graph*

4

Create a Graph User

Working with Graphs in Graph Studio, requires users with granted roles.

You can create Graph users with the correct set of roles and privileges using Oracle Database Actions.

Before you begin:

- Sign in to the OCI console using your Oracle Cloud credentials and navigate to your Oracle Autonomous AI Database instance.
- Access Database Actions from the Oracle Cloud Infrastructure Console as the ADMIN user. See Access Database Actions as ADMIN for more information.

You can then perform the following steps to create a graph user:

1. Click **Database Users** in the **Launchpad** page under the **Administration** group.
2. Click **Create User** on the Database Users page, in the All Users area.
3. Enter **User Name** , **Password** and enter the password again to confirm the password.
4. Switch on the **Graph** toggle to create a graph-enabled user.

   The `GRAPH_DEVELOPER` role gets automatically assigned to the user.
5. Switch on the **Web Access** toggle to provide the new user access to Database Actions in Autonomous AI Database.

   > **Note**
   >
   > You must provide Web Access to the new graph user in order to perform any of the following Database Actions:
   >
   > - Run SQL statements or queries in the SQL worksheet
   > - Load and access data from local files
6. Enter your desired **Quota on tablespace DATA**.
7. Click **Create User**.

   This creates a new user.

   See Lab 1 of the LiveLabs workshop for an example.

5

Access the Graph Studio Application

You can access the Graph Studio application in Autonomous AI Database from the Oracle Cloud Infrastructure console or with Database Actions.

Additionally, you can access some of the Graph Studio features programmatically through a Java or Python code.

Topics

Access Graph Studio Using Oracle Cloud Infrastructure Console

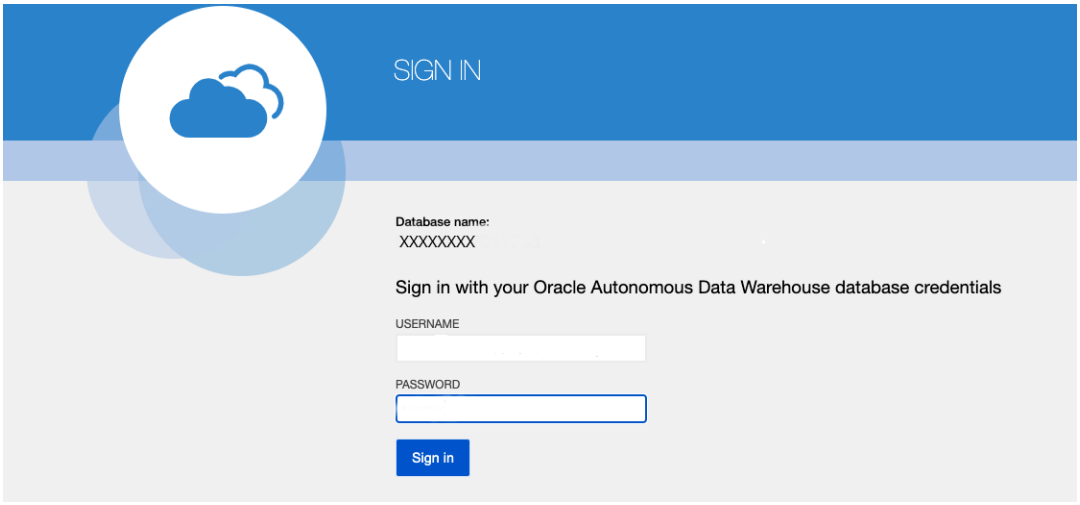
You can access the Graph Studio application from the Oracle Cloud Infrastructure Console as shown in the following steps:

1. Sign in to **Oracle Cloud**.
2. Select an Autonomous AI Database instance.

   This opens the Autonomous AI Database details page.
3. Click on **Tool configuration** tab.
4. Copy the **Public access URL** for **Graph Studio**.

   Graph Studio access URL gets copied to the clipboard.
5. Paste the URL in your browser to launch the Graph Studio application.

   The Graph Studio login screen opens as shown:



6. Enter your graph enabled **username** and **password** and then click **Sign In**.

You are now connected to Oracle Autonomous AI Database using Graph Studio.

Access Graph Studio Using Database Actions

You can access the Graph Studio application using Database Actions.

1. Sign in to **Oracle Cloud**.
2. Select an Autonomous AI Database instance and on the Autonomous AI Database page click **View all database actions** in the **Database actions** drop-down list.
3. Click **Graph Studio** in the **Development** tab.

   Graph Studio login screen opens in a new tab.
4. Enter your graph enabled **username** and **password** and then click **Sign In**.

   You are now connected to Oracle Autonomous AI Database using Graph Studio.

Access Graph Studio Features Using Autonomous AI Database Graph Client

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

This API provides the following capabilities:

- Authenticate with Autonomous AI Database
- Manage the Graph Studio environment
- Execute graph queries and algorithms against the graph server (PGX)
- Execute graph queries directly against the 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 AI Database Graph Client, ensure you meet all the prerequisite requirements explained in Prerequisites for Using Autonomous AI 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 PGQL property graph into memory, execute PGQL queries and run algorithms against the graph.

> **Note**
>
> See the Javadoc and Python API Reference for more information on `AdbGraphClient` API.

1. Start the interactive graph shell CLI and connect to your Autonomous AI Database instance with the `AdbGraphClient` using one of the following methods:

   Configuring the `AdbGraphClient` using Tenancy Details

- JShell
- Java
- Python

JShell

```
cd /opt/oracle/graph
./bin/opg4j --no_connect
For an introduction type: /help intro
Oracle Graph Server Shell 24.4.0
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.databaseOcid("<database_OCID>");
config.database("<DB_name>");
config.username("ADBDEV");
config.password("<password_for_ADBDEV>");
config.endpoint("https://<hostname-
prefix>.adb.<region>.oraclecloudapps.com/");

var client = new AdbGraphClient(config.build());
```

Python

```
cd /opt/oracle/graph
./bin/opg4py --no_connect
Oracle Graph Server Shell 24.4.0
>>> from opg4py.adb import AdbClient
>>> config = {
...          'tenancy_ocid': '<tenancy_OCID>',
...          'database': '<DB_name>',
...          'database_ocid': '<DB_OCID>',
...          'username': 'ADBDEV',
...          'password': '<password_for_ADBDEV>',
...          'endpoint': 'https://<hostname-
```

```
prefix>.adb.<region>.oraclecloudapps.com/'
... }
>>> client = AdbClient(config)
```

Configuring the `AdbGraphClient` using JDBC Connection

You can also configure the `AdbGraphClient` to use a JDBC connection to connect to your Autonomous AI Database instance (as shown in the following code). See Connect with JDBC Thin Driver in *Using Oracle Autonomous AI Database Serverless* on how to obtain the JDBC URL to connect to the Autonomous AI Database.

However, ensure that you have `READ` access to the `v$pdbs` view in your Autonomous AI Database instance. By default, the ADMIN user has `READ` access to the `v$pdbs` view. For all other users (non-administrator users), the `READ` access can be granted by the ADMIN (`GRANT SELECT ON v$pdbs TO <user>`).

- JShell
- Java
- Python

JShell

```
import oracle.pg.rdbms.*
opg4j> var conn = DriverManager.getConnection(<jdbcUrl>, <username>,
<password>)
opg4j> var config = AdbGraphClientConfiguration.fromConnection(conn,
<password>)
opg4j> var client = new AdbGraphClient(config)
```

Java

```
import oracle.pg.rdbms.*;
AdbGraphClientConfiguration config =
AdbGraphClientConfiguration.fromCredentials(<jdbcUrl>, <username>,
<password>);
AdbGraphClient client = new AdbGraphClient(config);
```

Python

```
>>> from opg4py.adb import AdbClient
>>> client = AdbClient.from_connection(<jdbcUrl>, <username>, <password>)
```

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
```

Python

```
>>> 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");
```

Python

```
>>> instance = client.get_pgx_instance()
>>> session = instance.create_session("adb-session")
```

4. Load a PGQL property graph from your Autonomous AI Database instance into memory.

- JShell
- Java
- Python

JShell

```
opg4j> var graph = session.readGraphByName("BANK_GRAPH",
GraphSource.PG_PGQL)
graph ==> PgxGraph[name=BANK_GRAPH,N=1000,E=5001,created=1647800790654]
```

Java

```
PgxGraph graph = session.readGraphByName("BANK_GRAPH",
GraphSource.PG_PGQL);
```

Python

```
>>> graph = session.read_graph_by_name("BANK_GRAPH", "pg_pgql")
```

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);
```

Python

```
>>> 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:

```
+-----------------------------------------------------+
| source | pagerank             | amount | destination |
+-----------------------------------------------------+
| 387    | 0.007302836252205922 | 1000.0 | 188         |
| 387    | 0.007302836252205922 | 1000.0 | 374         |
| 387    | 0.007302836252205922 | 1000.0 | 577         |
+-----------------------------------------------------+
```

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.pg.rdbms.GraphServer;
….
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 AI Database Graph Client

As a prerequisite requirement to get started with the `AdbGraphClient` API, you must:

- Provision an Autonomous AI Database instance in Oracle Autonomous AI Database.
- Obtain the following information if you are configuring the `AdbGraphClient` using the tenancy details. Otherwise, skip this step.

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.
database	Database name of your Autonomous AI Database instance	1. Open the OCI console and click **Oracle Database** in the left navigation menu. 2. Click **Autonomous AI Database**. 3. Select the required Autonomous AI Database under the **Display Name** column. 4. Note the **Database Name** under "General Information" in the **Autonomous AI Database Information** tab.
database OCID	The Oracle Cloud ID (OCID) of your Autonomous AI Database	1. Open the OCI console and click **Oracle Database** in the left navigation menu. 2. Click **Autonomous AI Database**. 3. Select the required Autonomous AI Database under the **Display Name** column. 4. Note the **Database OCID** under "General Information" in the **Autonomous AI Database Information** tab.
username	Graph enabled Autonomous AI Database username, used for logging into Graph Studio	See Create a Graph User for more information.
password	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.

Key	Description	More Information
endpoint	Graph Studio endpoint URL	1. Select your Autonomous AI Database instance and navigate to the Autonomous AI Database 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`

- Access Graph Studio and create a PGQL property graph.
- Download, install and start the Oracle Graph Java or Python client.

Using the PGX JDBC Driver with the AdbGraphClient API

Starting from Graph Server and Client Release 24.1.0, you can use the PGX JDBC driver with the `AdbGraphClient` API to query graphs stored in the memory of the graph server in Graph Studio on Autonomous AI Database.

To use the PGX JDBC driver to connect to your Autonomous AI Database instance, note the following:

- Register the PGX JDBC driver with the DriverManager:

```
import java.sql.DriverManager;
import oracle.pgx.jdbc.PgxJdbcDriver;
...
DriverManager.registerDriver(new PgxJdbcDriver());
```

- Use one of the following two ways to establish the connection using the PGX JDBC Driver:
  - **Using Properties**

```
properties = new Properties();
properties.put("tenancy_ocid", "<tenancy_OCID>");
properties.put("database_ocid", "<database_OCID>");
properties.put("database", "<database_name>");
properties.put("username", "<username>");
properties.put("password", "<password>");
Connection connection =
DriverManager.getConnection("jdbc:oracle:pgx:https://<hostname-
prefix>.adb.<region>.oraclecloudapps.com", properties);
```

  - **Using a Wallet**

```
Connection connection =
DriverManager.getConnection("jdbc:oracle:pgx:@<db_TNS_name>?
TNS_ADMIN=<path_to_wallet>", "<ADB_username>", "<ADB_password>")
```

Note that the JDBC URL in the preceding code samples, use `jdbc:oracle:pgx:` as the prefix.

Example 5-1 Using the PGX JDBC Driver to run graph queries in Autonomous AI Database

The following example establishes a connection using the PGX JDBC driver to connect to an Autonomous AI Database instance, starts the compute environment in Graph Studio, loads a graph into the graph server (PGX), creates a statement, and runs a PGQL query on the graph.

```
import java.sql.*;
import oracle.pgx.jdbc.*;
import oracle.pg.rdbms.*;
import oracle.pgx.api.*;

public class AdbPgxJdbc {

  public static void main(String[] args) throws Exception {

    DriverManager.registerDriver(new PgxJdbcDriver());


    try (Connection conn =
DriverManager.getConnection("jdbc:oracle:pgx:@<db_TNS_name>?
TNS_ADMIN=<path_to_wallet>","ADB_username","<ADB_password>")) {
      AdbGraphClient client = conn.unwrap(AdbGraphClient.class);
      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());
        }
      PgxSession session = conn.unwrap(PgxSession.class);
      PgxGraph graph = session.readGraphByName("BANK_PGQL_GRAPH",
GraphSource.PG_PGQL);
      Statement stmt = conn.createStatement();
      ResultSet rs = stmt.executeQuery("SELECT * "+
                                       "FROM GRAPH_TABLE ( BANK_PGQL_GRAPH
"+
                                       "MATCH (a IS ACCOUNTS) -[e IS
TRANSFERS]-> (b IS ACCOUNTS) "+
                                       "WHERE a.ID = 179 AND b.ID = 688 "+
                                       "COLUMNS (e.AMOUNT AS AMOUNT ))");
      while(rs.next()){
        System.out.println("AMOUNT = " + rs.getLong("AMOUNT"));
      }

    }
  }
}
```

The resulting output of the preceding code is as shown:

```
AMOUNT = 7562
```

6

Work with Graphs in Graph Studio

Graph Studio allows you to work with the two popular graph models, property graphs and RDF graphs.

You can easily create and manage either of the graph models. You can validate the graphs by executing queries and exploring their properties.

Topics:

Create a Graph

Graph Studio provides you with an intuitive user interface that enables you to create a graph easily.

You can create both property graphs and RDF graphs using Graph Studio.

Topics:

Create a Property Graph in Graph Studio

There are several ways to create a property graph using Graph Studio in your Autonomous AI Database instance.

Topics:

Create a Property Graph from Scratch

You can create graphs from scratch by using the `CREATE PROPERTY GRAPH` PGQL statement in the **Query Playground** page.

To create a graph from scratch:

1. Click **Graphs** on the left navigation menu and navigate to the Graphs page.
2. Click **</> Query** in the **Property Graph** tab and navigate to the Query Playground page.

3. Enter the **`CREATE PROPERTY GRAPH`** PGQL statement to create a PGQL graph. For example:

```
CREATE PROPERTY GRAPH BANK_GRAPH
  VERTEX TABLES (
    bank_accounts
      KEY ( id )
      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 ( amount, description, from_acct_id,
to_acct_id, txn_id )
  )OPTIONS (PG_PGQL)
```

4. Click **Run**.

   This creates a graph of type **PGQL Property Graph**. It is essentially a property graph view over data that is stored in the relational database tables.

Create a Property Graph from Existing Relational Tables

You can create a property graph from existing relational tables.

> **Note**
>
> - The **PG Objects** graph type is desupported. It is recommended that you create a **PGQL Property Graph** or **SQL Property Graph**.
> - SQL property graphs are supported only in Oracle AI Database 26ai. Therefore, if you are using an Autonomous AI Database instance with Oracle AI Database 26ai, then you have the option to create SQL property graphs.

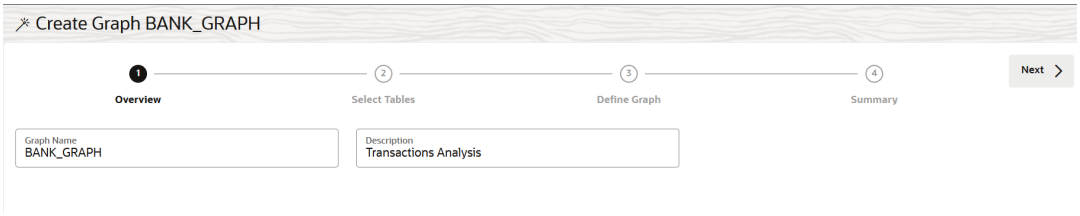
It is also important to note that when creating a graph, the property graph wizard will throw a warning if the source tables used to create a PGQL or SQL property graph include any Datetime data types in the primary key. Additionally, a warning will be issued if the tables contain composite vertex keys in the case of PGQL property graphs.

However, you can still create a property graph by ignoring the warning. But you cannot load the property graph into memory.

To create a property graph from existing relational tables:

1. Navigate to the Graphs page.
2. Select the **Property Graph** tab and click **Create Graph**.

   The property graph wizard opens displaying the **Overview** page.
3. Enter the **Graph Name**

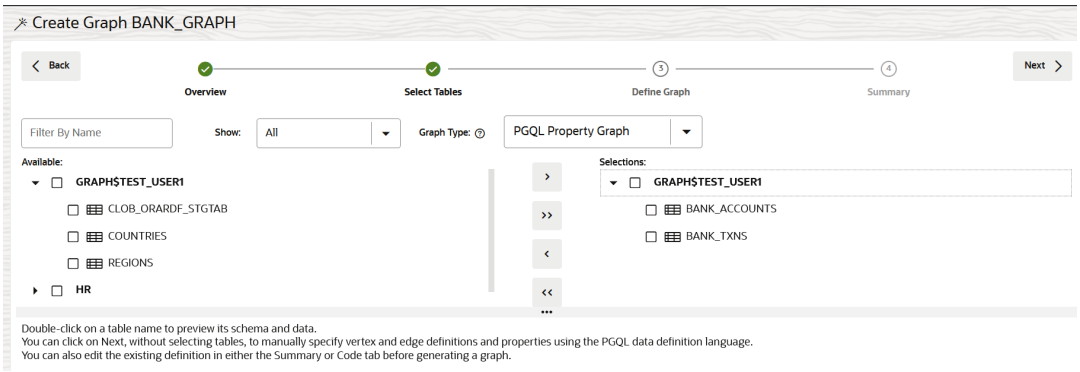


Note that the graph name is not case sensitive and is normalized to uppercase by default in Graph Studio. However, you can enable case sensitivity through the **Preserve Case** toggle as explained in step 11.

4. Optionally enter the graph **Description** and click **Next**.
5. Select the required **Graph Type**.

   Graph Studio supports the creation of two types of property graphs:

   - **SQL Property Graph:** The option to create a SQL property graph is available only if you are using an Autonomous AI Database instance with Oracle AI Database 26ai.
   - **PGQL Property Graph:** The option to create a PGQL property graph is available on all types of tenancies and supported on all database versions.
6. Select the data tables that are required as input for the graph and move them to the **Selections** section on the right.

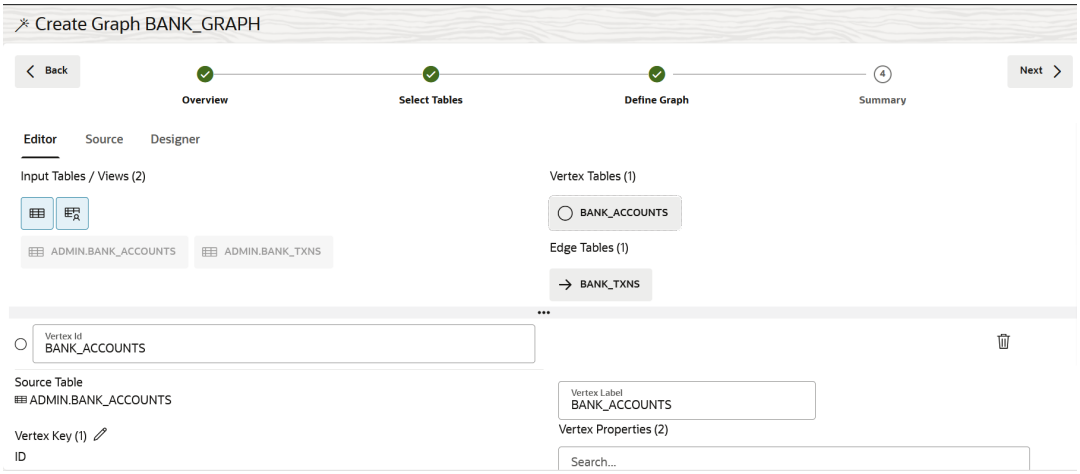


The following table shows a few supported input types and the corresponding default mapping when transformed into graph properties:

Oracle Data Type[1]	Oracle PGX Type
NUMBER	The following implicit type conversion rules apply: • NUMBER => LONG (for key columns) • NUMBER => DOUBLE (for non-key columns) • NUMBER(m) (number having precision m) with *m* <= *9* => INTEGER • NUMBER(m) (number having precision m) with *9* < *m* <= *18* => LONG • NUMBER(m,n) (number having precision m and scale n) => DOUBLE Note that this applies if n > 0. Otherwise, it follows the same mapping as NUMBER(x), where x = m-n (that is, subtracting the scale from the precision). The PGX type can then vary, depending on the x value as shown: – *x* <= *9* => INTEGER – *9* < *x* <= *18* => LONG – *x* > *18* => DOUBLE For instance, consider a scenario where n = -100 and m = 1. In this case, x = 101 (m-n), which is greater than 18. Extremely large numbers cannot be encoded to fit in INTEGER or LONG and therefore require the DOUBLE data type.
CHAR or NCHAR	STRING
VARCHAR, VARCHAR2, or NVARCHAR2	STRING
BINARY_FLOAT	FLOAT
BINARY_DOUBLE	DOUBLE
FLOAT	The following implicit type conversion rules apply: • FLOAT(m) with *m* <= *23* => FLOAT • FLOAT(m) with *23* < *m* => DOUBLE In the preceding entries, *m* is the variable for precision.
CLOB	STRING
DATE or TIMESTAMP	TIMESTAMP
TIMESTAMP WITH LOCAL TIME ZONE	TIMESTAMP
TIMESTAMP WITH TIME ZONE	TIMESTAMP WITH TIME ZONE

[1] Data types for **PGQL property graphs** and **SQL property graphs** share a one-to-one mapping with Oracle data types.

7. Click **Next** to view the suggested graph definition.

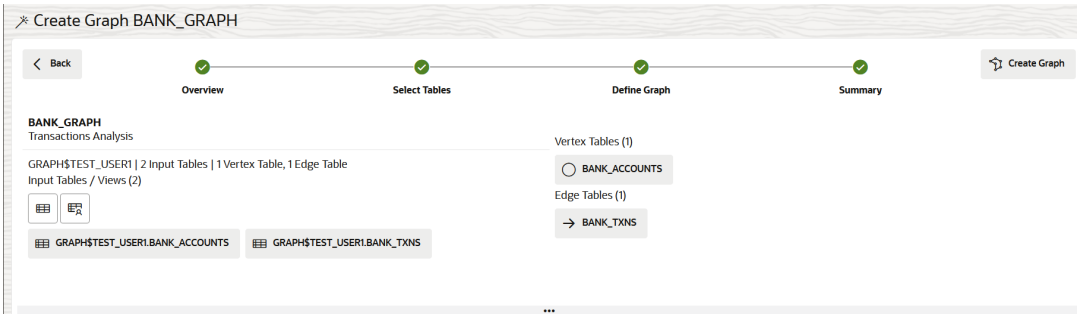


You can modify the graph definition if required.

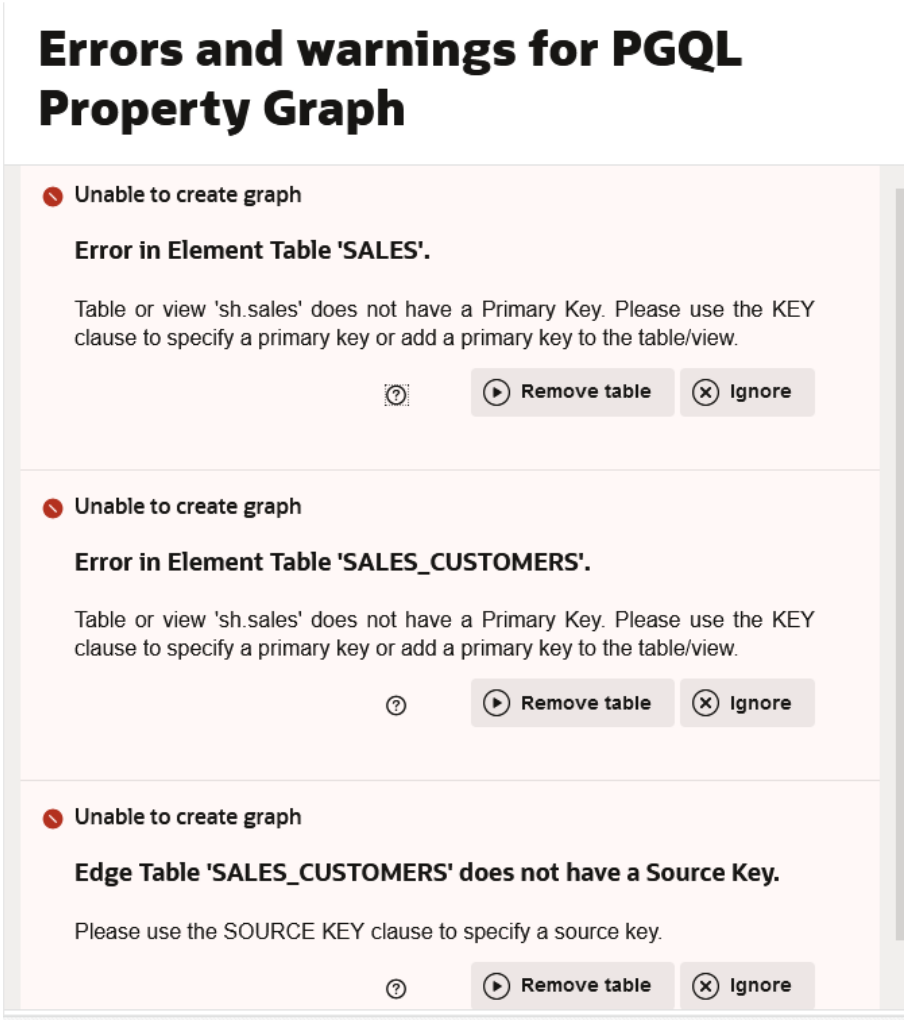
> **Note**
>
> Verify if the vertex and edge table keys are defined for the graph. These keys are generated automatically by the property graph wizard. In case the wizard is unable to generate the vertex and edge table keys, then you must manually specify these keys. Otherwise, the wizard will not proceed to the next step of graph creation. See Specify Vertex and Edge Table Keys for more information on how to add or edit the vertex and edge table keys.

See Add New Edges During Graph Creation for visually adding new edges between two vertices inside the **Designer** tab at this step.

8. Click **Next** to view the graph summary.



Graph Studio evaluates the graph definition and displays a summary of the graph if the validation is successful.
Otherwise it may report errors, warnings, or both:



The errors and warnings may vary depending on the graph type. Also, note the following:

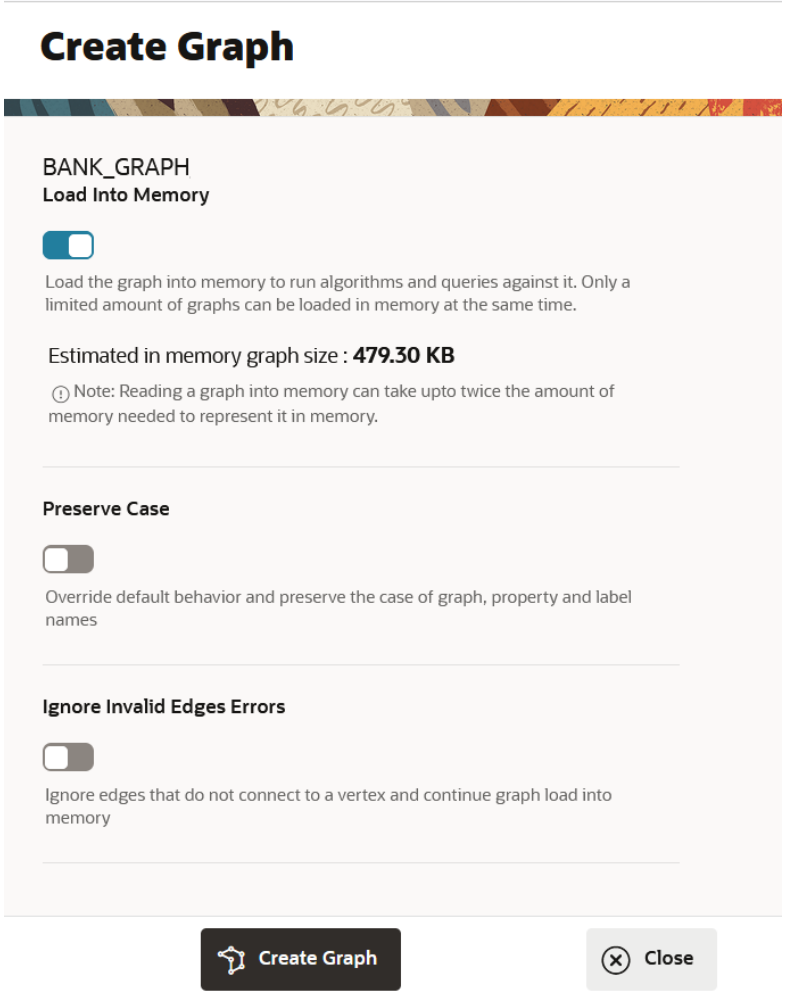
- **Errors:** Errors appear at the beginning in the **Errors and Warnings** slider. You need to resolve the errors in order to create a graph.
- **Warnings:** Warnings are reported following the errors. Graph Studio allows you to create a graph despite the warnings, but the graph cannot be loaded into memory. See Warnings During Property Graph Creation for more information on the warnings details when creating a property graph.

You can choose one of the following actions provided on the error or warning message:

- **Remove Column:** Removes the column from the vertex or edge table and the graph definition is updated and re-validated.
- **Remove Table:** Removes the vertex or edge table and the graph definition is updated and re-validated.
- **Ignore:** Dismisses the error or warning message. Ignoring a warning allows you to continue to the next step of creating a graph. However, ignoring an error does not allow you to proceed with the graph creation. If all the reported errors and warnings are ignored, the **Errors and Warnings** slider is automatically closed.
- **Remove All:** Removes all the tables and columns that cause errors or warnings and the graph definition is updated and re-validated.
- **Ignore All:** Closes the **Errors and Warnings** slider.

9. Click **Create Graph**.

   This opens the **Create Graph** slider as shown:



10. Optionally, switch on or off the **Load Into Memory** toggle.

    By default, the **Load Into Memory** toggle is disabled. If you had ignored any warnings reported on the graph definition, then the toggle remains disabled as the graph cannot be loaded into memory.
    The **Estimated in memory graph size** is also computed and displayed in the slider. Also, note the following with respect to the status of the compute environment:

    - **Detached:**
      - If the estimated graph size is less than the graph server (PGX) memory that is configured in the compute environment settings, then this new estimated value will be automatically saved as the default memory preference for the graph server (PGX). In this case, the slider will additionally display the following message:
        `This value will be saved as memory preference when compute environment is started.`
      - If the estimated graph size is greater than the maximum memory allowed to be allocated to the graph server (PGX) in the compute environment settings, then the following warning will be displayed in the slider:
        `A graph of this size will likely result in OutOfMemory errors during loading or analysis. Consider loading a subgraph instead.`

- **Attached:** If the estimated graph size is greater than the graph server (PGX) memory available for allocation in the compute environment settings, then the following warning will be displayed in the slider:
  `A graph of this size will likely result in OutOfMemory errors during loading or analysis. Consider loading a subgraph instead.`

11. Optionally, switch on or off the **Preserve Case** toggle.

    By default, this toggle is switched off. Enable the **Preserve Case** toggle if you wish to preserve the case for graph, property, and label names. In such a case, ensure to enclose the preserved names in quotes when referencing them later in SQL graph queries in Notebooks.

12. Optionally, switch on or off the **Ignore Invalid Edges Errors** toggle.

    The **Ignore Invalid Edges Errors** toggle determines the behavior for handling edges with missing source or destination vertices when the graph is loaded into memory. When set:

    - **ON:** It specifies that Graph Studio will ignore those edges with missing source or destination vertices.
    - **OFF:** This is the default option. It allows you to create the graph when there are edges with missing source or destination vertices. But Graph Studio throws an error when you attempt to load the graph into memory. However, you can reload the graph into memory from the Graphs page by switching on the **Ignore Invalid Edges Error** toggle. See Load Graph Into Memory for more information.

13. Click **Create Graph** to create the property graph.

Warnings During Property Graph Creation

When creating a property graph, the property graph wizard validates the designed graph and reports any validation errors or warnings.

You can still create a property graph by ignoring these warnings. However, in most cases, the graph cannot be loaded into the in-memory graph server (PGX).

The following table lists the warning messages that are generated during property graph creation.

Warning Message	Reason	How to Fix
`Vertex table <table_name> has composite key (<composite_key>) which will prevent this graph from being loaded into memory. See the documentation for more details`	Vertex table cannot have a composite key. Note that this applies only for PGQL property graphs.	If you **do not** plan to load this graph into the in-memory graph server (PGX) for analysis, you can ignore this warning. Otherwise, to fix this warning, choose one of the following options: • If you are using Oracle AI Database 26ai, then create a SQL property graph instead of a PGQL property graph. If the graph already exists, Graph Studio can help you to automatically convert it to a SQL Property graph. See Convert a PGQL Property Graph to SQL Property Graph for more information. • If you are using Oracle Database 19c, then first upgrade it to Oracle AI Database 26ai. You can then convert the PGQL property graph into a SQL property graph (see Convert a PGQL Property Graph to SQL Property Graph). • If SQL property graph is not an option, modify the vertex table which has composite keys to have a new column with unique IDs and declare that column as single primary key. This can be easily implemented using an `IDENTITY` column in the vertex table. Subsequently, you must alter all the edge tables pointing to this vertex to have an additional column that points to the new `IDENTITY` column. This can be achieved by first adding the column to the edge table and then having it filled through an `UPDATE` statement.

Warning Message	Reason	How to Fix
`Key column <column_name> of vertex table <table_name> has a data type which will prevent this graph from being loaded into memory. See the documentation for details and a list of supported datatypes`	Vertex table key must be one of the following types: VARCHAR, VARCHAR2, NVARCHAR2, CHAR, NCHAR, NUMBER	If you **do not** plan to load this graph into the in-memory graph server (PGX) for analysis, you can ignore this warning. Otherwise, to fix this warning, you can either: • Change the data type of the key column to be one of the supported types. • Alter the vertex table to add a new column with unique values of one of the supported types and declare it as the new key column.
`Key column <column_name> of edge table <table_name> has a data type which will prevent this graph from being loaded into memory. See the documentation for details and a list of supported datatypes`	Edge table key must be one of the following types: VARCHAR, VARCHAR2, NVARCHAR2, CHAR, NCHAR, NUMBER, BINARY_FLOAT, BINARY_DOUBLE, DATE, TIMESTAMP, TIMESTAMP_WITH_LOCAL_TIME_ZONE, TIMESTAMP_WITH_TIME_ZONE	If you **do not** plan to load this graph into the in-memory graph server (PGX) for analysis, you can ignore this warning. Otherwise, to fix this warning, you can either: • Change the data type of the key column to be one of the supported types. • Alter the edge table to add a new column with unique values of one of the supported types and declare it as the new key column.
`Column <column_name> of table <table_name> will prevent this graph from being loaded into memory. See the documentation for details and a list of supported datatypes`	Vertex or edge table columns must be one of the following types: VARCHAR, VARCHAR2, NVARCHAR2, CHAR, NCHAR, NUMBER, BINARY_FLOAT, BINARY_DOUBLE, CLOB, DATE, TIMESTAMP, TIMESTAMP_WITH_LOCAL_TIME_ZONE, TIMESTAMP_WITH_TIME_ZONE	If you **do not** plan to load this graph into the in-memory graph server (PGX) for analysis, you can ignore this warning. Otherwise, to fix this warning, choose one of the following options: • Ensure that you do not declare the columns with incompatible types as properties in your graph. • Include the columns in the graph definition, but exclude them when loading the graph into memory. See Manage Property Graphs for more information on loading a subset of graph properties into memory. • Try to convert the data types of the columns into one of the supported types.

Warning Message	Reason	How to Fix
`<Vertex/Edge> table key column(s) <column_names> in table (<table_name>) does not have an index.`	Index is missing in the underlying database table columns for one or more of the following - vertex key, edge key, source, and destination key columns. Graph Studio recommends indexing the key columns to optimize graph query traversals and ensuring faster retrieval of nodes and edges.	Click **Add Index** on the warning message to add index for the specific column. Ensure that you to have the `CREATE INDEX` privilege on the underlying table. Otherwise, request the owner of the table to create the index. In this case, if you ignore the warning, the graph will still be loaded into the in-memory graph server (PGX).
`Multiple labels in vertex/edge table <table_name> will prevent the graph from being loaded into memory.`	A vertex or an edge table can have only a single label.	If you **do not** plan to load this graph into the in-memory graph server (PGX) for analysis, you can ignore this warning. Otherwise, to fix this warning, change your graph definition and make sure each vertex and edge table only declares a single label.

Specify Vertex and Edge Table Keys

The property graph wizard in Graph Studio allows you to specify keys for the vertex and edge tables when creating a graph.

It is important to note the following key concepts:

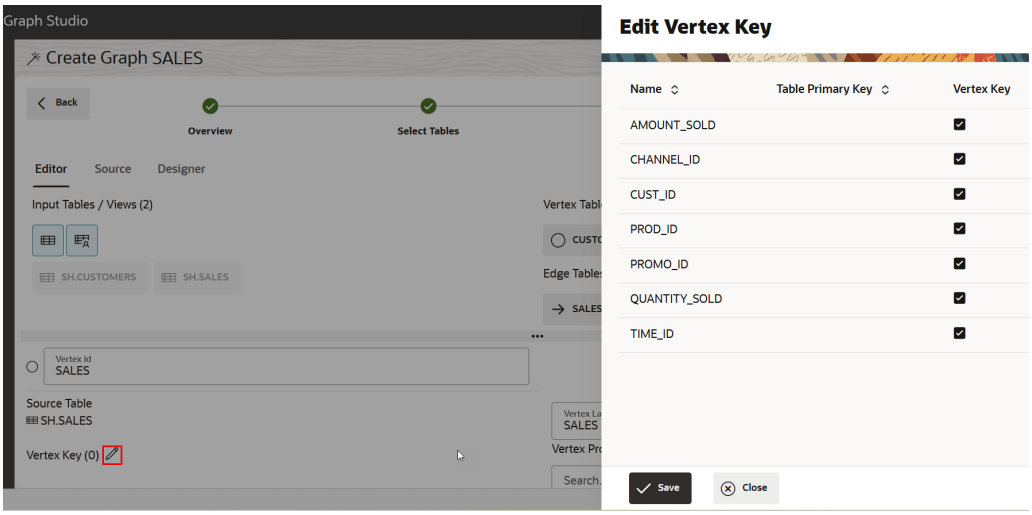
- All the vertex and edge tables of a graph must have vertex and edge keys defined respectively.
- By default, the wizard generates the vertex and edge table keys using the primary key of the underlying database tables for the vertex and edge tables respectively.
- By default, the edge source key and edge destination key for an edge table corresponds to a unique key (foreign key) of the source and destination tables respectively.
- If there is no primary key defined in the source database tables, then you must specify the required vertex or edge key in order to proceed with the graph creation.
- Similarly, you can specify the edge source key, referenced source vertex key, edge destination key, or referenced destination vertex key for an edge table, if they are not automatically generated.

Therefore, you can perform the following at the **Define Graph** step of the property graph wizard workflow:

- Specify a vertex key for a vertex table.
- Specify an edge key for an edge table.
- Specify an edge source key for an edge table.
- Specify an edge destination key for an edge table.
- Specify a source vertex key for an edge table.
- Specify a destination vertex key for an edge table.

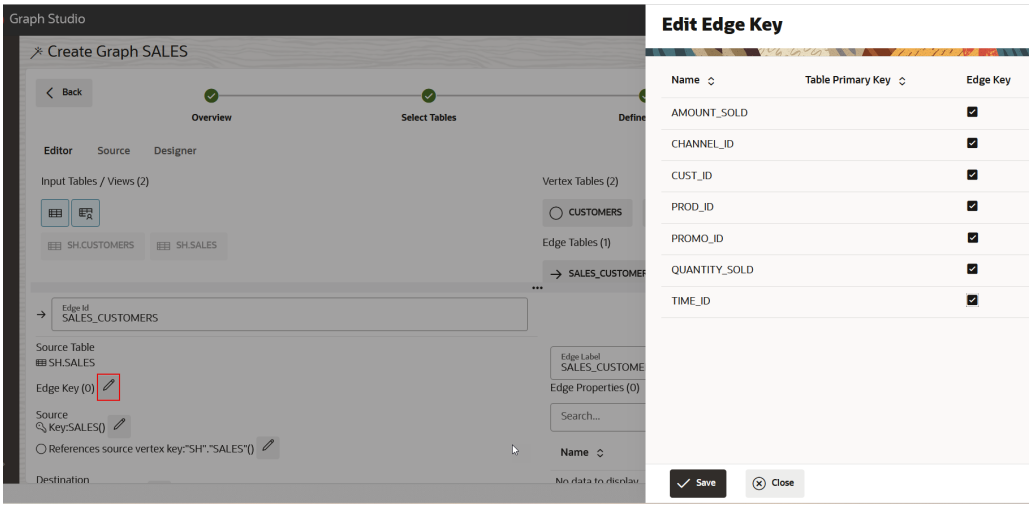
The following steps explain how to perform the preceding operations. The instructions assume that you are on the third step of the property graph wizard workflow.

1. To specify a vertex key for a vertex table:
   a. Click the required vertex table in the **Editor** tab.
   The **Source Table** name along with the **Vertex Key**, **Vertex Label** and **Vertex Properties** are displayed in the bottom pane.
   b. Click the **Edit Vertex Key** icon.
   The **Edit Vertex Key** slider opens as shown:



   Any existing primary key constraint is displayed in this key selection dialog.
   c. Select the required columns for the vertex key.
   Ensure you have selected at least one key column and the selected vertex key columns are unique.
   d. Click **Save**.
   The **Vertex Key** is saved.

   Alternatively, you can provide the vertex key directly in the **Source** tab using the `KEY` clause for the vertex tables.
2. To specify an edge key for an edge table:
   a. Click the required edge table in the **Editor** tab.
   The **Source Table** name along with the **Edge Key**, **Source** vertex key, **Destination** vertex key, **Edge Label** and **Edge Properties** are displayed in the bottom pane.
   b. Click the **Edit Edge Key** icon.
   The **Edit Edge Key** slider opens as shown:



Any existing primary key constraint is displayed in this key selection dialog.

c. Select the required columns for the edge key.
Ensure you have selected at least one key column and the selected edge key columns are unique.

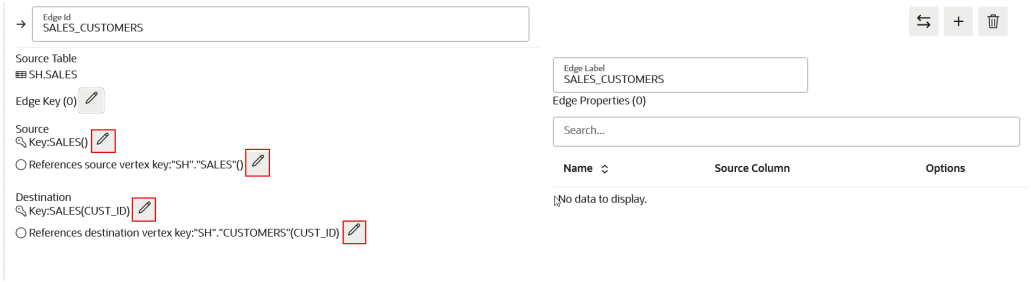
d. Click **Save**.
The **Edge Key** is saved.

Alternatively, you can provide the edge key directly in the **Source** tab using the `KEY` clause for the edge tables.

3. To specify an edge source key, edge destination key, source vertex key, or destination vertex key for an edge table:

a. Click the edge table in the **Editor** tab.

b. Click the  icon, corresponding to the **Source Key**, **References source vertex key**, **Destination Key**, or **References destination vertex key** which you wish to specify or change:



This opens the corresponding **Edit Edge Source Key**, **Edit Edge Source References**, **Edit Edge Destination Key**, or **Edit Edge Destination References** slider. Any existing key value is shown highlighted.

c. Select the required columns for the keys.
Ensure you have selected at least one key column and the selected key columns are unique.

d. Click **Save**.

Alternatively, you can provide the `SOURCE KEY`, `DESTINATION KEY`, referenced source or destination vertex keys directly in the **Source** tab for the edge tables.

Add New Edges During Graph Creation

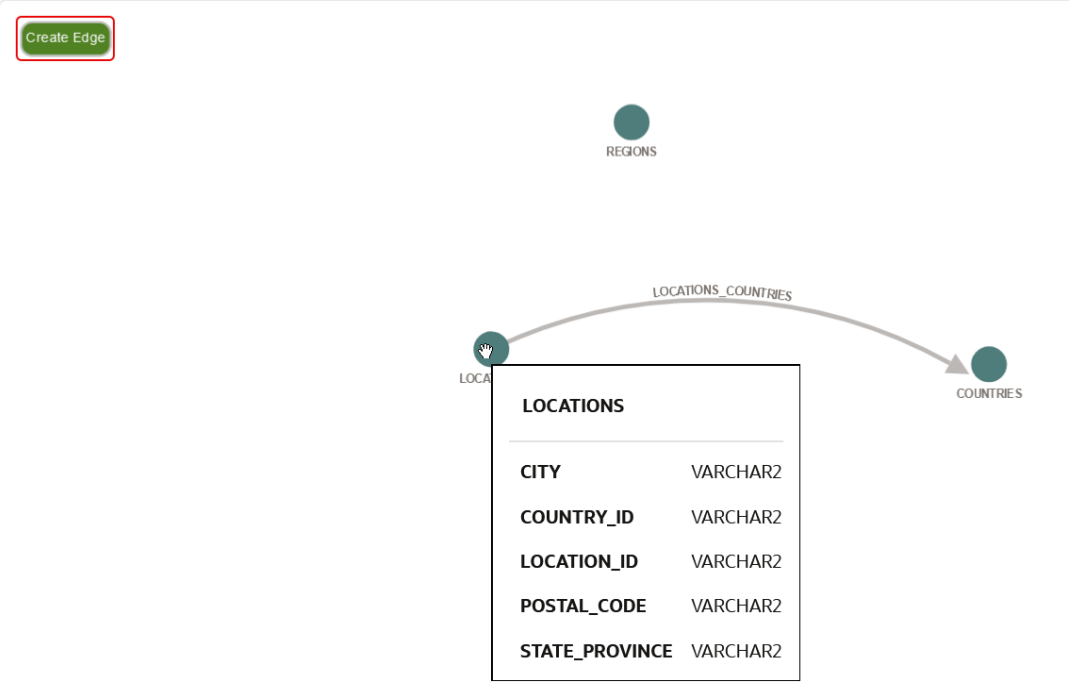
When creating or editing a graph, you can visually add new edges between two vertex tables through drag and drop action.

You can perform this action inside the **Designer** tab at the **Define Graph** step of the graph creation workflow.

The following steps describe the process for visually creating a new edge when using the property graph wizard. The instructions assume that you are at the **Define Graph** step of the property graph wizard, and more than one vertex tables are selected for building the graph.

1. Click the **Designer** tab.

   A preview of the graph to be created is displayed. For example:



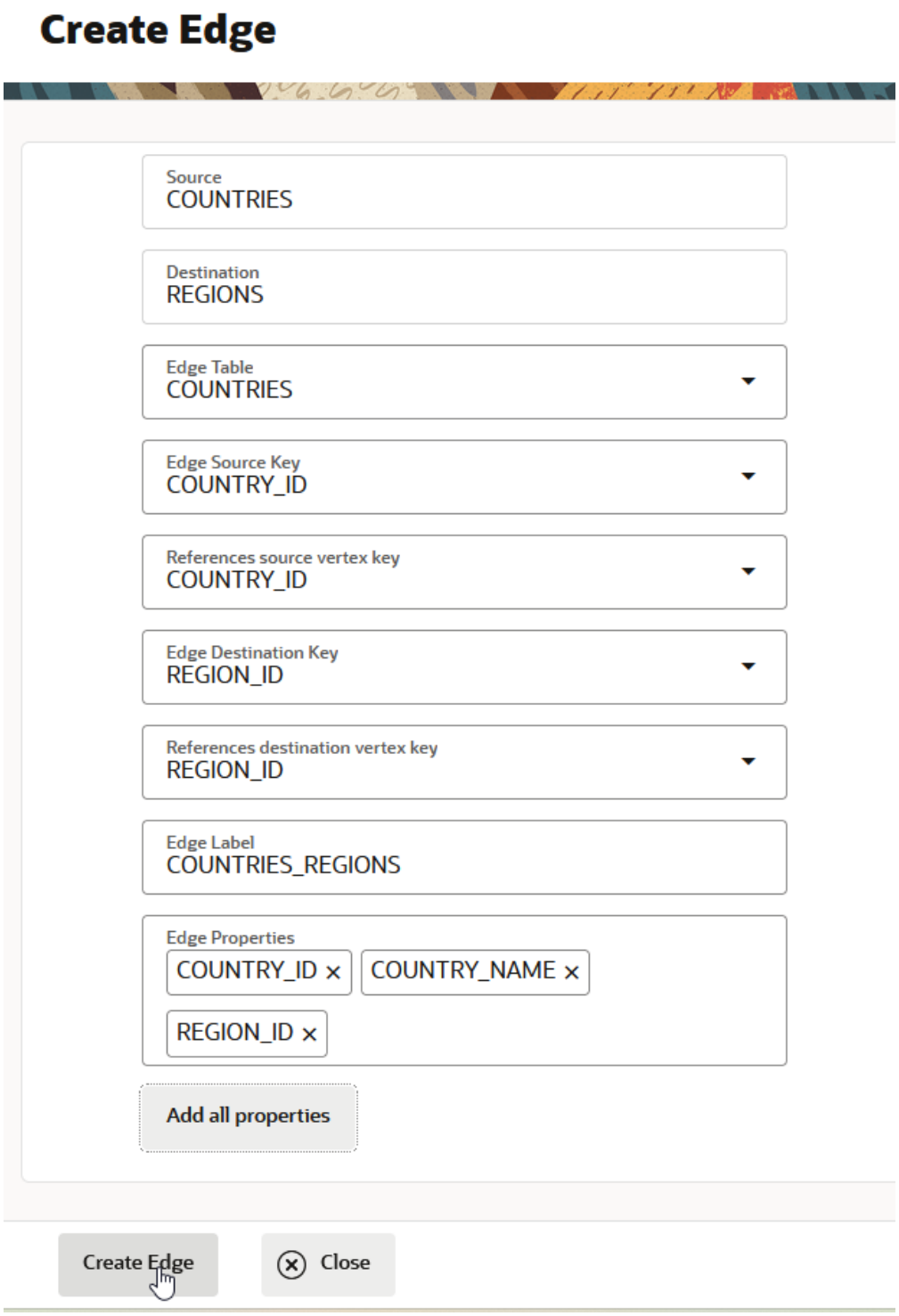
   You can drag and move the vertices as required inside the tab. When you hover over a vertex, a tooltip describing the vertex properties is displayed. Alternatively, you can right-click on a vertex to view the vertex properties.

2. Click **Create Edge** (shown highlighted in the preceding figure) to activate the mode to add a new edge.

   In this mode, starting a drag action from a vertex will start drawing an arrow (depicting an edge) from the source vertex until it is released on the destination vertex.

3. Drag an arrow from the desired source vertex and drop it on the target destination vertex to add a new edge.

   Note that if you drag an arrow from a vertex table to itself, then a self edge is created. This indicates that the source and destination tables are the same.
   The **Create Edge** slider opens as shown:



4. Select the **Edge Table**.

   The list of choices for the edge table is obtained from the input tables selected in the **Select Tables** step of the property graph wizard.

5. Select the **Edge Source key** and **Edge Destination key**.

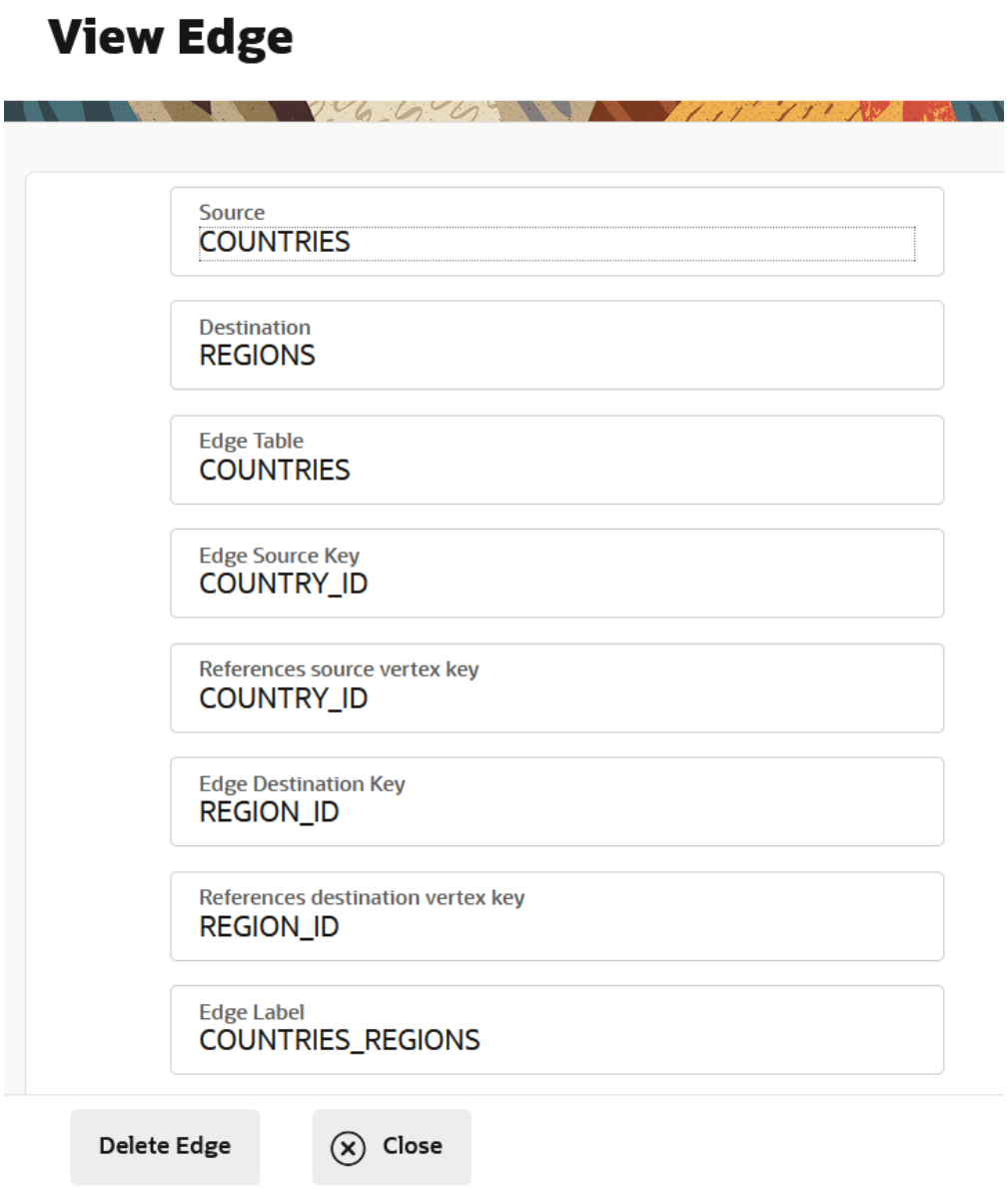
6. Select the source and destination vertex keys using the **References source vertex key** and **References destination vertex key** drop-downs respectively.

Note that these key columns must correspond to a unique (foreign) key of the source and destination vertex tables.

7. Optionally, modify the **Edge Label**.
8. Select the **Edge Properties**.

   You can choose any combination of the columns of the edge table as edge properties. By default, these edge properties will automatically be assigned the name of the column. If you wish to add all the columns of the edge table, then you can simply click **Add all properties**.
9. Click **Create Edge**.

   The new edge is shown displayed between the two vertex tables. You can click on the edge to review and verify the edge configuration as shown.



Optionally, you can click **Delete Edge** if you wish to delete the edge.

All edges added or deleted in the **Designer** tab will be reflected in the `CREATE PROPERTY GRAPH` statement in the **Source** tab.

Create a Property Graph by Editing an Existing Graph

You can edit an existing property graph on the Graphs page in Graph Studio.

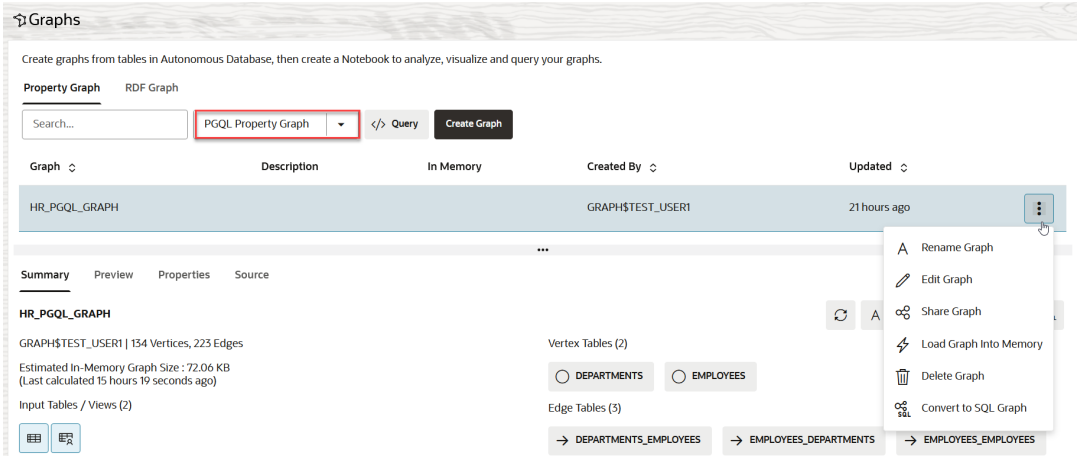
The following lists a few scenarios for editing a graph:

- Add or remove tables from the graph
- Rename labels for edges or vertices
- Alter the orientation of the edges
- Add or remove a vertex or edge property
- Rename a vertex or an edge property

To edit a graph, perform the following steps:

1. Navigate to the Graphs page using the **Graphs** menu link.
2. Select the graph type from the drop-down shown highlighted in the following figure.

   The list of property graphs to which you have access are displayed.
3. Select the graph which you want to edit and click open the additional options menu as shown:



4. Click **Edit Graph** in the context menu.

   The property graph wizard opens and displays the **Overview** page with the graph details. You can choose to perform one of the following actions:

   - **Save the edited graph as a new Property Graph:**

     a. Rename the graph by entering a new **Graph Name**.
     Note that the graph name is not case sensitive and is normalized to uppercase by default in Graph Studio. However, you can enable case sensitivity through the **Preserve Case** toggle in step c.

     b. Follow the graph creation workflow from step-4 to step-7 as explained in Create a Property Graph from Existing Relational Tables, and edit the graph as required.

> **⚠ Caution**
>
> When editing a graph, if you update the list of selected tables, then Graph Studio will generate a new property graph statement that will overwrite the current one.

c. Click the enabled **Save a Copy** button.
The **Edit Graph** slider opens.

The **Estimated in memory graph size** is computed and displayed in the slider. See Estimated in memory graph size for more information.

Optionally, enable the **Preserve Case** toggle to preserve the case for graph, property, and label names. In such a case, ensure to enclose the preserved names in quotes when referencing them in SQL graph queries in the Notebooks.

d. Click **Confirm** to save a copy of the graph with the new name.
The new property graph gets created.

- **Update the existing property graph:**

a. Use the same initial **Graph Name** in order to overwrite the existing graph.

b. Follow the graph creation workflow from step-4 to step-7 as explained in Create a Property Graph from Existing Relational Tables, and edit the graph as required.

> **⚠ Caution**
>
> When editing a graph, if you update the list of selected tables, then Graph Studio will generate a new property graph statement that will overwrite the current one.

c. Click the enabled **Save** button.
The **Edit Graph** slider opens.

The **Estimated in memory graph size** is computed and displayed in the slider. See Estimated in memory graph size for more information.

d. Click **Confirm** to overwrite the graph.

> **⚠ Caution**
>
> If you overwrite an existing property graph, then notebooks using these graphs may not work and hence they need to be manually updated.

Create a Property Graph from an RDF Graph

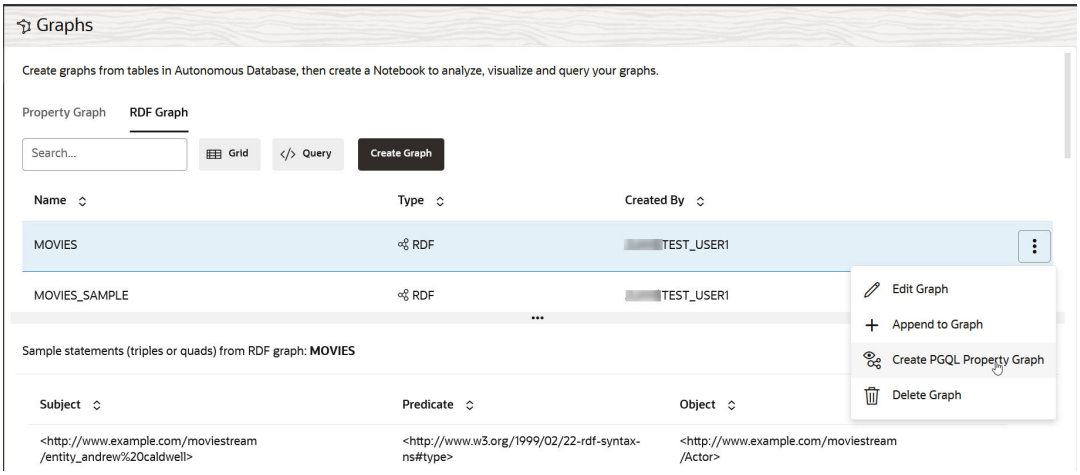
Graph Studio provides a modeler interface where you can map from an existing RDF graph to a create a PGQL property graph.

You can then load this graph into the graph server to run graph analytics.

Perform the following steps to invoke the modeler interface and follow the workflow to create a PGQL property graph from an RDF graph.

1. Navigate to the Graphs page using the **Graphs** menu link.
2. Click the **RDF Graph** tab.

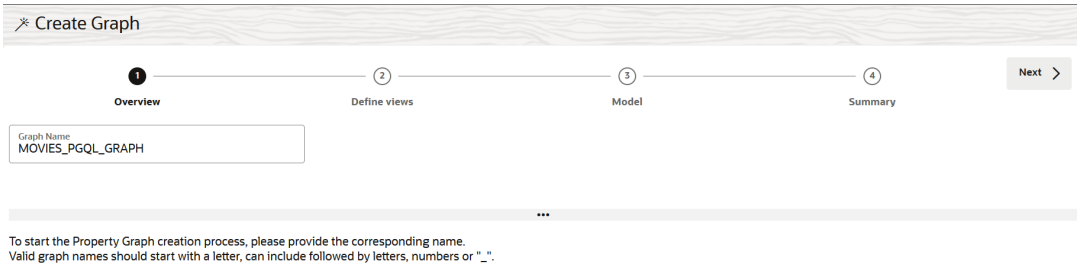
The list of RDF graphs to which you have access are displayed as shown:



3. Select the RDF graph from which you want to create a property graph and click open the additional options menu as shown in the preceding figure.

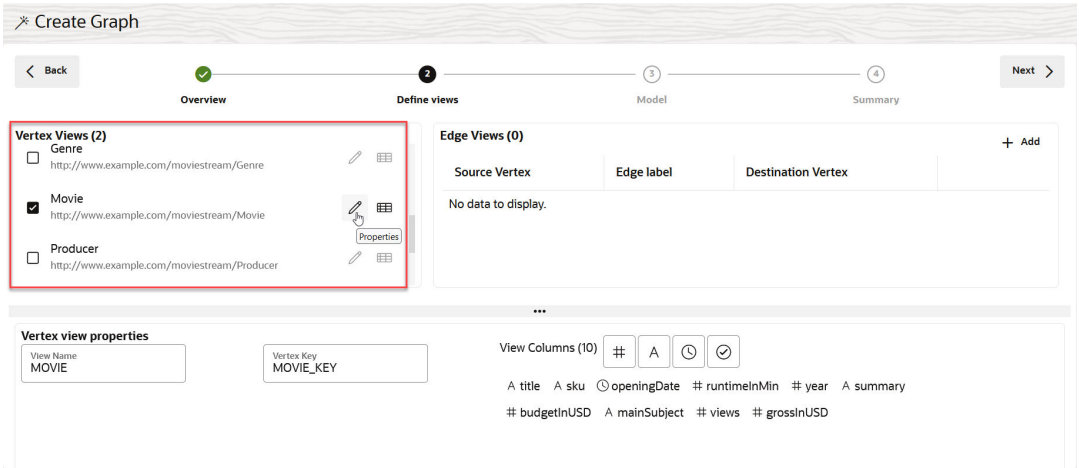
4. Click **Create PGQL Property Graph** in the context menu.

   The modeler interface opens and displays the **Overview** page as shown:



5. Enter the **Graph Name** and click **Next**.

   The **Define Views** page opens as shown:



This page displays the list of RDF classes for the graph in the top left **Vertex Views** pane. These RDF classes can be translated into vertices for the property graph. This step of the modeler interface also allows you to create the edges for the selected vertices in the top right **Edge Views** pane. For any selected vertex or edge view, you can view the corresponding **Properties** or **Sample data** details in the bottom pane of the page.

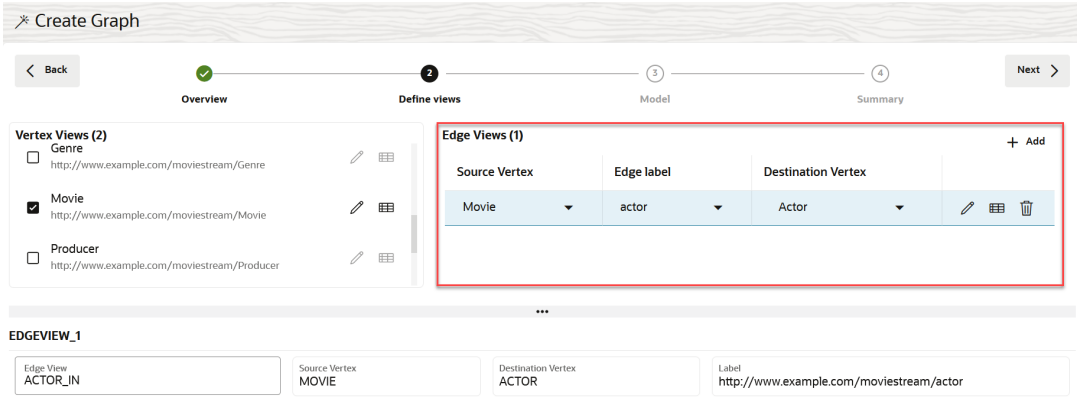
It is important to note that you must add atleast two vertex views and one edge view on this page.

6. Select two or more RDF classes to define the required **Vertex Views** (as shown highlighted in the preceding figure) for the graph.
7. Optionally, review the **Properties** or the **Sample data** for any selected vertex view in the bottom details pane.

   You can choose to perform any of the following actions, if required, when viewing the **Vertex view properties**:

   - Change the **View Name**.
   - Change the **Vertex Key** name.
   - View columns by applying the following column filters:
     - **Filter by Number**
     - **Filter by Text**
     - **Filter by Time**
     - **Filter by Boolean**
8. Add one or more edge views by clicking **Add** in the top right **Edge Views** section (shown highlighted in the figure in the following step).

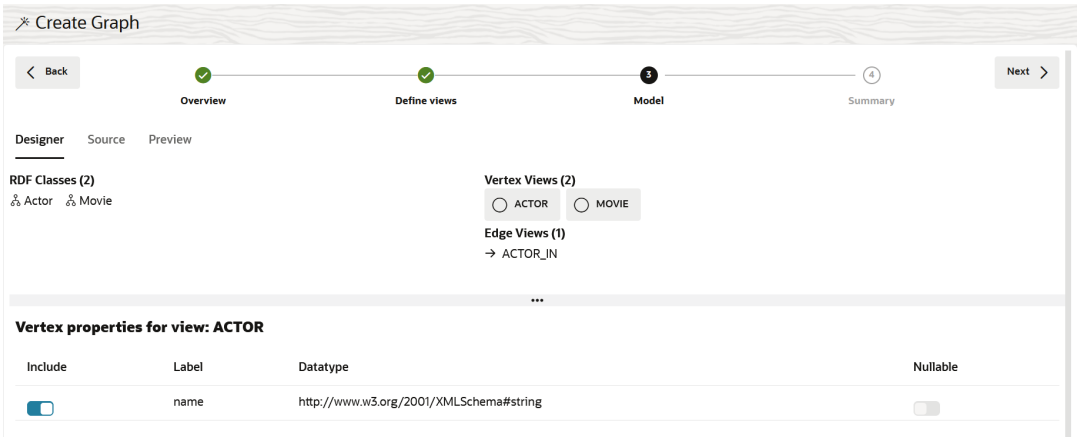
   A new row gets added to the panel where you can provide the source and destination vertices along with the edge label.
9. Select a **Source Vertex**, **Edge Label**, and **Destination Vertex**.



10. Optionally, choose to perform any of the following actions for any added edge view.

    - : Review the edge **Properties** and optionally, change the **Edge View** name in the bottom details section.
    - : Review the **Sample data**.
    - : Delete the edge view.
11. Click **Next** to proceed.

    The **Model** page of the workflow opens as shown:

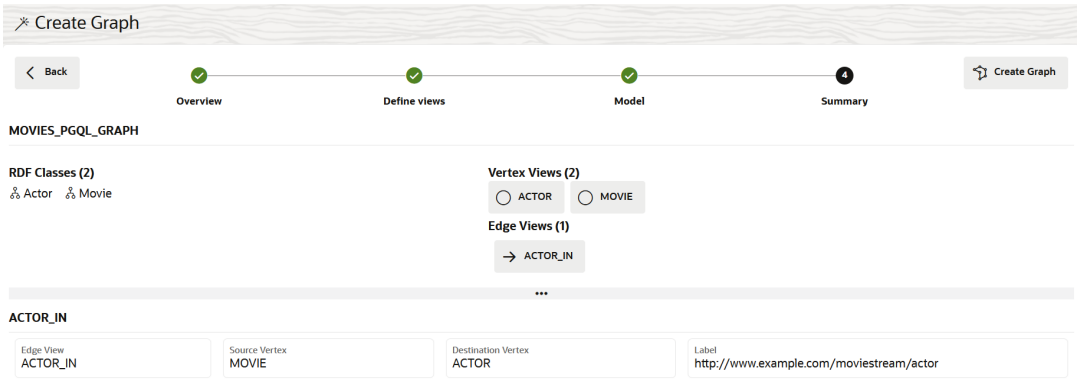


This page comprises the following three tabs:

- **Designer:** To review the selected vertices and edges of the property graph. Also, you can determine the properties to be included for a vertex and configure the nullable constraint for a vertex property.
- **Source:** To view the `CREATE PROPERTY GRAPH` source statement for the graph.
- **Preview:** To preview the modeled graph.

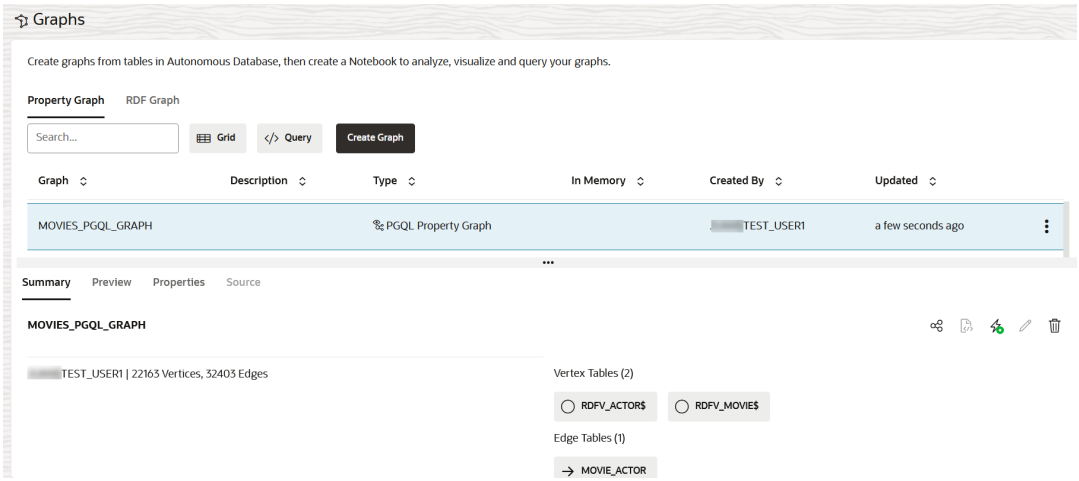
12. Optionally, click any vertex view in the **Designer** tab and choose to perform any of the following actions in the bottom **Vertex properties for view** details pane.
    - **Include:** Switch ON or OFF this toggle to indicate if a property is included or excluded. Note that atleast one property must be included for a vertex view. Otherwise, you cannot proceed to the next step of the workflow.
    - **Nullable:** Switch ON or OFF this toggle to indicate the nullable constraint for a property.
        - `TRUE`: Vertices with `NULL` (missing) values for the property will be included.
        - `FALSE`: Vertices with `NULL` (missing) values for the property will be excluded.

      Note that atleast one `FALSE` property must be included.
13. Click **Next** to view the property graph **Summary**.



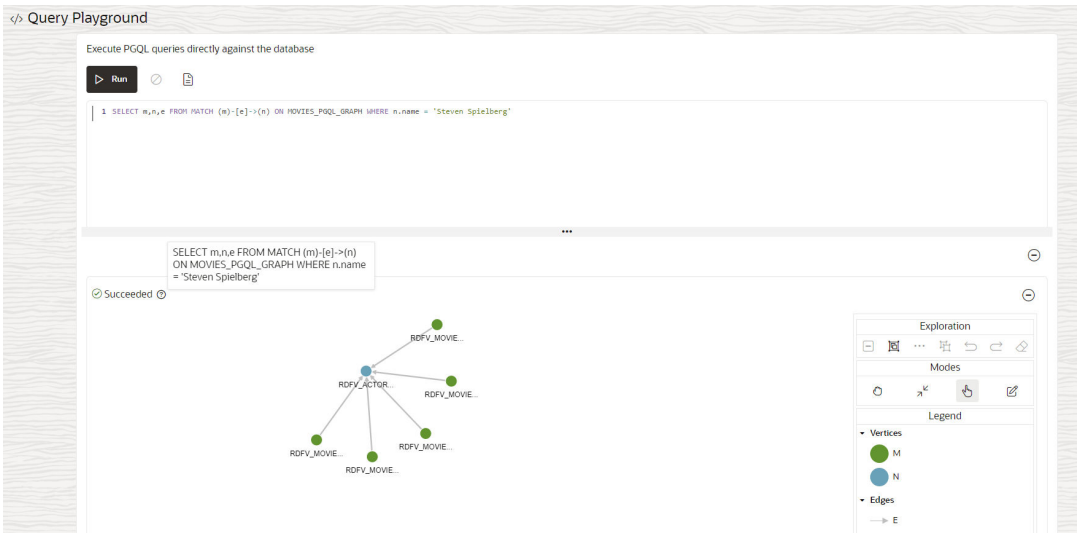
14. Click **Create Graph**.

    The graph creation job is initiated on the **Jobs** page. Once the job completes successfully, you can view the newly created PGQL property graph on the **Graphs** page in the **Property Graph** tab.



Once the graph is created, you can run PGQL queries on the graph in the Query Playground page or analyze and visualize the graph using a Notebook.

The following figure shows an example PGQL query that is executed on the graph in the Query Playground page:



Create an RDF Graph in Graph Studio

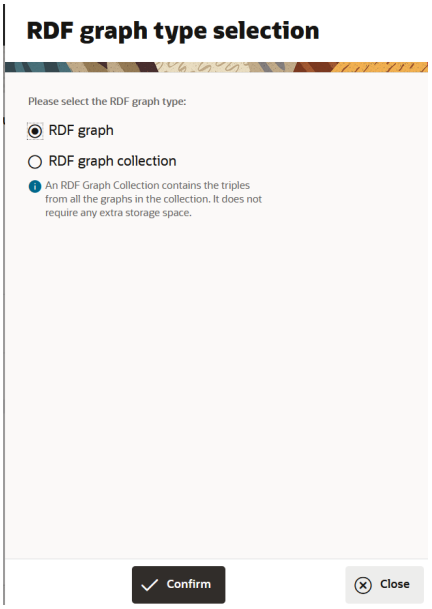
You can create an RDF graph or an RDF graph collection using Graph Studio in Oracle Autonomous AI Database.

> **Note**
>
> You can also see the Oracle LiveLabs workshop, Working with RDF Graphs in Graph Studio, for a complete example on creating, querying and visualizing an RDF graph.

1. Navigate to the **Graphs** page.
2. Select the **RDF Graph** tab.

   All the RDF graphs to which you have access are displayed.
3. Click **Create Graph**.

The **RDF graph type selection** slider opens as shown:



4. Choose to create an RDF graph or an RDF graph collection as required.
   - To create a single RDF graph by importing RDF data from Oracle Cloud Infrastructure Object Storage:
     a. Select **RDF graph** as the **RDF graph type**.
     b. Click **Confirm**.
     RDF wizard displays the **OCI Storage input files** page. See Use RDF Wizard to Create an RDF Graph for more information.
   - To create an RDF graph collection from existing RDF graphs:
     a. Select **RDF graph collection** as the **RDF graph type**.
     b. Click **Confirm**.
     RDF wizard displays the **General** page. See Use RDF Wizard to Create an RDF Graph Collection for more information.

Use RDF Wizard to Create an RDF Graph

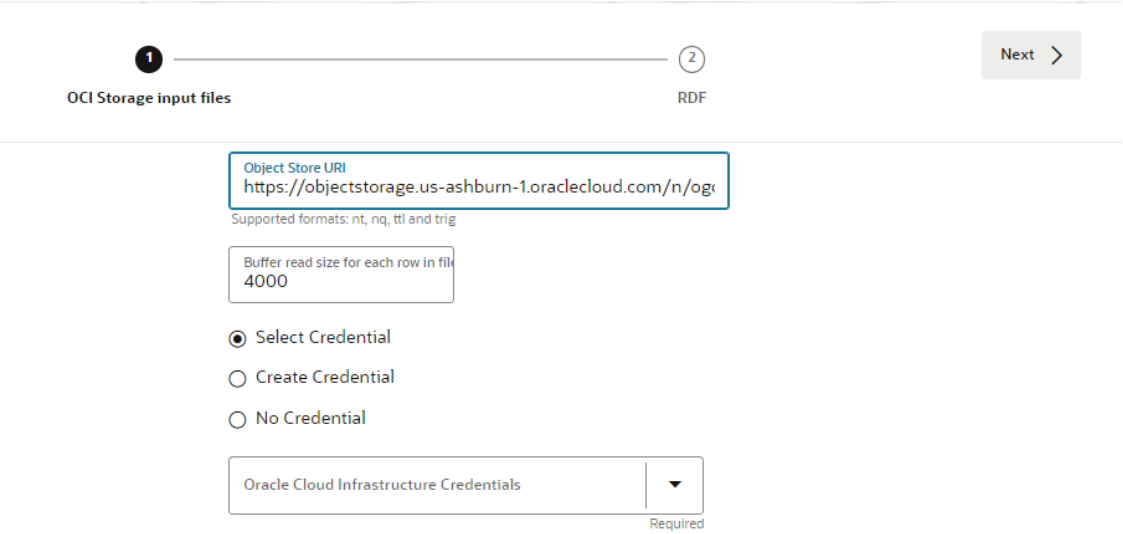
You can create a new RDF graph using the RDF wizard feature in Graph Studio.

As a prerequisite, you must upload the RDF data to Oracle Cloud Infrastructure Object Storage. You can then access the RDF data store from Graph Studio with or without credentials. See Perform Prerequisites to Use RDF Graph Wizard for more information.

The RDF wizard consists of two pages:

- On the first page, you can provide the OCI object store credentials to create a new credential or select any existing object store credential. Otherwise, you can simply provide a pre-authenticated request URL to access the object store without credentials.
- On the second page, you can provide the RDF graph information.

To create an RDF graph, perform the following steps in the RDF wizard:



1. Enter the **URI** path or the **Pre-Authenticated Request URL** to the RDF object store in your OCI bucket.
2. Optionally, enter the **Buffer read size for each row in file**.
3. Choose the required **Credential** option:
   - Select an existing credential:
     a. Click **Select Credential**.
     b. Select a credential name from the **Oracle Cloud Infrastructure Credentials** drop-down list.
     On selection, the **Oracle Cloud Infrastructure User Name** value is automatically populated.
   - Create a new credential:
     a. Click **Create Credential**.
     b. Enter a **Credential Name**.
     c. Enter your **Oracle Cloud Infrastructure User Name**.
     d. Enter the **Auth Token** value.
   - Click **No Credential** to access the object store using pre-authenticated request URL.
4. Click **Next**.
5. Enter the RDF **Graph Name**.
6. Click **Create**.

   This redirects you to the Jobs page, where the RDF graph creation job is initiated.

   Successful completion of the job indicates that the RDF data is imported and the RDF graph is created successfully. The newly created graph appears on the Graphs page in the **RDF Graph** tab.

Perform Prerequisites to Use RDF Graph Wizard

Prior to using the RDF graph wizard utility in Graph Studio, you must upload the RDF data to Oracle Cloud Infrastructure Object Storage.

You can then access the RDF data store from Graph Studio with or without credentials.

You must perform the following prerequisite actions using the same Oracle Cloud credentials used for creating the graph user. See Create a Graph User for more information.

Topics:

- Get the URI or Pre-Authenticated Request URL to Access the Object Store
- Get the Object Store Credentials

Get the URI or Pre-Authenticated Request URL to Access the Object Store

You must determine the URI or the pre-authenticated request URL for the RDF source data object in Oracle Cloud Infrastructure Object Storage which is to be imported in Graph Studio.

Perform the following steps to find the URI or the pre-authenticated request URL for the RDF object store:

1. Sign in to the OCI console using your Oracle Cloud credentials.
2. Open the navigation menu and select **Storage**.
3. Under **Object Storage & Archive Storage**, select **Buckets** and navigate to your Object Storage.
4. Select a **Compartment** to view the list of buckets that are existing in that compartment.

   Optionally, If you need to create a new bucket, then perform the following steps:

   a. Select your **Compartment** and click **Create Bucket**.
   The **Create Bucket** dialog opens.

   b. Enter the **Bucket Name** and click **Create**.
   The bucket is created and appears on the **Buckets** table.
5. Select the required bucket **Name**.

   The objects uploaded to the bucket are listed in the **Objects** section on the **Bucket Details** page.
   Optionally, if you need to create a new object store, then perform the following steps:

   a. Click on the required bucket and navigate to the **Bucket Details** page.

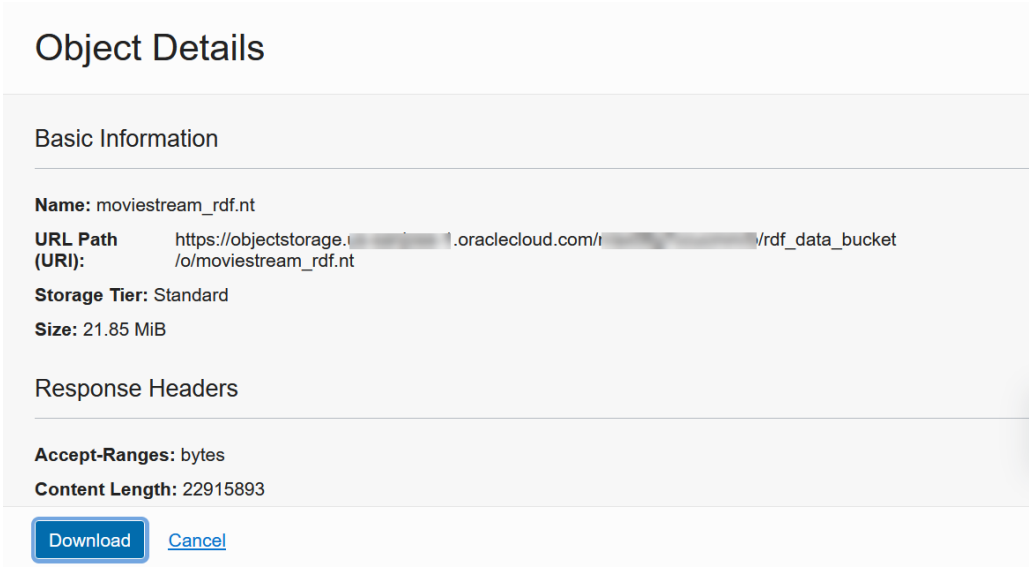
   b. Click **Upload** in the **Objects** section.
   The **Upload Objects** slider opens.

   c. Select the file containing RDF data on your local system and click **Upload**.

   > **Note**
   >
   > - Files with extensions `.nt` (N-triples), `.nq` (N-quads), `.trig` (TriG), and `.ttl` (Turtle) are supported in Graph Studio.
   > - Graph Studio supports only five million rows of data for `.ttl` and `.trig` files. In case these files contain more than 5 million rows, then you must convert your input `.ttl` or `.trig` file to a `.nt` file.

   d. Click **Close** to return to the Bucket Details page.
   The uploaded file is listed under **Objects** section.
6. Select the Actions menu for the required Object and click one of the following options as required:

- Click **View Object Details** if you want to access the object store with credentials. You can determine the URI path on the Object Details page as shown:



  The **URL Path (URI)** field displays the URI to access the object store.

- Click **Create Pre-Authenticated Request** if you want to access the object store without credentials. To obtain the pre-authenticated request URL on the **Create Pre-Authenticated Request** page, see the Oracle Cloud Infrastructure Documentation for more information.

Get the Object Store Credentials

You need to determine your object store credentials if you want to authorize Graph Studio to access the RDF data source objects in Oracle Cloud Infrastructure Object Storage.

> **Note**
>
> This section only applies if Graph Studio must access the object store using credentials. You can skip this section if you are using a pre-authenticated request URL to access the object store.

Perform the following steps to determine the `username` and `password` (auth token) to access the RDF object store:

1. Sign in to the OCI console using your Oracle Cloud credentials.
2. Click the **avatar** icon in the top right corner to open your profile.

   Note the first entry under **Profile**. This is your OCI user name.
   The OCI user name is the `username` to be used to access the object store.
3. Create an auth token:
   a. Click **User Settings** in the **Profile** menu.
   b. Click **Auth Tokens** on the left side under **Resources**.
   c. Click **Generate Token**.
      The **Generate Token** dialog opens.
   d. Enter a token **Description**.
   e. Click **Generate Token**.

The auth token is generated. Copy the token string immediately. Save it for later use as you cannot retrieve the token after closing the dialog box.

The auth token is the `password` to be used to access the object store.

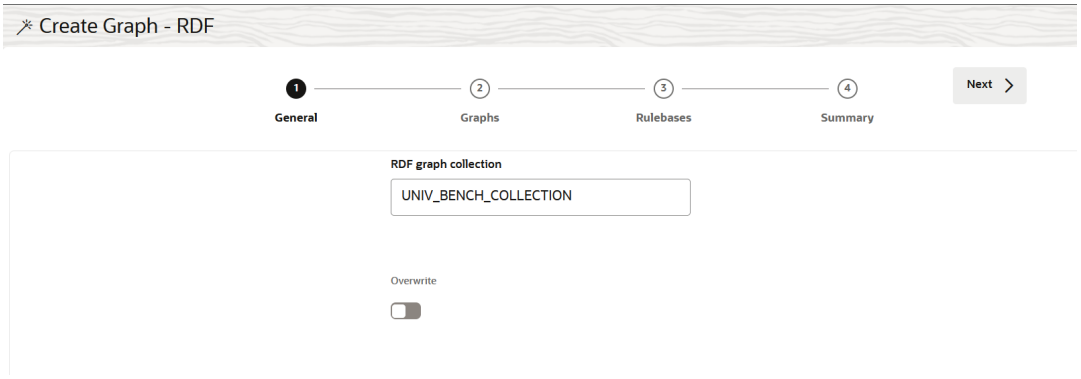
Use RDF Wizard to Create an RDF Graph Collection

You can create an RDF graph collection, with one or more existing graphs, using the RDF wizard feature in Graph Studio.

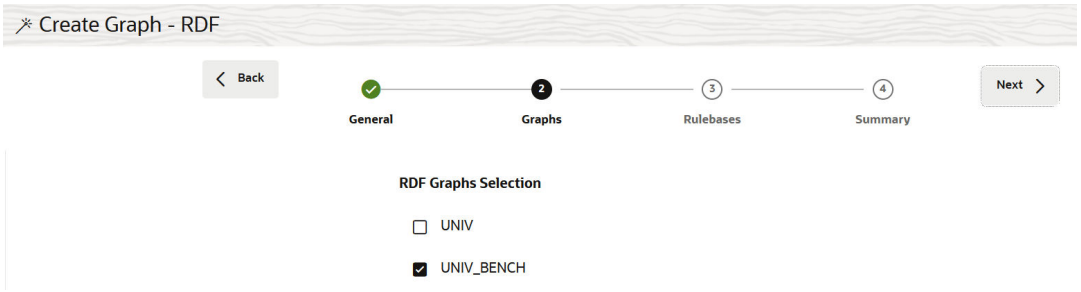
Optionally, you can perform inferencing by applying a rulebase to the graph collection. Therefore, an RDF graph collection is a virtual combination of one or more RDF graphs. Additionally, it may include entailments when a rulebase is used.

To create an RDF graph collection, perform the following steps in the RDF wizard:

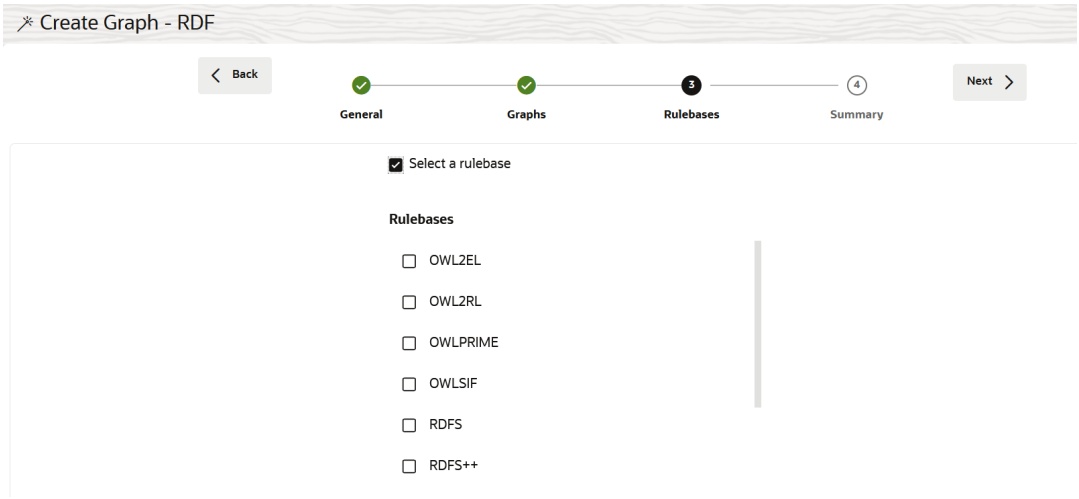
1. Enter the name for the **RDF graph collection** in the **General** step as shown:



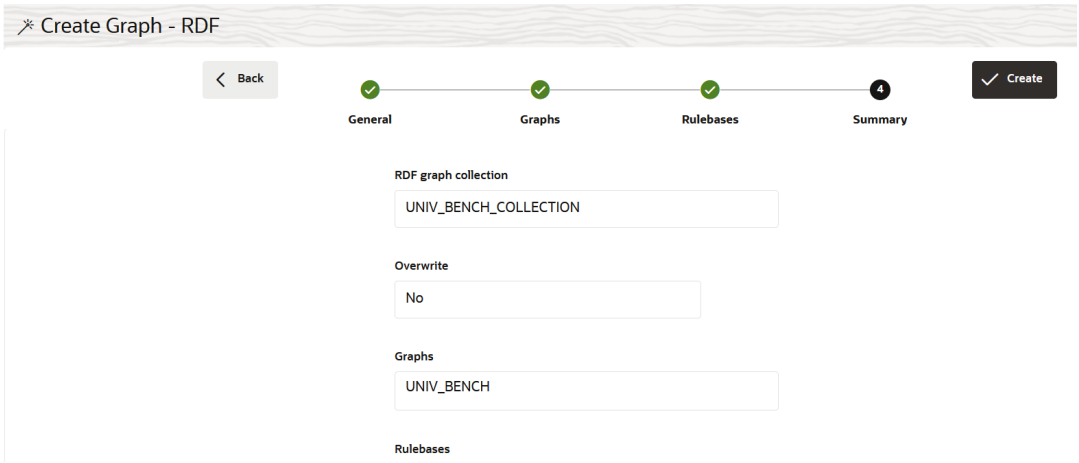
2. Optionally, switch the **Overwrite** toggle to overwrite an existing graph collection, if you have provided an existing graph collection name in the preceding step.
3. Click **Next** to go to the **Graphs** step.



4. Select one or more RDF graphs under **RDF Graphs Selection**.
5. Click **Next** to go to the **Rulebases** step.



6. Optionally, if you want to perform inferencing operation, then select **Select a rulebase** and select the required **Rulebase** for the graph collection. Otherwise, you can skip this step.
7. Click **Next** to view the **Summary** of the parameters selected for creating an RDF graph collection.



If you have selected rulebases, then Graph Studio validates if an entailment for the selected RDF graphs and rulebases in the collection already exists. If there is no valid entailment, then a new one is created. As creating a new entailment is a long running process, an appropriate warning is displayed when creating a new entailment.

8. Click **Create**.

   The job to create an RDF graph collection is initiated on the Jobs page. On successful completion of the job, the newly created RDF graph collection is listed on the Graphs page in the **RDF Graph** tab.

Manage Graphs

You can explore and manage your graphs in Graph Studio.

Topics:

- Manage Property Graphs
- Manage RDF Graphs

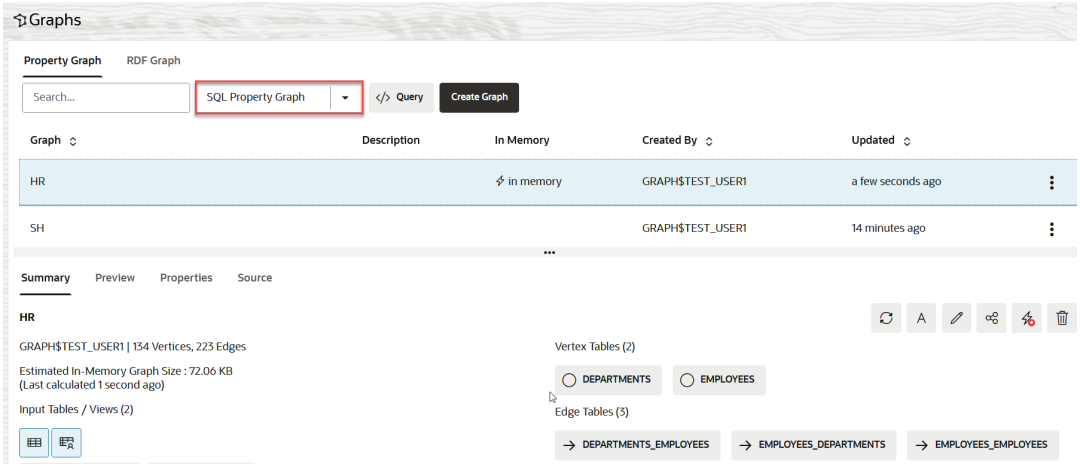
Manage Property Graphs

You can load a property graph into memory, share, edit, rename, delete or preview a graph.

To manage property graphs:

1. Click **Graphs** on the left navigation menu and navigate to the Graphs page.
2. Select the **Property Graph** tab to view the list of property graphs for which you have access in the Autonomous AI Database.

   The type of property graph displayed by default on the Graphs page is determined by the drop-down value shown highlighted in the following figure:



   The drop-down supports one or both of the following values depending on the database version used in your Autonomous AI Database instance:

   - **SQL Property Graph:** This is the default property graph type displayed on the Graphs page if you are using Oracle AI Database 26ai.
   - **PGQL Property Graph:** The PGQL property graph type is available on all types of tenancies and supported on all database versions. This is the only supported type if you are using Oracle Database 19c.

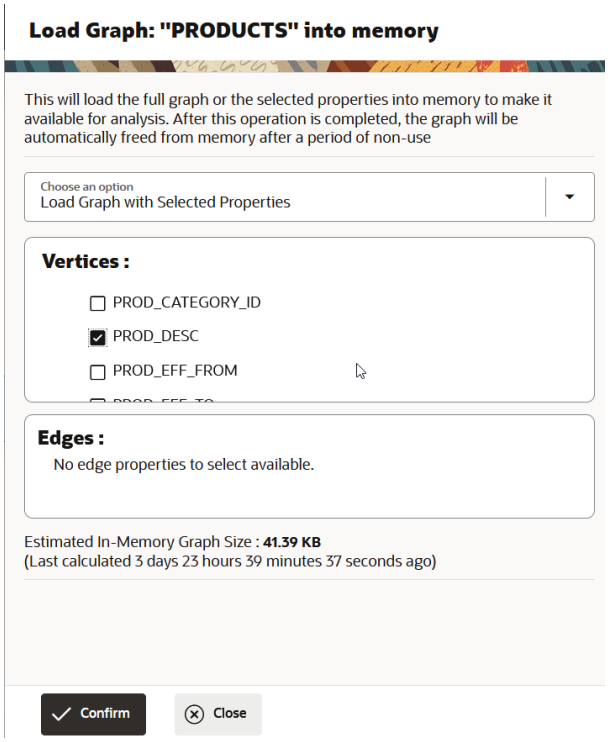
   In the preceding figure, the **In Memory** column indicates the property graphs that are loaded into memory. You must ensure to load a full graph or selected properties into memory before accessing it, as the notebook interpreters operate only on the in-memory graphs. See Available Notebook Interpreters for more information on notebook interpreters.

3. Select any property graph.

   The details of the graph are displayed in the graph details section of the Graphs page. Note that this section also displays the previously computed **Estimated In-Memory Graph Size**.

4. Optionally, click to perform any one of the following **actions** on the property graph:

Action	Description
	To recompute the graph metadata to refresh metadata information about the graph which might have become stale, like the total number of vertices and edges. This action also recomputes and updates the **Estimated In-Memory Graph Size**.

Action	Description
	To rename the graph.
	To edit the graph.
	To share the graph with other users.
	To load the complete graph or selected properties into memory for analysis. **Note** Optionally, you can load a **PGQL Property Graph** by name directly in the notebook. See Load Graphs into Memory Programmatically for an example. When you click the **Load Graph Into Memory** icon, the **Load Graph into memory** slider is displayed:  You can choose one of the following options: • **Load Graph with All Properties**: In this case, the complete graph with all its properties will be loaded into memory. • **Load Graph with Selected Properties**: The slider will display the list of available vertex and edge properties for the graph. By default, all the properties are selected. You can choose to select (or deselect) specific vertex or edge properties that you wish to load into memory. Note that the properties that are unsupported by the graph server (PGX) will appear disabled.

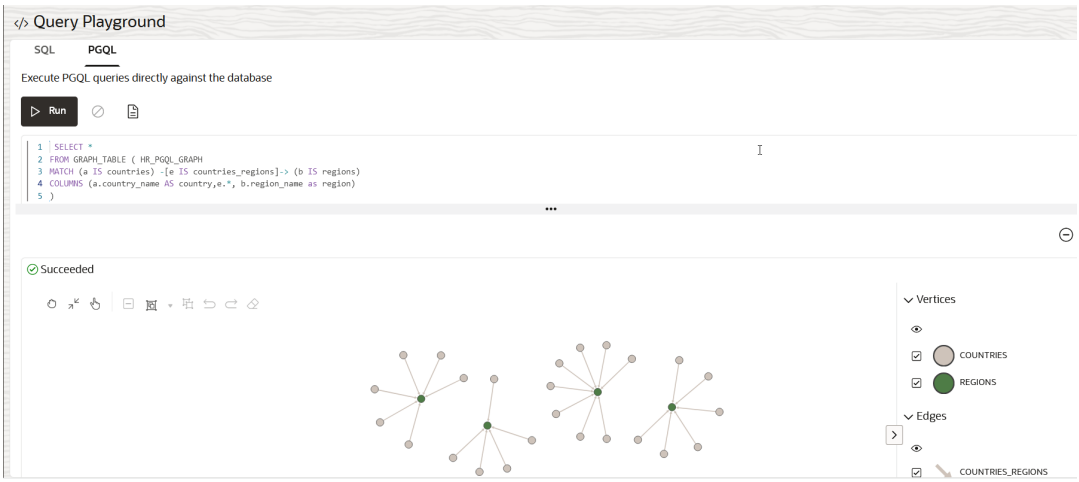
Action	Description
	To delete the graph.
	To convert a PGQL property graph into SQL graph. Note that this option is supported only if you are using an Autonomous AI Database instance with Oracle AI Database 26ai. Also, this action is available only for PGQL property graphs. See Convert a PGQL Property Graph to SQL Property Graph for more information.

This executes the desired action on the property graph.

5. Optionally, click the **</> Query** button if you wish to query and validate the selected property graph in the Query Playground page.

   The following example shows running SQL graph queries on a SQL property graph in the **SQL** tab of the Query Playground page. Note that the SQL tab is displayed only for Autonomous AI Database based on Oracle AI Database 26ai.

   The following example shows running PGQL graph queries on a PGQL property graph in the **PGQL** tab of the Query Playground page:



Also, note the following when using the Query Playground page:

- `CREATE PROPERTY GRAPH` and `DROP PROPERTY GRAPH` statements are supported.
- `SELECT` queries are supported, and the `SELECT` query results can be visualized, if applicable.
- `INSERT` and `UPDATE` queries are not supported.
- All queries are executed as the currently logged in user.
- The current query will be persisted in the browser's local storage once executed.
- You cannot run multiple statements or queries separated by semi-colon in the **SQL** tab. Only one SQL statement or query is allowed.

Convert a PGQL Property Graph to SQL Property Graph

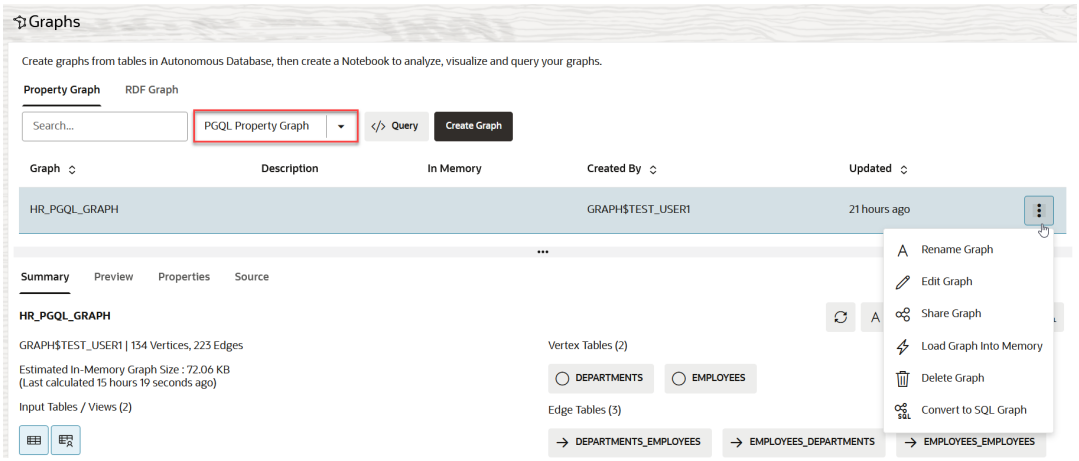
Graph Studio allows you to convert an existing PGQL property graph to SQL property graph.

Before you begin the migration operation, note the following:

- SQL property graphs are supported only on . Therefore, ensure that you are using an Autonomous AI Database instance with Oracle AI Database 26ai.
- PGQL property graphs based on database views cannot be migrated. If you attempt to migrate a PGQL graph based on views, then an error message is displayed and the original PGQL property graph is preserved.
- The migration operation does not delete the original PGQL property graph.
- The following describes a few basic characteristics of the newly created SQL property graph:
  - The SQL graph will have the original name of the PGQL property graph and the original graph will be renamed by appending `_PGQL` at the end of the name.
  - The SQL graph will consist of the same vertex and edge tables as the original PGQL property graph.
  - The SQL graph will be owned by the user who triggered the migration operation, regardless of the owner of the PGQL property graph.

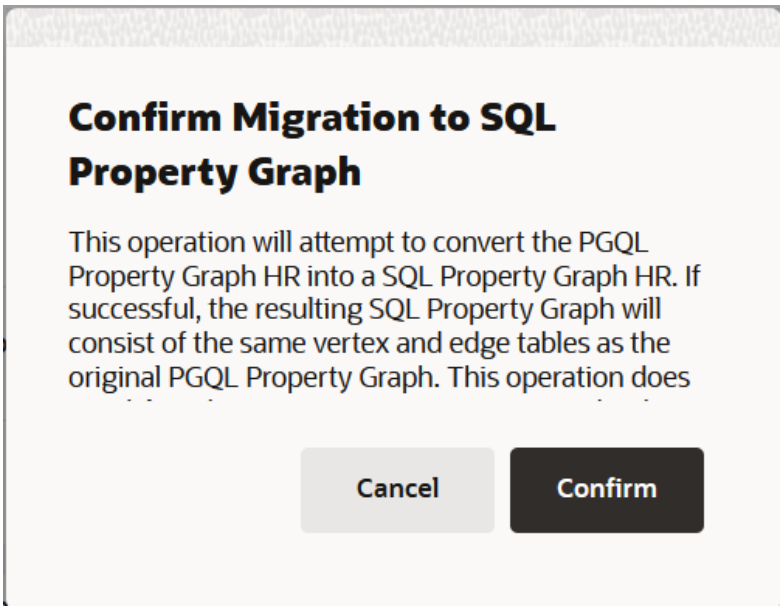
1. Navigate to the Graphs page using the **Graphs** menu link.
2. Click the **Property Graph** tab.
3. Select **PGQL Property Graph** from the drop-down shown highlighted in the following figure.

   The list of PGQL property graphs to which you have access are displayed.
4. Select the graph that you wish to migrate and click open the additional options menu as shown:



5. Click **Convert to SQL Graph** in the context menu.

   The **Confirm Migration to SQL Property Graph** window opens as shown:



6. Click **Confirm**.

   If the migration operation is successful, then both the newly created SQL property graph and the renamed original PGQL property graph are displayed on the Graphs page. Otherwise, only the PGQL property graph in its original state is displayed.

Manage RDF Graphs

You can explore and validate an RDF graph or an RDF graph collection in Graph Studio.

Topics:

- Explore and Validate an RDF Graph
- Explore and Validate an RDF Graph Collection

Explore and Validate an RDF Graph

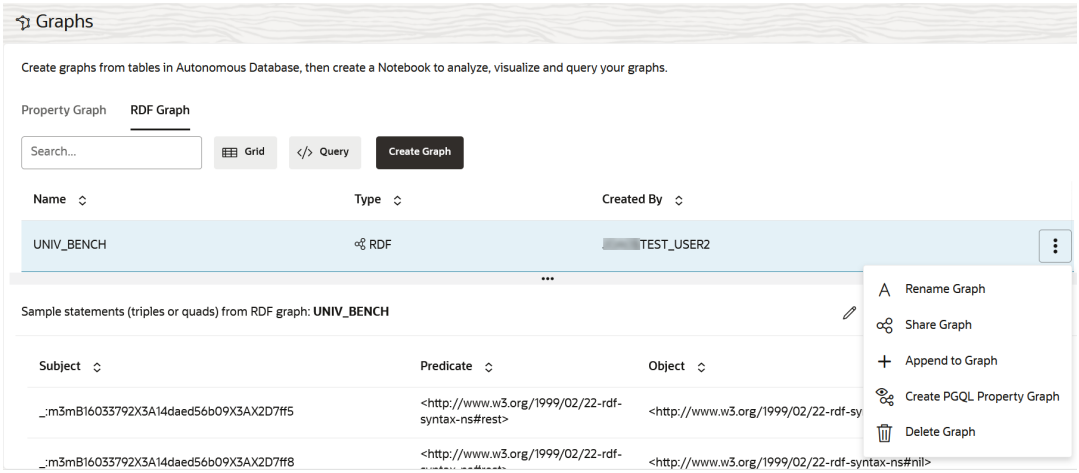
You can view the list of RDF graphs to which you have access in Graph Studio and explore their properties.

Also, you can execute SPARQL queries on an RDF graph in the Query Playground page.

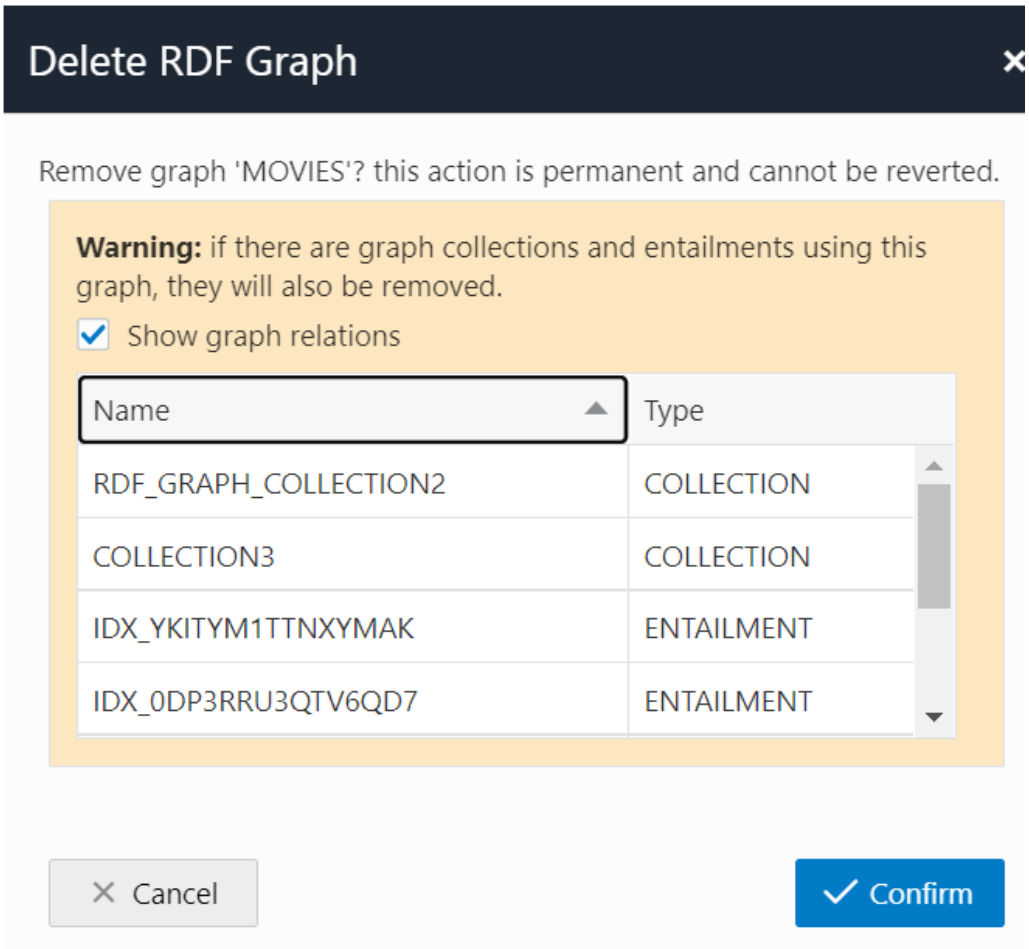
1. Navigate to the **Graphs** page.
2. Select the **RDF Graph** tab.

   All the RDF graphs to which you have access are displayed.
3. Select the required row having the **Type** as **RDF**.

   The graph properties are displayed on the bottom panel as shown:



   You can view the RDF statements that are loaded for the graph.
4. Optionally, click open the additional options menu to perform any of the following actions.
   - Click **Rename Graph** to rename an RDF graph.
   - Click **Share Graph** to share an RDF graph with another user. See Share an RDF Graph for more information.
   - Click **Append to Graph** to append RDF data obtained from OCI Object Storage to an existing RDF graph. See Append RDF Data to an RDF Graph for more information.
   - Click **Create PGQL Property Graph** to create a property graph from an RDF graph.
   - Click **Delete** to delete an RDF graph.
     It is important to note that when an RDF graph is deleted, Graph Studio removes all the RDF graph collections and entailments using this RDF graph. An appropriate warning is displayed as shown:



5. Optionally, click the **</> Query** button and run any SPARQL query on the selected RDF graph type in the Query Playground page.

   For example:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX ms: <http://www.example.com/moviestream/>
SELECT DISTINCT ?gname
WHERE {
?movie ms:actor/ms:name "Keanu Reeves" ;
ms:genre/ms:genreName ?gname .
}
ORDER BY ASC(?gname)
```

   The query gets executed and the resulting query output is displayed.

Append RDF Data to an RDF Graph

You can import RDF data from the OCI Object Storage and append this data to an existing RDF graph in Graph Studio.

To append RDF data to an RDF graph, perform the following steps:

1. Navigate to the **Graphs** page.

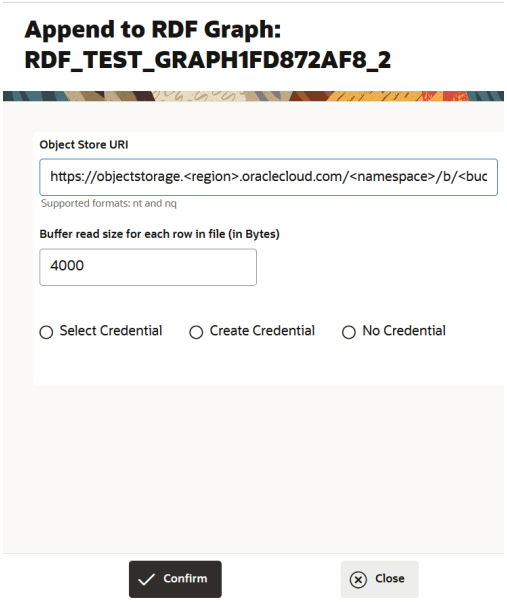
2. Select the **RDF Graph** tab.

   You can see the list of RDF graphs to which you have access.

3. Select the RDF graph to which additional RDF data needs to be appended.

4. Click **Append to Graph** from the additional options menu.

   The **Append to RDF Graph** slider opens as shown:



5. Enter the **Object Store URI** path or the **Pre-Authenticated Request URL** to access the RDF object store in your OCI bucket.

   See Get the URI or Pre-Authenticated Request URL to Access the Object Store for more information.

6. Choose one of the following credential options:

   - Selecting an existing credential:
     a. Click **Select Credential**.
     b. Select a credential name from the **Oracle Cloud Infrastructure Credentials** drop-down list.
   - Creating a new credential:
     a. Click **Create Credential**.
     b. Enter a **Credential Name**.
     c. Enter your **Oracle Cloud Infrastructure User Name**.
     d. Enter the **Auth Token** value.
   - Click **No Credential** to access the object store using pre-authenticated request URL.

7. Click **Confirm**.

   The job to load data from the given RDF data source gets initiated. On successful completion of the job, the new data gets appended to the RDF graph.

   You can verify by selecting the RDF graph to which the RDF data was appended on the Graphs page. The bottom panel displays the RDF statements for both the initial and appended RDF data.

Share an RDF Graph

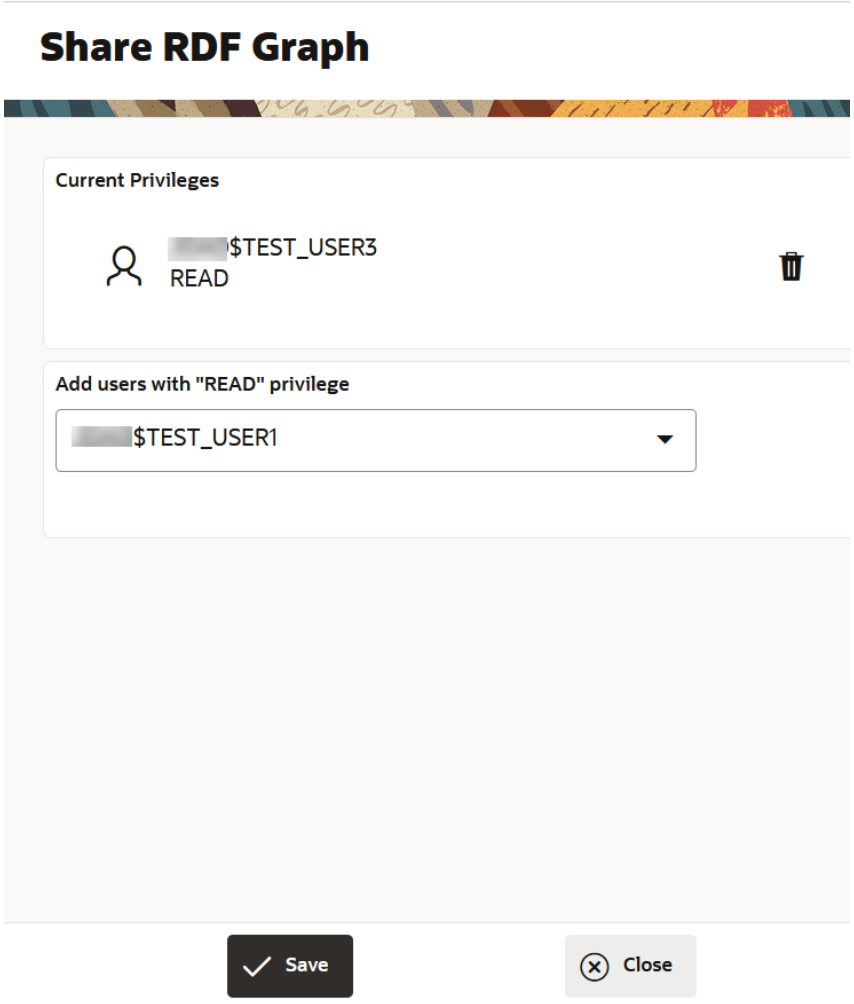
You can share your RDF graph or RDF graph collection to allow other users to run SPARQL queries on the graph.

In order to query a shared RDF graph or RDF graph collection, the specified user must have READ privilege on the graph.
Perform the following steps for sharing RDF graphs.

1. Navigate to the **Graphs** page.
2. Select the **RDF Graph** tab.

   All the RDF graphs and RDF graph collections to which you have access are displayed.
3. Select the required graph row.
4. Select the **Share Graph** option either from the additional options menu or by directly clicking the icon in the bottom panel of the Graphs page.

   The **Share RDF Graph** slider opens as shown.



   All existing users who already have `READ` privilege on the graph are shown listed under **Current Privileges**.
5. Select the user with whom you intend to share the graph from the drop-down.

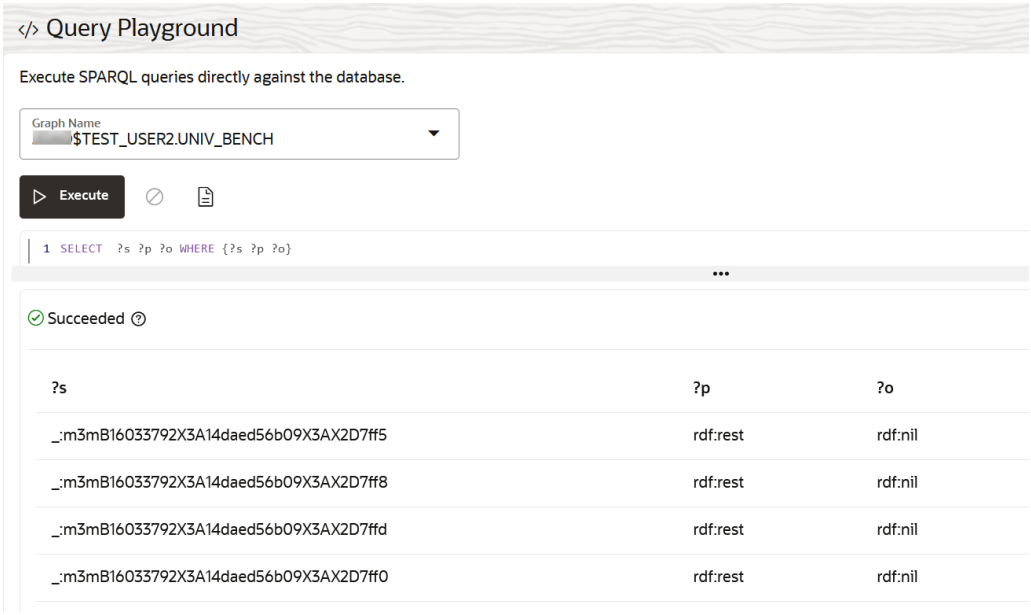
For example, in the preceding figure, the logged user (*$TEST_USER2*) shares the RDF graph with a different user, *$TEST_USER1*.

6. Click **Save** to share the RDF graph with the user.

   The **Current Privileges** section gets updated and displays the new user with whom the graph is shared.

   Also, note the following:

   - The new user can access the shared graph on the **Graphs** page only for querying purpose. All other graph actions such as **Rename Graph**, **Append to Graph**, **Share Graph**, **Create PGQL Property Graph**, and **Delete Graph** remain disabled for the user.
   - The user can run SPARQL queries on the shared graph in the **Query Playground** page. For example:



     As seen in the preceding figure, the shared graph appears in the drop-down along with its owner name.

   - Similarly, the user can also query the shared graph using the RDF interpreter in a notebook paragraph.

   - If you want to revoke the graph sharing privilege for a specific user, then click the icon against the user in the **Share RDF Graph** slider.

Explore and Validate an RDF Graph Collection

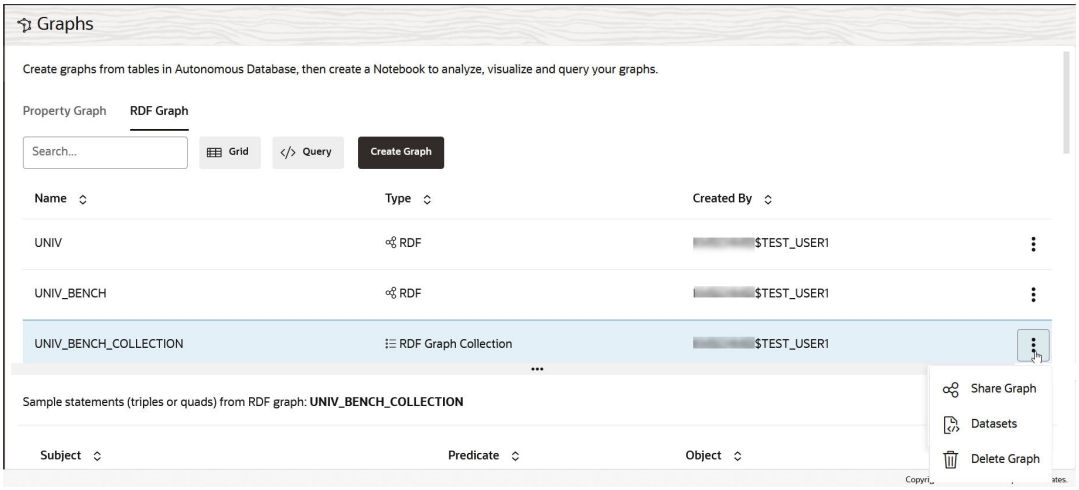
You can view the list of RDF graph collections to which you have access in Graph Studio and explore their properties.

Also, you can execute SPARQL queries on an RDF graph collection in the Query Playground page.

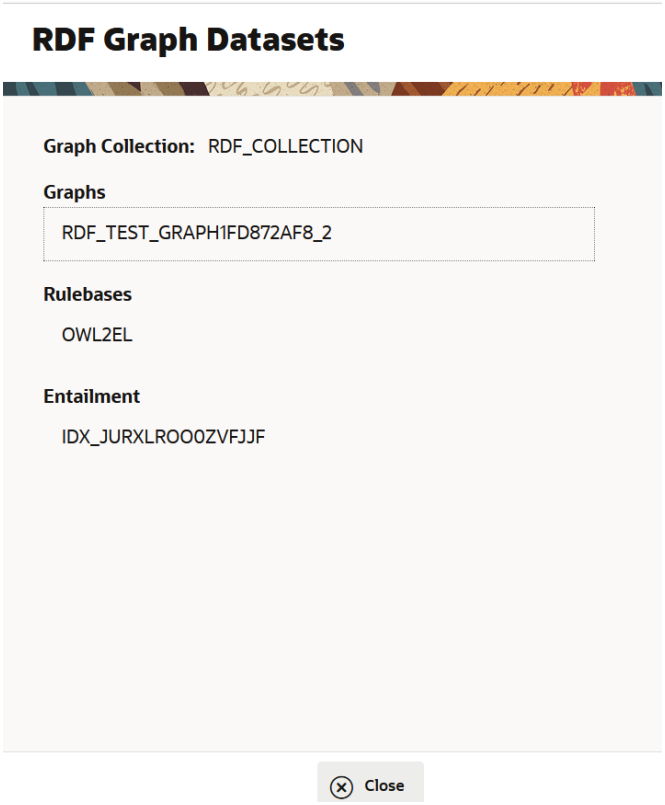
1. Navigate to the **Graphs** page.
2. Select the **RDF Graph** tab.

   All the RDF graphs and RDF graph collections to which you have access are displayed.
3. Select the required row having the **Type** as **RDF Graph Collection**.

   The graph collection properties are displayed on the bottom panel as shown:



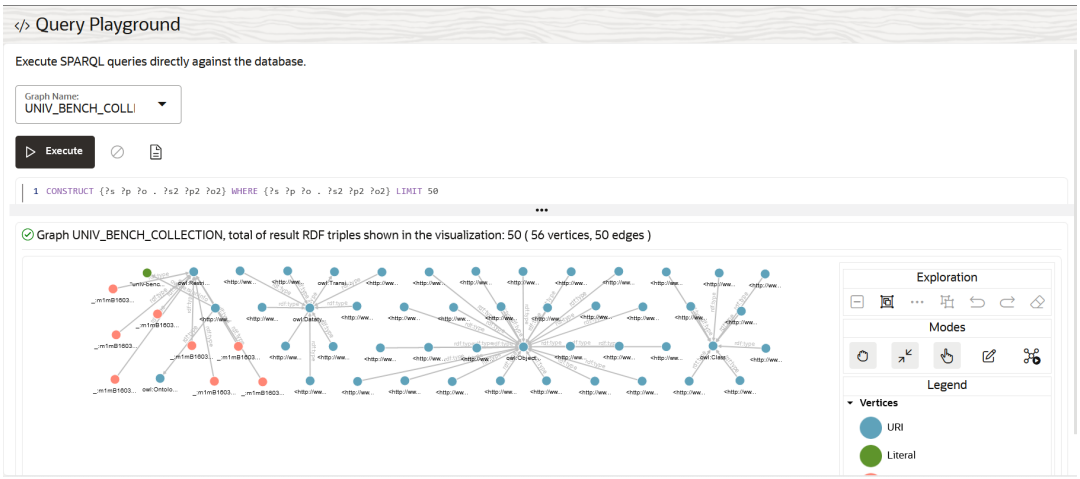
   You can view the RDF statements that are loaded for the graph collection.
4. Optionally, click open the additional graph options menu to perform any of the followings actions.
   - Click **Share Graph** to share an RDF graph collection with another user. See Share an RDF Graph for more information.
   - Click **Datasets** to view the summary of the selected RDF graph collection as shown:



- Click **Delete** to delete an RDF graph collection.

5. Optionally, click the **</> Query** button and execute any SPARQL query on the selected RDF graph collection in the Query Playground page.

   For example:



   The query is executed successfully and the resulting query output is displayed.

7

Work with Notebooks in Graph Studio

After you create a graph, you can analyze it and visualize the results by using a notebook.

> **⚠ Caution**
>
> Different Graph Studio users working on the same Autonomous AI Database Serverless instance can share the same CPU and memory resources when executing code in notebooks. Therefore, while designing your applications, it is recommended that you consider ways to mitigate any potential risks caused by shared hardware resources.

> **ⓘ Note**
>
> The graph visualization panel in the notebook paragraphs is redesigned to enhance user experience. However, if you wish to use the previous graph visualization interface, select **Preferences** from the username drop-down menu (on the top right) and disable the **Enable Oracle Graph Visualization Library** option.

Topics

- Example Notebooks

About Notebooks

A notebook is an interactive, browser-based object that enables data engineers, data analysts, and data scientists to be more productive by developing, organizing, executing, and sharing code, and by visualizing results without using the command line or needing to install anything. Notebooks enable you to execute code and to work interactively with long workflows.

You can create any number of **notebooks**, each of which can be a collection of documentation, snippets of code, and other visualizations. You can enter your input in **paragraphs**, each of which is configured to be run with a particular **interpreter**. See Available Notebook Interpreters to view the different notebook interpreters supported in Graph Studio.

In order to run the notebook paragraphs using the interpreters, Graph Studio must attach itself to an internal compute environment. This attachment happens implicitly when you open a notebook. See About Implicit Environment Creation Through Notebooks for more information.

After running a notebook paragraph, you can display the results in different ways, such as tables, charts, or as an interactive graph.

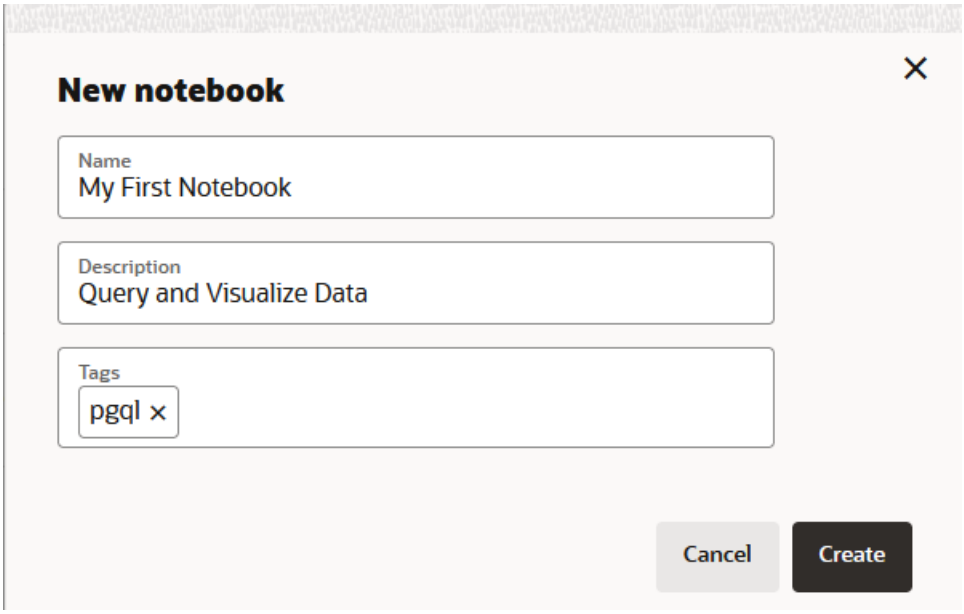
Create a Notebook

You can create a notebook to query, analyze and visualize a graph.

The following are the steps to create a notebook:

1. Click **Notebooks** on the left navigation menu and navigate to the Notebooks page.
2. Click **Create** on the top-right side of the page.

   The **Create Notebook** window opens.



3. Enter the **Name** of the notebook.

   Notebooks can be organized into a directory hierarchy. To create a new directory or to add or to move a notebook to a directory, simply give the notebook a name with slashes to indicate the directory structure.

For example, the notebook name `dir1/dir2/MyNotebook` will create a notebook named `MyNotebook` inside a directory `dir2`, which is inside a root directory `dir1`.

4. Optionally enter **Description** and **Tags**.
5. Click **Create**.

   This creates a new notebook which opens to a blank paragraph page.

Export a Notebook

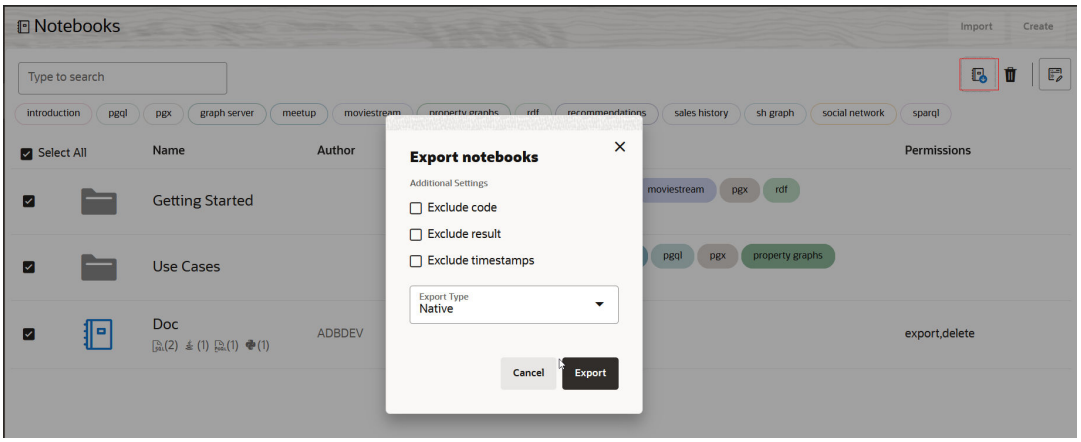
You can export one or more selected notebooks from Graph Studio to your local system.

You can choose to export the notebook in Native (`.dsnb`) file format, Jupyter (`.ipynb`) format, Zeppelin (`.zpln`) format, or HTML (`.html`) format. However, any functionality (such as tags, layout, dynamic forms, and so on) that is not supported by the chosen format will not be exported.
Perform the following steps to export a notebook:

1. Navigate to the **Notebooks** page.
2. Click **Select Notebooks** on the top right corner of the page.
3. Select one or more notebooks that you wish to export and click **Export Notebooks** (as shown highlighted in the following figure).

   Alternatively, to export an individual notebook, you can click open a specific notebook and click **Export Notebook** in the notebook toolbar at the top of the page.

   The **Export notebooks** window opens as shown:



4. Select the **Export Type**.
5. Optionally, select any **Additional Settings** options.
6. Click **Export**.

   The notebooks are exported and saved in your local system.

Find a Notebook

You can search for a notebook on the Notebooks page.

On the Notebooks page, you can use the search bar to search for a notebook by title, description, or tags. Additionally you can use keyboard shortcuts when working with notebooks.

1. Click **Notebooks** on the left navigation menu and navigate to the Notebooks page.
2. Enter the **name** of the notebook to find in the search bar.

   This opens the desired notebook.

Import a Notebook

You can import previously exported notebooks into Graph Studio from your local system.

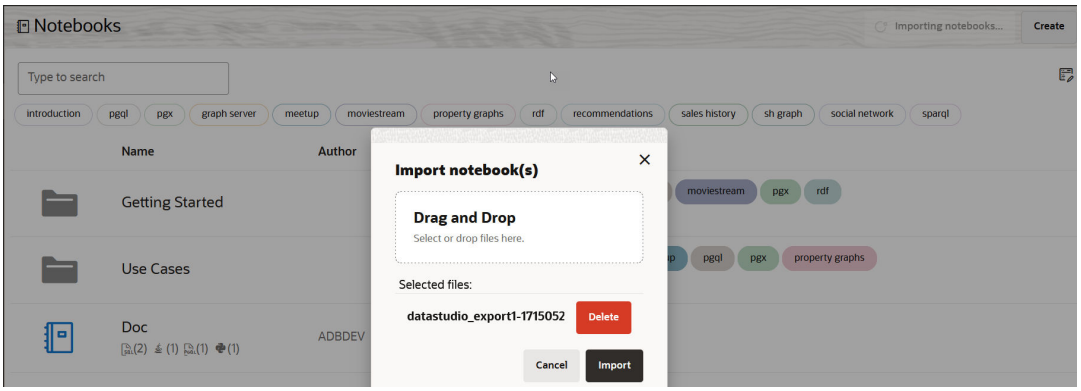
The following file formats are supported for importing a notebook:

- `.dsnb`: Native file format
- `.zpln`: Zeppelin file format
- `.ipynb`: Jupyter file format

Perform the following steps to import a notebook:

1. Navigate to the **Notebooks** page.
2. Click **Import** on the top right corner of the page.

   The **Import notebook(s)** window opens as shown:



3. Select one or more files from your local system or drag and drop the required files in the **Drag and Drop** section.
4. Optionally, review and verify the **Selected files**. Click **Delete** if you wish to remove a selected file.
5. Click **Import**.

   The files are imported as notebooks in Graph Studio.

Move a Notebook

You can move a notebook to another directory in Graph Studio.

Notebooks can be moved from:

- the notebooks main workspace in to a directory or conversely
- one directory to another

The following are the steps to move a notebook:

1. Navigate to the **Notebooks** page.
2. Click to open the **notebook** you want to move.
3. Click the **Modify Notebook** icon on the notebook toolbar at the top of the page.

   The window to modify the notebook details opens.
4. Enter a **Name** with the new directory path as required. This path determines the destination directory where you want to move the notebook.

   > **Note**
   >
   > Notebooks can be organized into a directory hierarchy. To create a new directory or to add or to move a notebook to a directory, simply give the notebook a name with slashes to indicate the directory structure.

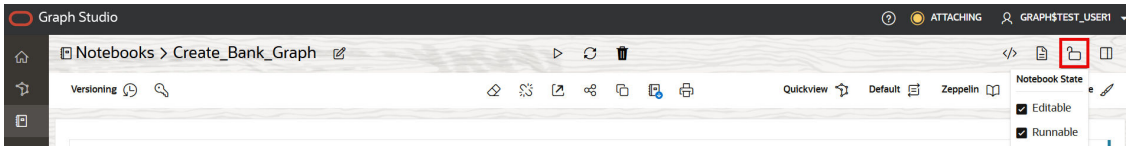
   For example, the notebook name `dir1/dir2/MyNotebook` will create a notebook named `MyNotebook` inside a directory `dir2`, which is inside a root directory `dir1`.
5. Optionally, enter the **Description** and the **Tags**.
6. Click **Save**.

   The notebook is moved to the destination directory.

Notebook States

When sharing a notebook, you can control the actions that a user can perform in the notebook by setting up the notebook state.

You can view the notebook state by clicking the **Update Notebook State** icon on the top right of the notebook as shown highlighted in the following figure:



You can configure the notebook state by selecting or deselecting the checkboxes for **Editable** and **Runnable** options. Depending on what actions you wish the users to perform, you can set any one of the following three states:

- **Editable** and **Runnable** (default): This allows a user to edit and run the notebook paragraphs.
- **Non-editable** and **Runnable**: This allows a user to run the notebook paragraphs, but the user cannot make any changes in the notebook.
- **Non-editable** and **Non-runnable**: This disallows a user to edit or run the notebook paragraphs.

Also, note the following:

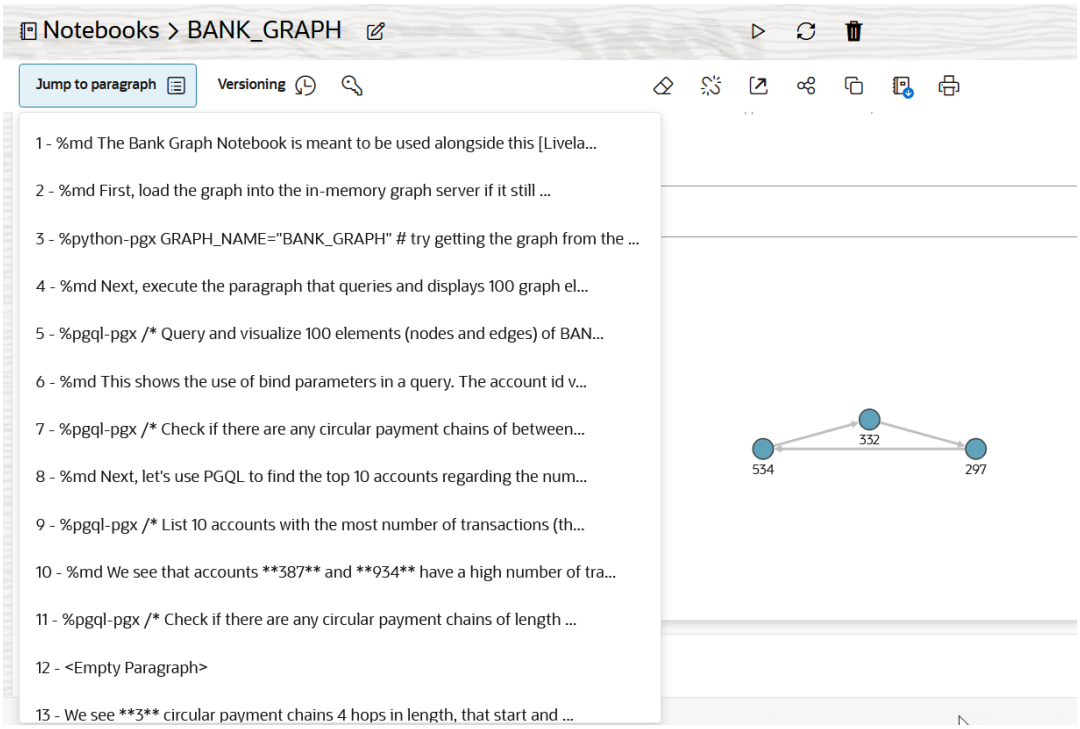
- In a non-editable notebook state, although a user cannot edit a notebook paragraph, some actions like changing the notebook layout, paragraph visibility, and paragraph results are allowed. However, these changes are not persistent.
- **Editable** and **Non-runnable** state is not supported.

Jump to a Paragraph

You can jump to a specific paragraph inside a notebook.

1. Navigate to the **Notebooks** page and click open the required notebook.
2. Click **Jump to paragraph** on the top left of the notebook toolbar at the top of the page.

   A drop-down menu listing all the paragraphs in the notebook opens as shown:



   Note that each paragraph is displayed by its title. If a paragraph is untitled, then a snippet of the first line of code or a placeholder (`<Empty Paragraph>`) is displayed.

3. Select the desired paragraph from the drop-down menu.

   The control shifts to the selected paragraph.

Available Notebook Interpreters

An interpreter executes code input and renders the output visually.

The following types of interpreters are supported:

> **Note**
>
> Graph Studio allows you to configure memory for the interpreters. See Manually Manage the Compute Environment for more information.

Topics

Markdown Interpreter

You can format text using Markdown interpreter in a notebook paragraph.

Markdown paragraphs start with `%md` and accept Markdown syntax as input. When executed, the underlying Markdown interpreter converts the input into HTML output. You can use the Markdown interpreter to explain your notebook in a formatted way and to add media elements like images or even videos.

> **Tip**
>
> You can hover over the bottom part of a notebook paragraph and click the  **Add Markdown Paragraph** icon to open a Markdown paragraph instantly in the notebook.

The following is an example of a Markdown paragraph:

```
%md
# My First Notebook
This is my first paragraph
```

Java (PGX) Interpreter

Java (PGX) paragraphs start with `%java-pgx` and expose the full Java language (based on JDK 11) as well as all the available Java (PGX) APIs.

See the Javadoc for more information on the Java APIs.

> **Tip**
>
> You can hover over the bottom part of a notebook paragraph and click the  **Add Java-PGX Paragraph** icon to open a Java (PGX) paragraph instantly in the notebook.

Some variables are built-in to make interaction with PGX easier:

- `session`: the `PgxSession` object bound to your user. You can access all graphs currently loaded into memory via the `session` object. Note that sessions time out after a while of not being used. A new session will be created when you log back in to the notebook; thus, the underlying session ID is not always the same.
- `instance`: the `ServerInstance` pointing to the PGX server.
- `visualQuery`: a helper object to convert PGQL queries into visualizable output.

The following imports are available on all Java (PGX) paragraphs:

```
import java.io.*
import java.util.concurrent.TimeUnit
import org.apache.commons.io.*
import oracle.pgx.common.*
import oracle.pgx.common.mutations.*
import oracle.pgx.common.types.*
import oracle.pgx.api.*
import oracle.pgx.api.admin.*
import oracle.pgx.config.*
import oracle.pg.rdbms.pgql.*
import oracle.pg.rdbms.pgql.pgview.*
import oracle.pgx.api.filter.*
import oracle.pgx.api.PgxGraph.SortOrder
import oracle.pgx.api.PgxGraph.Degree
import oracle.pgx.api.PgxGraph.Mode
import oracle.pgx.api.PgxGraph.SelfEdges
import oracle.pgx.api.PgxGraph.MultiEdges
import oracle.pgx.api.PgxGraph.TrivialVertices
```

The following is an example of a Java (PGX) paragraph:

```
%java-pgx
var g = session.getGraph("MY_FIRST_GRAPH") // reference in-memory graphs by
name
session.createAnalyst().pagerank(g) // run algorithms
```

You can also define new helper classes/functions inside paragraphs. For example:

```
%java-pgx
import java.lang.Math // import

// can define new classes
public class Functions {
    public static double haversine(double lat1, double lon1, double lat2,
double lon2) {
        double delta_lon = (lon2 - lon1) * Math.PI / 180;
        double delta_lat = (lat2 - lat1) * Math.PI / 180;
        double a = Math.pow(Math.sin(delta_lat / 2 ), 2) + Math.cos(lat1 *
Math.PI / 180) * Math.cos(lat2 * Math.PI / 180) *
Math.pow(Math.sin(delta_lon / 2), 2);
        double c = 2 * Math.asin(Math.sqrt(a));
        double r = 6371; // Radius of the Earth in kilometers. Use 3956 for
miles
        return c * r;
    }
}

Functions.haversine(30.26, 97.74, 48.13, 11.58)
```

Internally, the Java (PGX) interpreter operates on the same PGX session as the Python (PGX) interpreter. So, any analysis results computed in Python (PGX) paragraphs are available for querying in subsequent Java (PGX) paragraphs.

The following example show the PageRank values computed on a graph in a Python (PGX) paragraph. The `pagerank` property on the graph is then queried in the subsequent Java (PGX) paragraph.

```
%python-pgx
g = session.get_graph("MY_FIRST_GRAPH")
analyst.pagerank(g,tol=0.001,damping=0.85,max_iter=100,norm=False,rank='pagera
nk')


%java-pgx
session.getGraph("MY_FIRST_GRAPH").queryPgql("SELECT x.pagerank FROM MATCH
(x)").print(out,10,0)
```

See Known Issues for Graph Studio to learn about any known problems when executing a Java (PGX) paragraph.

Python (PGX) Interpreter

Python (PGX) paragraphs start with `%python-pgx` and allows you to use the available PyPGX APIs.

See the Python API Reference for more information on PyPGX APIs.

> **Tip**
>
> You can hover over the bottom part of a notebook paragraph and click the  **Add Python-PGX Paragraph** icon to open a Python (PGX) paragraph instantly in the notebook.

The following variables are built-in for easier PGX interaction when using a Python paragraph:

- `session`: the `PgxSession` object bound to your user. You can access all graphs currently loaded into memory via the `session` object. Note that sessions time out after a while of not being used. A new session will be created when you log back in to the notebook; thus, the underlying session ID is not always the same.
- `instance`: the `ServerInstance` pointing to the PGX server.
- `visual_query`: a helper object to convert PGQL queries into visualizable output.
- `analyst`: a helper object providing access to all built-in graph analytics such as PageRank and Betweenness Centrality.

The following import is available by default on all Python (PGX) paragraphs:

```
import pypgx
```

Also, the Python (PGX) interpreter supports the following Python libraries. However, you must import these modules in order to use them in a Python (PGX) paragraph.

- `NumPy`
- `scikit-learn`
- `oracledb`
- `Matplotlib`
- `pandas`
- `SciPy`
- `requests`
- `openpyxl`

The following is an example of a Python (PGX) paragraph which runs a built-in algorithm to counts the number of triangles inside a graph:

```
%python-pgx
# Reference in-memory graphs by name
graph = session.get_graph("FIRST_GRAPH")
# Running an algorithm to determine the number of triangles in a graph
analyst.count_triangles(graph, True)
```

You can also define new helper classes/functions inside Python paragraphs. For example:

```
%python-pgx
import math
# Define helper classes/functions
class Functions:
    def haversine (lat1, lon1, lat2, lon2):
```

```
       delta_lon = (lon2 - lon1) * math.pi/180
       delta_lat = (lat2 - lat1) * math.pi/180
       a = math.pow(math.sin(delta_lat/2),2) + math.cos(lat1 * math.pi/180) *
math.cos(lat2 * math.pi / 180) * math.pow(math.sin(delta_lon / 2), 2)
       c = 2 * math.asin(math.sqrt(a))
       r = 6371 #  Radius of the Earth in kilometers. Use 3956 for miles
       return c * r
Functions.haversine(30.26, 97.74, 48.13, 11.58)
```

Internally, the Python (PGX) interpreter operates on the same PGX session as the Java (PGX) interpreter. So, any analysis results computed in Java (PGX) paragraphs are available for querying in subsequent Python (PGX) paragraphs.

The following example show the PageRank values computed on a graph in a Java (PGX) paragraph. The `pagerank` property on the graph is then queried in the subsequent Python (PGX) paragraph.

```
%java-pgx
var g = session.getGraph("MY_FIRST_GRAPH")
session.createAnalyst().pagerank(g)
```

```
%python-pgx
session.execute_pgql("SELECT x.pagerank FROM MATCH (x) ON
MY_FIRST_GRAPH").print()
```

PGQL (PGX) Interpreter

You can run PGQL queries that are supported in the graph server (PGX) in your notebook paragraphs.

See the PGQL Specification for more information on PGQL queries.

PGQL (PGX) paragraphs start with `%pgql-pgx` and accept PGQL queries supported in the graph server(PGX) as input.

> **Tip**
>
> You can hover over the bottom part of a notebook paragraph and click the  **Add PGQL-PGX Paragraph** icon to open a PGQL (PGX) paragraph instantly in the notebook.

The following is an example of a PGQL(PGX) paragraph:

```
%pgql-pgx
SELECT v, e FROM MATCH (v)-[e]->() ON MY_FIRST_GRAPH
```

Internally, the PGQL-PGX interpreter operates on the same PGX session as the Java (PGX) interpreter or the Python (PGX) interpreter. So, any analysis results computed in Java (PGX) paragraphs or Python (PGX) paragraphs are available for querying in the subsequent PGQL (PGX) paragraphs.

For example, the vertex ranking computed for each vertex using the `Vertex Betweenness Centrality` algorithm in the Java (PGX) paragraph is used for querying in the following PGQL (PGX) paragraph:

```
%java-pgx
var g = session.getGraph("MY_FIRST_GRAPH")
session.createAnalyst().approximateVertexBetweennessCentrality(g, 3)


%pgql-pgx
SELECT city, e
FROM MATCH (city) -[e] -> () ON MY_FIRST_GRAPH
ORDER BY city.approx_betweenness
```

PGQL (RDBMS) Interpreter

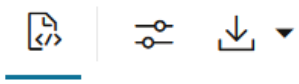
You can run PGQL queries directly against your property graph data in database using the PGQL-RDBMS interpreter in Graph Studio.

In addition to creating the property graphs from the Graphs page, you can now create these graphs directly in the database using the PGQL-RDBMS interpreter.

> **Tip**
>
> You can hover over the bottom part of a notebook paragraph and click the  **Add PGQL-RDBMS Paragraph** icon to open a PGQL (RDBMS) paragraph instantly in the notebook.

The following example shows the creation of a **PGQL Property Graph** using the `CREATE PROPERTY GRAPH` statement in a PGQL (RDBMS) paragraph.

```
%pgql-rdbms
CREATE PROPERTY GRAPH bank_graph
    VERTEX TABLES (
      bank_accounts
        KEY (id)
        LABEL Accounts
        PROPERTIES (id, name)
    )
    EDGE TABLES (
      bank_txns
        KEY (txn_id)
        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_PGQL)
```

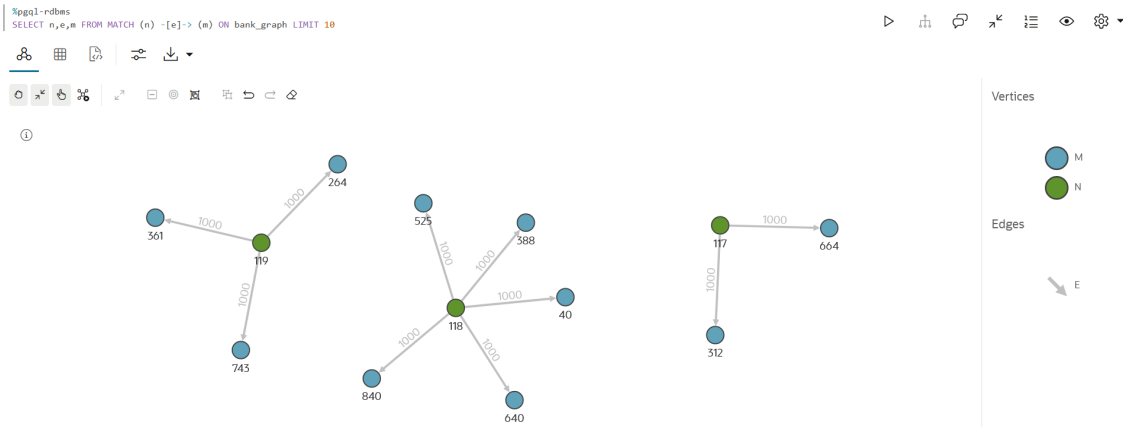


```
Graph successfully created
```

PGQL (RDBMS) paragraphs begin with `%pgql-rdbms`.

You can then run PGQL `INSERT`, `SELECT`, `UPDATE` or `DELETE` queries directly on the graph without having to load the graph into memory. See Executing PGQL Queries Against PGQL Property Graphs in *Oracle AI Database Graph Developer's Guide for Property Graph* for more information.

For example, the following figure shows the graph visualization output using a PGQL `SELECT` query on the **PGQL Property Graph** created in the earlier example:



Also, see the table in Supported PGQL Features and Limitations for more information on the supported PGQL functionalities for the graphs in the database.

Supported PGQL Features and Limitations

This section provides the complete list of supported and unsupported PGQL functionalities in PGQL queries that can be performed directly on the PGQL and SQL property graphs in the database and those that are run after loading the graphs into memory.

Feature	PGQL on RDBMS (PGQL Property Graph)	PGQL on RDBMS (SQL Property Graph[1])	PGQL on the Graph Server (PGX)
`CREATE PROPERTY GRAPH`	Supported	Supported	Supported Limitations: • No composite keys for vertices • Properties need to be column references; arbitrary property expressions are not supported unless the graph is first created in the database and then loaded into the graph server (PGX).
`DROP PROPERTY GRAPH`	Supported	Supported	Not Supported
Fixed-length pattern matching	Supported	Supported	Supported
Variable-length pattern matching goals	Supported: • Reachability • Path search prefixes: – `ANY` – `ANY SHORTEST` – `SHORTEST k` – `ALL` • Path modes: – `WALK` – `TRAIL` – `SIMPLE` – `ACYCLIC` Limitations: • Path search prefixes: – `ALL SHORTEST` – `ANY CHEAPEST` – `CHEAPEST k`	Not Supported	Supported: • Reachability • Path search prefixes: – `ANY` – `ANY SHORTEST` – `SHORTEST k` – `ALL SHORTEST` – `ANY CHEAPEST` – `CHEAPEST k` – `ALL` • Path modes: – `WALK` – `TRAIL` – `SIMPLE` – `ACYCLIC`

Feature	PGQL on RDBMS (PGQL Property Graph)	PGQL on RDBMS (SQL Property Graph[1])	PGQL on the Graph Server (PGX)
Variable-length pattern matching quantifiers	Supported: • * • + • ? • { n } • { n, } • { n, m } • { , m }	Not Supported	Supported: • * • + • ? • { n } • { n, } • { n, m } • { , m } Limitations: • ? is only supported for reachability • In case of ANY CHEAPEST and TOP k CHEAPEST, only * is supported
Variable-length path unnesting	Supported: • ONE ROW PER STEP Limitation: Quantifier * not supported Not supported: • ONE ROW PER VERTEX	Not Supported	Supported: • ONE ROW PER VERTEX • ONE ROW PER STEP Limitation: • * quantifier is not supported
OPTIONAL MATCH	Not supported	Not supported	Supported
GROUP BY	Supported	Supported	Supported
HAVING	Supported	Supported	Supported
Aggregations	Supported: • COUNT • MIN, MAX, AVG, SUM • LISTAGG • JSON_ARRAYAGG Limitations: • ARRAY_AGG	Supported: • COUNT • MIN, MAX, AVG, SUM • LISTAGG Not supported: • ARRAY_AGG • JSON_ARRAYAGG	Supported: • COUNT • MIN, MAX, AVG, SUM • LISTAGG • ARRAY_AGG Not Supported: • JSON_ARRAYAGG
DISTINCT • SELECT DISTINCT • Aggregation with DISTINCT (such as, COUNT(DISTINCT e.prop))	Supported	Supported	Supported
SELECT v.*	Supported	Not Supported	Supported
ORDER BY (+ASC/DESC), LIMIT, OFFSET	Supported	Supported	Supported

Feature	PGQL on RDBMS (PGQL Property Graph)	PGQL on RDBMS (SQL Property Graph[1])	PGQL on the Graph Server (PGX)
Data Types	All available Oracle RDBMS data types supported	All available Oracle RDBMS data types supported	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	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)`
		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)`
			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)	Supported: • Relational: +, -, *, /, %, - (unary minus) • Arithmetic: =, <>, <, >, <=, >= • Logical: `AND`, `OR`, `NOT` • String: || (concat)	Supported: • Relational: +, -, *, /, %, - (unary minus) • Arithmetic: =, <>, <, >, <=, >= • Logical: `AND`, `OR`, `NOT` • String: || (concat)

Feature	PGQL on RDBMS (PGQL Property Graph)	PGQL on RDBMS (SQL Property Graph[1])	PGQL on the Graph Server (PGX)
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`	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	Supported: • `IS NULL`, `IS NOT NULL` • `JAVA_REGEXP_LIKE` (based on `CONTAINS`) • `LOWER`, `UPPER` • `SUBSTRING` • `ABS`, `CEIL/CEILING`, `FLOOR`, `ROUND` • `EXTRACT` • `ID`, `VERTEX_ID`, `EDGE_ID` • `LABEL`, `LABELS`, `IS [NOT] LABELED` • `ALL_DIFFERENT` • `IN_DEGREE`, `OUT_DEGREE` • `CAST` • `CASE` • `IN` and `NOT IN` • `MATCHNUM` • `ELEMENT_NUMBER` • `IS [NOT] SOURCE [OF]`, `IS [NOT] DESTINATION [OF]` • `VERTEX_EQUAL`, `EDGE_EQUAL`
User-defined functions	Supported: • PL/SQL functions • Functions created via the Multilingual Engine (MLE)	Supported: • PL/SQL functions • Functions created via the Multilingual Engine (MLE)	Supported: • Java UDFs • JavaScript UDFs
Subqueries: • Scalar subqueries • `EXISTS` and `NOT EXISTS` subqueries • `LATERAL` subquery	Supported: • `EXISTS` and `NOT EXISTS` subqueries • Scalar subqueries • `LATERAL` subquery	Supported subqueries: • EXISTS • NOT EXISTS Not supported: • Scalar subqueries • `LATERAL` subquery	Supported
`GRAPH_TABLE` operator	Supported Extension: • `BASE GRAPHS` clause in `CREATE PROPERTY GRAPH` for creating graphs based on metadata of other graphs	Not supported	Supported
`INSERT/UPDATE/DELETE`	Supported	Not supported	Supported

Feature	PGQL on RDBMS (PGQL Property Graph)	PGQL on RDBMS (SQL Property Graph[1])	PGQL on the Graph Server (PGX)
INTERVAL literals and operations	Not supported	Not supported	Supported literals: • SECOND • MINUTE • HOUR • DAY • MONTH • YEAR Supported operations: • Add INTERVAL to datetime (+) • Subtract INTERVAL from datetime (-)

1 SQL Property Graphs are supported only in Oracle AI Database 26ai.

SPARQL (RDF) Interpreter

Graph Studio provides a SPARQL (RDF) interpreter which allows you to run SPARQL queries on an RDF graph in a notebook paragraph.

See SPARQL Protocol and RDF Query Language (SPARQL) for more information on W3C SPARQL 1.1 standard.

To use the SPARQL (RDF) interpreter, you must specify `%sparql-rdf` at the beginning of the notebook paragraph and then input the SPARQL query.

Tip

You can hover over the bottom part of a notebook paragraph and click the  **Add RDF Paragraph** icon to open an SPARQL (RDF) paragraph instantly in the notebook.

You can run the following types of SPARQL queries:

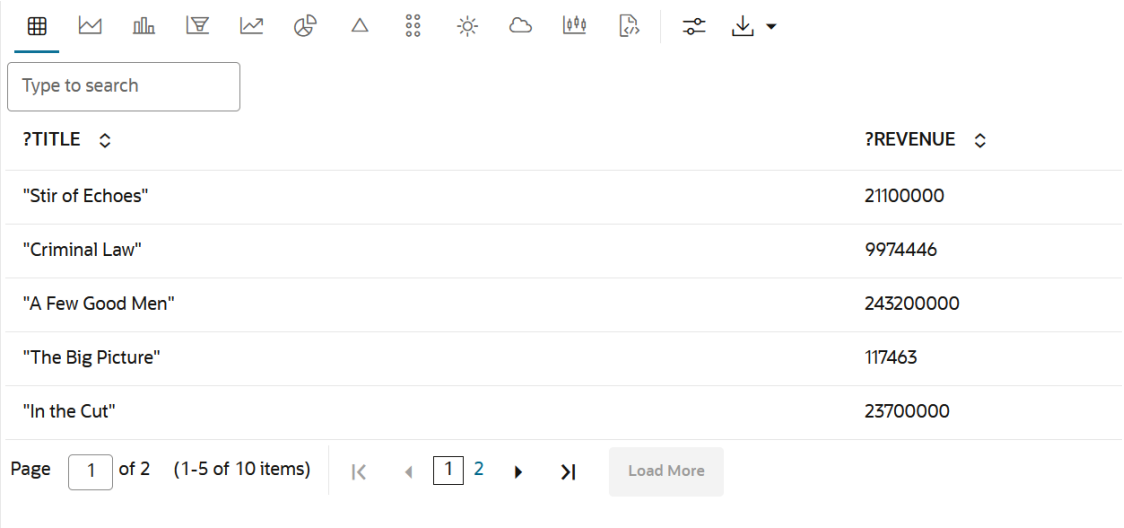
- `SELECT`
- `ASK`
- `CONSTRUCT`
- `DESCRIBE`
- `INSERT`, `DELETE`, `CLEAR`, and other supported SPARQL queries for graph update operations. See SPARQL 1.1 Update Specification for more information.

Also, note that execution of SPARQL `SELECT` and `ASK` queries return a tabular output and execution of SPARQL `CONSTRUCT` and `DESCRIBE` queries return a graph view of the resulting output.

If your user account is associated with just one RDF graph, then you can directly run the SPARQL query as shown:

```
%sparql-rdf
PREFIX  rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX  xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX   ms: <http://www.example.com/moviestream/>

SELECT ?title ?revenue
WHERE {
  ?movie ms:actor ?actor .
  ?actor ms:name "Kevin Bacon" .
  ?movie ms:title ?title .
  ?movie ms:grossInUSD ?revenue
}
```

The preceding `SELECT` SPARQL query is automatically applied on the default RDF graph existing in the account. The query aims to project the `title` and `revenue` in USD of all movies starring `"Kevin Bacon"`, using multiple triple patterns in the `WHERE` clause. On execution, the query output is displayed in a tabular format as shown:



In case you have multiple RDF graphs in your account, then a selection box is displayed when you run the first SPARQL query in the notebook. You can select the desired graph and then rerun the paragraph. This selection is automatically applied to all other SPARQL (RDF) paragraphs.

The following example performs a SPARQL update operation. The example uses a SPARQL `INSERT` query to add new data for a movie.

```
%sparql-rdf
#####################################################################
# Insert new data for Minions: The Rise of Gru
#####################################################################

PREFIX  rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX  xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX   ms: <http://www.example.com/moviestream/>
```

```
INSERT DATA {
  ms:movie_4004 ms:title "Minions: The Rise of Gru" ;
                ms:year "2022"^^xsd:decimal ;
                ms:openingDate "2022-07-01"^^xsd:date ;
                ms:runtimeInMin "87"^^xsd:decimal ;
                ms:director ms:entity_kyle%20balda ;
                ms:views "100"^^xsd:decimal .
}
```

SQL Interpreter

Graph Studio provides a SQL interpreter which allows you to run SQL statements in a notebook paragraph.

To use the SQL interpreter, you must specify `%sql` at the beginning of the notebook paragraph and then input the SQL statement. You can run only one SQL statement in a single paragraph.

> **Tip**
>
> You can hover over the bottom part of a notebook paragraph and click the  **Add SQL Paragraph** icon to open a SQL paragraph instantly in the notebook.

The database connection is established for the currently logged in user. For example, the following SQL statement retrieves the name of the user logged on to the database.

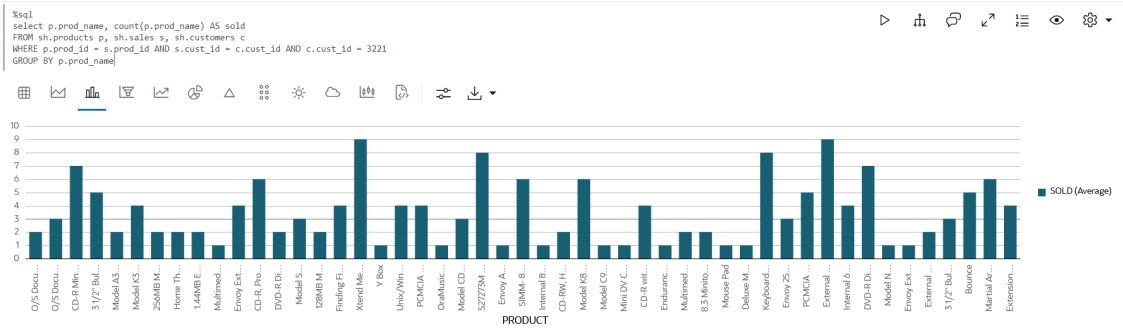
```
%sql
-- Get Current user
SELECT SYS_CONTEXT('USERENV','CURRENT_USER') FROM DUAL;
```

The following examples describe a few scenarios using the SQL interpreter.

Example: Visualization Using Charts

You can visualize any tabular output from a SQL query using charts in a notebook paragraph. For example, the following SQL query to determine the products bought by a specific customer, is visualized using a Bar Chart:

```
%sql
SELECT p.prod_name, count(p.prod_name) AS sold
FROM sh.products p, sh.sales s, sh.customers c
WHERE p.prod_id = s.prod_id AND s.cust_id = c.cust_id AND c.cust_id= 3221
GROUP BY p.prod_name;
```



Example: Creating, Querying, Visualizing, and Deleting SQL Property Graphs

If you are using an Autonomous AI Database instance with Oracle AI Database 26ai, then you can create, query, and visualize SQL property graphs using the SQL interpreter.

The following code uses the `CREATE PROPERTY GRAPH` DDL statements for creating a SQL property graph in a notebook paragraph:

```
%sql
CREATE PROPERTY GRAPH bank_sql_pg
  VERTEX TABLES (
    bank_accounts
      KEY (id)
      LABEL account
      PROPERTIES ALL COLUMNS
  )
  EDGE TABLES (
    bank_txns
      KEY (txn_id)
      SOURCE KEY (from_acct_id) REFERENCES bank_accounts (id)
      DESTINATION KEY (to_acct_id) REFERENCES bank_accounts (id)
      LABEL transfer
      PROPERTIES ALL COLUMNS
  );
```

You can query the SQL property graph using SQL graph queries.

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

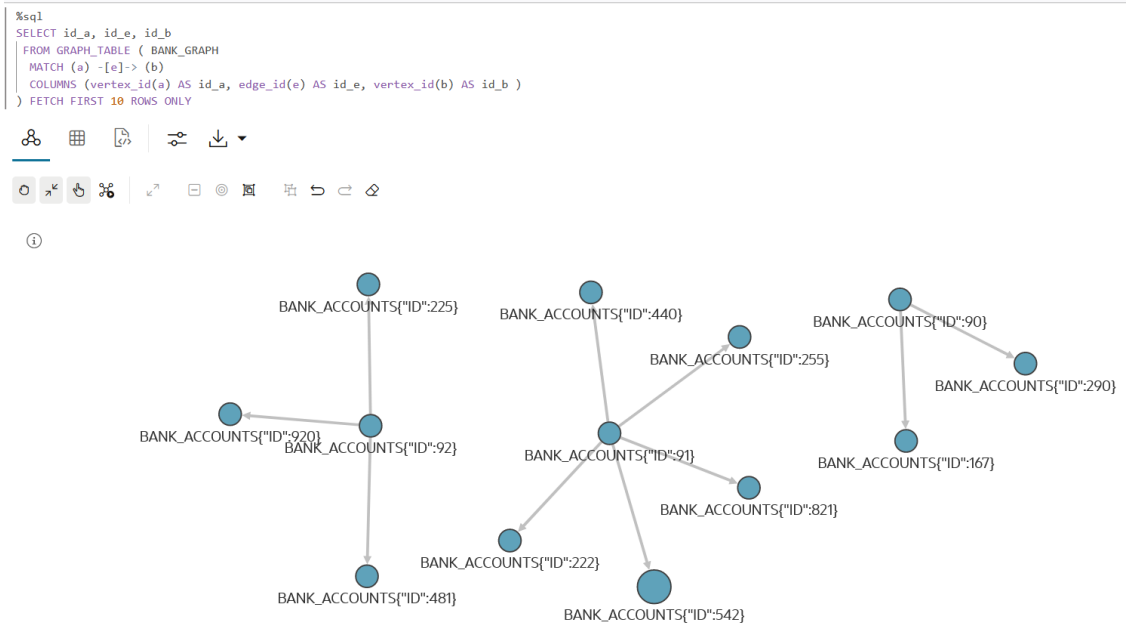
The preceding query produces the following output:

```
ACC_A    AMOUNT    ACC_B
816    8781    287
816    6381    590
816    9011    934
816    6890    289
816    4443    812
```

You can also visualize the output of SQL graph queries. In order to visualize the vertices and edges of the SQL graph query, you must return the vertex and edge IDs. For example:

```
SELECT id_a, id_e, id_b
FROM GRAPH_TABLE ( BANK_GRAPH
MATCH (a) -[e]-> (b)
COLUMNS (vertex_id(a) AS id_a, edge_id(e) AS id_e, vertex_id(b) AS id_b )
) FETCH FIRST 10 ROWS ONLY
```

Note that the `COLUMNS` clause in the preceding query uses the `VERTEX_ID` and `EDGE_ID` operators. The visualization output of the SQL graph query is as shown:



Finally, you can delete the SQL property graph using the `DROP PROPERTY GRAPH` DDL statement as shown:

```
%sql
DROP PROPERTY GRAPH bank_sql_pg;
```

> **See Also**
>
> - SQL DDL Statements for Property Graphs in *Oracle AI Database Graph Developer's Guide for Property Graph*
> - SQL Graph Queries in *Oracle AI Database Graph Developer's Guide for Property Graph*
> - Vertex and Edge Identifiers in *Oracle AI Database Graph Developer's Guide for Property Graph*

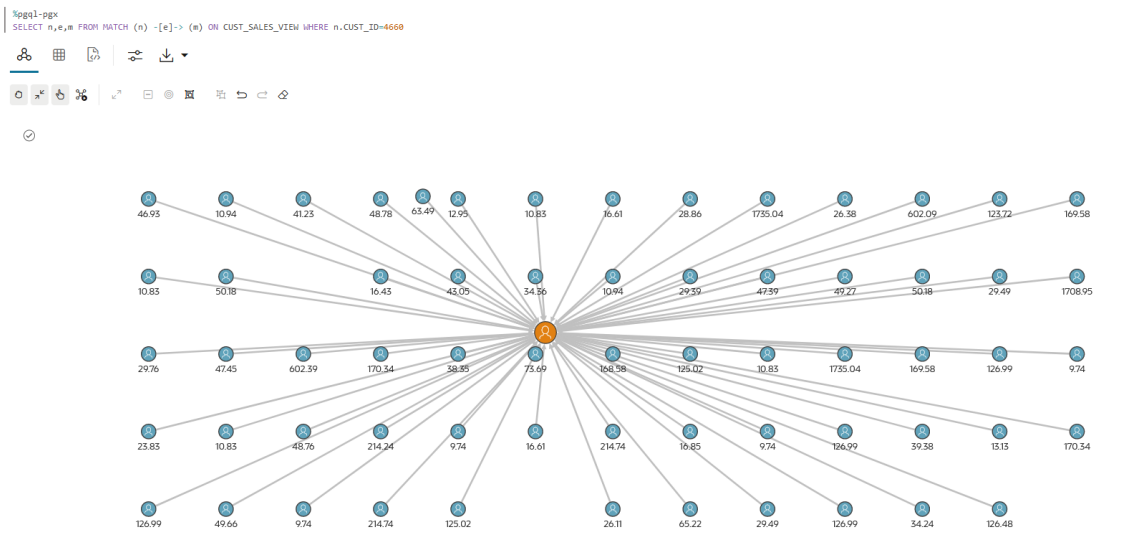
Example: Creating and Using Custom Database Views for PGQL Property Graphs

Another example scenario is to create custom database views using the SQL interpreter, which are then used to create a property graph. Note that this example scenario applies only for PGQL property graphs.

As shown in the following sequence of SQL paragraphs, database views are created on the `SALES` and `CUSTOMERS` tables in `SH` schema. Also, the primary key and foreign key constraints are defined for the views.

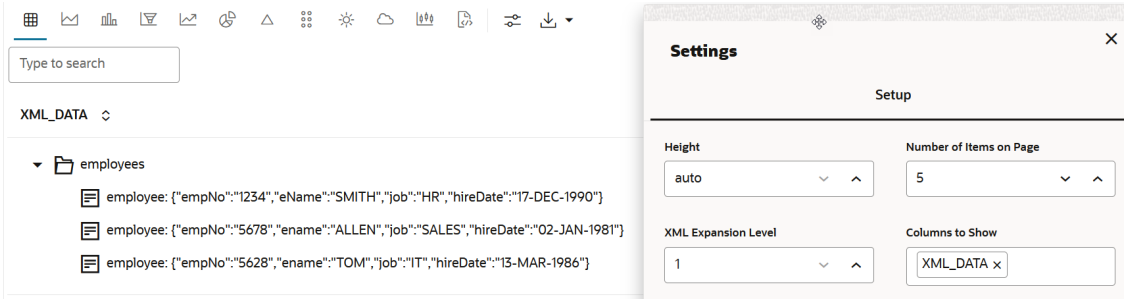
```
%sql
CREATE VIEW sh_customers
AS SELECT cust_id, cust_first_name, cust_last_name, country_id, cust_city,
cust_state_province
FROM sh.customers;


%sql
ALTER VIEW sh_customers
ADD CONSTRAINT shcustomers_id PRIMARY KEY (cust_id)
DISABLE NOVALIDATE;


%sql
CREATE VIEW sh_sales
AS SELECT rownum sale_id, cust_id, prod_id, channel_id, promo_id,
quantity_sold, amount_sold
FROM sh.sales;


%sql
ALTER VIEW sh_sales
ADD CONSTRAINT shsales_id PRIMARY KEY (sale_id)
DISABLE NOVALIDATE;


%sql
ALTER VIEW sh_sales
ADD CONSTRAINT shsale_cust_fk FOREIGN KEY (cust_id)
REFERENCES sh_customers DISABLE NOVALIDATE;
```

You can then create a **PGQL Property Graph** graph using these database views (see Create a Property Graph from Existing Relational Tables) and then perform graph visualizations in a PGQL (PGX) paragraph as shown:



Example: XML Support in Table Visualization

Graph Studio provides support for visualizing tabular data with `XMLType` and `CLOB` data type columns. The results of these columns are parsed and rendered as tree of items. You can modify the rendering by changing the **XML Expansion Level** in the table visualization settings. The default is 1.



Custom Algorithm (PGX) Interpreter

Using the custom algorithm (PGX) interpreter, you can write your own custom PGX graph algorithms in a notebook paragraph in Graph Studio.

A custom algorithm (PGX) paragraph starts with `%custom-algorithm-pgx` and a custom graph algorithm can be written using Java syntax. See the PGX Algorithm APIs in the Javadoc for more information.

On running the custom algorithm (PGX) paragraph, the algorithm gets compiled. You can then use the compiled algorithm in a Java (PGX) or Python (PGX) paragraph.

> **Tip**
>
> You can hover over the bottom part of a notebook paragraph and click the  **Add CUSTOM-ALGORITHM-PGX Paragraph** icon to open a custom algorithm (PGX) paragraph instantly in the notebook.

For example, consider the following graph algorithm:

```
%custom-algorithm-pgx
package oracle.pgx.algorithms;

import oracle.pgx.algorithm.annotations.GraphAlgorithm;
import oracle.pgx.algorithm.PgxGraph;
import oracle.pgx.algorithm.VertexProperty;
import oracle.pgx.algorithm.annotations.Out;

@GraphAlgorithm
public class IndegreeCentrality {
  public void indegreeCentrality(PgxGraph g, @Out VertexProperty<Integer>
indegreeCentrality) {
    g.getVertices().forEach(n ->
        indegreeCentrality.set(n, (int) n.getInDegree())
    );
  }
}
```

After running the preceding code, you can integrate the compiled algorithm (`indegreeCentrality`) in a Java (PGX) or Python (PGX) paragraph as shown:

- %java-pgx
- %python-pgx

%java-pgx

```
var graph = session.getGraph("HR_GRAPH")
var centrality = graph.createVertexProperty(PropertyType.INTEGER,
"centrality")
var algorithm = session.getCompiledProgram("indegreeCentrality")
algorithm.run(graph, centrality)
graph.queryPgql("SELECT x.centrality, x.last_name FROM MATCH (x:employees)
ORDER BY x.centrality DESC LIMIT 10").print(out,10,0)
```

%python-pgx

```
graph = session.get_graph("HR_GRAPH")
centrality = graph.create_vertex_property("integer", "centrality")
algorithm = session.get_compiled_program("indegreeCentrality")
algorithm.run(graph, centrality)
```

```
graph.query_pgql("SELECT x.centrality, x.last_name FROM MATCH (x:employees)
ORDER BY x.centrality DESC LIMIT 10").print()
```

The graph query produces the following output:

```
+------------------------+
| centrality | last_name |
+------------------------+
| 14         | King      |
| 9          | Kaufling  |
| 8          | Weiss     |
| 8          | Vollman   |
| 8          | Fripp     |
| 8          | Mourgos   |
| 7          | Kochhar   |
| 6          | Zlotkey   |
| 6          | Russell   |
| 6          | Cambrault |
+------------------------+
```

See Using Custom PGX Graph Algorithms in *Oracle AI Database Graph Developer's Guide for Property Graph* for more information.

Also, see Built-In Algorithms on GitHub for detailed information about the supported graph built-in algorithms.

Conda Interpreter

Using the Conda interpreter, you can create a custom Conda environment by installing specific third-party Python packages and use the activated environment in a Python(PGX) notebook paragraph.

Conda is an open source package management system and environment management system for Python. Conda supports multiple environments with different versions of Python and other libraries.

To use the Conda interpreter, you must specify `%conda` at the beginning of a notebook paragraph. See About the Default Conda Environment to learn more about the base Conda environment in Graph Studio.

The Conda environment and package management can be performed only by ADMIN users. An ADMIN user can be any graph-enabled user with `GRAPH_ADMINISTRATOR` role or the default ADMIN user in your Autonomous AI Database instance. The ADMIN user can create a Conda environment, install the required packages, and upload the environment. The uploaded environment is persisted internally and is shared only by the graph users. Other graph users can then simultaneously access, download, and work on one or more Conda environments in their respective notebook sessions.

> **Note**
>
> You do not have to install any additional third-party software through this Conda feature in order to use any of the graph features of Oracle Autonomous AI Database.

> **⚠ Caution**
>
> Oracle is not responsible for vulnerability management and license compliance of all the third-party Python packages installed in a Conda environment using this feature. It is solely your responsibility.

As a graph user, you can download and activate the preinstalled environment. You can then access the activated Conda environment from a Python(PGX) notebook paragraph to quickly develop and visualize analytical workloads. Also, you can switch between different preinstalled Conda environments.

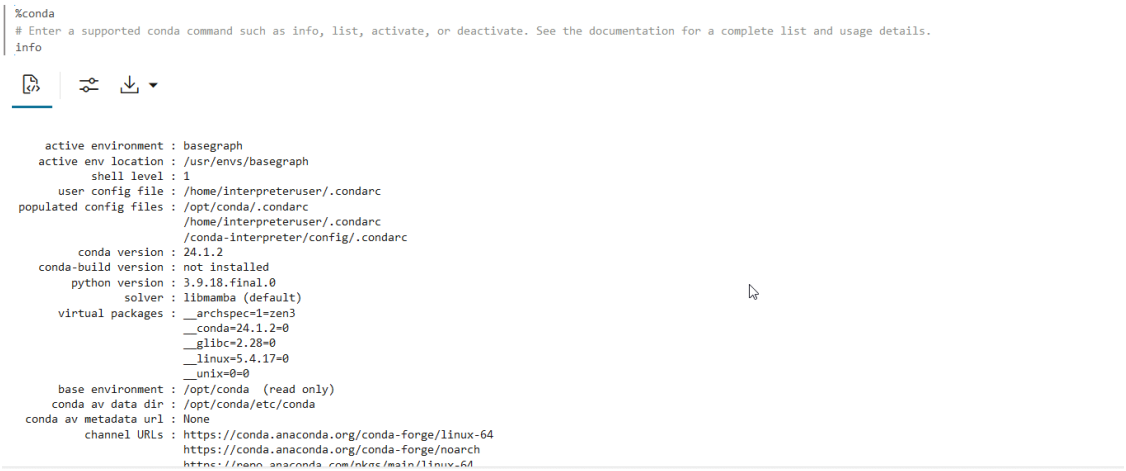
The following sections explain more on the supported Conda interpreter tasks:

Topics

About the Default Conda Environment

Graph Studio uses the `basegraph` environment as the default Conda environment.

For instance, before you start creating or downloading a Conda environment, run the Conda `info` command in a Conda paragraph:



As seen in the preceding output, the `basegraph` environment is set as the default Conda environment. To view the default packages in the `basegraph` environment, you can run the Conda `list` command.

It is important to note the following:

- It is recommended that you do not install any third-party Python library in the default `basegraph` environment.
- The `oracle-pypgx-client` package, which is required to work with PyPGX APIs, is available in the `basegraph` environment by default. Therefore, to work using this graph Python client library along with other external Python packages, you must create a Conda

environment by copying the default `basegraph` environment. See step-2 in Create and Publish a Conda Environment for an example.

Supported Conda Interpreter Tasks

You can learn the different tasks that are supported by the Conda interpreter in Graph Studio.

The following table describes the supported `conda` commands and the users authorized to perform these tasks:

Task	Command	Authorized Users
Create a new Conda environment using a specific Python version	`create -n <env_name> python==<python_version>`	• ADMIN[1]
Create a Conda Environment by copying the default `basegraph` environment	`copy-local-env -n <env_name>`	• ADMIN
Install an external package from public Conda channel in a Conda environment	`install -n <env_name> <package_name>`	• ADMIN
Uninstall a specific package from a Conda environment	`uninstall -n <env_name> <package_name>`	• ADMIN
Upload a Conda environment to internal storage	`upload <env_name> --description '<write_description>' -t <tag_name> <tag_value>`	• ADMIN
Get information about the Conda installation	`info`	• ADMIN • Graph User[2]
List the packages installed in the active environment	`list`	• ADMIN • Graph User
Search on a specific package in the Conda environment	`search <package_name>`	• ADMIN
Get specific command-line help	`<conda_command> --help`	• ADMIN • Graph User
Download and unpack a specific Conda environment from internal storage	`download <env_name> --skip-if-exists`	• ADMIN • Graph User
List all the uploaded Conda environments	`list-saved-envs`	• ADMIN • Graph User
List all the available Conda environments	`env list`	• ADMIN • Graph User
List the local Conda environments created by the user	`list-local-envs`	• ADMIN • Graph User
Activate a Conda environment	`activate <env_name>`	• ADMIN • Graph User
Deactivate a Conda environment	`deactivate`	• ADMIN • Graph User
Remove a Conda environment locally	`env remove -n <env_name>`	• ADMIN • Graph User
Delete a persisting Conda environment	`delete <env_name>`	• ADMIN

1 Default `ADMIN` user in your Autonomous AI Database instance or a graph-enabled user with `GRAPH_ADMINISTRATOR` role.
2 See Create a Graph User.

Create and Publish a Conda Environment

All administrative tasks for managing the Conda environment can be performed only by the ADMIN user.

The following example describes the steps to create a new Conda environment, install external Python packages, and persist the environment in internal storage. Note that these tasks can be performed only by the ADMIN user.

1. Navigate to the **Notebooks** page and open a new notebook.
2. Create a new Conda environment in a Conda paragraph.

> **Tip**
>
> You can hover over the bottom part of a notebook paragraph and click the **Add Conda Paragraph** icon to open a Conda paragraph instantly in the notebook.

The following describes two choices for creating a new Conda environment. You can choose the option that applies to you:

- To work with **PyPGX APIs and other external Python packages**, run the following command:

  ```
  %conda
  copy-local-env -n graphenv
  ```

  The following example creates a Conda environment, `graphenv`, by copying the `basegraph` environment:

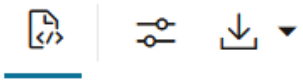
  

- To work with **external Python packages only**, create a Conda environment by running the following command:

  ```
  %conda
  create -n graphenv python==3.6.8
  ```

  The following example creates a Conda environment, `graphenv`, with the specified Python version:

```
%conda
create -n graphenv python==3.6.8
```

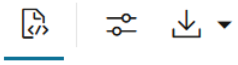


```
Collecting package metadata: ...working... done
Solving environment: ...working... done

## Package Plan ##

  environment location: /home/interpreteruser/.conda/envs/graphenv

  added / updated specs:
    - python==3.6.8


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    _libgcc_mutex-0.1          |             main           3 KB
    certifi-2021.5.30          |   py36h06a4308_0         141 KB
    libedit-3.1.20210910       |       h7f8727e_0         191 KB
    libffi-3.2.1               |    hf484d3e_1007          52 KB
    libgcc-ng-9.1.0            |       hdf63c60_0         8.1 MB
```

3. Install any third-party Python package in the newly created `graphenv`. For example, the following command installs the `pandas 1.3.5` package in the `graphenv`.

```
%conda
install -n graphenv pandas=1.3.5
```

```
%conda
install -n graphenv pandas=1.3.5
```



```
Collecting package metadata: ...working... done
Solving environment: ...working... done

## Package Plan ##

  environment location: /home/interpreteruser/.conda/envs/graphenv

  added / updated specs:
    - pandas=1.3.5


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    certifi-2023.5.7           |   py39h06a4308_0         154 KB
    pandas-1.3.5               |   py39h8c16a72_0        12.3 MB
    ------------------------------------------------------------
                                           Total:        12.5 MB

The following packages will be UPDATED:

  certifi                          2022.12.7-py39h06a4308 0 --> 2023.5.7-py39h06a4308 0
```

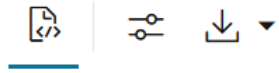
As an ADMIN user, you can also choose to install a different Python version other than the one provided in the `basegraph` environment. For this, you must first activate the Conda environment created in the preceding step. Then you can uninstall the default Python library and install the required Python version as shown:

```
activate <env_name>
uninstall python
install python=3.9
```

4. Upload the Conda environment as shown:

```
%conda
upload graphenv --overwrite --description 'Conda environment with Pandas'
```

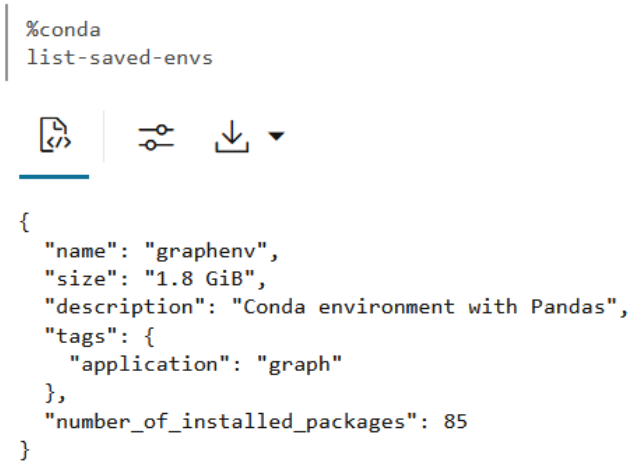
```
%conda
upload graphenv --overwrite --description 'Conda environment with Pandas'
```



```
Uploading conda environment graphenv
Upload successful for conda environment graphenv
```

5. Optionally, verify by listing all the uploaded environments as shown:

```
%conda
list-saved-envs
```



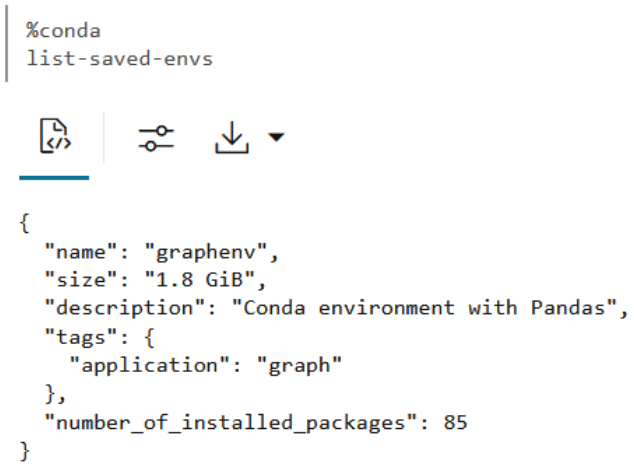
Work with Preinstalled Conda Environments

As a graph user, you can download and activate a preinstalled Conda environment.

You can then access the activated environment in a Python(PGX) paragraph. The following example describes the steps for a graph user to work with a preinstalled Conda environment.

1. Navigate to the **Notebooks** page and open a new notebook.
2. List all the available preinstalled Conda environments:

```
%conda
list-saved-envs
```

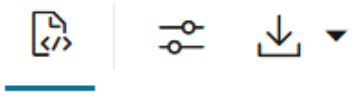


3. Download the required Conda environment.

   The following example downloads the saved `graphenv`:

```
%conda
download graphenv
```

```
%conda
download graphenv
```



```
Downloading conda environment graphenv
Download successful for conda environment graphenv
```

Note the following:

- If you wish to skip the download in case the Conda environment already exists, then you can run the following command:

  ```
  download <env_name> --skip-if-exists
  ```

- If you wish to overwrite an already downloaded Conda environment, then you can run the command as shown:

  ```
  download <env_name> --overwrite
  ```

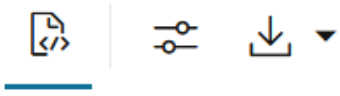
- You can download multiple Conda environments and can always switch between your environments by using the Conda `activate <env>` command.
- If the environment download exceeds the maximum local storage limit of 8 GB, then the Conda interpreter throws an error. In such a case, you can remove an environment from the local storage, using the following command, and repeat the download operation:

  ```
  env remove -n <env_name>
  ```

4. Activate the required environment.

```
%conda
activate graphenv
```

```
%conda
activate graphenv
```



```
Conda environment 'graphenv' activated
```

When you activate a specific Conda environment, the earlier active environment is automatically deactivated. Therefore, when you are working with multiple environments, it is recommended that you activate the required environment before switching to another.

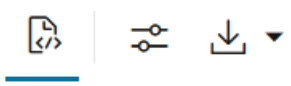
5. Access the environment in a Python(PGX) paragraph.

   As a prerequisite, perform the following steps:

   - Run the Conda `info` or `env list` command and verify that you have activated the required environment. If not, run the Conda `activate` command, as described in the preceding step, to activate the required environment.
   - Run the Conda `list` command to verify that the activated environment contains the required packages that you need to access in the Python(PGX) paragraph.
   - This step applies only if you want to work with the PyPGX APIs. Verify that the output of the Conda `list` command shows the `oracle-pypgx-client` package. If this package is not available in the activated environment, then you cannot work using the PyPGX APIs. See step-2 in Create and Publish a Conda Environment for more information.

   Once you have verified the active environment and the packages installed in the active environment, then you can access the environment in the Python(PGX) paragraph. For instance, the following example uses the `pandas` package in the activated conda environment to convert a PGQL result set into Pandas dataframe.

```
%python-pgx
import pandas
graph = session.read_graph_by_name("BANK_GRAPH", "pg_view")
analyst.pagerank(graph)
query="SELECT n.id, n.pagerank FROM MATCH(n) limit 5"
rs=graph.execute_pgql(query)
print(rs.to_pandas())
```



```
   id  pagerank
0   4  0.119279
1   1  0.221505
2   5  0.170062
3   2  0.221944
4   6  0.025000
```

Use OCI Vault Secret Credentials

As the default ADMIN user of the Autonomous AI Database instance, you can access secret credentials stored in Oracle Cloud Infrastructure (OCI) Vault, in a Python notebook paragraph in Graph Studio.

Topics:

- Prerequisites to Use OCI Vault Secret Credentials
- Attach Vault Secret Credentials to Graph Studio
- Attach and Access a Secret in a Python Notebook Paragraph

Prerequisites to Use OCI Vault Secret Credentials

Before you begin to use Oracle Cloud Infrastructure (OCI) Vault secret credentials in Graph Studio, you must first perform a few prerequisite steps.

The following steps describe the process to configure an OCI Vault and secrets in your Autonomous AI Database instance, enable the resource prinicipal, and attach the vault credential in Graph Studio. Ensure you are the default ADMIN user of the Autonomous AI Database instance to access resources and run OCI operations at tenancy level or at the compartment level.

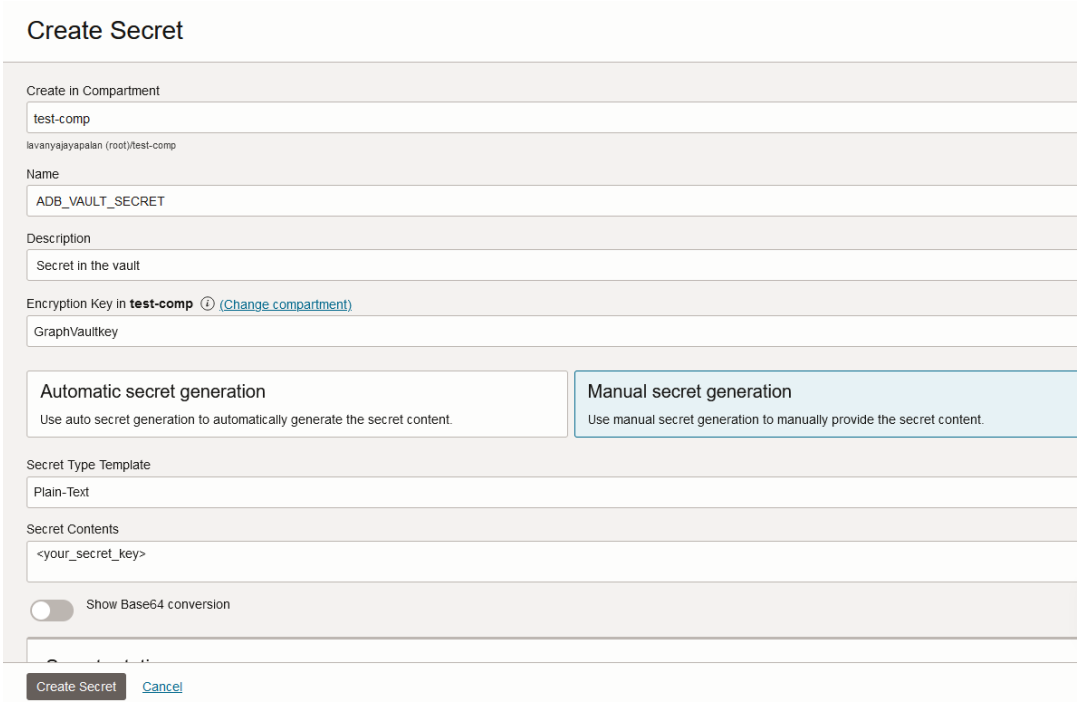
1. Create a **Vault** in your Autonomous AI Database instance.

   See Creating a Vault for more information.

2. Create a **Master Encryption Key** for the vault.

   See Creating a Master Encryption Key for more information.

3. Create a **Secret** specifying the encryption key created in the previous step.



   See Creating a Secret in a Vault for more information.

4. Create a **Dynamic Group** to provide access to the vault in your Autonomous AI Database instance.

   a. Click **Identity & Security** in the OCI Console.

   b. Click **Domains** under **Identity** and select the required domain.

   c. Click **Dynamic groups** under **Identity domain**.

   d. Click **Create dynamic group**.

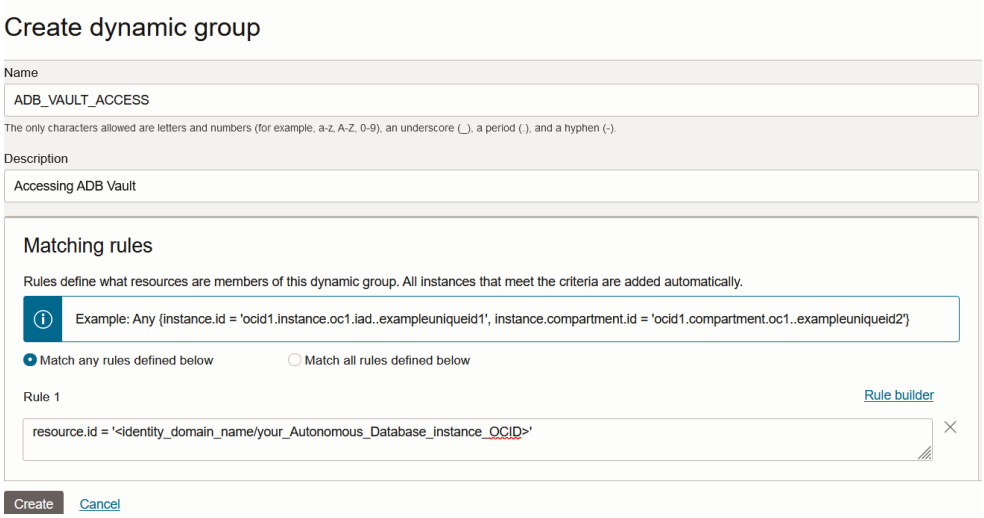
      i. Enter **Name** and **Description**.

ii. Add a **Rule** to specify that your Autonomous AI Database instance is part of the dynamic group as shown in the following code:

```
resource.id = '<your_Autonomous_Database_instance_OCID>'
```

In case the tenancy uses an identity domain, then you need to also include the domain name as shown:

```
resource.id = '<identity_domain_name/
your_Autonomous_Database_instance_OCID>'
```



Note that you can find the database OCID on the Autonomous AI Database page under **General Information** in the **OCID** field.

See Use Resource Principal with Autonomous AI Database for more information on how to define a rule.

iii. Click **Create**.

5. Create a **Policy** for the dynamic group (created in the previous step) to allow access to the vault, keys, and secrets.

   a. Click **Identity & Security** in the OCI Console.

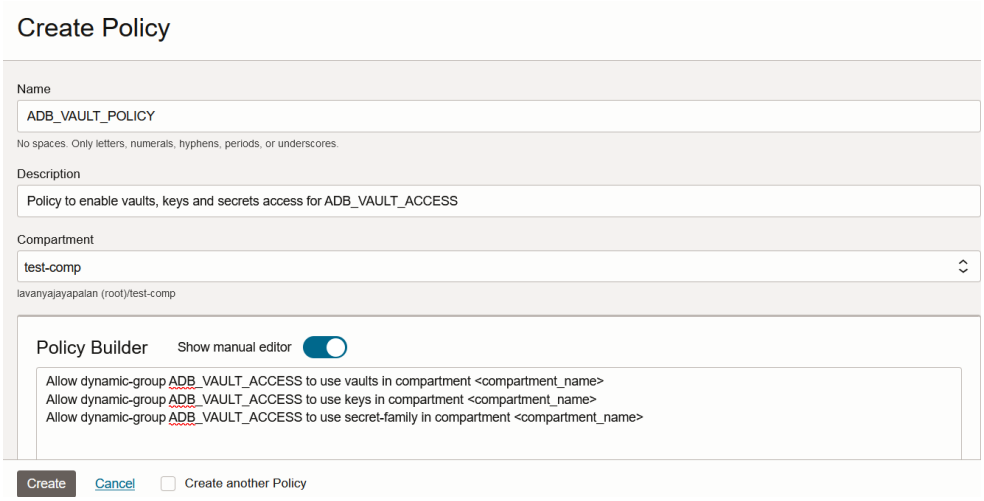
   b. Click **Policies** under **Identity**.

   c. Click **Create Policy**.

      i. Enter **Name** and **Description**.

      ii. Select the required **Compartment**.

      iii. Add the policy statements (as shown in the following figure) using the **Show manual editor** option:



iv. Click **Create**.

6. Copy the OCID for the secret from the **Secret Details** page under **Secret Information** in the **OCID** field.
7. Login to Graph Studio as the ADMIN user, and enable the resource principal (see Use Resource Principal to Access Oracle Cloud Infrastructure Resources) by running the following code in a SQL paragraph.

```
%sql
BEGIN
    DBMS_CLOUD_ADMIN.ENABLE_RESOURCE_PRINCIPAL();
END;
```

   Alternatively, you can connect to **Database Actions** on your Autonomous AI Database instance, and run the preceding code on the **SQL** page.
8. Attach the secret credentials to Graph Studio by following the steps in Attach Vault Secret Credentials to Graph Studio.

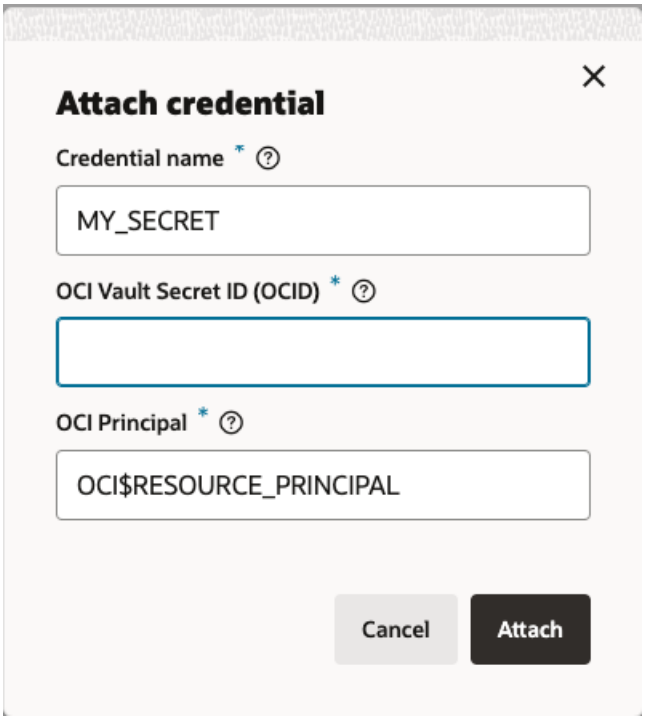
Attach Vault Secret Credentials to Graph Studio

As the ADMIN user, you can attach a credential to Graph Studio which can be later accessed in a Python notebook paragraph.

Perform the following steps as the ADMIN user to upload a secret from Oracle Cloud Infrastructure (OCI) Vault to Graph Studio. The steps assume that you have already created an OCI vault and stored a secret as described in Prerequisites to Use OCI Vault Secret Credentials.

1. Click **Credentials** on the left navigation menu and go to the Credentials page.
2. Click **Attach from OCI Vault**.

   The **Attach credential** dialog opens as shown:



3. Enter a **Credential name**.
4. Enter the **OCI Vault Secret ID (OCID)** (that was copied earlier).
5. Enter the **OCI Principal** value as *`OCI$RESOURCE_PRINCIPAL`*.
6. Click **Attach**.

   Graph Studio will fetch the credential from OCI Vault and the newly created credential gets listed on the **Credentials** page.

Attach and Access a Secret in a Python Notebook Paragraph

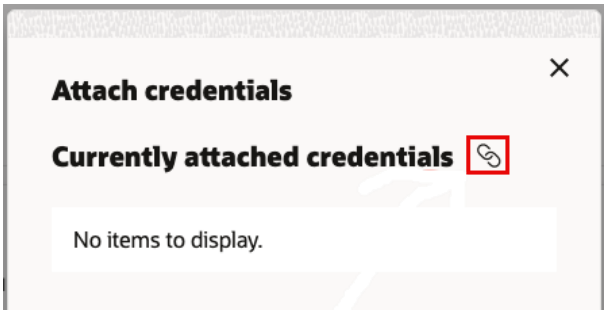
As the ADMIN user, you can attach the credential created in Graph Studio to a notebook. You can then access the secret in a Python paragraph.

Ensure that you meet all the prerequistes described in Prerequisites to Use OCI Vault Secret Credentials.

Perform the following steps to attach and access a secret in a Python notebook paragraph:

1. Click open a notebook in the **Notebooks** page.
2. Click the **Credential** icon on the top left of the page.

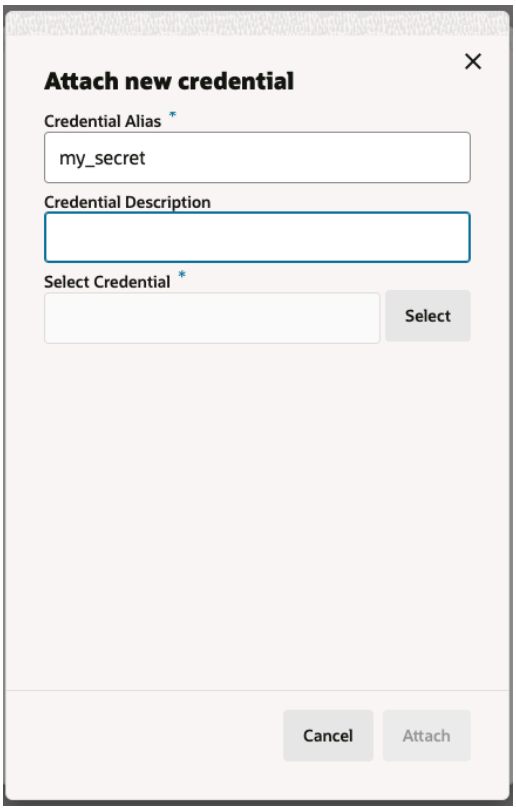
   The **Attach credentials** window opens as shown:



The window displays the currently attached credentials to the notebook. It also allows you to attach a new credential.

3. Click the **Attach** icon, shown highlighted in the preceding figure.

   The **Attach new credential** window opens as shown.



4. Enter the **Credential Alias** and **Credential Description**.
5. Click **Select** and choose a secret from the list.
6. Click **Attach**.

   The newly added credential gets attached to the notebook.

7. Access the secret in a Python paragraph by referencing it using the alias (provided earlier) as shown.

> **⚠ Caution**
>
> The following code snippet is for illustrative purposes only. It is recommended that you do not print secrets in plain text in a paragraph output to maintain confidentiality.

```
%python
from ds_interpreter_client.context.ds_context import PyDataStudioContext

ds = PyDataStudioContext()
print('My secret: ' + ds.get_credential('my_secret'))
```

The following shows the output on running the preceding code:

8. Optionally, share the notebook with another graph user by clicking **Share Notebook** in the notebook toolbar at the top of the page.

   Note that sharing the notebook with any permission allows the graph user to run the Python code in the previous step successfully. However, the user cannot view or attach the credential to their notebooks.
   Also, if you do not wish the user to modify the paragraph, then ensure that you do not grant **Modify Paragraph** permission. Alternatively, you can update the notebook state as **Non-editable**. See Notebook States for more information.

Reference Graphs in Notebook Paragraphs

In order to reference graphs in notebook paragraphs that belong to the PGX interpreter group, the graph must be loaded into the graph server memory.

In addition to loading graphs into memory from the Graphs page (see), you can also perform this action using the following two ways:

Topics:

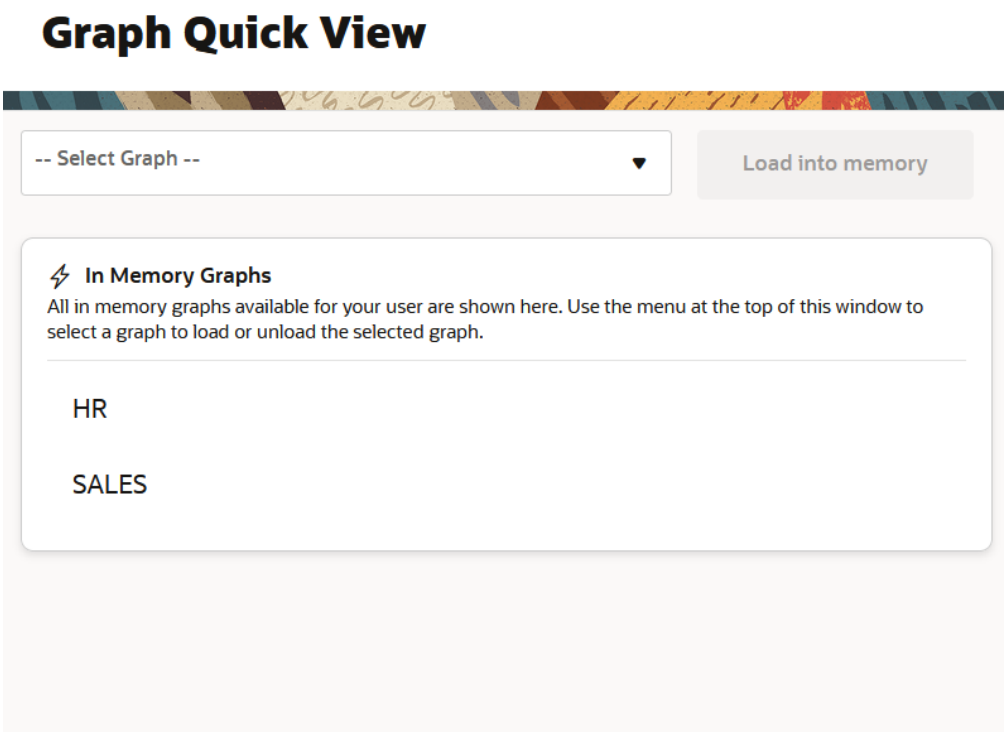
- Load Graphs Into Memory Using the Quickview Option
- Load Graphs into Memory Programmatically

Load Graphs Into Memory Using the Quickview Option

Graph Studio allows you to easily load a graph into memory using the **Quickview** option inside a notebook.

1. Navigate to the **Notebooks** page and click open a notebook.
2. Click **Quickview** at the top of the notebook.

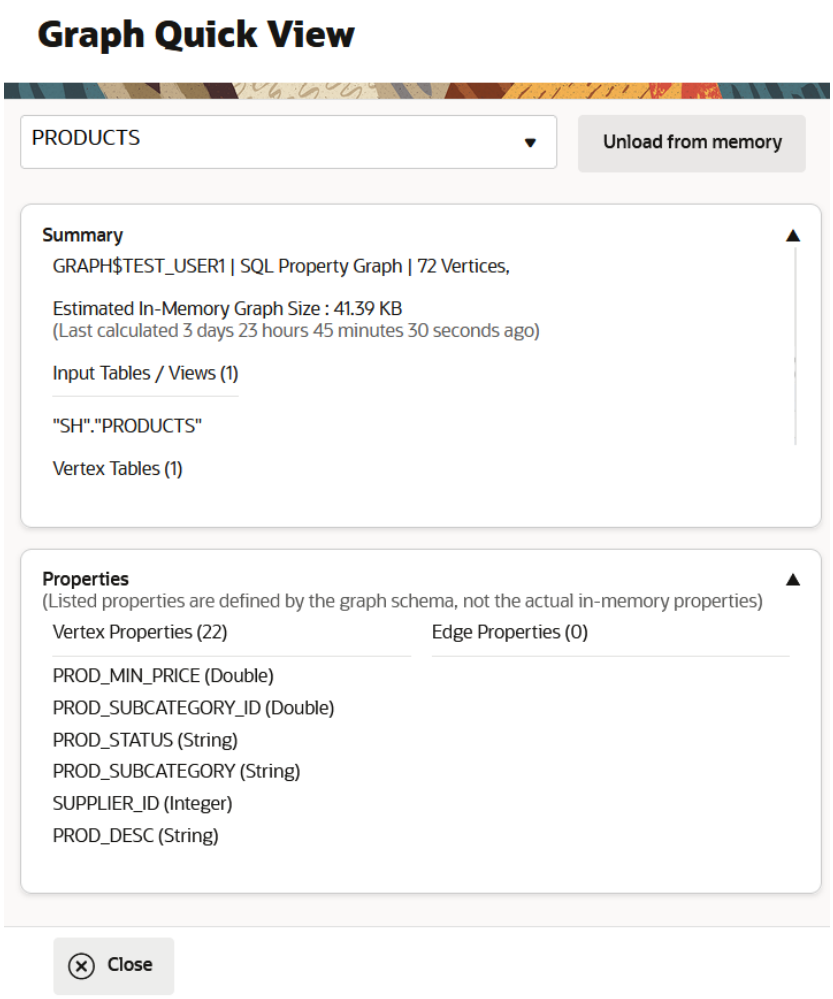
   The **Graph Quick View** slider opens as shown:



   As seen in the preceding figure, all user owned graphs that are already loaded into memory are displayed by default.

3. Select the graph that you wish to load into memory using the **Select Graph** drop-down.

   The slider displays the graph summary along with the **Load into memory** button as shown:



As seen in the preceding figure, the graph details are displayed under the following collapsible sections:

- **Summary:** This shows the graph summary such as the number of vertices and edges in the graph, the underlying source vertex and edge tables, and the estimated in-memory graph size.
- **Properties:** This shows the vertex and edge properties of the graph.

Also, note the following:

- Graphs that are already loaded into memory are indicated by ⚡ in the **Select Graph** drop-down.
- In case you choose to select a graph that is already loaded into memory, then the **Graph Quick View** slider displays the **Unload from memory** button.
- For graphs that cannot be loaded into memory, the **Load into memory** button is disabled.
- If the **Load into memory** or **Unload from memory** button is disabled, then hovering over the button provides you with information on why the specific button is disabled.

4. Click **Load into memory**.

A job to load the graph into memory is initiated at the background. On successful completion of the job, the graph will be listed with the ⚡ icon. In case the job fails, an error message will be displayed.
Also, note that while a graph loading action is in progress, you can continue to load other graphs into memory.

5. Optionally, verify that the graph is loaded into memory.

   For example, run the following code from a PGQL (PGX) paragraph and view the results:

```
%pgql-pgx
SELECT *
FROM GRAPH_TABLE ( country
MATCH (a IS countries) -[e IS countries_regions]-> (b IS regions)
COLUMNS (e.country_name AS country, b.region_name AS region)
)
```

Load Graphs into Memory Programmatically

You can use the `readGraphByName()` API to programmatically load graphs into the graph server memory.

The following example loads a **SQL Property Graph** named `BANK_GRAPH` into memory using the `readGraphByName()` API.

- %java-pgx
- %python-pgx

%java-pgx

```
var graph = session.readGraphByName("BANK_GRAPH", GraphSource.PG_SQL)
```

%python-pgx

```
graph = session.read_graph_by_name("BANK_GRAPH", "pg_sql")
```

The following example loads a **PGQL Property Graph** named `BANK_GRAPH` into memory using the `readGraphByName()` API.

- %java-pgx
- %python-pgx

%java-pgx

```
var graph = session.readGraphByName("BANK_GRAPH", GraphSource.PG_PGQL)
```

%python-pgx

```
graph = session.read_graph_by_name("BANK_GRAPH", "pg_pgql")
```

Once a graph is loaded into memory, you can access the graph in any subsequent notebook paragraphs. For example, you can reference the graph in a PGQL (PGX) paragraph as shown:

```
%pgql-pgx
SELECT *
FROM GRAPH_TABLE ( bank_graph
MATCH (a IS accounts) -[e IS transfers]-> (b IS accounts)
COLUMNS (e.amount AS amount)
) FETCH FIRST 10 ROWS ONLY
```

Store a PgxFrame in Database

You can store a `PgxFrame` output to relational database tables.

The outputs of the property graph machine learning algorithms are `PgxFrame`(s) and this data structure can be stored in the database. The columns and rows of the `PgxFrame` correspond to the columns and rows of the database table.

The following example converts a PGQL result set to a `PgxFrame`, which is then stored as a table to the database.

%java-pgx

```
var g = session.readGraphByName("BANK_GRAPH", GraphSource.PG_VIEW)
var query = "SELECT s.acct_id FROM MATCH (s) LIMIT 10"
var rs = g.queryPgql(query)
if (rs != null) {
  rs.toFrame().write().db()
    .tablename("accounts") // name of the DB table
    .overwrite(true)
    .store();
}
```

%python-pgx

```
g = session.read_graph_by_name("BANK_GRAPH", "pg_view")
query = "SELECT s.acct_id FROM MATCH (s)"
rs = g.execute_pgql(query)
if (rs != None):
    rs.to_frame().write().db().table_name("accounts").overwrite(True).store()
```

On executing the notebook paragraph, the `PgxFrame` data gets inserted in the appropriate database table. You can verify this by viewing and querying the database table using Database Actions. See SQL Page in Database Actions for more information on running SQL statements in Database Actions.

Also, note the following:

- The generated table name and column names are case-sensitive. The preceding code example creates a database table having a lowercase name `"accounts"` with a column named `"acct_id"`.
  If however, the query is:

  ```
  "SELECT s.acct_id as ACCT_ID FROM MATCH (s) limit 10"
  ```

  and table name is specified as `tablename("ACCOUNTS")`, then the database table will have a uppercase name `"ACCOUNTS"` with a column named `"ACCT_ID"`.
- If a database table with the same name is already existing, then you can use the overwrite mode by setting `overwrite(true)` as seen in the preceding example. The previous table gets truncated and the data is then inserted. By default, the value is set to `false`.
- If you are using an Always Free Autonomous AI Database instance (that is, one with only 1 OCPU and 20GB of storage), then you must also specify that only one connection must be used when writing the `PgxFrame` to the table in a Java (PGX) notebook paragraph. For example, you must invoke `write()` as shown:

  ```
  rs.toFrame().write().db().connections(1).tablename("accounts").overwrite(tr
  ue).store();
  ```

Visualize Output of Paragraphs

If a paragraph returns data rows separated by `\n` (newline) and columns separated by `\t` (tab) with the first row as the header row, Graph Studio will render the result visually.

In addition to table-based visualization, the results of PGQL queries can be rendered using graph visualization. `%pgql-pgx` paragraphs will be rendered as graph visualization automatically, if possible.

The following example shows the Java and the Python interpreter using a helper object to generate graph visualization output:

%java-pgx

```
out.println(visualQuery.queryPgql("SELECT v,e,m FROM MATCH (v)-[e]->(m) ON SH
LIMIT 50"))
```

%python-pgx

```
print(visual_query.query_pgql("SELECT v, e, m FROM MATCH (v)-[e]->(m) ON SH
LIMIT 50"))
```

Only a subset of queries can be visualized. If a query cannot be visualized, the notebook will render the result set as a table instead.

Apply Machine Learning on a Graph

You can use machine learning on your property graph data in Graph Studio using the PGX machine learning library.

The following are a few of the supported machine learning algorithms:

- DeepWalk
- Supervised GraphWise
- Unsupervised GraphWise
- Pg2vec

See Using the Machine Learning Library (PgxML) for Graphs in *Oracle AI Database Graph Developer's Guide for Property Graph* for more information.
Running machine learning algorithms is supported in a notebook paragraph using the following interpreters:

- Java (PGX): See `oracle.pgx.api.mllib` package in Java API Reference for more information.
- Python (PGX): See the PyPGX MLlib package in Python API Reference for more information.

For example, the following steps describe the usage of the DeepWalk model on a graph in a notebook paragraph.

1. Load the required graph into memory and reference the graph in the notebook.

   See Load Graphs into Memory Programmatically for more information.

2. Build a DeepWalk model using customized hyper-parameters.

   - %java-pgx
   - %python-pgx

   ### %java-pgx

   ```
   import oracle.pgx.api.mllib.DeepWalkModel
   var model = session.createAnalyst().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-pgx

```
model = analyst.deepwalk_builder(min_word_frequency= 1,
                                 batch_size= 512,
                                 num_epochs= 1,
                                 layer_size= 100,
                                 learning_rate= 0.05,
                                 min_learning_rate= 0.0001,
                                 window_size= 3,
                                 walks_per_vertex= 6,
                                 walk_length= 4,
                                 sample_rate= 0.00001,
                                 negative_sample= 2,
                                 validation_fraction= 0.01)
```

3. Train the DeepWalk model on the graph data.

- %java-pgx
- %python-pgx

%java-pgx

```
model.fit(g)
```

%python-pgx

```
model.fit(g)
```

You can now perform one or more of the following functionalities on the DeepWalk model:

4. Compute the loss value on the data.

- %java-pgx
- %python-pgx

%java-pgx

```
var loss = model.getLoss()
```

%python-pgx

```
loss = model.loss
```

5. Fetch similar vertices for a list of input vertices.

- %java-pgx
- %python-pgx

%java-pgx

```
import oracle.pgx.api.frames.*
List<java.lang.Object> vertices = Arrays.asList("3244407212344026742",
"371586706748522153")
var batchSimilars = model.computeSimilars(vertices, 2 )
batchSimilars.print(out,10,0)
```

%python-pgx

```
vertices = ["3244407212344026742", "371586706748522153"]
batch_similars = model.compute_similars(vertices, 2)
batch_similars.print()
```

The output results in the following format:

```
+-------------------------------------------------------------------+
| srcVertex           | dstVertex           | similarity            |
+-------------------------------------------------------------------+
| 3244407212344026742 | 3244407212344026742 | 1.0                   |
| 3244407212344026742 | 3510061098087750671 | 0.2863036096096039    |
| 371586706748522153  | 371586706748522153  | 1.0                   |
| 371586706748522153  | 2128822953047004384 | 0.3220503330230713    |
+-------------------------------------------------------------------+
```

6. Retrieve and store all trained vertex vectors to the database.

- %java-pgx
- %python-pgx

%java-pgx

```
var vertexVectors = model.getTrainedVertexVectors().flattenAll()
vertexVectors.write().db().name("deepwalkframe").tablename("vertexVectors")
.overwrite(true).store()
```

%python-pgx

```
vertex_vectors = model.trained_vectors.flatten_all()
vertex_vectors.write().db().table_name("vertex_vectors").overwrite(True).st
ore()
```

If you are using an Always Free Autonomous AI Database instance (that is, one with only 1 OCPU and 20GB of storage), then you must also specify that only one connection must be used when writing the `PgxFrame` to the table in a Java (PGX) notebook paragraph. For example, you must invoke `write()` as shown:

```
vertexVectors.write().db().name("deepwalkframe").tablename("vertexVectors")
.overwrite(true).connections(1).store()
```

The columns in the database table for the flattened vectors will appear as:

```
+-----------------------------------------------------+---------------------
+
| vertexid                      | embedding_0         | embedding_1
|
+-----------------------------------------------------+---------------------
+
```

7. Store the trained model to the database.

- %java-pgx
- %python-pgx

%java-pgx

```
model.export().db().modelstore("bank_model").modelname("model").descriptio
n("DeepWalk Model for Bank data").store()
```

%python-pgx

```
model.export().db(model_store="bank_model",
                  model_name="model", model_description="DeepWalk Model
for Bank data", overwrite=True)
```

The model gets stored as a row in the model store table.

8. Load a pre-trained model from the database.

- %java-pgx
- %python-pgx

%java-pgx

```
var model =
session.createAnalyst().loadDeepWalkModel().db().modelstore("bank_model").m
odelname("model").load()
```

%python-pgx

```
model = analyst.get_deepwalk_model_loader().db(model_store="bank_model",
                                               model_name="model")
```

9. Destroy a DeepWalk model.

- %java-pgx
- %python-pgx

%java-pgx

```
model.destroy()
```

%python-pgx

```
model.destroy()
```

Dynamic Forms

Graph Studio allows the creation of dynamic forms. A dynamic form is a user input field that is generated from the code of a paragraph.

The following two ways of creating dynamic forms are supported.

Topics:

- Create Fixed Dynamic Forms
- Create Programmatic Dynamic Forms

Create Fixed Dynamic Forms

Fixed dynamic forms use values that are hard-coded in the paragraph code to generate the dynamic form.

The structure for the dynamic form is `${my-form-info}`. On execution, the placeholders in the code will be replaced with the custom user input in the respective input fields.

The following input fields are currently supported:

- Use **Textbox** to input any string of characters.

  ```
  ${<name>(<label>)=<default_value>}
  ```

  In the preceding code:

  - **`name`**: The name of the dynamic form. It is displayed on top of the dynamic form. If you want to reference a dynamic form multiple times in a paragraph, you can assign the same `name` to do so and it will only be displayed once.
  - **`label`**: The label that is displayed on top of the dynamic form. A customized label can be specified using `${name (myLabel)}`.
  - **`default_value`** (optional): The default value that is given to the dynamic form when it is first created.

  For example:

  ```
  %md
  My name is ${textbox(Title of textbox)=Graph Studio}
  ```

- Use **Select** to choose a value from a drop-down list.

  ```
  ${<name>(<label>)=<default_value>,<option_value_a>(<option_label_a>)|
  <option_value_b>(<option_label_b>)}
  ```

  In the preceding code:

  - **`name`**: The name of the dynamic form. It is displayed on top of the dynamic form. If you want to reference a dynamic form multiple times in a paragraph, you can assign the same `name` to do so and it will only be displayed once.
  - **`default_value`** (optional): The default value that is given to the dynamic form when it is first created. It must be one of the *option* values.
    - An *option* comprises `option_value` and `option_label`. The `option_value` is used to reference which `default_value` should be selected, and the (optional) `option_label` is displayed in the dropdown list or in respective boxes created by a checkbox.
    - An `option_value` can be either a string or a numeric value.
    - Options are separated with the `|` character in parsed forms.

  For example:

  ```
  %md
  Country: ${country=US,US(United States)|UK|JP}
  ```

- Use **Multiple Select** to select one or multiple values from a list.

  ```
  $
  {selectMultiple(<join_parameter>):<name>(<label>)=<default_value>,<option_v
  alue_a>(<option_label_a>)|<option_value_b>(<option_label_b>)}
  ```

  In the preceding code:

  – **`name`**: The name of the dynamic form. It is displayed on top of the dynamic form. If you want to reference a dynamic form multiple times in a paragraph, you can assign the same `name` to do so and it will only be displayed once.

  – **`label`**: The label that is displayed on top of the dynamic form.

  – **`default_value`** (optional): The default value that is given to the dynamic form when it is first created. It must be one of the *option* values.

    * An *option* comprises `option_value` and `option_label`. The `option_value` is used to reference which `default_value` should be selected, and the (optional) `option_label` is displayed in the dropdown list.

    * An `option_value` can be either a string or a numeric value.

    * Options are separated with the `|` character in parsed forms.

  – **`join_parameter`**: The value that will be inserted between multiple selected values. For instance, consider that a Multiple Select dynamic form having two elements `A` and `B` with a join parameter of `or`. If the user selects both `A` and `B` and runs the paragraph, then the result will be `A or B`.

  For example:

  ```
  ${selectMultiple(OR):country=US|JP, US(United States)|UK|JP}
  ```

- Use **Slider** to select within a specified range.

  ```
  %md
  ${slider(<minimum>,<maximum>,<step_size>):<name>(<label>)=<default_value>}
  ```

  In the preceding code:

  – **`name`**: The name of the dynamic form. It is displayed on top of the dynamic form. If you want to reference a dynamic form multiple times in a paragraph, you can assign the same `name` to do so and it will only be displayed once.

  – **`minimum`**: The minimum value of the slider. Must be a number.

  – **`maximum`**: The maximum value of the slider. Must be a number.

  – **`step_size`**: The step size of the slider. Must be a number and divider of (maximum - minimum).

  – **`default_value`** (optional): The default value that is given to the dynamic form when it is first created (`minimum <= default_value <= maximum`).

  For example:

  ```
  %md
  My age is: ${slider(18.0,30.0,5.0):My Age=25.0}
  ```

- Use **Checkbox** to select one or more specified values.

  ```
  $
  {checkbox(<join_parameter>):<name>(<label>)=<default_value>,<option_value_a
  >(<option_label_a>)|<option_value_b>(<option_label_b>)}
  ```

  In the preceding code:

  - **`name:`** The name of the dynamic form. It is displayed on top of the dynamic form. If you want to reference a dynamic form multiple times in a paragraph, you can assign the same `name` to do so and it will only be displayed once.
  - **`default_value`** (optional): The default value that is given to the dynamic form when it is first created. It must be one of the *option* values.
    - An *option* comprises `option_value` and `option_label`. The `option_value` is used to reference which `default_value` should be selected, and the (optional) `option_label` is displayed in the dropdown list or in respective boxes created by a checkbox.
    - An `option_value` can be either a string or a numeric value.
    - Options are separated with the `|` character in parsed forms.
  - **`join_parameter`**: The value that will be inserted between multiple selected values. For instance, consider that a Checkbox dynamic form having two elements `A` and `B` with a join parameter of `or`. If the user selects the checkbox for both `A` and `B` and runs the paragraph, then the result will be `A or B`.

  For example:

  ```
  %md
  ${checkbox( or ):country(Country)=US|JP, US(United States)|UK|JP}
  ```

- Use **Date Picker** to select a date.

  ```
  ${date(<date_format>):<name>(<label>)=<default_value>}
  ```

  In the preceding code:

  - **`name`**: The name of the dynamic form. It is displayed on top of the dynamic form. If you want to reference a dynamic form multiple times in a paragraph, you can assign the same `name` to do so and it will only be displayed once.
  - **`date_format`** (optional, recommended): The date format that is used for displaying the selected date in the input field and for formatting the resulting date when the paragraph is run.
  - **`default_value`** (optional): The default value that is given to the dynamic form when it is first created. It must be specified according to the `date_format` or in `yyyy-MM-dd` format if the `date_format` is not provided.

  For example:

  ```
  %md
  ${date(EEEE):myName(my-label)=1994-06-15T09:00:00}
  ```

- Use **Time Picker** to select a time.

  ```
  ${time(<time_format>):<name>(<label>)=T13:30}
  ```

  In the preceding code:

  - **`name`**: The name of the dynamic form. It is displayed on top of the dynamic form. If you want to reference a dynamic form multiple times in a paragraph, you can assign the same `name` to do so and it will only be displayed once.
  - **`time_format`** (optional, recommended): The time format that is used for displaying the selected time in the input field and for formatting the resulting time when the paragraph is run.

  For example:

  ```
  %md
  ${time(hh:mm:ss):myName(my-label)=1994-06-15T09:00:00}
  ```

- Use **DateTime Picker** to select one or more specified values.

  ```
  ${dateTime(<dateTime_format>):<name>(<label>)=<default_value>}
  ```

  In the preceding code:

  - **`name`**: The name of the dynamic form. It is displayed on top of the dynamic form. If you want to reference a dynamic form multiple times in a paragraph, you can assign the same `name` to do so and it will only be displayed once.
  - **`dateTime_format`** (optional, recommended): The `dateTime` format that is used for displaying the selected date and time in the input field and for formatting the resulting date and time when the paragraph is run.
  - **`default_value`** (optional): The default value that is given to the dynamic form when it is first created. It must be specified according to the `dateTime_format` or in `yyyy-MM-dd HH:mm` format if the `dateTime_format` is not provided.

  For example:

  ```
  %md
  ${dateTime(YYYY-M-dd hh:mm:ss):myName(my-label)=1995-06-15T09:00:00}
  ```

Create Programmatic Dynamic Forms

Graph Studio allows you to programmatically create dynamic forms using the Java (PGX) and Python (PGX) interpreters.

You can pass dynamic values (such as variables, arrays, and so on) through Java or Python code to generate dynamic forms.
As a prerequisite step, you must first **import** the context that allows you to display the forms and define your own variable name and instantiate your context.

- %java-pgx
- %python-pgx

%java-pgx

```
import oracle.datastudio.interpreter.common.context.JavaDataStudioContext
JavaDataStudioContext ds = interpreter.getJavaDataStudioContext()
```

%python-pgx

```
from ds_interpreter_client.context.ds_context import PyDataStudioContext
ds = PyDataStudioContext()
```

The `ds` context allows you to display the forms and define your own variable name.
The following steps describe the programmatic creation of the **Textbox**, **Select**, **Select Multiple**, **Slider**, **Checkbox**, **Date Picker**, **Time Picker**, and **DateTime Picker** forms. It is important to note that only when you run the notebook paragraph with the dynamic form value (or the default value), then the values are persisted on page reload.

- Create a **Textbox** dynamic form which allows you to input any string of characters.

- %java-pgx
- %python-pgx

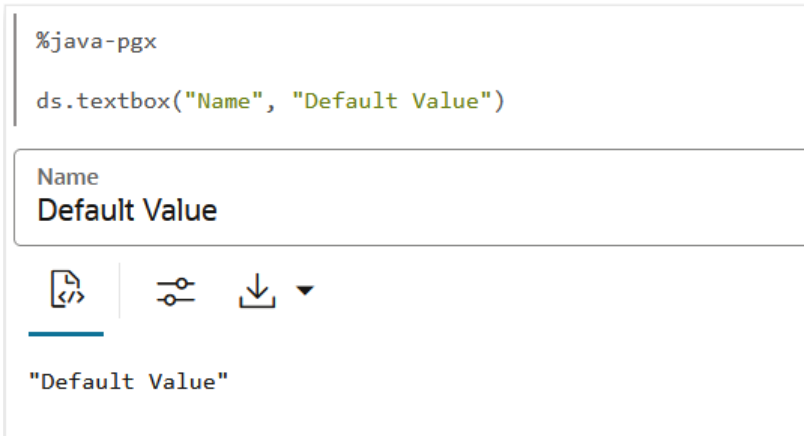
%java-pgx

```
ds.textbox("<name>", "<default_value>")
```

In the preceding code:

* **`name`**: The name of the dynamic form. It is displayed on top of the dynamic form. If you want to reference a dynamic form multiple times in a paragraph, you can assign the same `name` to do so and it will only be displayed once.
* **`default_value`** (optional): The default value that is given to the dynamic form when it is first created.

For example:



%python-pgx

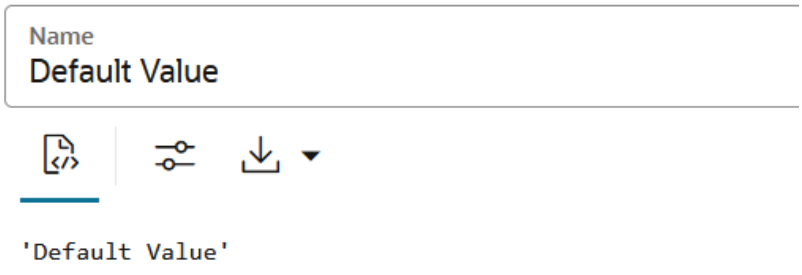
```
ds.textbox(name="<name>", default_value="<default_value>")
```

In the preceding code:

* **name**: The name of the dynamic form. It is displayed on top of the dynamic form. If you want to reference a dynamic form multiple times in a paragraph, you can assign the same name to do so and it will only be displayed once.
* **default_value** (optional): The default value that is given to the dynamic form when it is first created.

For example:

```
%python-pgx

ds.textbox("Name", "Default Value")
```




```
'Default Value'
```

- Create a **Select** dynamic form which allows you to select a value from a drop-down menu.

- %java-pgx
- %python-pgx

%java-pgx

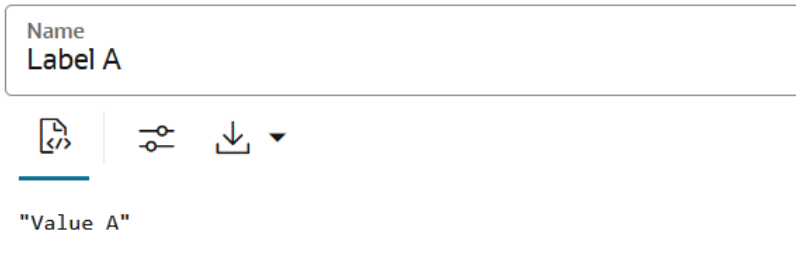
```
import oracle.datastudio.common.forms.ParamOption
List<ParamOption<String>> options = new ArrayList<>()
options.add(new ParamOption<>("<option_value_a>", "<option_label_a>"))
options.add(new ParamOption<>("<option_value_b>", "<option_label_b>"))
ds.select("<name>", options, "<default_value>")
```

In the preceding code:

* **name**: The name of the dynamic form. It is displayed on top of the dynamic form. If you want to reference a dynamic form multiple times in a paragraph, you can assign the same name to do so and it will only be displayed once.
* **default_value** (optional): The default value that is given to the dynamic form when it is first created.

For example:

```
%java-pgx

import oracle.datastudio.common.forms.ParamOption

List<ParamOption<String>> options = new ArrayList<>()
options.add(new ParamOption<>("Value A", "Label A"))
options.add(new ParamOption<>("Value B", "Label B"))
ds.select("Name", options, "Value A")
```



%python-pgx

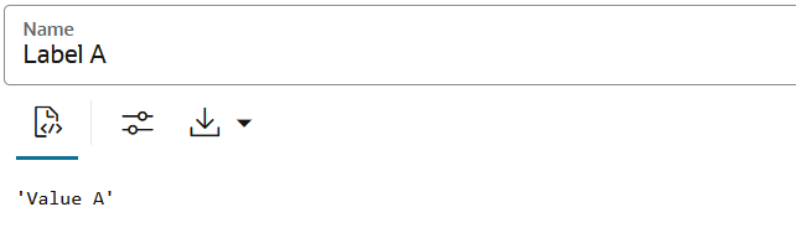
```
options = [("<option_value_a>", "<option_label_a>"),("<option_value_b>",
"<option_label_b>")]
ds.select(name="<name>", options=options, default_value="<default_value>")
```

In the preceding code:

* **`name`**: The name of the dynamic form. It is displayed on top of the dynamic form. If you want to reference a dynamic form multiple times in a paragraph, you can assign the same `name` to do so and it will only be displayed once.
* **`default_value`** (optional): The default value that is given to the dynamic form when it is first created.

For example:

```
%python-pgx

options = [("Value A", "Label A"),("Value B", "Label B")]
ds.select("Name", options, "Value A")
```



- Create a **Select Mulitple** dynamic form which allows you to select one or more values from a drop-down list.

- %java-pgx

- %python-pgx

%java-pgx

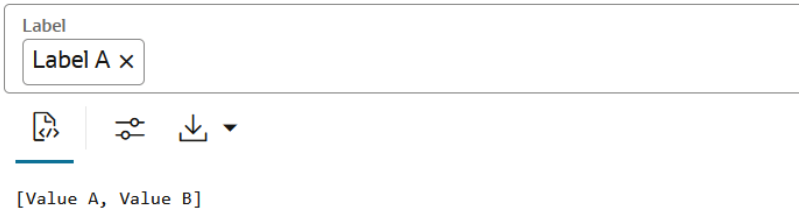
```
List<ParamOption<String>> options = new ArrayList<>();
options.add(new ParamOption<>("<option_value_a>", "<option_label_a>"));
options.add(new ParamOption<>("<option_value_b>", "<option_label_b>"));
List<String> defaultValues = new ArrayList<>();
defaultValues.add("<default_value>");
ds.selectMultiple("<name>", options, defaultValues, "<label>")
```

In the preceding code:

* **`name`**: The name of the dynamic form. It is displayed on top of the dynamic form. If you want to reference a dynamic form multiple times in a paragraph, you can assign the same `name` to do so and it will only be displayed once.
* **`label`** (optional): The label that is displayed on top of the dynamic form.
* **`default_value`** (optional): The default value that is given to the dynamic form when it is first created. It must be one of the *option* values.
    * An *option* comprises `option_value` and `option_label`. The `option_value` is used to reference which `default_value` should be selected, and the (optional) `option_label` is displayed in the drop-down list.
    * An `option_value` can be either a string or a numeric value.
    * Options are separated with the `|` character in parsed forms.

For example:

```
%java-pgx
List<ParamOption<String>> options = new ArrayList<>();
options.add(new ParamOption<>("Value A", "Label A"));
options.add(new ParamOption<>("Value B", "Label B"));
List<String> defaultValues = List.of("Value A");
ds.selectMultiple("Name", options, defaultValues, "Label")
```




```
[Value A, Value B]
```

%python-pgx

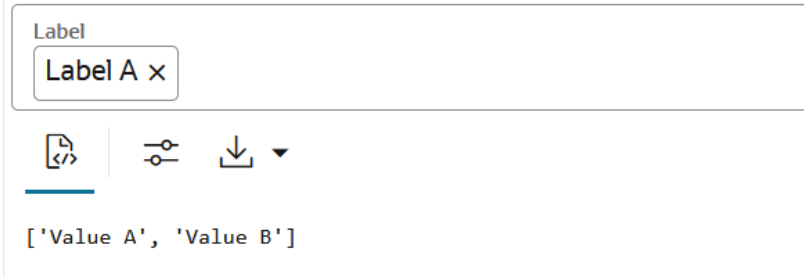
```
options = [('<option_value_a>', '<option_label_a>'),('<option_value_b>',
'<option_label_b>')]
ds.select_multiple(name='<name>', options=options,
default_value=['<default_value>'], label='<label>')
```

In the preceding code:

* **`name`**: The name of the dynamic form. It is displayed on top of the dynamic form. If you want to reference a dynamic form multiple times in a paragraph, you can assign the same `name` to do so and it will only be displayed once.
* **`label`** (optional): The label that is displayed on top of the dynamic form.
* **`default_value`** (optional): The default value that is given to the dynamic form when it is first created. It must be one of the *option* values.
    * An *option* comprises `option_value` and `option_label`. The `option_value` is used to reference which `default_value` should be selected, and the (optional) `option_label` is displayed in the drop-down list.
    * An `option_value` can be either a string or a numeric value.
    * Options are separated with the `|` character in parsed forms.

For example:

```
%python-pgx
options = [('Value A', 'Label A'),('Value B', 'Label B')]
ds.select_multiple('Name', options, ['Value A'], 'Label')
```



- Create a **Slider** dynamic form which allows you to choose a number from a given range.

- %java-pgx
- %python-pgx

%java-pgx

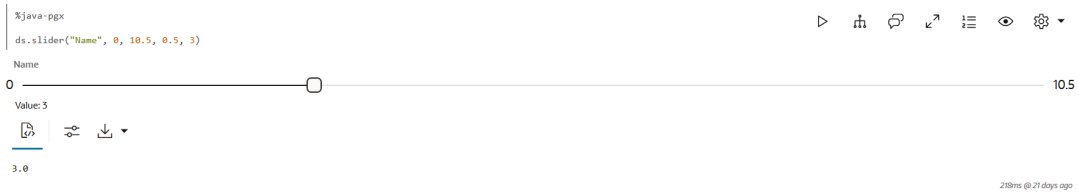
```
ds.slider("<name>", <minimum>, <maximum>, <step_size>, <default_value>)
```

In the preceding code:

* **`name`**: The name of the dynamic form. It is displayed on top of the dynamic form. If you want to reference a dynamic form multiple times in a paragraph, you can assign the same `name` to do so and it will only be displayed once.
* **`minimum`**: The minimum value of the slider. Must be a number.
* **`maximum`**: The maximum value of the slider. Must be a number.
* **`step_size`**: The step size of the slider. Must be a number and divider of (maximum - minimum).

* **`default_value`** (optional): The default value that is given to the dynamic form when it is first created (`minimum <= default_value <= maximum`).

For example:



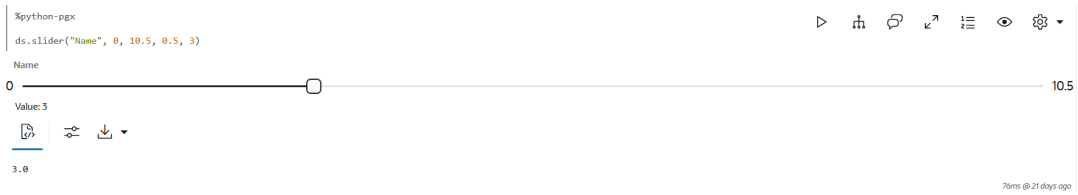
%python-pgx

```
ds.slider(name="<name>", min=<minimum>, max=<maximum>, step=<step_size>,
default_value=<default_value>)
```

In the preceding code:

* **`name`**: The name of the dynamic form. It is displayed on top of the dynamic form. If you want to reference a dynamic form multiple times in a paragraph, you can assign the same `name` to do so and it will only be displayed once.
* **`minimum`**: The minimum value of the slider. Must be a number.
* **`maximum`**: The maximum value of the slider. Must be a number.
* **`step_size`**: The step size of the slider. Must be a number and divider of (maximum - minimum).
* **`default_value`** (optional): The default value that is given to the dynamic form when it is first created (`minimum <= default_value <= maximum`).

For example:



- Create a **Checkbox** dynamic form which allows you to select one or multiple values.

%java-pgx

```
import oracle.datastudio.common.forms.ParamOption

List<ParamOption<String>> options = new ArrayList<>()
options.add(new ParamOption<>("<option_value_a>", "<option_label_a>"))
options.add(new ParamOption<>("<option_value_b>", "<option_label_b>"))
```

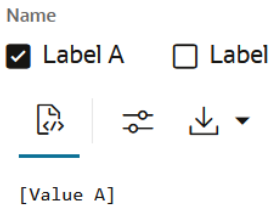
```
List<String> defaultValues = new ArrayList<>()
defaultValues.add("<default_value>")
ds.checkbox("<name>", options, defaultValues)
```

In the preceding code:

* **`name`**: The name of the dynamic form. It is displayed on top of the dynamic form. If you want to reference a dynamic form multiple times in a paragraph, you can assign the same `name` to do so and it will only be displayed once.
* **`default_value`** (optional): The default value that is given to the dynamic form when it is first created. It must be one of the *option* values:
  * An *option* comprises `option_value` and `option_label`. The `option_value` is used to reference which `default_value` should be selected, and the (optional) `option_label` is displayed in respective boxes created by a checkbox.
  * An `option_value` can be either a string or a numeric value.
  * Options are separated with the `|` character in parsed forms.

For example:

```
%java-pgx

import oracle.datastudio.common.forms.ParamOption

List<ParamOption<String>> options = new ArrayList<>()
options.add(new ParamOption<>("Value A", "Label A"))
options.add(new ParamOption<>("Value B", "Label B"))
List<String> defaultValues = new ArrayList<>()
defaultValues.add("Value A")
ds.checkbox("Name", options, defaultValues)
```




```
[Value A]
```

%python-pgx

```
options = [("<option_value_a>", "<option_label_a>"),("<option_value_b>",
"<option_label_b>")]
ds.checkbox(name="<name>", options=options,
default_value=["<default_value>"])
```

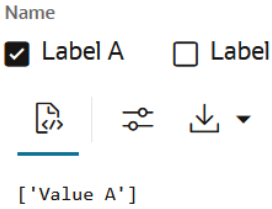
In the preceding code:

* **`name`**: The name of the dynamic form. It is displayed on top of the dynamic form. If you want to reference a dynamic form multiple times in a paragraph, you can assign the same `name` to do so and it will only be displayed once.
* **`default_value`** (optional): The default value that is given to the dynamic form when it is first created. It must be one of the *option* values:
  * An *option* comprises `option_value` and `option_label`. The `option_value` is used to reference which `default_value` should be selected, and the (optional) `option_label` is displayed in respective boxes created by a checkbox.

* An `option_value` can be either a string or a numeric value.
* Options are separated with the | character in parsed forms.

For example:

```
%python-pgx

options = [("Value A", "Label A"),("Value B", "Label B")]
ds.checkbox("Name", options, ["Value A"])
```




```
['Value A']
```

- Create a **Date Picker** dynamic form which allows you to select a date.

- %java-pgx
- %python-pgx

%java-pgx

```
ds.datePicker("<name>", "<date_format>", "<default_value>")
```

In the preceding code:

* **`name`**: The name of the dynamic form. It is displayed on top of the dynamic form. If you want to reference a dynamic form multiple times in a paragraph, you can assign the same `name` to do so and it will only be displayed once.
* **`date_format`** (optional, recommended): The date format that is used for displaying the selected date in the input field and for formatting the resulting date when the paragraph is run.
* **`default_value`** (optional): The default value that is given to the dynamic form when it is first created. It must be specified according to the `date_format` or in `yyyy-MM-dd` format if the `date_format` is not provided.

For example:

```
%java-pgx

ds.datePicker("Name", "yyyy/MM/dd", "1990/01/01")
```


```
"1990/01/01"
```

%python-pgx

```
ds.date_picker(name="<name>", format="<date_format>",
default_value="<default_value>")
```

In the preceding code:

* **`name`**: The name of the dynamic form. It is displayed on top of the dynamic form. If you want to reference a dynamic form multiple times in a paragraph, you can assign the same `name` to do so and it will only be displayed once.
* **`date_format`** (optional, recommended): The date format that is used for displaying the selected date in the input field and for formatting the resulting date when the paragraph is run.
* **`default_value`** (optional): The default value that is given to the dynamic form when it is first created. It must be specified according to the `date_format` or in `yyyy-MM-dd` format if the `date_format` is not provided.

For example:

```
%python-pgx

ds.date_picker("Name", format="yyyy/MM/dd", default_value="2020/12/10")
```


```
'2020/12/10'
```

- Create a **Time Picker** dynamic form which allows you to select a time.

- %java-pgx
- %python-pgx

%java-pgx

```
ds.timePicker("<name>", "<time_format>", "<default_value>")
```

In the preceding code:

* **name**: The name of the dynamic form. It is displayed on top of the dynamic form. If you want to reference a dynamic form multiple times in a paragraph, you can assign the same `name` to do so and it will only be displayed once.
* **time_format** (optional, recommended): The time format that is used for displaying the selected time in the input field and for formatting the resulting time when the paragraph is run.
* **default_value** (optional): The default value that is given to the dynamic form when it is first created. It must be specified according to the `time_format` or in `HH:mm` format if the `time_format` is not provided.

For example:

```
%java-pgx

ds.timePicker("Name", "HH:mm:ss", "12:11:01")
```


```
"12:11:01"
```

%python-pgx

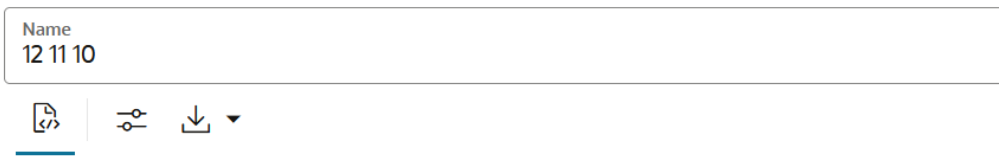
```
ds.time_picker(name="<name>", format="<time_format>",
default_value="<default_value>")
```

In the preceding code:

* **name**: The name of the dynamic form. It is displayed on top of the dynamic form. If you want to reference a dynamic form multiple times in a paragraph, you can assign the same `name` to do so and it will only be displayed once.
* **time_format** (optional, recommended): The time format that is used for displaying the selected time in the input field and for formatting the resulting time when the paragraph is run.
* **default_value** (optional): The default value that is given to the dynamic form when it is first created. It must be specified according to the `time_format` or in `HH:mm` format if no `time_format` is provided.

For example:

```
%python-pgx
ds.time_picker(name='Name', format='HH mm ss', default_value='12 11 10', label='Label')
```




```
'12 11 10'
```

- Define a **DateTime Picker** dynamic form which allows you to select a date and a time.

- %java-pgx
- %python-pgx

%java-pgx

```
ds.dateTimePicker("<name>", "<dateTime_format>", "<default_value>")
```

In the preceding code:

* **`name`**: The name of the dynamic form. It is displayed on top of the dynamic form. If you want to reference a dynamic form multiple times in a paragraph, you can assign the same `name` to do so and it will only be displayed once.
* **`dateTime_format`** (optional, recommended): The `dateTime` format that is used for displaying the selected date and time in the input field and for formatting the resulting date and time when the paragraph is run.
* **`default_value`** (optional): The default value that is given to the dynamic form when it is first created. It must be specified according to the `dateTime_format` or in `yyyy-MM-dd HH:mm` format if the `dateTime_format` is not provided.

For example:

```
%java-pgx

ds.dateTimePicker("Name", "yyyy-MM-dd HH:mm:ss", "1998-12-30 12:11:01")
```


```
"1998-12-30 12:11:01"
```

%python-pgx

```
ds.date_time_picker("<name>", format="<dateTime_format>",
default_value="<default_value>")
```

In the preceding code:

* **`name`**: The name of the dynamic form. It is displayed on top of the dynamic form. If you want to reference a dynamic form multiple times in a paragraph, you can assign the same `name` to do so and it will only be displayed once.
* **`dateTime_format`** (optional, recommended): The `dateTime` format that is used for displaying the selected date and time in the input field and for formatting the resulting date and time when the paragraph is run.
* **`default_value`** (optional): The default value that is given to the dynamic form when it is first created. It must be specified according to the `dateTime_format` or in `yyyy-MM-dd HH:mm` format if the `dateTime_format` is not provided.

For example:

```
%python-pgx

ds.date_time_picker("Name", format="yyyy-MM-dd HH:mm:ss", default_value="2010-12-11 12:10:02")
```

Customize Dynamic Form Layout

You can use the `columnSpan` and `nextRow` parameters to enhance data entry and readability in dynamic form layouts.

The layout of a dynamic form is based on a 12-column grid. By default, each form spans four columns, allowing up to three forms per row. When the total column width exceeds 12, forms automatically wrap to the next row. The layout is responsive:

- Small screens: Forms collapse to four columns (one form per row).
- Medium screens: 8-column grid (two forms per row).
- Large screens: Full 12-column grid (three forms per row).

As a prerequisite step, you must first **import** the context that allows you to display the forms and define your own variable name and instantiate your context.

- %java-pgx
- %python-pgx

%java-pgx

```
import oracle.datastudio.interpreter.common.context.JavaDataStudioContext
JavaDataStudioContext ds = interpreter.getJavaDataStudioContext()
```

%python-pgx

```
from ds_interpreter_client.context.ds_context import PyDataStudioContext
ds = PyDataStudioContext()
```

- The following example describes how to programmatically create a half-width Textbox (6 columns):

  - %java-pgx
  - %python-pgx

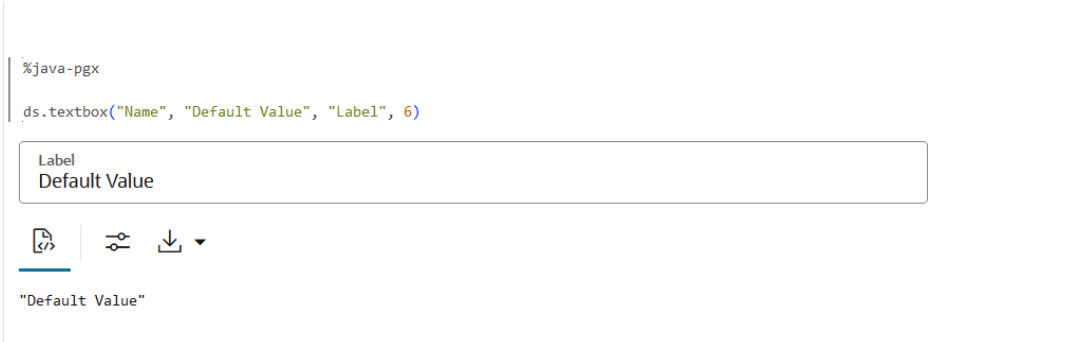
  ## %java-pgx

  ```
  ds.textbox("<name>", "<defaultValue>", <"label">, <columnSpan> )
  ```

  In the preceding code:

  * **`name`**: The name of the dynamic form. It is displayed on top of the dynamic form if no *<label>* is set. If you want to reference a dynamic form multiple times in a paragraph, you can assign the same `name` to do so and it will only be displayed once.
  * **`defaultValue`** (optional): The default value that is given to the dynamic form when it is first created.
  * **`label`** (optional): The label that is displayed on top of the dynamic form.
  * **`columnSpan`**: Number of columns (out of 12) a form occupies. The default value is four (three forms per row).

  For example:

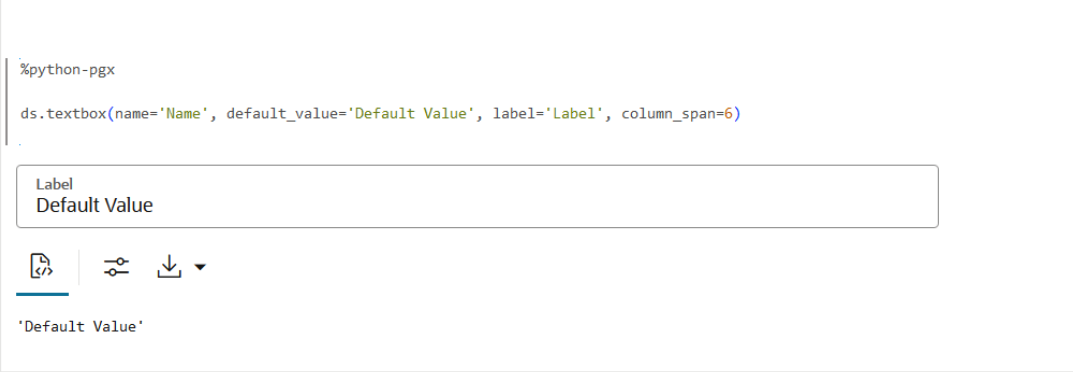
%python-pgx

```
ds.textbox(name="<name>", default_value="<default_value>",
label="<label>", column_span="<column_span_value>"))
```

In the preceding code:

* **`name`**: The name of the dynamic form. It is displayed on top of the dynamic form if no *<label>* is set. If you want to reference a dynamic form multiple times in a paragraph, you can assign the same `name` to do so and it will only be displayed once.
* **`default_value`** (optional): The default value that is given to the dynamic form when it is first created.
* **`label`** (optional): The label that is displayed on top of the dynamic form.
* **`column_span`**: Number of columns (out of 12) a form occupies. The default value is four (three forms per row).

For example:



- The following example describes how to programmatically create a form on a new row:

- %java-pgx
- %python-pgx

%java-pgx

```
ds.textbox("<name>", "<defaultValue>", <"label">, <columnSpan>, <nextRow> )
```

In the preceding code:

* **`name`**: The name of the dynamic form. It is displayed on top of the dynamic form if no *<label>* is set. If you want to reference a dynamic form multiple times in a paragraph, you can assign the same `name` to do so and it will only be displayed once.
* **`defaultValue`** (optional): The default value that is given to the dynamic form when it is first created.

* **`label`** (optional): The label that is displayed on top of the dynamic form.
* **`columnSpan`**: Number of columns (out of 12) a form occupies. The default value is four (three forms per row).
* **`nextRow`**: A boolean parameter that determines if the form should start on a new row or not. Note that you **must** explicitly set `columnSpan` when using `nextRow`.

For example:

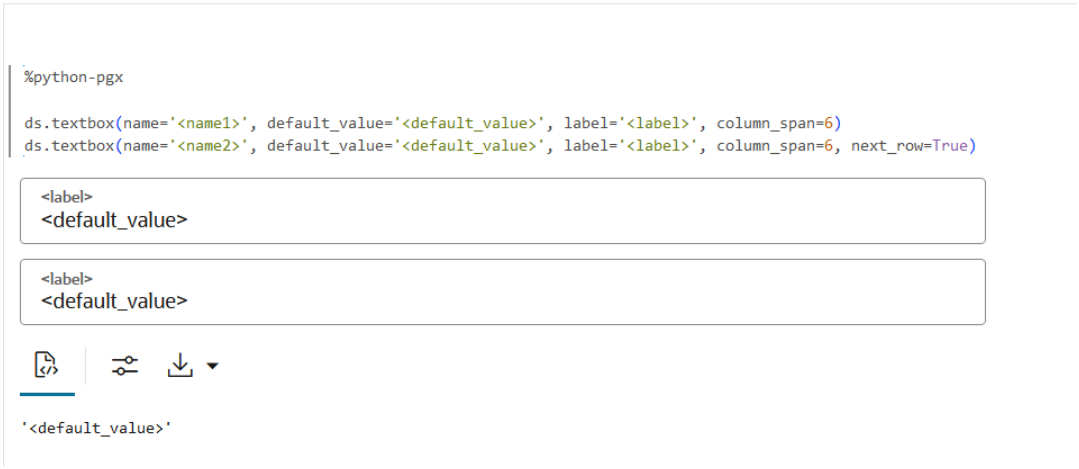
%python-pgx

```
ds.textbox(name="<name>", default_value="<default_value>",
label="<label>", column_span="<column_span_value>,
next_row="<next_row_value>"))
```

In the preceding code:

* **`name`**: The name of the dynamic form. It is displayed on top of the dynamic form if no *<label>* is set. If you want to reference a dynamic form multiple times in a paragraph, you can assign the same `name` to do so and it will only be displayed once.
* **`default_value`** (optional): The default value that is given to the dynamic form when it is first created.
* **`label`** (optional): The label that is displayed on top of the dynamic form.
* **`column_span`**: Number of columns (out of 12) a form occupies. The default value is four (three forms per row).
* **`next_row`**: A boolean parameter that determines if the form should start on a new row or not.

For example:



Notebook Forms

Graph Studio allows you to create notebook forms which can be made available to the entire notebook.

The notebook form appears at the top of the notebook. In this way, a form created in one paragraph can have its value used in other paragraphs of the same notebook.

This is in contrast to Dynamic Forms which appear under a paragraph and whose scope is limited within the paragraph in which they are created.

Topics:

- Create Fixed Notebook Forms
- Create Programmatic Notebook Forms

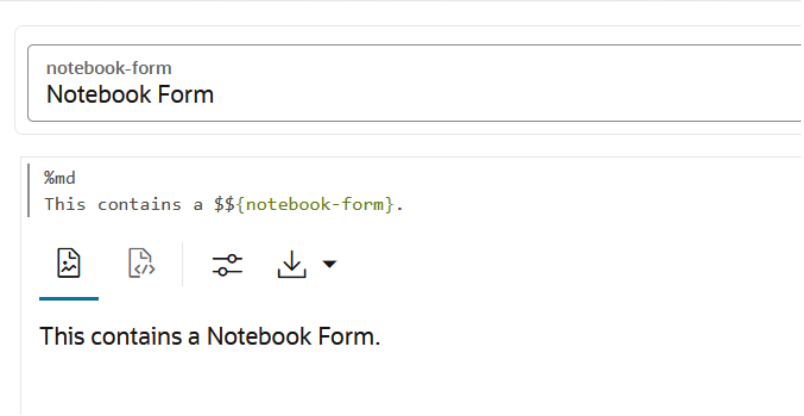
Create Fixed Notebook Forms

Fixed notebook forms are similar to fixed dynamic paragraph forms and use values that are hard-coded in the code.

The structure for the notebook form is `$${<notebook-name>}`. It is similar to Create Fixed Dynamic Forms, except that it uses the escape character ($) twice. For example:

```
%md
This contains a $${notebook-form}.
```

This will show a notebook form as shown:



In case there is a need to use the fixed notebook form syntax in a paragraph's code without creating forms, then that can be achieved by preceding $$ with a backslash \. For instance, `\$${name}` syntax will not create a form and will be parsed as `$${name}` in paragraph results.

If the escape functionality of the backslash \ is undesirable, then that itself can be escaped with another backslash \ (`\\$${name}`).

Create Programmatic Notebook Forms

Graph Studio allows you to programmatically create notebook forms using the Java (PGX) and Python (PGX) interpreters.

The prerequiste step (to import the context) and the methods to generate these forms are similar to the methods described for Create Programmatic Dynamic Forms. However, the methods take an optional argument (`true`) to indicate that the form should be a notebook form as shown. This optional argument is `false` by default.

- %java-pgx
- %python-pgx

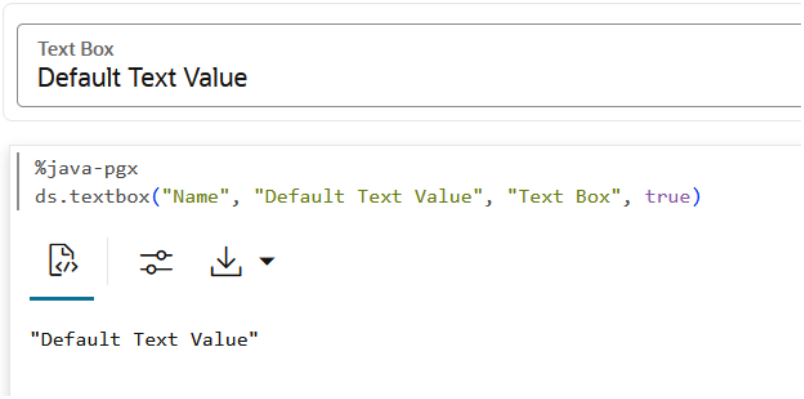
%java-pgx

```
ds.textbox("<name>", "<default_value>", "<label>", true)
```

In the preceding code:

- **`name`**: The name of the notebook form. It is displayed on top of the notebook form if no label is set. If you want to reference a notebook form multiple times in a paragraph, you can assign the same `name` to do so and it will only be displayed once.
- **`label`** (optional): The label that is displayed on top of the notebook form.
- **`default_value`** (optional): The default value that is given to the notebook form when it is first created.

For example:



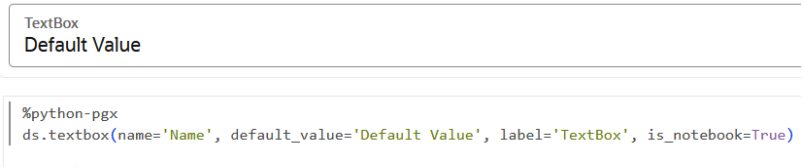
%python-pgx

```
ds.textbox(name='<name>', default_value='<default_value>', label='<label>',
is_notebook=True)
```

In the preceding code:

- **`name`**: The name of the notebook form. It is displayed on top of the notebook form if no label is set. If you want to reference a notebook form multiple times in a paragraph, you can assign the same `name` to do so and it will only be displayed once.
- **`label`** (optional): The label that is displayed on top of the notebook form.
- **`default_value`** (optional): The default value that is given to the notebook form when it is first created.

For example:



Paragraph Dependencies

You can add dependencies between paragraphs.

The dependents of a paragraph are automatically executed after the original paragraph itself or any graph manipulation on the original paragraph is executed.

To start dependency mode, click the **Dependencies** button in the paragraph settings bar.

Dependency Mode

In dependency mode, you can select dependent paragraphs that will be executed after the current paragraph has finished running.

You can save or cancel your changes by clicking **Save** or **Cancel**.

Viewing Dependents

To view dependencies of a paragraph when not in dependency mode, select the paragraph. The dependents will be displayed with a light blue border and a number indicating that paragraph's position in the execution order.

Keyboard Shortcuts for Notebooks

When working with notebooks, you can use keyboard shortcuts to trigger actions by using only the keyboard.

You can open an overview of all keyboard shortcuts and perform a search for shortcuts by using the context menu in the top-right corner. If the page you are currently on does not have any keyboard shortcuts, this menu item will not appear. You can also search for shortcuts by pressing Ctrl+Shift+F.

See Keyboard Shortcuts for Graph Studio in the *Accessibility Guide for Oracle Cloud Services* for more information on keyboard shortcuts for notebooks in Graph Studio.

Example Notebooks

Graph Studio includes a set of examples.

You can find these examples in the Notebooks section.

- Getting Started: BANK_GRAPH
- Getting Started: Intro to PGQL using the SH property graph
- Getting Started: Get started with an in-memory graph
- Getting Started: SPARQL Introduction
- Getting Started: Using the built-in notebooks
- Use Cases: Graph Queries on the SH sample data
- Use Cases: Part 1 Exploring Social Networks: A Guide to Oracle Graph

Each of these notebooks contains a set of Markdown paragraphs that explain each step of the example.

Each example notebook is ready to execute but ***read-only*** by default, so that they remain unchanged for other users of Graph Studio. To remove the read-only state, first create a private copy of the notebook by clicking **Clone** at the top of the example notebook.

After the private copy has been created, click **Unlock** to remove its read-only state.

After the private copy is unlocked, you can run each paragraph one-by-one by clicking **Run**.

8

Work with Templates in Graph Studio

A template allows you to persist graph visualization and notebook settings.

You can apply these custom built templates to your notebook.

Topics

Create a Template

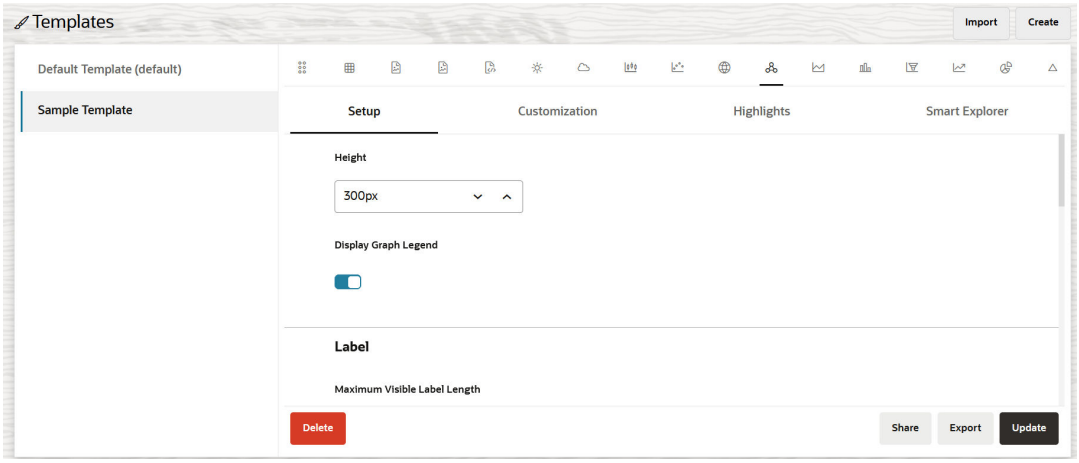
You can create custom templates that you can use in your notebook to quicky format your graph visualization.

The following are the steps to create a template:

1. Click **Templates** on the left navigation menu and navigate to the Templates page.
2. Click **Create**.

   The **New Template** window opens.
3. Enter the **Name** of the template.
4. Click **Create**.

   This creates a new template and the new template name gets listed on the left pane. Graph Studio displays the default settings for the template on the right pane. The right pane is again divided into two sections. The left section lists the menu options for the various components that can be configured in a template and the right section displays the corresponding parameter settings for the selected menu item as shown:



5. Select the component to be formatted from the menu and configure the required settings.

6. Click **Update** at the bottom-right of the page.

   The template gets saved with the custom settings.
   You can also import and export settings using the **Import** and **Export** buttons.

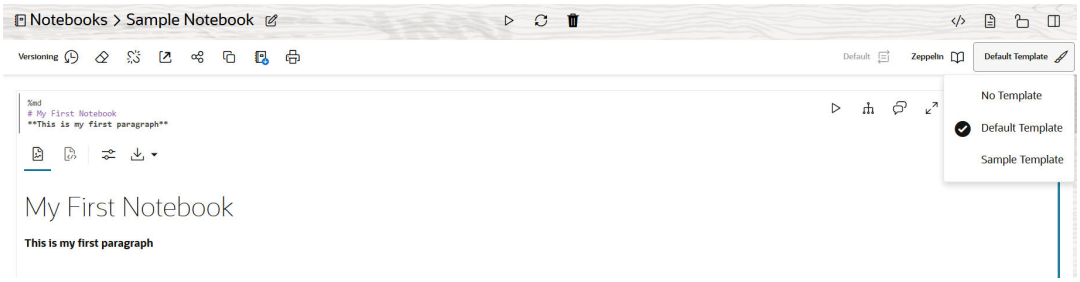
> **Note**
>
> Templates are imported and exported using the JSON file format only.

Use a Template in a Notebook

You can apply a custom template to your notebook.

The following are the steps to use a custom template in a notebook:

1. Click **Notebook** on the left navigation menu and navigate to the Notebooks page.
2. Open a **Notebook**.
3. Select the required **template** as shown:



   The custom settings in the selected template gets applied to the notebook.

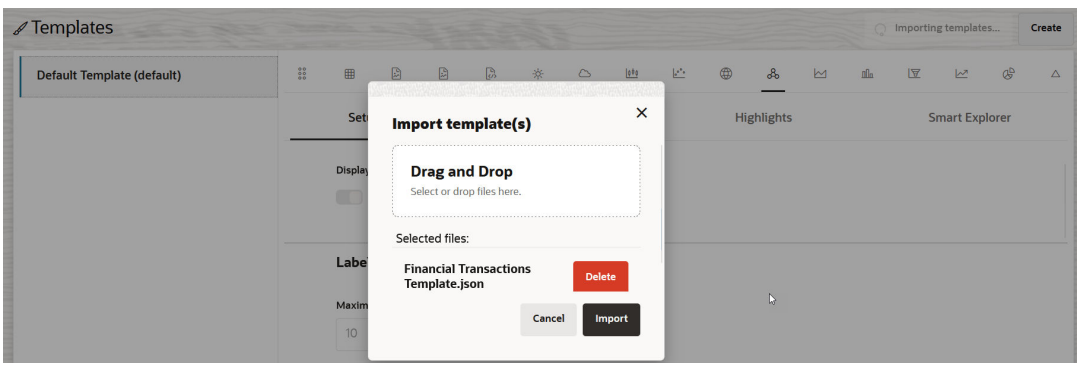
Import a Template

Graph Studio allows you to import a previously exported template in JSON format from your local system.

Perform the following steps to import one or more templates:

1. Navigate to the **Templates** page.
2. Click **Import** on the top right corner of the page.

   The **Import template(s)** window opens as shown:



3. Select one or more files from your local system or drag and drop the required files in the **Drag and Drop** section.
4. Optionally, review and verify the **Selected files**. Click **Delete** if you wish to remove a selected file.
5. Click **Import**.

   The files are imported as templates in Graph Studio.

Manage Templates

Graph Studio allows you to update, share, export, or delete an existing template.

To perform any one of the supported actions on an existing template:

1. Navigate to the **Templates** page.
2. Select the desired template on the left pane.

   Choose to perform any one of the following actions:

   - **Update**:
     a. Modify the required parameter values for the template.
     b. Click **Update** to update the template.
   - **Share**:
     a. Click **Share** to share the template.
     The **Share template** window opens and displays the default template permissions.
     b. Select the user or role from the **Add New Permissions** drop-down list.
     c. Click **Add** and set the permissions for the selected user.
     d. Click **Save** to share the template.
   - **Export**:
     – Click **Export** to export the template.
     The template gets saved in JSON format to your local system.
   - **Delete**:
     a. Click **Delete**.
     b. Confirm **Delete** to delete the template in Graph Studio.

9

Visualize and Interact with Graph Data in Graph Studio

You can visualize graph data in the form of a graph or table visualization.

Graph Studio provides the option to switch between graph or table visualization.

> **Note**
>
> All the graph visualization features explained in the following topics apply for property graphs. In case of RDF graphs, only selected visualization features are supported. Those features that do not apply will appear grayed out on the graph visualization panel for RDF graphs.

Topics

About Graph Visualization and Manipulation

The graph visualization feature allows you to visually explore a graph directly in the graph visualization panel.

Graph visualization and manipulation actions are available in several parts of the Graph Studio user interface, including:

- Property graph wizard - through the **Preview** tab in the Define Graph step.
- Graphs page - through the **Preview** tab in the graph details section for a selected graph.
- Notebooks.

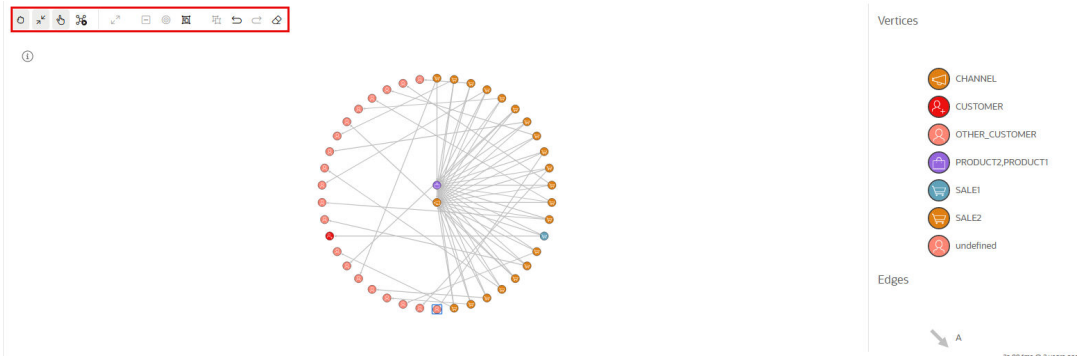
> **Note**
>
> Some graph visualization and manipulation features are not enabled in Preview mode.

Manipulate a Graph Visualization

Graph manipulation lets you interact with a loaded graph visualization.

To manipulate a graph:

1. Navigate to the toolbar (shown highlighted in the following figure) on the Graph Visualization panel.



2. Hover over any one of the **icons** to view a tooltip describing its purpose.

   The following actions are available from the graph manipulation toolbar or tooltip:

   - **Expand** fetches n-hop neighbors of selected vertices or neighbors that fulfill certain criteria if Smart Expand is used.
   - **Drop** removes selected vertices from the view.
   - **Focus** shifts the focus of the view; it drops everything and fetches n-hop neighbors of the selected vertex.
   - **Group** groups selected vertices into a ***super vertex***.
     You can customize the appearance of super vertices by using the graph visualization property **Grouped Vertex** in the Highlights tab of graph visualization settings modal.
   - **Ungroup** ungroups a group (that is, ungroups a super vertex).
   - **Undo** undoes (reverses the effect of) the last action.
   - **Redo** repeats the last action.
   - **Reset** resets the visualization to its default state.

3. Select the desired **action**.

   The graph is altered accordingly.

   You can also manipulate a graph visualization using the following features:

   - **Smart Explorer:** Lets you specify conditions for properties for navigation and destination vertices and edges that must be fulfilled when expanding or grouping vertices.
     See Expand Vertices Using Smart Expand for details on expanding vertices.

     See Group Vertices Using Smart Group for details on grouping vertices.
   - **Visible Graph Mode:** Allows you to store your graph data in a variable which can be used in further graph queries.
     See Enable Visible Graph Mode for more information.

Enable Visible Graph Mode

Visible Graph mode allows you to store your visible graph along with any graph manipulation actions in a variable. You can later use this variable in your further queries.

To enable visible graph mode and to use the visible graph mode variable:

1. Click **Settings** on the Visualization panel.

   This opens the **Settings** dialog.

2. Click the **Graph Exploration** tab.
3. Switch on the **Enable Visible Graph Mode** toggle in the **Visible Graph Sharing** section.

4. Optionally, change the default name of the variable in the **Global Variable Name** field.
5. Click the icon to copy the visible graph mode variable name to the clipboard.
6. Click **X** on the top-right to close the **Settings** dialog.

   The graph data gets stored in the variable. You can now query the vertices and edges of the graph using the variable as shown:

   - **Vertices**: `<visible_graph_mode_variable_name>.get("V")`
   - **Edges**: `<visible_graph_mode_variable_name>.get("E")`
7. Use the variable in your further queries.

   The following example creates a prepared statement for a query. The visible graph mode variable is used in the `setArray` method to set the bind variable to an array of values.

```
%java-pgx
var prepared_stmnt = graph.preparePgql("SELECT * FROM MATCH (v) WHERE
v.acct_id IN ?");
prepared_stmnt.setArray(1, visible_graph.get("E"));
var r = prepared_stmnt.executeQuery();
out.println(prepared_stmnt.executeQuery());
```

Expand Vertices Using Smart Expand

Smart Expand allows you to expand vertices based on specified conditions for properties of navigation and destination vertices or edges.

You can configure Smart Expand on a graph visualization as described in the following steps:

1. Click **Settings** on the Visualization panel.

   This opens the **Settings** dialog.
2. Click the **Smart Explorer** tab and click **New Smart Explorer**.

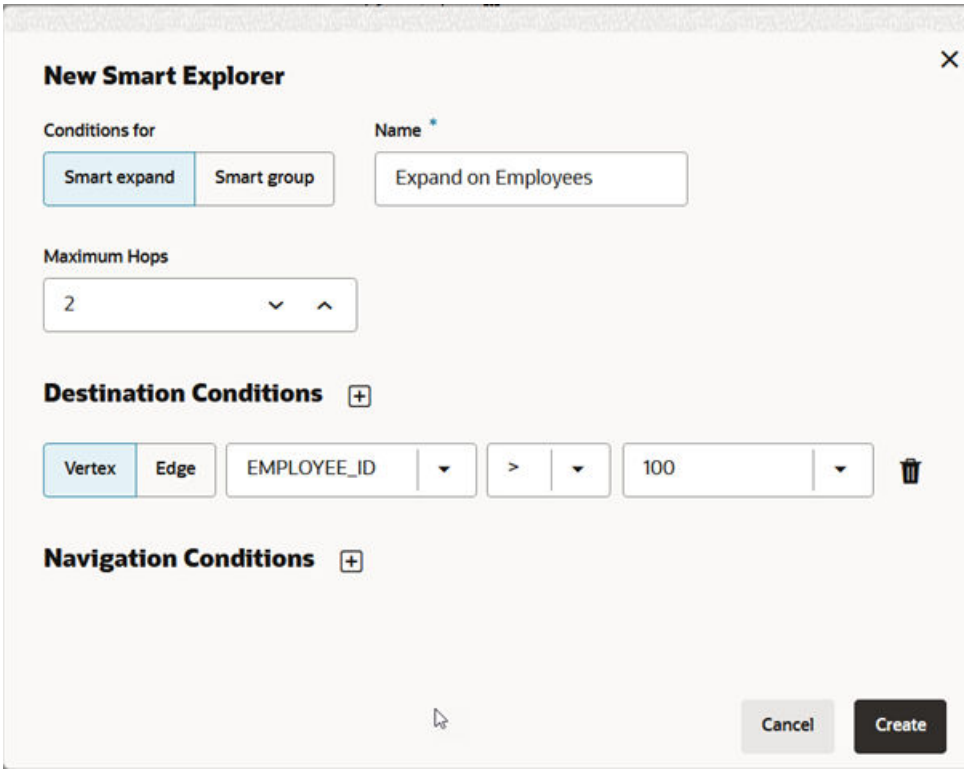
   The **New Smart Explorer** window opens.
3. Set the **Conditions for** field to **Smart Expand**.
4. Enter a **Name**.
5. Optionally, select the **Maximum Hops** count.

   This value determines the maximum path length. Smart Expand does not return vertices or edges that are in any path longer than this path length. The default value is `infinite`.

6. Optionally, click ⊞ to add **Destination Conditions** to identify the destination vertices or edges when expanding a selected vertex.

   Destination conditions are conditions that you apply to the last vertex or edge in the path. It does not apply to the vertices selected for expand.

   A row to create a new condition appears as shown:



   Each condition includes the following options:

   - target vertex or edge element that the navigation condition applies to
   - property of the target element
   - operator to apply (such as, =, <, >, and so on)
   - value to be fulfilled for the operator and property

   It uses numeric comparison if the property value is convertible to number and lexicographic comparison otherwise.

   Repeat this step to add as many destination conditions as required.

7. Optionally, if there are multiple destination conditions, then join your conditions by clicking **Match All** or **Match Any** as required.

8. Optionally, click ⊞ to add one or more **Navigation Conditions** that need to be fulfilled when expanding a vertex.

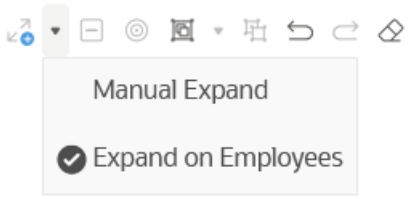
   Note the following:

   - Navigation conditions are conditions applied to the vertices or edges that are not the origin vertex or the destination vertex, but those that are on a path that connects the origin and destination vertex.
   - The conditions that you specify are applied to the vertices or edges that are on the path of your expand. It does not apply to the vertices selected for expand.

The options for adding a navigation condition and joining multiple conditions is same as described in the preceding steps for destination conditions.

9. Click **Create**.
10. Click **X** on the top-right to close the **Settings** dialog.
11. Click the **Expand** drop-down list in the exploration toolbar to view the list of Smart Expand names.
12. Select the required Smart Expand **Name**.

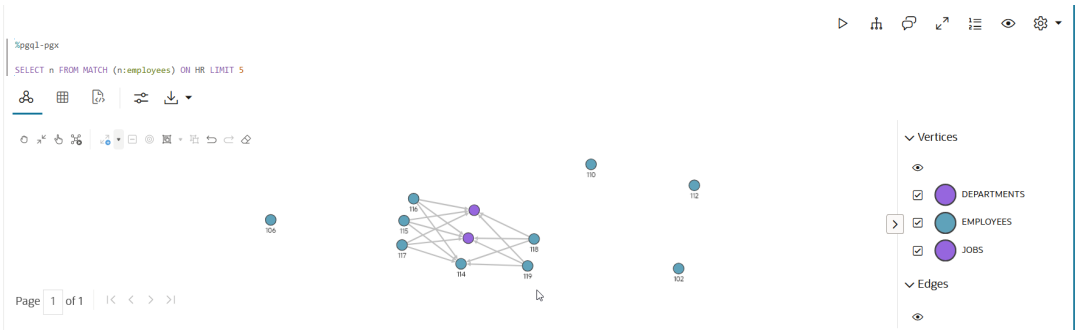
    For example:



13. Select a specific **vertex** or multiple **vertices** on the graph and click the **Expand** action on the graph manipulation toolbar.

> **Tip**
>
> Alternatively, you can apply Smart Expand from the tool tip. You can display the tool tip by using a right-click on the selected vertex.

Smart Expand fetches a shortest path to the vertex or vertices that are within the specified maximum path length, fulfilling the navigation and destination conditions for the selected vertex or vertices.
The following example shows expanding on an employee vertex which fetches all two-hop neighbors with `employee_id > 100`.



> **Note**
>
> If you do not configure the maximum hop count, navigation or destination conditions for Smart Expand, then the graph expands on the default infinite hop count value.

Group Vertices Using Smart Group

Smart Group allows you to group vertices based on specified vertex conditions or edge conditions or a combination of both.

There are two ways you can apply Smart Group:

- **Automatic Smart Group**: Applies grouping to the entire graph.
- **Manual Smart Group**: Applies grouping to the selected vertices that fulfill the specified conditions. But, if no vertices are selected, it applies to the entire graph.

To configure and to apply Smart Group for your graph:

1. Click **Settings** on the Visualization panel.

   This opens the **Settings** dialog.

2. Click the **Smart Explorer** tab and click **New Smart Explorer**.

   The **New Smart Explorer** window opens.

3. Set the **Conditions for** field to **Smart Group**.

4. Enter a group **Name**.

   > **Tip**
   >
   > You can use this **Group Name** in Highlights to customize the appearance of grouped vertices.

5. Switch on the **Automatic** toggle.

   > **Note**
   >
   > Switch off the **Automatic** toggle for manual Smart Group.

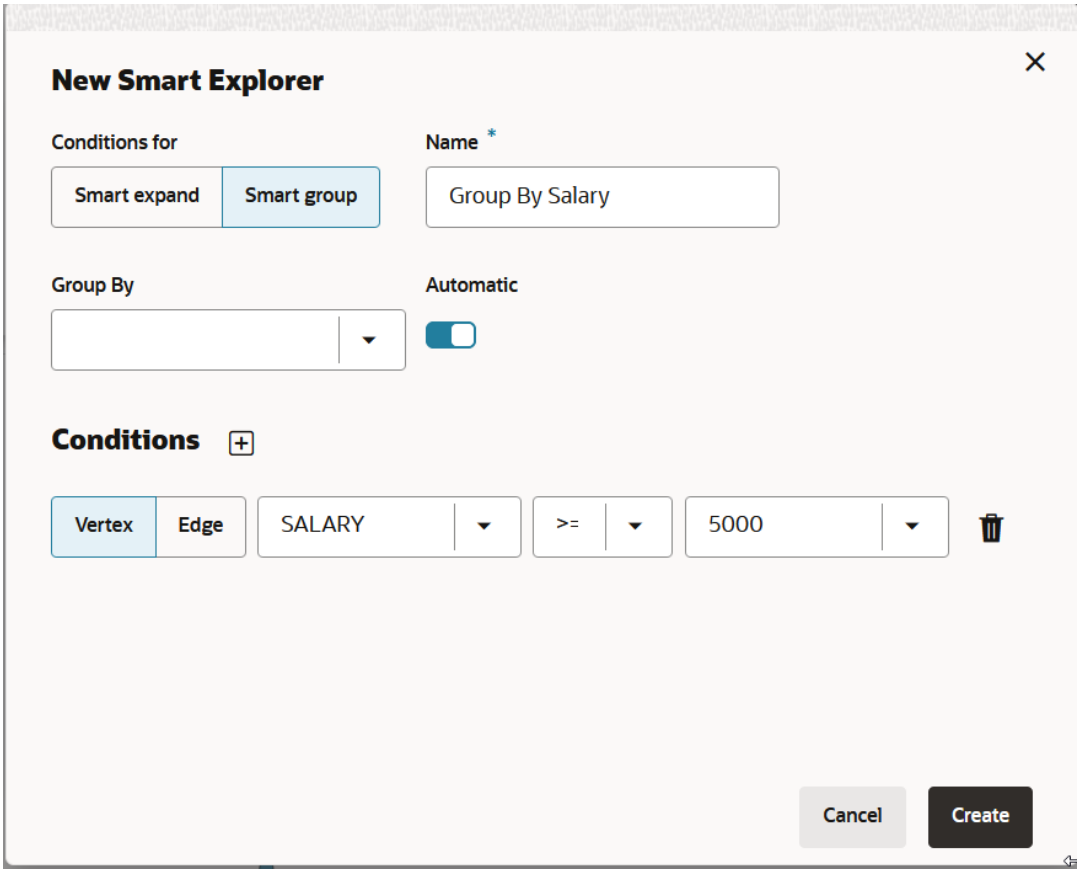
6. Optionally, select property value from the **Group By** drop-down list.

   If `Group By` is set, Smart Group creates one group per each available value of the specified property from all vertices fulfilling given conditions. Otherwise, Smart Group results in just one group containing all allowable vertices.

   If Smart Group has any edge conditions, then the created groups are further split into separate parts where all vertices are reachable just through edges fulfilling specified edge conditions.

7. Click  to add a condition for grouping.

   A row to create a new conditions appears as shown:



Each condition includes the following options:

- target vertex or edge element that the condition applies to
- property of the target element
- operator to apply (such as, =, <, > and so on)
- value to be fulfilled for the operator and property

It uses numeric comparison if the property value is convertible to number and lexicographic comparison otherwise.

8. Set the required condition on the target **Vertex** or **Edge** element as applicable.
9. Optionally, join your conditions by clicking **Match All** or **Match Any** as required.

> **Note**
>
> The join options are displayed only when you have multiple conditions.

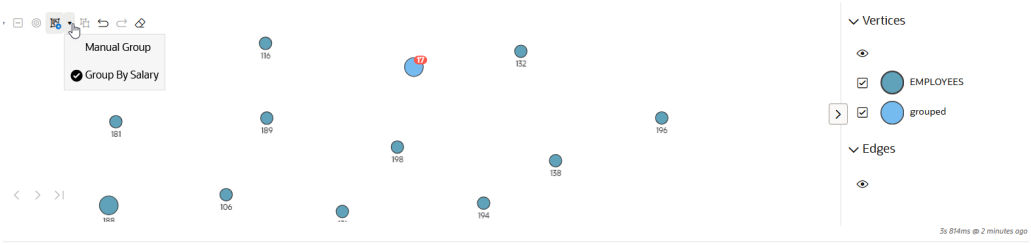
10. Click **Create** to add one or more conditions.
11. Click **X** on the top-right to close the **Settings** dialog.

    If the Smart Group is configured as automatic, then the conditional grouping is applied on the whole graph displayed in the visualization panel.

    Otherwise, perform the following steps to apply **Manual Smart Group**:

    a. Click the **Group** drop-down list in the exploration toolbar.

    b. Select the required Smart Group **Name** as shown:



The preceding example shows grouping of all employee vertices with `salary >= 5000`.

c. Click specific **vertices** on the graph and click the **Group** action on the graph manipulation toolbar.

Vertices fulfilling the configured conditions are grouped together.

> **Note**
>
> - If Smart Group has an edge condition, then you can select vertices that are connected by the edge relationship.
> - If you do not select vertices on the graph, then the manual Smart Group is applied to the whole graph.

Annotate a Graph

Graph Annotation mode allows you to add vertices and edges on a graph visualization. You can also add or edit the graph's properties for visualization.

To annotate a graph:

1. Set to graph **Annotation Mode** on the Graph Visualization panel.
2. Annotate the **graph visualization** by performing one of the following actions:
   - Add a new vertex by clicking anywhere in the graph visualization canvas.
   - Create a new edge by dragging the mouse from the source vertex to the target vertex.
   - Move a vertex by dragging the mouse while holding the Shift key or with initial long click on it.
   - Add properties to new vertices and edges or edit the properties of existing ones.

   All your edits are added to the graph manipulation action stack, so you can undo, redo, or clear them using appropriate graph manipulation actions. The `addedByUser` and `editedByUser` properties are added automatically to vertices and edges that you create or edit, so that you can use them in Graph Highlights operations.

> **Note**
>
> All graph annotations persist only on the graph visualization and not on the actual graph itself. You can remove the graph annotations by resetting the graph visualization to its default state.

Visualize a Dynamic Graph

Graph Studio allows you to visualize the evolution of a graph over time. This time-based analysis provides great insights on the graph data.

To visualize a dynamic graph, you must have a date or a time property in your graph data. It can either be a vertex or an edge property.

You must then configure the graph visualization settings to use these properties as shown in the following steps:

1. Click **Settings** on the Visualization panel.

   This opens the **Settings** dialog.
2. Click the **Graph Exploration** tab.
3. Switch on the **Enable Network Evolution** toggle.
4. Select a network element from the **Based On** drop-down list.

   You can configure the network evolution to be based on vertices or edges or both.

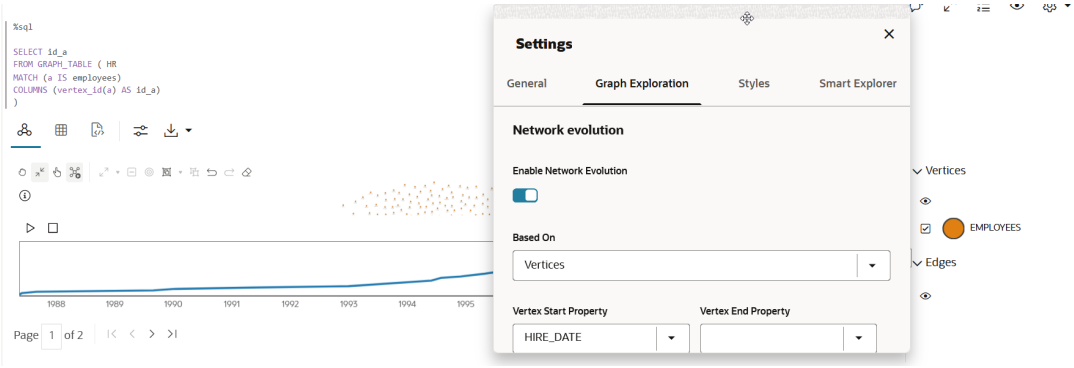
   Depending on your selection, you must select one or more of the following properties:

   - **Vertex Start Property**: Select the name of the property to use for the vertex filtering. The time frame for the graph will be after the *Vertex Start Property*.
   - **Vertex End Property**: Optionally, select the name of the property to use for the vertex filtering.
     The time frame for the graph will be before the *Vertex End Property*.
   - **Edge Start Property**: Select the name of the property to use for the edge filtering. The time frame for the graph will be after the *Edge Start Property*.
   - **Edge End Property**: Optionally, select the name of the property to use for the edge filtering.
     The time frame for the graph will be before the *Edge End Property*.
5. Select the data type value from the **Data Type of the Property** drop-down list.

   Note that Graph Studio supports only `Integer` and `Date` type property values.
6. Optionally, enable **Advanced Settings** if you want to explore advanced network evolution features and select one or more of the following options:

   - **Values to Exclude**: Select values to additionally filter vertices or edges.
   - **Behavior**: Select the behavior of the excluded values.
   - **Increment**: Select the interval size.
   - **Chart Type**: Select the type of the chart to be used to show the network evolution.
   - **Height**: Select a value to specify the height of the network evolution chart.
   - **Milliseconds Between Steps**: Select a value to specify how often does the playback advance in ms.
   - **Number of Items per Step**: Select a value to specify how many steps are taken per time out during playback.
7. Click **X** on the top-right to close the **Settings** dialog.

   A time bar showing the network evolution of your graph data is displayed at the bottom of the graph visualization as shown:



You can view the graph animation by clicking the **Play Network Evolution** button. The animation shows the changes in the graph network over time.

Additionally, you can activate and deactivate network evolution, by clicking **Activate Network Evolution** which is show highlighted in the preceding figure.

Use Live Search in Graph Visualization

Using the Live Search feature in Graph Studio, you can search the currently displayed graph and add live fuzzy search score to each item.

Perform the following steps to configure and apply Live Search in your graph visualization. The steps assume that a graph is displayed in the visualization panel.

1. Click **Settings** on the Visualization panel.

   This opens the **Settings** dialog.

2. Switch ON the **Enable Live Search** toggle in the **General** tab.

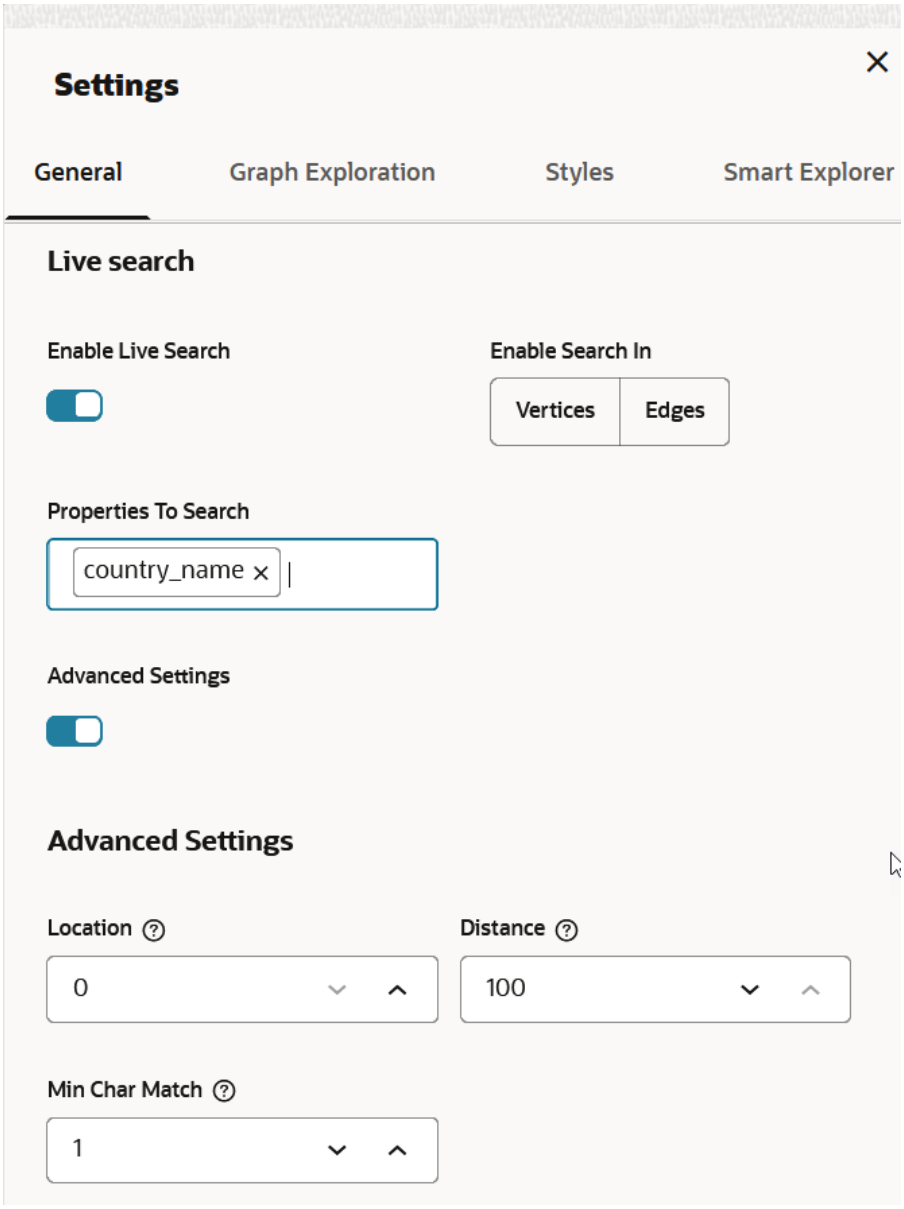
   This enables the search, adds the search input to the visualization, and allows you to further customize the search. It is important to note that you can only search the graph that is currently displayed in the visualization panel, and not the entire graph as stored in the database.

3. Select whether you want to search the properties of either **Vertices** or **Edges**, or both under **Enable Search In**.

4. Select one or more **Properties To Search** based on what you selected in the previous step.

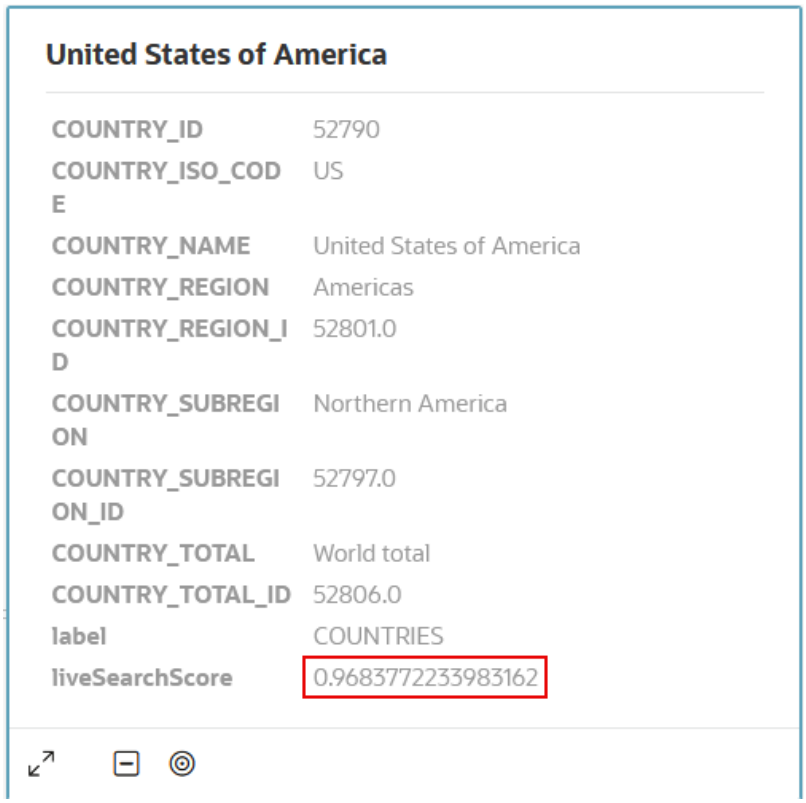
   Note that if you disable search for any graph element (vertices or edges) for which you already had selected the properties, then those properties will be stored and added back when you enable search again for that graph element.

   The following figure shows an example of configuring Live Search. As seen, Live Search is enabled for the vertex property, `country_name`.



5. Optionally, enable **Advanced Settings** if you wish to fine-tune the search even more and configure one or more of the following options:
   - **Location**: This determines approximately where in the text property the pattern is expected to be found. For instance, location value `0` indicates that the pattern is matched from the beginning of the text. Location value `1` indicates that the pattern will be matched from the second letter of the text and so on.
   - **Distance**: This 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.
   - **Min Char Match**: The minimum length of the pattern that needs to match.
6. Close the **Settings** dialog and rerun the visualization query.

The search input will be displayed towards the right side of the graph visualization. If you start typing the search keyword, the search will add a score to every vertex or edge, based on the settings and the search match. The Live search score can be viewed inside the tooltip, that can be triggered by right-clicking a vertex or edge. For example:



Manage the Graph Display Size

You can use the **Initial Display Size** visualization setting to control the number of graph elements (vertices and edges) to be displayed in the graph visualization panel.

The default **Initial Display Size** value is 100. For instance, consider the following example visualization:

In the preceding figure, the graph display size (shown highlighted) shows that:

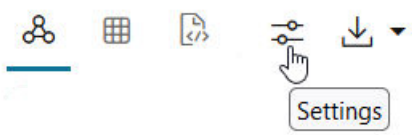
- 55 vertices out of the total 117 vertices are displayed.
- 47 edges out of the total 106 edges are displayed.
- 100 graph elements out of the total 223 elements are displayed.
- The progress bar shows the percentage of the graph that is displayed in the panel along with the percentage value.

You can move the slider to dynamically change the number of vertices or edges that you wish to display for visualization.

Settings for Graph Visualization

The Settings modal lets you specify options that control how graph data is displayed when it is visualized.

You can invoke the settings modal by clicking the settings icon as shown highlighted in the following figure:



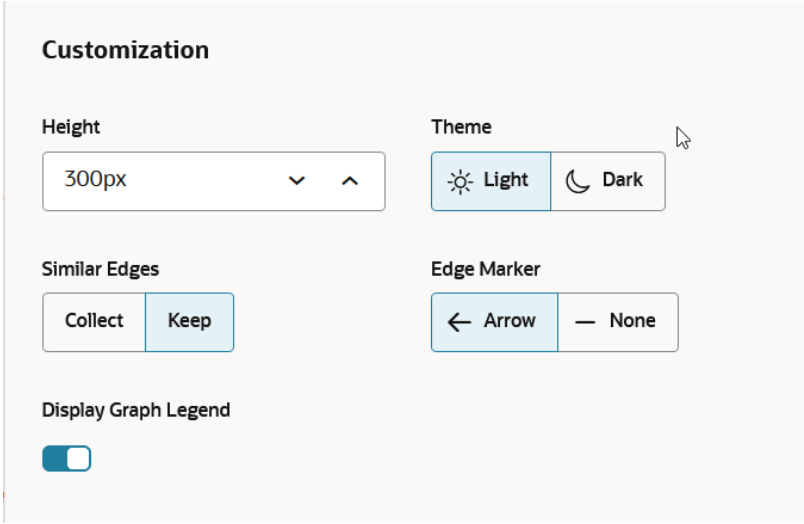
The Settings modal contains the following tabs that group the options according to their scope:

General

The **General** tab contains the general settings that affect the entire visualization, including search-related options. This tab comprises the following sections:

Customization

These are visualization settings that affect the visual aspects of the display.

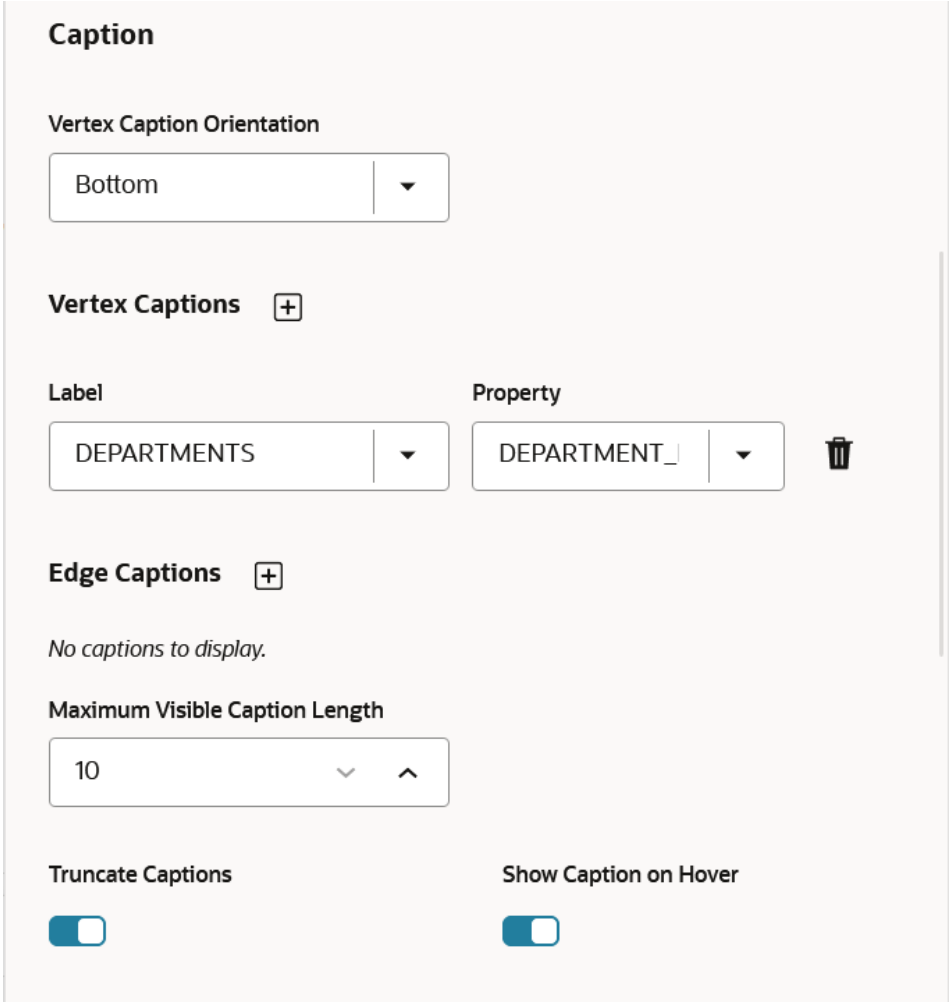


Customization settings include the following options:

Option	Description
Height	Height of the visualization. Setting the value to 0 will take the default height.
Theme	Toggles the visualization between light and dark theme (useful for presentations).
Similar Edges	Similar edges can be collected when this button is checked. Toggled edges give no overview of specific edges but a generalized tooltip.
Edge Marker	Determines if the outgoing edges have an arrow to show the flow direction.
Display Graph Legend	The graph legend will be displayed when this toggle is enabled.

Caption

The **Caption** section is displayed as shown:

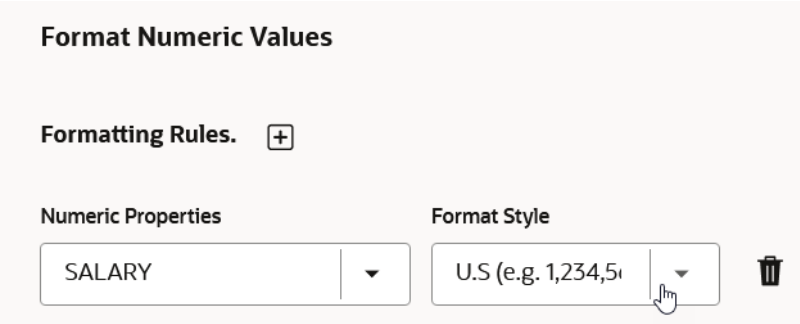


Caption settings include the following options:

Option	Description
Vertex Caption Orientation	Determines where the selected property will be displayed. Options are: *Bottom*, *Center*, *Top*, *Right*, *Left*.
Vertex Captions	A configurable list of captions based on vertex labels and their associated properties. You can add or remove captions, and the selected properties will be displayed on the corresponding vertices in the graph.
Edge Captions	A configurable list of captions based on edge labels and their associated properties. You can add or remove captions, and the selected properties will be displayed on the corresponding edges in the graph.
Maximum Visible Caption Length	Maximal `char` length of a truncated caption.
Truncate Captions	If enabled, captions will be truncated at a specific length as specified in the previous option.
Show Caption on Hover	If enabled, full captions will appear as a tooltip when hovering over a vertex.

Format Numeric Values

You can customize the display of numeric properties in a graph by applying various formatting styles for better readability. This section is displayed only if the graph contains numeric properties.



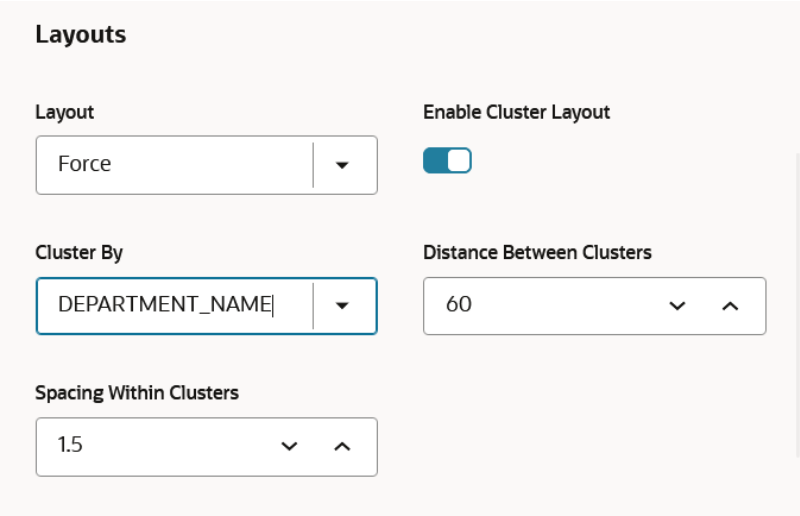
As seen in the preceding figure, you can add or delete a format style for a numeric property.

To add a formatting rule for a numeric property, click the **+** icon and configure the following options:

- **Numeric Properties:** Determines the numeric property to be formatted.
  Note that only unformatted numeric properties are listed in this drop-down.
- **Format Style:** Determines the display style for the selected numeric property.

Layouts

Graph Studio supports different graph layouts. Each layout has its own algorithm, which computes the positions of the vertices and affects the visual structure of the graph.



The following graph layout options are supported:

Option	Description
Random Layout	Positions the vertices in random positions within the viewport.

Option	Description
Grid Layout	Positions the vertices in a well-spaced grid. It supports the following configurable property: • **Spacing**: Sets the space between the elements in the grid.
Circle Layout	Positions vertices in a circle. It supports the following configurable property: • **Radius**: Sets the radius of the circle.
Concentric Layout	Positions vertices in concentric circles. It supports the following configurable property: • **Minimum Vertex Spacing**: Sets the minimum spacing in between vertices. It is used for radius adjustment.
Force Layout	Attempts to create an aesthetically-pleasing graph based on the structure of the graph, with the goal of positioning the vertices in the viewport so that all the edges are of approximately equal length and there are as few crossing edges as possible. It has the following configurable properties: • **Enable Cluster Layout**: Determines if cluster based layout is enabled. If this parameter is switched *ON*, then the following cluster options will be displayed: – **Cluster By**: By default, the cluster layout uses the first element in vertex labels to form the cluster. Alternatively, this can be set to the property name of a vertex, and the clusters will be formed based on the property value. – **Distance Between Clusters**: Influences the forces among clusters (that is, to push clusters away from each other). – **Spacing Within Clusters**: Determines how close different vertices are rendered next to each other within the clusters.
Hierarchical Layout	Organizes the graph using a DAG (Directed Acyclic Graph) system. It is especially suitable for DAGs and trees. It supports the following configurable properties: • **Ranking Algorithm**: Specifies the type of algorithm used to rank the vertices. Possible values are Network Simplex, Tight Tree and Longest Path. • **Network Simplex**: Assigns ranks to each vertex in the input graph and iteratively improves the ranking to reduce the length of the edges. • **Tight Tree**: Constructs a spanning tree with tight edges by adjusting the ranks of the input vertex. The length of a tight edge matches its `minlen` attribute. • **Longest Path**: Pushes vertices to the lowest layer possible, leaving the bottom ranks wide and leaving edges longer than necessary. • **Direction**: Specifies the direction of the graph. Possible values are Top Bottom, Bottom Top, Left Right, and Right Left • **Alignment of Rank Nodes**: Determines the alignment of the ranked vertices. Possible values are Up Left, Up Right, Down Left and Down Right • **Vertex Separation**: Sets the horizontal separation between the vertices. • **Edge Separation**: Sets the horizontal separation between the edges. • **Rank Separation**: Sets the separation between two ranks (levels) in the graph.

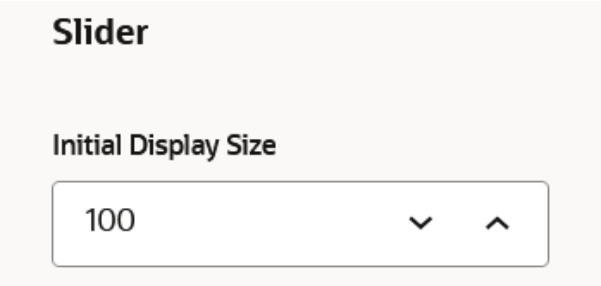
Option	Description
Radial Layout	Displays the dependency chain of a graph by using an outwards expanding tree structure. It can be especially useful if the graph data has a hierarchical structure and contains many children for each parent vertex. It has the following configurable properties: • **Starting Point (left, top, right, bottom)**: Defines the starting point of the radial layout and thus allows you to change the orientation. • **Arc Degree slider (0° - 360°)**: Specifies the arc degree of the circle used for the radial layout. Higher arc degree values can help to detangle the network; lower values make it more compact. • **Packing slider (0 - 5)**: Reduces the separation gap between neighboring vertices if they share the same parent vertex. If set to 0, no packing will be applied. • **Intelligent Separation**: Reduces the separation gap proportionally to the depth level of each vertex.

Option	Description

Geographical Layout	Enables you to overlay the graph on a map, given that latitude and longitude coordinates exist as graph properties on the graph's vertices. It has the following configurable properties:
	• **Latitude Property**: The vertex property to use for determining the latitude of a vertex.
	• **Longitude Property**: The vertex property to use for determining the longitude of a vertex.
	• **Map Type**: You can select map type either in map visualization or graph visualization settings, or provide your own sources and layers. Supported types are:
	– *World Map* ("oracle-elocation")
	– *OSM Positron* (default)
	– *OSM Bright*
	– *OSM Darkmatter*
	– *Custom type*: Custom type has the following two additional fields. It is important to note that you must provide these attribute properties separately from visualization because of security reasons.
	* **Sources**: Provide your own sources in JSON format which will be used in the map. For example:
	`{ "oracle-elocation": { "type": "raster", "tiles": ["https://elocation.oracle.com/mapviewer/mcserver/ELOCATION_MERCATOR/world_map_mb/{z}/{y}/{x}.png"], "tileSize": 256, "minzoom": 0, "maxzoom": 18 } }`
	* **Layers**: Provide the layers that you want to display on the map as an array of JSON elements. For example:
	`[{ "id": "elocation-tiles", "type": "raster", "source": "oracle-elocation" }]`
	Also, note the following when visualizing a graph on a map:
	• You can change the viewport of the map by clicking and dragging the mouse on the map.
	• You can zoom into the map using the + / - buttons, through the scrolling wheel of your mouse, or through pinching motions using your track pad.
	• You can also use the `Shift` key and then click and drag the mouse to define a field. The view will zoom into that area, changing both viewport and zoom level at the same time.
	• You can change the orientation and angle of the viewport by pressing `Ctrl` and then clicking and dragging the mouse on the map.

Slider

You can configure the **Initial Display Size** parameter to customize the total number of graph elements to be shown in the graph visualization.



See Also: Manage the Graph Display Size

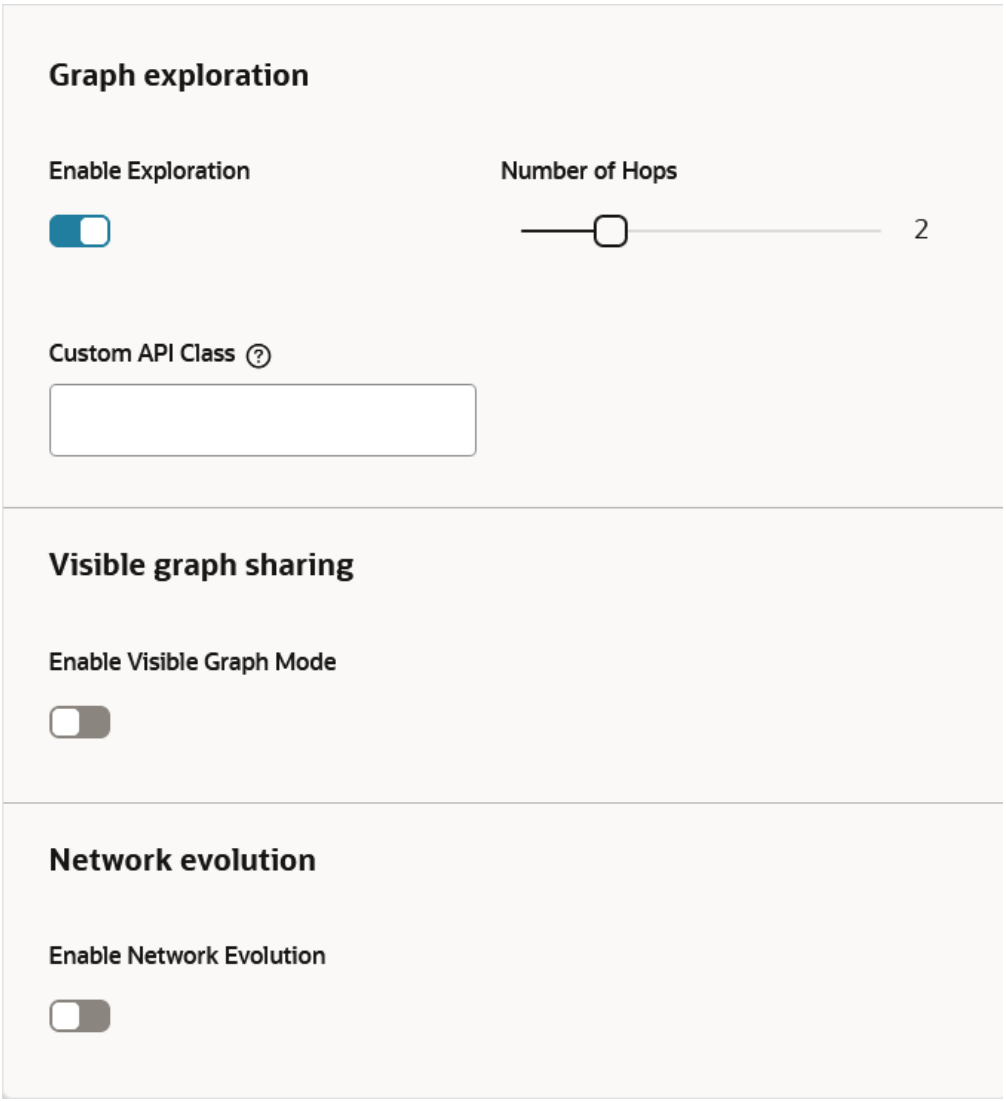
Live Search

The **Live Search** section appears as shown:

You can enable *Live Search* to search the displayed graph. Live fuzzy search score is added to each item and you can create a Style which visually shows the results of the search in the graph immediately. See Use Live Search in Graph Visualization for more information.

Graph Exploration

The **Graph Exploration** tab appears as shown in the following figure:



The **Graph Exploration** tab comprises the following sections:

Graph exploration

- **Number of Hops**: You can specify the number of hops for graph manipulation.

Visible graph sharing

- **Enable Visible Graph Mode**: You can enable or disable the visible graph mode. See Enable Visible Graph Mode for more information.

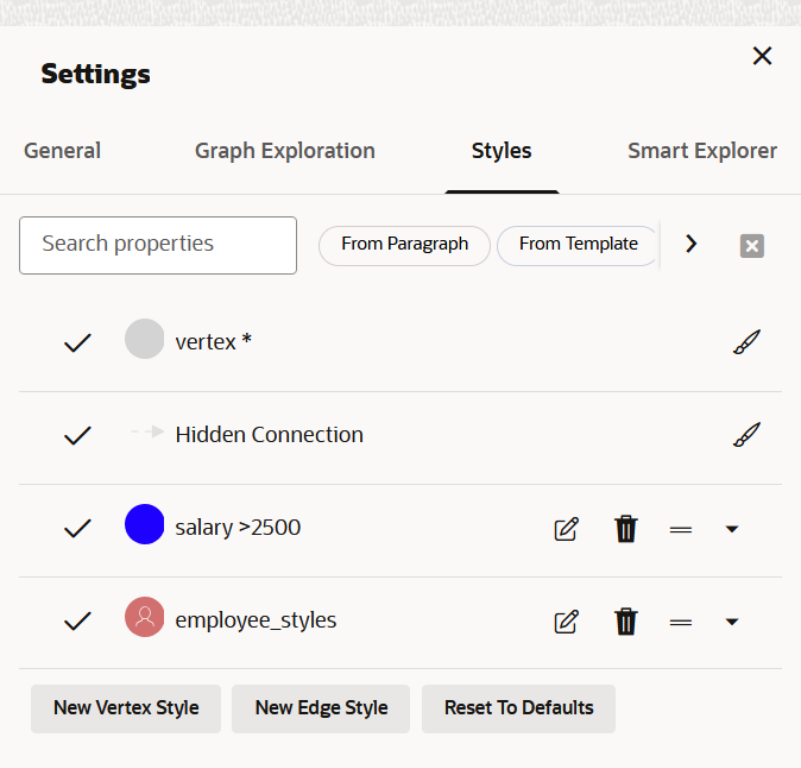
Network Evolution

- **Enable Network Evolution:** Enables you to visualize the evolution of a graph over time.

See Visualize a Dynamic Graph for more information.

Styles

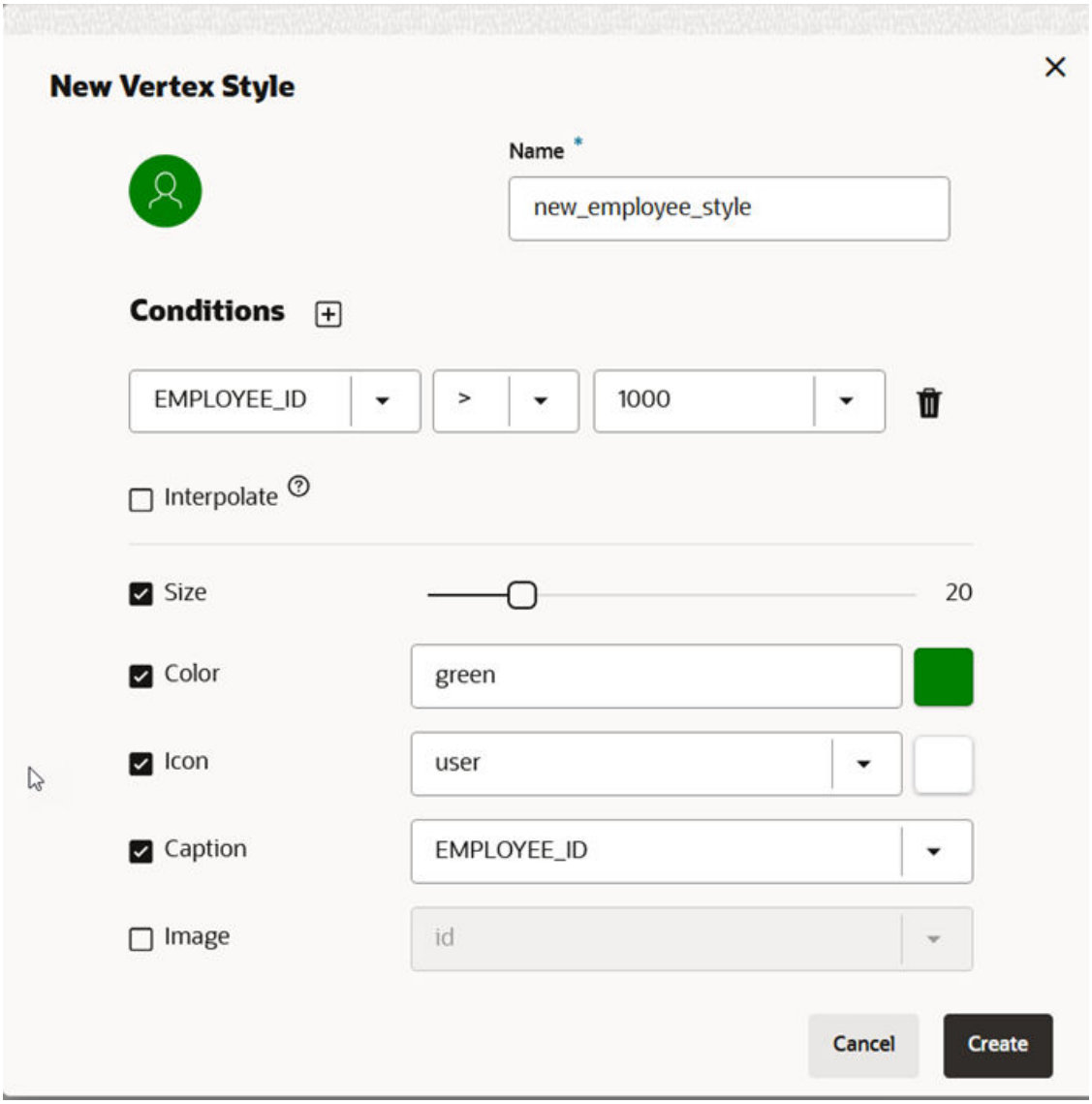
The **Styles** tab allows you to customize the appearance of vertices and edges based on search criteria. It allows you to modify collectively styling such as color, size, or icons for vertices or edges that match the search criteria.



On the Styles tab, you can:

- View the list of existing styles for vertices and edges.
- Filter the list of styles based on an input string or applicable tags.
- Drag and reorder the items on the list.
- Create custom vertex or edge styles.
- Edit an existing style.
- Enable or disable a style.

The following describes the properties and operations when creating a new style:



- **Name**: Name of the style. This adds a text in the graph legend for elements where this style applies. This field is mandatory for styles creation. Style value can be either constant or interpolation based on some property value. Interpolation settings include:
  - The property of the element
  - The minimum or maximum value (If not specified, the minimum or maximum property value from all matched elements will be used, and the highlight will be applied proportionally between the selected minimum and maximum values of the specified property.)
- **Conditions** The search condition lets you define how vertices or edges that are influenced by a style are filtered. To configure a search criteria, you must specify the element type to search for (vertex or edge), search conditions, condition operator (*match all* or *any*). Each condition includes the property of the given element, the operator you want to apply (=, <, <=, >, >=, !=, ~, *), the property value that needs to be fulfilled for the operator. It uses numeric comparison if the property value is convertible to number and lexicographic comparison otherwise.
- The following options apply to highlight the vertices or edges that match the search criteria:

- **Size**: Sets the size of the vertex or edge to the specific value. If interpolation is selected, the slider will have two ends and the size of the vertex or edge is interpolated based on the result of the search criteria.
- **Color**: Sets the color of the vertex or edge. If interpolation is selected, the combobox will allow to add multiple colors. All vertices or edges are interpolated between these colors based on the result of the search criteria.
- **Icon**: Sets an icon to the vertex (not applicable to edges). If interpolation is selected, multiple icons can be selected.
- **Caption**: Sets the caption to the vertex or edge.
- **Image**: Sets an image to the vertex based on an `href` property (not applicable to edges).
- **Animations**: Allows to set certain animation `css` classes to vertices and edges (such as flashing, dotted-line, animated dotted-line, pulsating) and duration of an animation cycle.

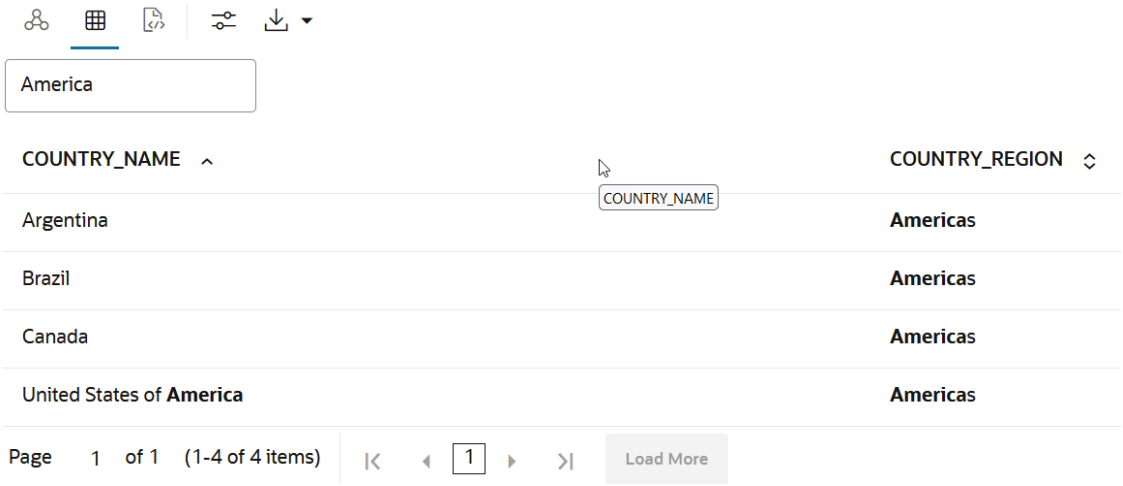
Smart Explorer

The **Smart Explorer** tab supports the following actions:

- **Smart Expands:** Allows you to expand vertices based on the specified conditions for properties of navigation and destination vertices or edges. See Expand Vertices Using Smart Expand for more information.
- **Smart Groups:** Allows you to group vertices based on specified conditions. See Group Vertices Using Smart Group for more information.

About Table Visualization

You can visualize the result of a graph query in tabular format. The table can be sorted by columns in ascending or descending order.



Additionally, the table can be filtered for a specific search term. Rows that do not contain this term are hidden from view and the remaining rows highlight the location of the search term within the row.

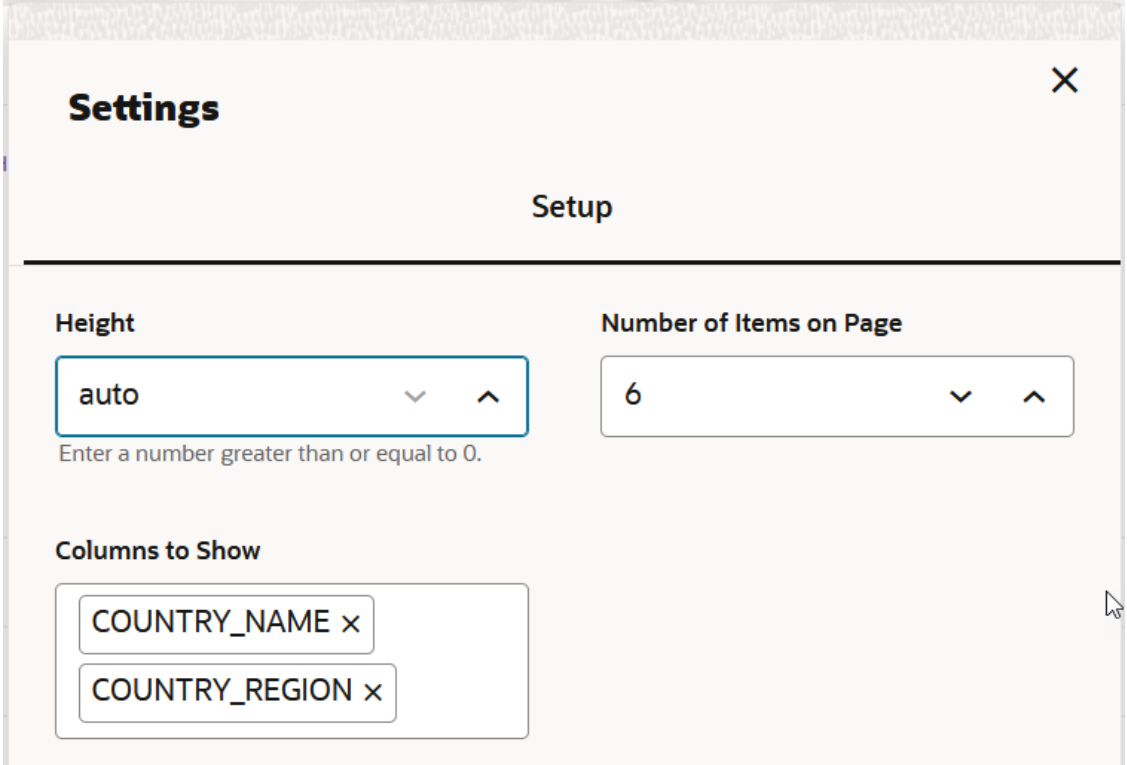
Topics

- Settings for Table Visualization

Settings for Table Visualization

You can format the table by configuring the options in the Settings dialog.

The **Settings** dialog for a table visualization is as shown:



The **Setup** tab contains the following options.

- **Height**: This parameter changes the height of the visualization. Setting the value to 0 will take the default height.
- **Columns to Show**: This parameter controls the columns (from the query results) to be displayed in the **Table**. You can also change the order of the columns by removing and adding them again at the desired position. The changes are reflected immediately in the table.
- **Number of Items on Page**: This sets the pagination size. By default five items per page are displayed.

10

Interactive Graph Visualization in Oracle APEX Applications

Using the APEX Graph Visualization plug-in, you can visualize and interact with property graphs on your Autonomous AI Database instance in an APEX application.

Topics

About the APEX Graph Visualization Plug-in

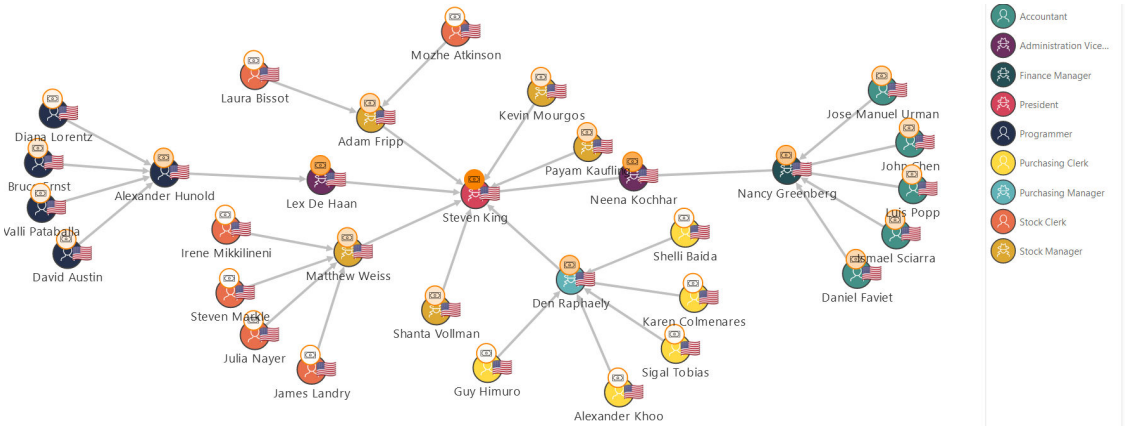
The APEX Graph Visualization plug-in integrates a Java Script Library that supports graph visualization in APEX applications.

See *Property Graph Visualization Developer's Guide and Reference* for more information.

The plug-in mainly allows you to:

- Construct a property graph for visualization from the graph data in your Autonomous AI Database instance.
- Explore the graph vertices and edges. You can also select and visualize these graph elements individually or in groups.
- Interact with the graph visualization by performing various actions such as changing the graph layouts, grouping or ungrouping selected vertices, removing selected vertices or edges, and so on.
- Style the vertices and edges in the graph by configuring the style settings such as size, color, icon, label values, and so on.
- Visualize and study the evolution of the graph over time.

The following figure shows an example of graph visualization in an APEX application using the plug-in:



Note that the plug-in supports icons in the Font APEX library.

Prerequisites for Using the APEX Graph Visualization Plug-in

Review the prerequisites for using the Graph Visualization plug-in in APEX applications.

1. Ensure that the schema associated with the APEX application workspace, where the Graph Visualization plug-in is imported, is a graph-enabled schema. To enable graph for a schema:
   a. Access Database Actions as an ADMIN user. See Access Database Actions as ADMIN for more information.
   b. Click **Database Users** in the **Launchpad** page under the **Administration** group.
   c. Locate the user card for your schema on the **User Management** page and click the Actions (three vertical dots) icon to open the context menu.
   d. Select **Enable Graph**.
      Graph gets enabled for the schema.

      Alternatively, you can also select **Edit**, turn on the **Graph** toggle on the **Edit User** page, and click **Apply Changes**.
2. The target application into which you want to import the plug-in exists in your APEX instance.
3. The target application is connected to the desired database (19c or 26ai) and the property graph to be used for visualization exists in the default database schema.
4. Note that the Graph Visualization plug-in version in the Oracle APEX 24.2 GitHub repository is supported only on APEX 24.2 version.

Get Started with the APEX Graph Visualization Plug-in (Oracle AI Database 26ai)

Get started with the APEX Graph Visualization plug-in in your APEX application on your Autonomous AI Database instance using Oracle AI Database 26ai.

Before you begin, ensure that you meet the prerequisites described in Prerequisites for Using the APEX Graph Visualization Plug-in.

1. Download the **Graph Visualization (Preview)** plug-in (`region_type_plugin_graphviz.sql`) from the Oracle APEX GitHub repository.
2. Sign in to your APEX workspace in your Autonomous AI Database instance.

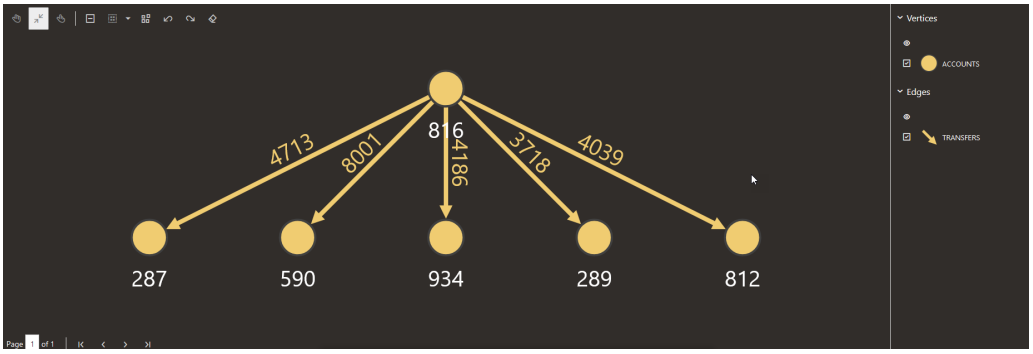
3. Create the `DBMS_GVT` package in your APEX workspace.
   a. Download the `optional-23ai-only/gvt_sqlgraph_to_json.sql` file from the Oracle APEX GitHub repository.
   b. Upload and run the `gvt_sqlgraph_to_json.sql` script in your APEX workspace (see Uploading a SQL Script).
4. Import the downloaded plug-in script (`region_type_plugin_graphviz.sql`) file into your target APEX application (see Importing Plug-ins).
5. Implement the plug-in in an application page to perform various graph visualizations.

   The following basic example describes the steps to visualize a graph existing in your database using the Graph Visualization plug-in.

   a. Open the application page in **Page Designer**.
   b. Select the **Rendering** tab on the left pane of the Page Designer.
   c. Right-click an existing component and add a new region component.
   d. Select the new region and configure the following attributes in the **Region** tab of the **Property Editor** on the right pane of the Page Designer:
      i. Enter the Identification **Title**.
      ii. Select **Graph Visualization (Preview)** as Identification **Type**.
      iii. Select the source **Location** as **Local Database**.
      iv. Select the **Type** value.
      You can choose either **SQL Query** or **PropertyGraph** as the Type value.
      v. Embed the SQL graph query to retrieve the graph data.
      Depending on the type selected in the previous step, you can provide the query as shown in the following examples:
      - **SQL Query:** Enter the SQL graph query input as shown:

```
SELECT *
  FROM GRAPH_TABLE (
           BANK_SQL_PG
           MATCH (a IS accounts) -[e IS transfers]-> (b IS
accounts)
           WHERE a.id = 816
           COLUMNS(vertex_id(a) AS id_a, edge_id(e) AS id_e,
vertex_id(b) AS id_b)
       )
```

      - **PropertyGraph :** Provide the SQL graph query as shown:
        – **Graph Name:** Select the SQL property graph name.
        – **Match Clause:** Enter the `MATCH` clause of the graph query. For example: `(a IS accounts) -[e IS transfers]-> (b IS accounts)`
        – **Columns Clause:** Enter the `COLUMNS` clause of the graph query. For example:
        `(vertex_id(a) AS id_a, edge_id(e) AS id_e, vertex_id(b) AS id_b)`
        – **Where Clause:** Optionally, enter the `WHERE` clause of the query. For example, `a.id = 816`.
   e. Run the application page to visualize the graph rendered by the plugin.



> **Note**
>
> The APEX Graph Visualization plug-in on Oracle DataOracle AI Database 26ai does not support graphs that use vertex or edge keys with `DATE` or `TIMESTAMP` data types. Visualizing graph query results on graphs with `DATE` or `TIMESTAMP` keys may result in only a subset of graph data being shown.

6. Optionally, if you wish to implement pagination in the preceding graph visualization, then perform the following steps:

   a. Switch ON the **SQL Query Supports Pagination** setting in the **Attributes** tab of the Property Editor for the graph visualization component in your APEX application.

   b. Set the **Page Size** value in the **Attributes** tab of the Property Editor.

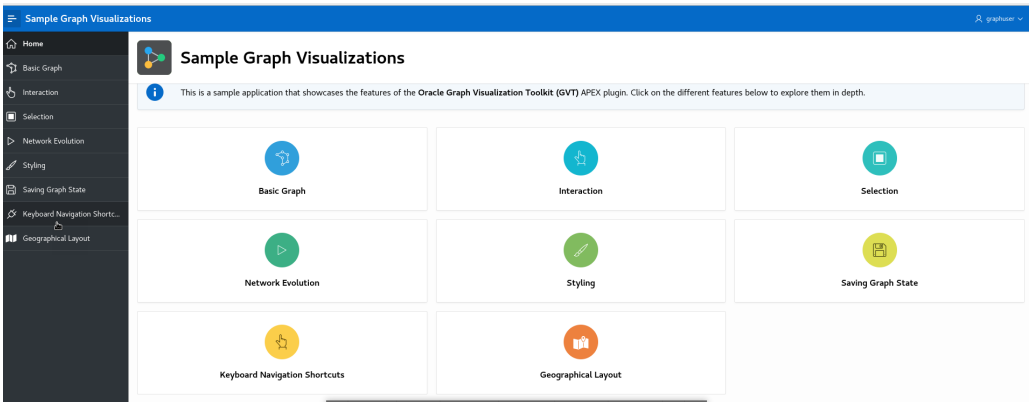
   c. Save and rerun the application page.

      The graph gets rendered with pagination.

7. Optionally, you can import and run the **Sample Graph Visualizations** application from Oracle APEX GitHub repository.

   - Import the `sample-apps/sample-graph-visualizations/sample-graph-visualizations_23ai.sql` into your APEX instance and install the application by following the steps in Importing an Application.

     When installing the sample application, ensure that you have the `CREATE VIEW` privilege for installing the supporting objects.
     You can directly run the sample application once it is installed.



Also, note that the sample application requires a secure `HTTPS` connection. If you want to disable secure connection, then perform the following steps:

> ⚠ **Caution**
>
> It is **not** recommended to disable secure connections in production deployment.

i. Navigate to the sample application home page in **App Builder**.
ii. Click **Shared Components**.
iii. Click **Authentication Schemes** under **Security**.
iv. Click the **Current** authentication scheme.
v. Click the **Session Sharing** tab and turn off the **Secure** switch.
vi. Click **Apply Changes** and then run the application.

Get Started with the APEX Graph Visualization Plug-in (Oracle Database 19c)

Get started with the APEX Graph Visualization plug-in in your APEX application on your Autonomous AI Database instance using Oracle Database 19c.

Before you begin, ensure that you meet the prerequisites described in Prerequisites for Using the APEX Graph Visualization Plug-in.

1. Download the **Graph Visualization (Preview)** plug-in from Oracle APEX GitHub repository.
2. Sign in to your APEX workspace in your Autonomous AI Database instance.
3. Import the downloaded plug-in script (`region_type_plugin_graphviz.sql`) file into your target APEX application by following the steps in Importing Plug-ins in the *Oracle APEX App Builder User's Guide*.
4. Implement the plug-in in an application page to perform graph visualization.

   The following basic example describes the steps to visualize a graph existing in your Autonomous AI Database instance using the Graph Visualization plug-in.

   a. Open the application page in **Page Designer**.
   b. Select the **Rendering** tab on the left pane of the Page Designer.
   c. Right-click an existing component and add a new region component.
   d. Select the new region and configure the following attributes in the **Region** tab of the **Property Editor** on the right pane of the Page Designer:
      i. Enter the Identification **Title**.
      ii. Select **Graph Visualization (Preview)** as Identification **Type**.
      iii. Select the source **Location** as **Local Database**.
      iv. Select **Type** as **SQL Query**.
      v. Run a SQL query, which wraps a PGQL query in the `ORA_PGQL_TO_JSON` PL/SQL function, to retrieve the graph data.

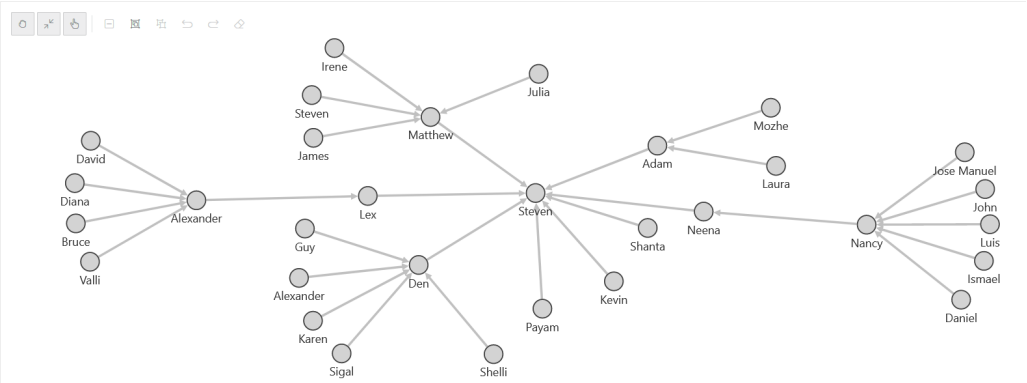
For example:

```
SELECT
  ORA_PGQL_TO_JSON(query => 'SELECT e FROM MATCH
    (e:employees) ON OEHR_EMPLOYESS LIMIT 20')
  FROM DUAL;
```

It is important to note the following:

- The plugin accepts the input graph data containing the vertex and edge information in JSON format only. This is supported by the `ORA_PGQL_TO_JSON` PL/SQL function which takes a PGQL query as input and returns the graph output in JSON structure.
- The graph referenced in the PGQL query must exist in your Autonomous AI Database instance.

e. Run the application page to visualize the graph rendered by the plug-in.



5. Optionally, if you wish to implement pagination in the preceding graph visualization, then perform the following steps:

   a. Switch ON the **SQL Query Supports Pagination** setting in the **Attributes** tab of the Property Editor for the graph visualization component in your APEX application.

   b. Bind the `page_start` and `page_size` parameters when calling the `ORA_PGQL_TO_JSON` function in the SQL query as shown in the following example code:

   ```
   SELECT
     ORA_PGQL_TO_JSON(query => 'SELECT e FROM MATCH
       (e:employees) ON OEHR_EMPLOYESS LIMIT 20',:page_start,:page_size)
     AS result FROM DUAL;
   ```

   c. Set the **Page Size** value in the **Attributes** tab of the Property Editor.

   Note that the **page_start** value is automatcially set.

   d. Save and rerun the application page.

   The graph gets rendered with pagination.

6. Optionally, download the **Sample Graph Visualizations** application from Oracle APEX GitHub repository.

   This application demonstrates the use of the Graph Visualization plug-in.

   a. Import the downloaded `sample-apps/sample-graph-visualizations/sample-graph-visualizations_19adb.sql` into your APEX instance by following the steps in Importing an Application.

b. Run the sample application from the application home page in App Builder.

Configure Attributes for the APEX Graph Visualization Plug-in

Learn how to customize your graph visualization using the Graph Visualization plug-in attributes in your APEX application.

You can configure the attributes for the plug-in component in the Attributes tab (Property Editor) on the right pane of the Page Designer. The attributes are grouped as per their scope in the following panels:

Topics:

Settings

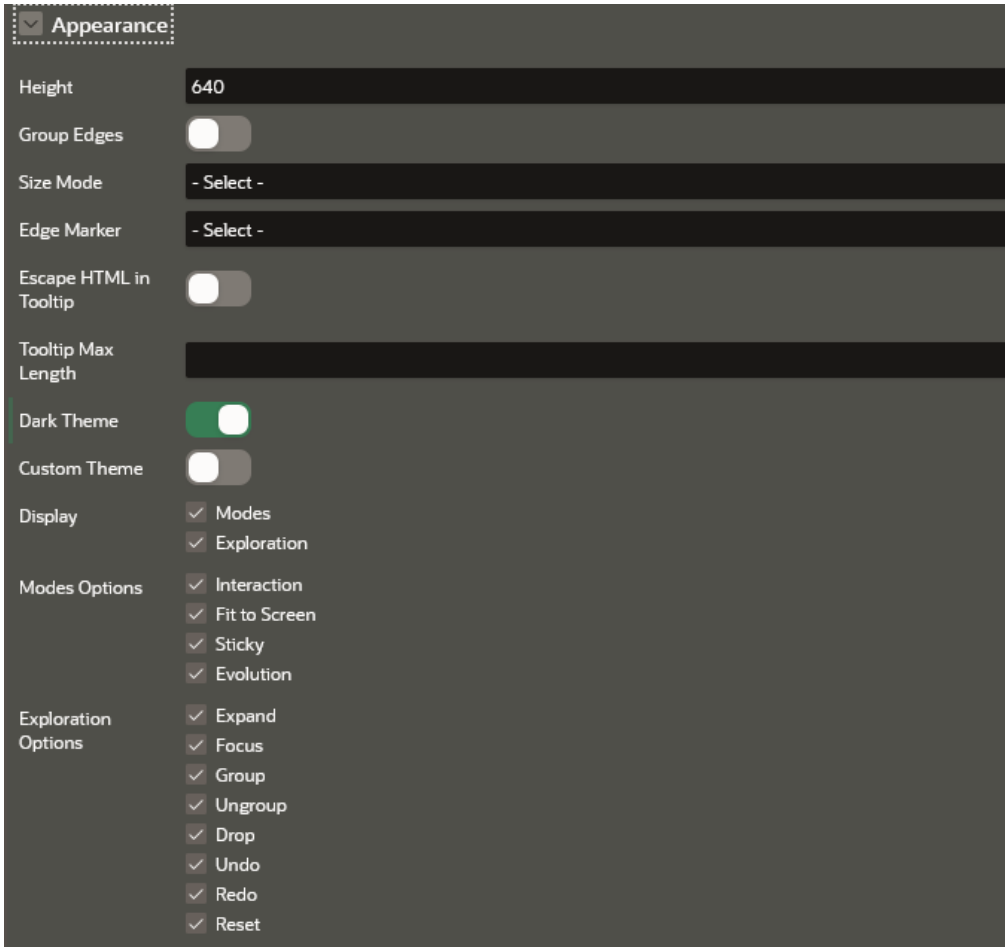
The **Settings** panel appears as shown:

The following table describes the attributes in the **Settings** panel:

Attribute	Description
SQL Query supports Pagination	Switch on this toggle if you are implementing the paginate interface.
Page Size	An integer value that determines the number of vertices and edges to be displayed per page if you enabled **SQL Query supports Pagination**.
Live Search	Switch on this toggle to enable Live Search when visualizing the graph.
Show Legend	Switch on this toggle to display the legend for the graph visualization.
Legend Width	An integer value that controls the legend width if you have enabled **Show Legend**. Default is 150.

Appearance

The **Appearance** panel appears as shown:



The following table describes the attributes in the **Appearance** panel:

Attribute	Description
Height	An integer value (in `px`) to set the size of the graph visualization panel. Default value is `400 px`.
Group Edges	When this option is enabled, multiple edges between the same source and target vertex will be grouped together in the graph. The grouped edges will be shown as a single edge with a number on it, indicating how many edges have been grouped.
Size Mode	Two size modes are supported: • **Normal** (default) • **Compact**
Edge Marker	Supported edge markers are: • **None** • **Arrow** (default)
Escape HTML in Tooltip	Switch on this toggle if you wish to escapes HTML content used on vertex or edge tooltip.
Tooltip Max Length	An integer value that determines the maximum length of characters for the tooltip. Default value is 100.
Dark Theme	Enable this toggle to switch to a dark theme.

Attribute	Description
Custom Theme	Enable this toggle if you wish to configure a custom theme for the following: • **Background Color**: Enter a color code or pick a color for the background. • **Text Color**: Enter a color code or pick a color for the text.
Display	You can enable or disable the **Modes** and **Exploration** options. Supported **Modes Options** are: • **Interaction**: • **Fit to Screen** • **Sticky** • **Evolution** Supported **Exploration Options** are: • **Expand**: To retrieve n-hops neighbors of selected vertices. • **Focus**: To shift the focus of view; it drops everything and fetches n-hops neighbors of the selected vertex. • **Group**: To group selected multiple vertices and collapse them into a single one. • **Ungroup**: To select a group of collapsed vertices and ungroup them. • **Drop**: To remove selected vertices or edges from the visualization. • **Undo**: To undo the last action. • **Redo**: To redo the last action. • **Reset**: To reset the visualization to its default state.

Layout

The **Layout** panel allows you to choose one of the following layout options:

- *Circle*
- *Concentric*
- *Force* (default)
- *Grid*
- *Hierarchical*
- *Radial*
- *Geographical*

The layout configuration parameters may vary for different layouts.

Force Layout

The *Force* layout configuration parameters are described in the following table:

Attribute	Description
Spacing	Spacing determines how close different vertices are rendered next to each other. Default is 1.5.
Alpha Decay	Controls the rate at which the simulation's internal alpha value, which influences node movement, decreases over time, gradually stabilizing the force layout. Default is 0.01.
Velocity Decay	Determines how fast a simulation ends. Default is 0.1.

Attribute	Description
Edge Distance	The simulation tries to set each edge to the specified length. This can affect the padding between vertices. Default is 100.
Vertex Charge	Influences the underlying forces (for example, to remain within the viewport, to push vertices away from each other, and so on). If **Enable Cluster** is true, then it influences the forces among clusters. Default is -60.
Enable Cluster	Switch on this toggle if you wish to enable cluster based layout.
Cluster By	By default, the cluster layout (if enabled) uses the first element in `vertex.labels` to form the cluster. It can also be set to the property name of a vertex, and the clusters will be formed based on the property value.
Hide Unclustered Vertices	Determines whether to display vertices that do not belong to any cluster. Default is false.

Circle, Concentric, and Radial Layouts

The following layouts require only the **Spacing** configuration:

- *`Circle`*: Spacing sets the radius of the circle. Default is 2.
- *`Concentric`*: Spacing sets the minimum spacing in between vertices. It is used for radius adjustment. Default is 2.
- *`Radial`*: Spacing sets separation gap between neighboring vertices if they share the same parent vertex. If set to 0, then spacing will not be applied. Default is 2.

Grid Layout

The *`Grid`* layout supports the following configuration options:

- **Spacing**: Spacing sets the space between elements in the grid. Default is 2.
- **Rows**: Determines the number of rows in the grid.
- **Columns**: Determines the number of columns in the grid.

The default number of rows and columns are dynamically calculated depending on the height and the width of the graph visualization panel.

Hierarchical Layout

The *`Hierarchical`* layout configuration parameters are described in the following table:

Attribute	Description
Rank Direction	Alignment of the ranked vertices. Supported options are - `Up to Left`, `Up to Right`, `Down to Left`, `Down to Right`, `Top to Bottom`, `Bottom to Top`, `Left to Right`, `Right to Left`.
Ranker	Specifies the type of algorithm used to rank the vertices. Supported algorithms are: *Network Simplex*, *Tight Tree*, and *Longest Path*.
Vertex Separation	Sets the horizontal separation between the vertices.
Edge Separation	Sets the horizontal separation between the edges.
Rank Separation	Sets the separation between two ranks(levels) in the graph.

Geographical Layout

The `Geographical` layout configuration parameters are described in the following table:

Attribute	Description
Map Type	Select map type in map visualization or graph visualization settings, or provide your own sources and layers.
Longitude	Specify the vertex property to use for determining the longitude of a vertex.
Latitude	Specify the vertex property to use for determining the latitude of a vertex.
App ID	Specify the *appId* to fetch maps from `http://maps.oracle.com/elocation`. If omitted, a generic *appId* will be used.
Show Information	Enabling this toggle, displays an info box in the visualization that shows the latitude and longitude of the mouse position and the zoom level of the map.
Navigation	Displays the navigation controls towards the top right region of the map.
Markers	Displays location markers on the map

> **See Also**
>
> Layouts page in *Property Graph Visualization Developer's Guide and Reference*

Captions

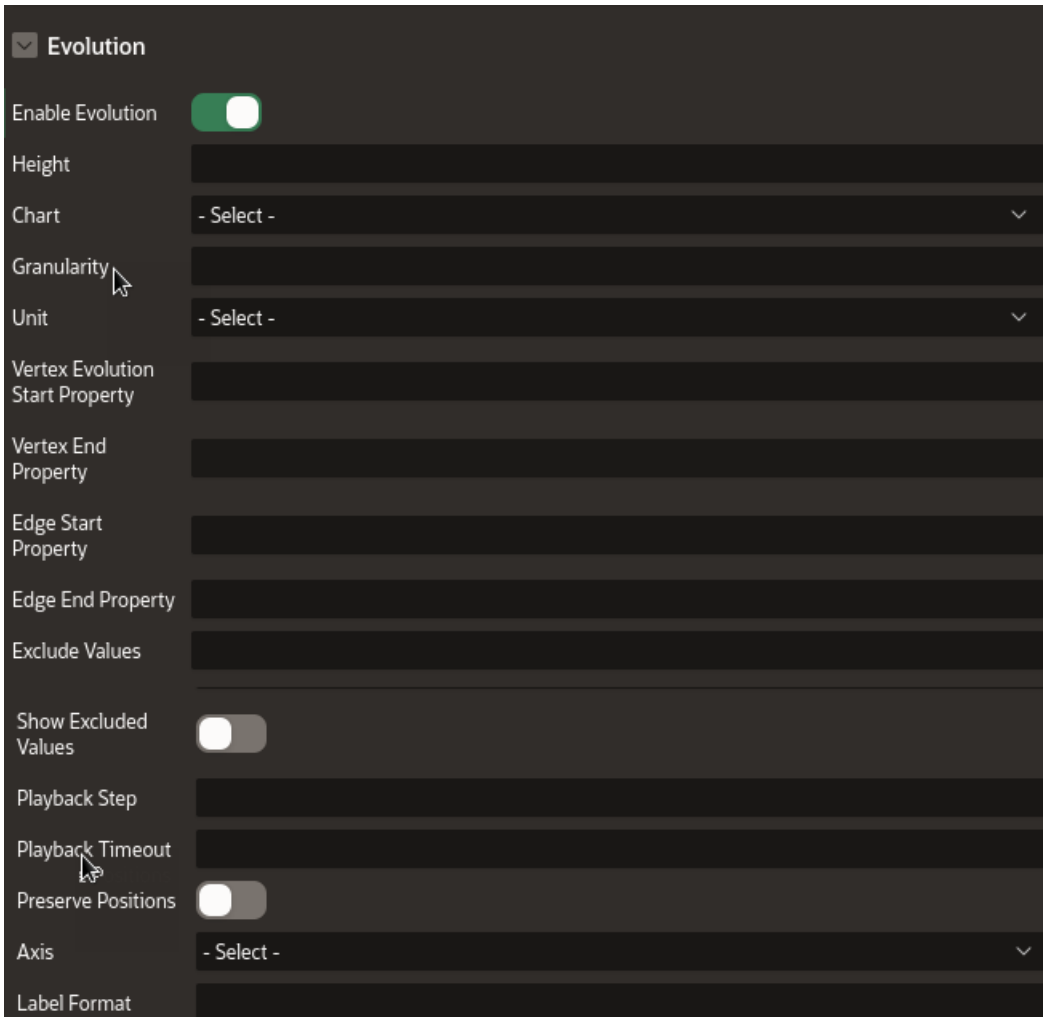
The **Captions** panel appears as shown:

The following table describes the attributes in the **Caption** panel:

Attribute	Description
Vertex Caption	Specify the property to be displayed as the vertex label.
Edge Caption	Specify the property to be displayed as the edge label.
Maximum Caption Length	Specify the maximum length of the caption.
Show full label text on hover	Enable this toggle if you wish to display the vertex and edge caption when hovering over a specific vertex or edge.

Evolution

The **Evolution** panel appears as shown:



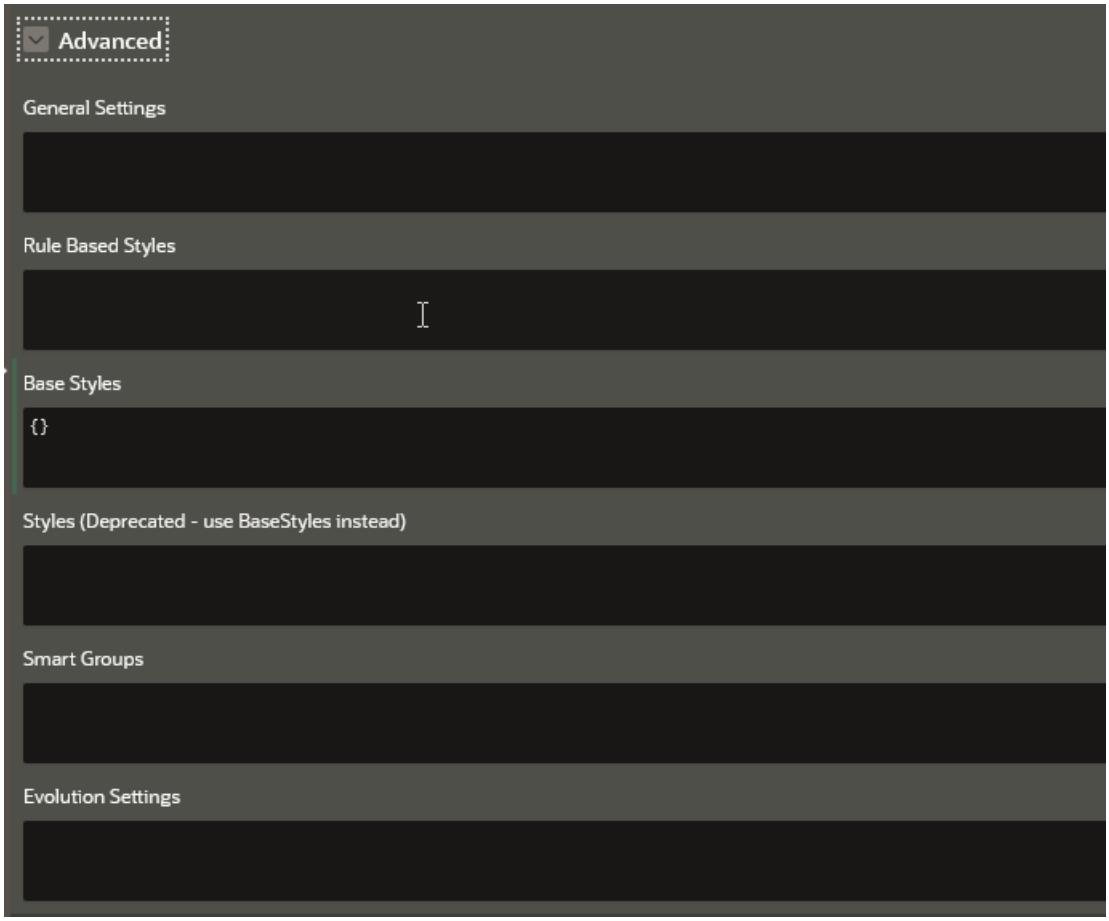
The following table describes the attributes in the **Evolution** panel:

Attribute	Description
Enable Evolution	Switch on this toggle to enable network evolution in the graph visualization.
Height	Specify the height of the chart.
Chart	Select the chart type - *Bar* or *Line*.
Granularity	Specify the aggregation granularity for the input unit.
Unit	Select the unit of time for the increment.
Vertex Evolution Start Property	Select the name of the property to use for the vertex filtering. The time frame for the graph will be after the *Vertex Evolution Start Property*.
Vertex End Property	Select the name of the property to use for the vertex filtering. The time frame for the graph will be before the *Vertex End Property*.

Attribute	Description
Edge Start Property	Select the name of the property to use for the edge filtering. The time frame for the graph will be after the *Edge Start Property*.
Edge End Property	Select the name of the property to use for the edge filtering. The time frame for the graph will be before the *Edge End Property*.
Exclude Values	Specify one or more values to be excluded.
Show Excluded Values	Enable this toggle if you wish to display the excluded values.
Playback Step	Specify a value to determine how often does the playback advance in ms.
Playback Timeout	Specify a value to determine how many steps are taken per time out during playback.
Preserve Positions	If switched on, network evolution will keep the original vertex positions of the graph during playback.
Axis	Select one of the supported values - *vertices*, *edges*, or *both*.
Label Format	Specify a string that represents the format in which the date must be displayed. Note that the format must include either YYYY, MM, or DD. Otherwise, the format will be ignored.

Advanced Options

The **Advanced** panel appears as shown:



The Advanced panel allows you to configure custom and default styling for your graph visualization using the following options:

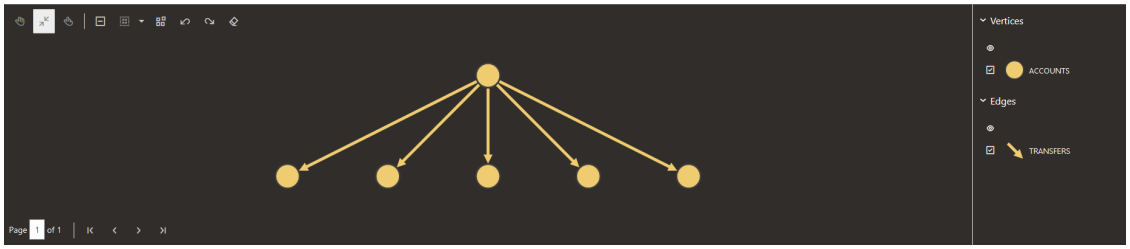
General Settings

You can specify the general graph visualization settings (see settings) in JSON format.

For instance, the following JSON example specifies the theme, legend width, and layout configurations:

```
{
"theme": "dark",
"layout": "hierarchical",
"legendWidth": "20"
}
```

The corresponding graph visualization is as shown:



Rule-Based Styles

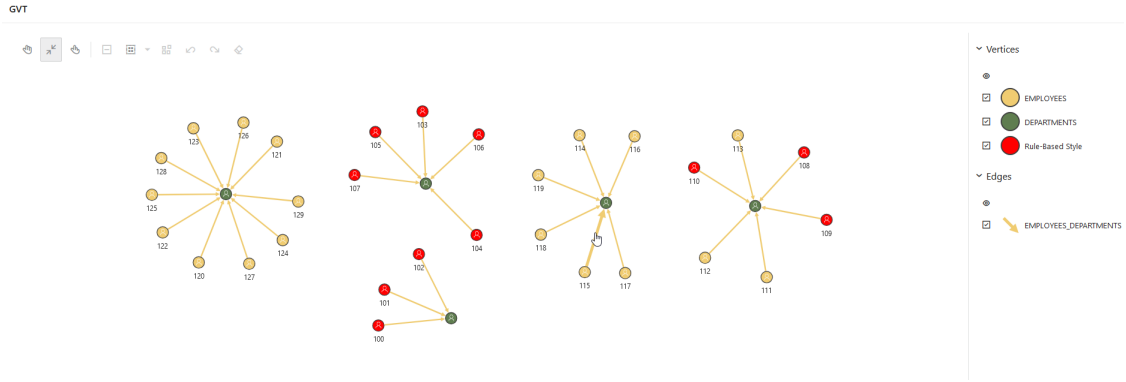
Rule-based style expressions are used to specify the target element into which the given style must be applied. The applied custom style is reflected in the legend panel as well. See Rule Expressions in *Property Graph Visualization Developer's Guide and Reference* for more information.

For instance, the following JSON example creates a custom color style for employee IDs ranging from 100 to 110:

```
[
  {
    "_id": 1,
    "component": "vertex",
    "stylingEnabled": true,
    "target": "vertex",
    "visibilityEnabled": true,
    "conditions": {
      "operator": "and",
      "conditions": [
        {
          "property": "EMPLOYEE_ID",
          "operator": ">=",
          "value": "100"
        },
        {
          "property": "EMPLOYEE_ID",
```

```
          "operator": "<=",
          "value": "110"
        }
      ]
    },
    "legendTitle": "Rule-Based Style",
    "style": {
      "color": "red"
    }
  }
]
```

The corresponding graph visualization is as shown:



For more examples, see Rule-Based Styles in *Property Graph Visualization Developer's Guide and Reference*.

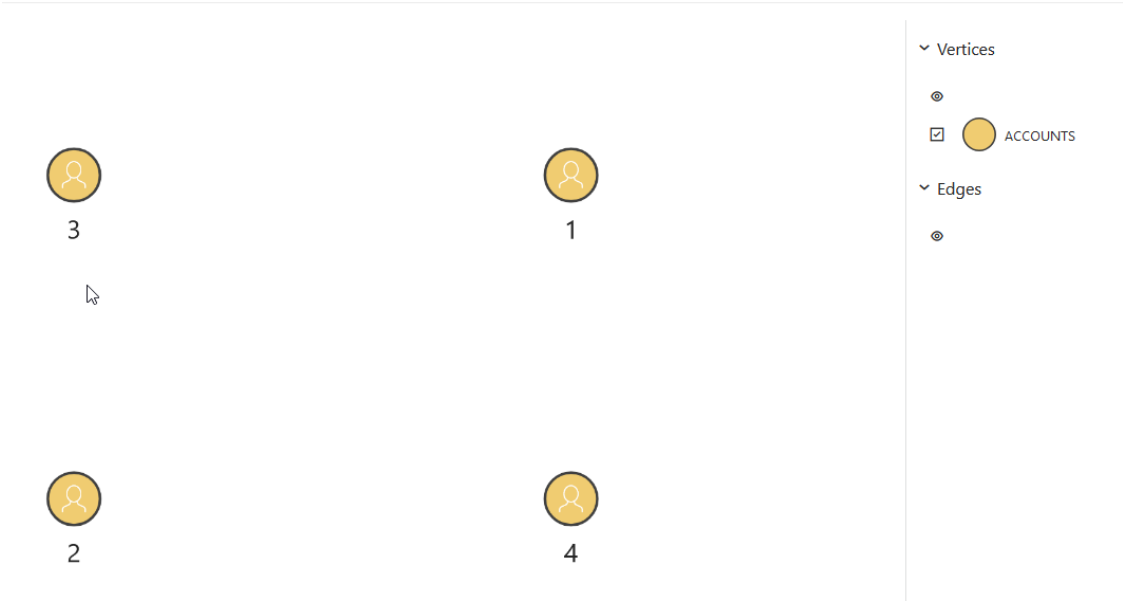
Base Styles

Base style expressions are used to overwrite the default styling for the vertices and edges in the graph.

For instance, the following JSON example overwrites the default vertex styling:

```
{
   "vertex":{
      "size":12,
      "label":"${properties.EMPLOYEE_ID}",
      "icon":"fa-user"
    }
}
```

The corresponding graph visualization is as shown:



For more examples, see Base Styles in *Property Graph Visualization Developer's Guide and Reference*.

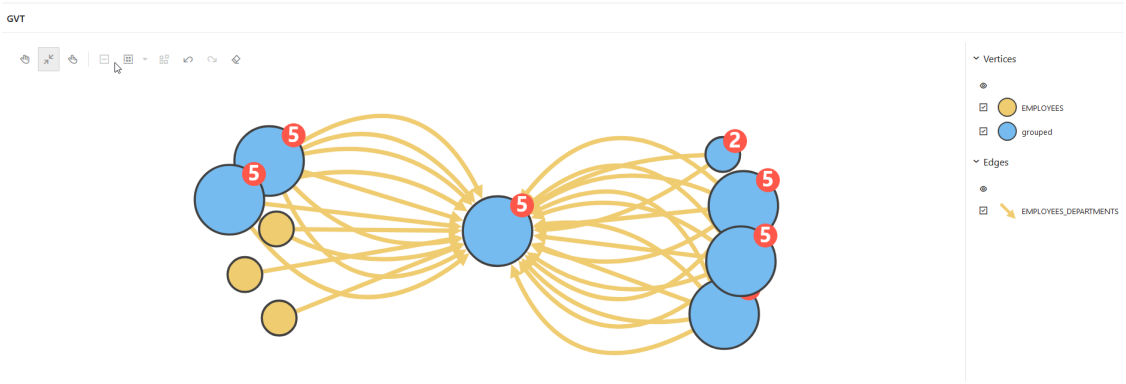
Smart Groups

You can specify the configuration for applying smart grouping in JSON format.

For instance, the following JSON example groups employees by their `JOB_ID`:

```
[
  {
    "_id": 1,
    "name": "Group By Job",
    "type": "group",
    "automatic": true,
    "enabled": true,
    "groupBy": "JOB_ID",
    "conditions": {
      "operator": "or",
      "conditions": [

      ]
    }
  }
]
```

The corresponding graph visualization is as shown:



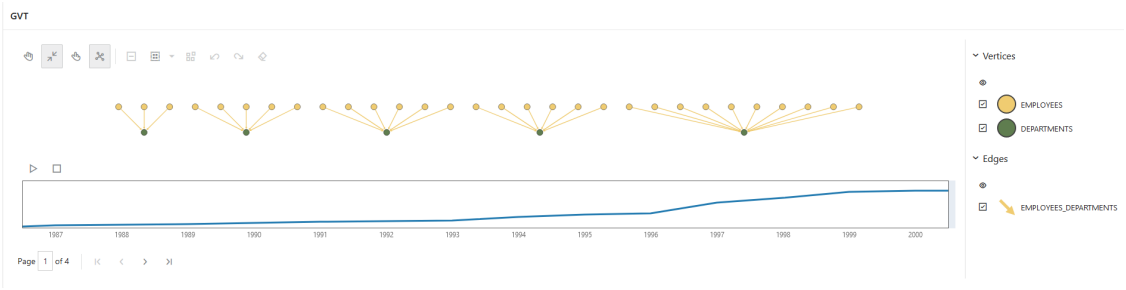
Evolution Settings

You can provide the configuration for network evolution in JSON format.

For example:

```
{
    "vertex": {
      "start": "properties.HIRE_DATE"
    },
    "unit": "year",
    "chart": "line"
}
```

The corresponding graph visualization is as shown:



Callback Options

The **Callbacks** panel appear as shown:

The **Callbacks** panel comprises the following options:

Attribute	Description
Expand	To expand a selected vertex in the graph visualization, see Expand for more information.
FetchActions	To retrieve the graph actions from a data source, refer to fetchActions for more information.
Persist	To persist the graph actions to a data source, refer to persist for more information.
UpdateGraphdata	Callback to handle events when the graph data is updated.

Expand

You can expand a selected vertex in the graph and fetch the adjacent vertices using the **Expand** attribute in the Property Editor of the Page Designer.

1. Switch to the **Processing** tab on the left pane of the Page Designer and navigate to the **After Submit** node.
2. Right-click and select **Create Process** from the context menu.
3. Enter the process **Name**.
4. Specify **Type** as **Execute Code**.
5. Select the source **Location** as **Local Database**.
6. Select the source **Language** as **PL/SQL** and enter the following code in the PL/SQL input box.

```
DECLARE data clob;
id VARCHAR2(100) := apex_application.g_x01;
graph VARCHAR2(100) := "<graph-name>";
hops NUMBER := <hops>;
n NUMBER := hops - 1;
query VARCHAR2(1000) := 'SELECT e1 FROM MATCH ANY (x) ->{,' || n || '} (y)
ON ' || graph || ', MATCH (y) -[e1]-> () ON ' || graph || ' WHERE id(x) =
```

```
''' || id || '''';
BEGIN
SELECT ORA_PGQL_TO_JSON(query) INTO data FROM sys.dual;
htp.p(data);
END;
```

In the preceding code:

- <graph_name>: Name of the graph
- <hops>: Number of hops to be expanded

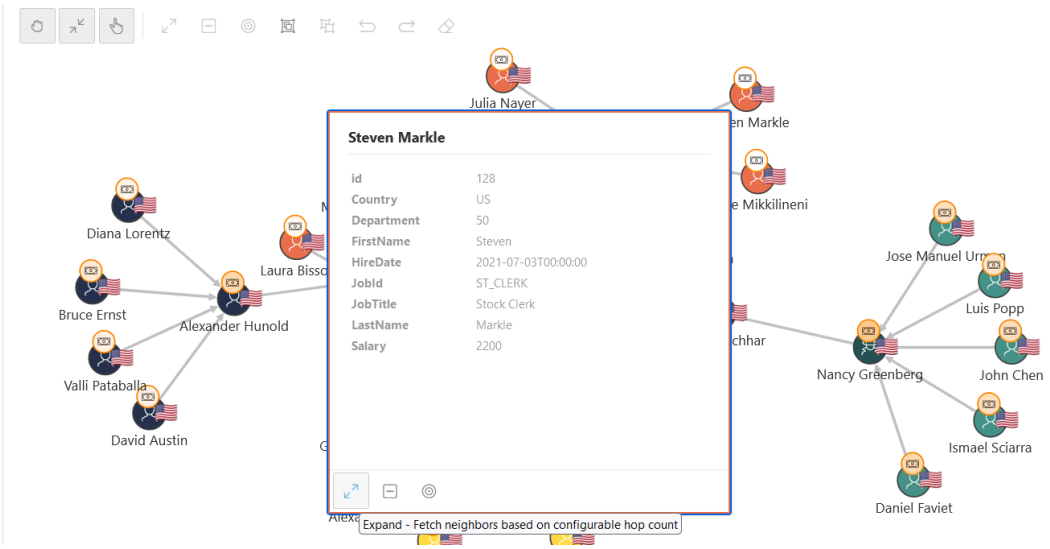
Note that the process takes the vertex `id` to be expanded as input and returns the resulting output as JSON.

7. Select the execution **Point** as **Ajax Callback**.
8. Switch to the **Rendering** tab on the left pane of the Page Designer and select the graph visualization component.
9. Switch to the **Attributes** tab on the right pane and enter the following code in the **Expand** input box in the **Callbacks** panel.

```
const data = await apex.server.process('<process_name>', {
    x01: ids[0]
}, { dataType: 'text' });
try {
    return JSON.parse(data);
} catch (error) {
    return [];
}
```

In the preceding code, <*process_name*> refers to the name of process that was provided at step-3.

10. Click **Save**.
11. Run the application page and you can now click expand on a specific vertex in the graph as shown:



11

Work with Jobs in Graph Studio

A **job** in Graph Studio is a potentially long-running asychronous operation composed of a set of **tasks**.

Topics

About Jobs

A ***job*** is identified by an id, a name, a status, and a set of ***tasks***. Additionally, a *job* includes information about the operation's input, progress logs, progress output, and result (if succeeded).

Types of jobs in Graph Studio may include:

- Creating an RDF graph.
- Loading a graph into memory to perform analytics.
- Starting, stopping and restarting the internal compute environment.

A *job* starts when an operation (for example: create graph, load into memory) is executed. During a job execution, the job status is set to `RUNNING` and the progress logs and outputs are updated to keep track of the executed tasks and processed entities. At the end of the job execution, the job's status depends on the success or failure of its tasks:

- A failed job has a `FAILED` or `TIMEOUT` status and produces no result.
- A successful job has a `SUCCEEDED` status and produces a result.

A job can be canceled at any time during the job execution. When a job is canceled, its job status is set to `CANCELED` and no result is produced.

Inspect a Job

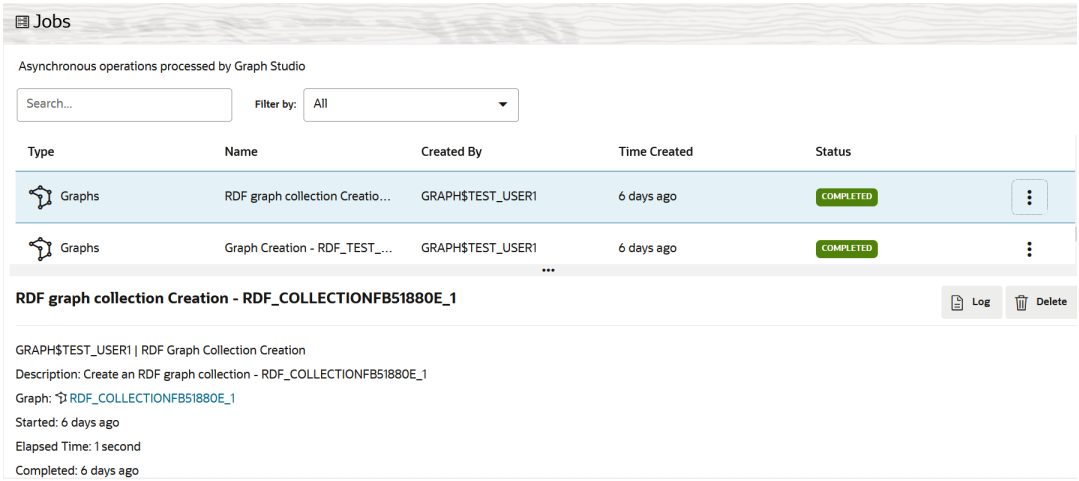
You can inspect (that is, preview) a job.

The Jobs page provides features to review the current status, progress logs and outputs of the existing jobs in Graph Studio.
To inspect the details of a job:

1. Click **Jobs** on the left navigation menu and navigate to the Jobs page.
2. Select the desired **job** on the Jobs page.

   The details section of the Jobs page shows information about the job.

   For example, a Graph Creation job will show the name of the job along with the graph name, the creator, and the associated data tables. Note that a running job displays the start date and elapsed time of the job execution. When a job execution finishes due to a success, failure, or cancellation, the details section will include the ending time of the job execution.



   In addition, you can inspect the progress output, which includes the list of processed, queued, and failed entities or tasks. For a running job, the status of the job includes the progress percentage.

   The Jobs page refreshes automatically without the need to manually refresh the page via the browser. In addition, older, successful jobs get deleted automatically from the list. Failed jobs stay for further inspection until explicitly deleted.

Review a Job Log

A job is also described by a log file.

The log file list out the tasks that have been started, executed, or finished that are part of the job itself. If a task has failed, the log will display the reason behind the task failure. If a job has been canceled, the log will display the last executed task, as well as details for the canceled tasks.
To review a Job log:

1. Select a **job** on the Jobs page.

   The job details are displayed in the job details section.
2. Click **See Logs** in the job details section.

   The log details are displayed.

Cancel a Job

You can cancel a running job.

This action cannot be undone. After the job is canceled, all changes done to the entities affected by the job execution will be rolled back.

To cancel a job during its execution:

1. Select a **job** in progress on the Jobs page.

   The job details are displayed in the job details section.

2. Click **Cancel Job** in the job details section.

   The job is cancelled.

Retry a Job

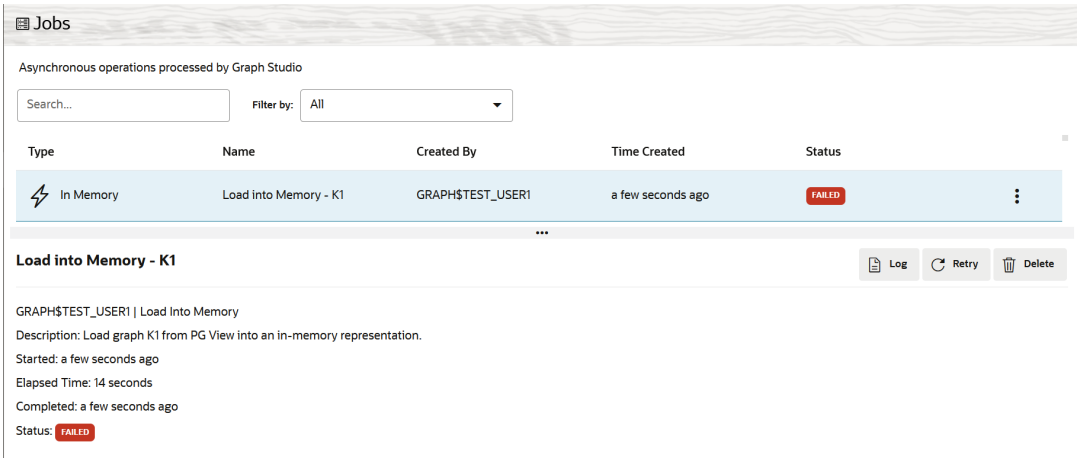
You can retry the execution of a failed or canceled job.

When a job execution is retried, the job operation will be executed using the stored input. To retry a job:

1. Select a failed or canceled job to retry on the **Jobs** page.

   > **Note**
   >
   > **Retry** option is not supported for job requests related to managing the compute environment.

2. Click **Retry** in the details section of the selected job.



   Provide additional information if requested.

   Retrying a job removes the information about the previous job execution. Thus, after the retry operation completes, the job status, progress, and logs will reflect the execution of the retried job.

Delete a Job

You can delete a job that has successfully finished, has failed, or has been canceled.

To delete a job:

1. Select a **job** for deleting on the Jobs page.
2. Click **Delete Job** in the details section of the selected job.

Deleting a job removes the information about the job execution including the input, progress log, and output. However, the job result and any changes made by the job are not affected by this operation.

Retention of Finished Jobs

To optimize disk space usage, completed successful jobs are kept for 10 days after their creation if they have not already been removed.

What to do When a Job Fails

A job's execution can fail for any of several reasons, including incorrect input, storage or memory quota issues, timeouts or database connection problems.

If you want to re-execute a failed job, review the log and look for potential causes of the failure. For example, if the operation to load a graph into memory failed due to storage quota exceeded, increase the storage size of your Autonomous AI Database and try again.

If retrying the job keeps failing unexpectedly, please submit a support request. See Submit a Service Request in the Appendix for more details on how to create and submit a service request.

12

Manage the Compute Environment

Graph Studio must be attached to an internal compute environment in order to perform all graph analysis tasks.

Topics

About the Compute Environment

The internal compute environment in Graph Studio allows you to run notebooks and accelerates analysis by running algorithms and queries parallelized in memory.

Graph Studio can attach to or detach from the internal compute environment automatically. This ensures efficient use of computing resources, thereby saving cost.

The attachment happens at the background when you load property graphs into memory and also implicitly when working with notebooks in Graph Studio. See About Implicit Environment Creation Through Notebooks for more information.

When not in use for a certain period of time, Graph Studio detaches itself from the compute environment. On detachment, any in-memory data stored in the environment is deleted.

> **Note**
>
> The data deletion during the detachment process is only limited to in-memory copies of property graph data and transient analysis results like in-memory algorithms or query results. Graphs and notebooks (including input and generated output of paragraphs) remain persisted in your Autonomous AI Database and are available even in detached state.

Graph Studio automatically reconnects to the compute environment when you reload the property graph into memory or rerun your notebook from the top.

The status of the **Compute Environment** is indicated on the top right of the header.

The compute environment also allows you to configure your preferred memory settings for the graph server and the notebook interpreters. You can also choose to save the values as the default memory settings to be used for creating the Graph Studio environment.

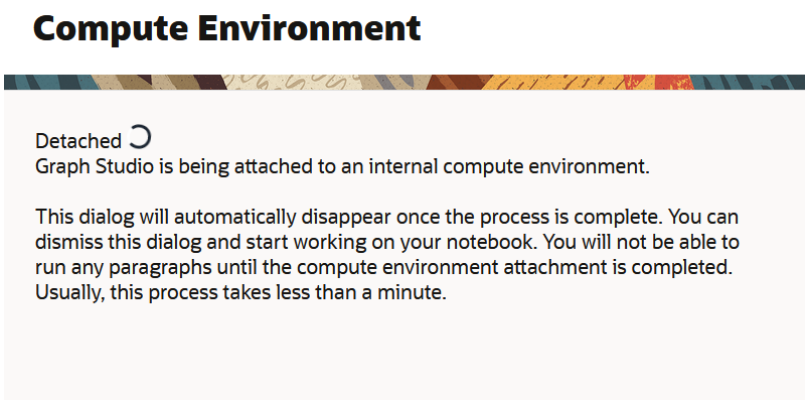
About Implicit Environment Creation Through Notebooks

The internal compute environment, required to run paragraphs in notebooks, is implicitly created when you create a new or open an existing notebook in Graph Studio.

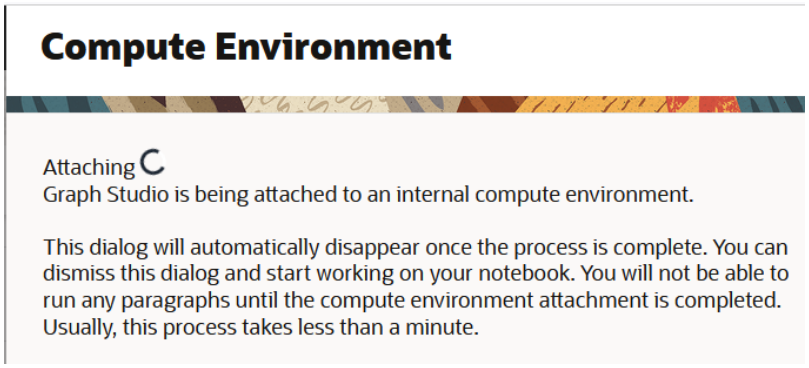
Graph Studio displays a message dialog indicating the environment status and the progress of the environment creation when a notebook is opened. Once the environment is attached, the message dialog automatically disappears.

Optionally, you can choose to **Dismiss** the message and continue to work on your notebook. However, you cannot run the notebook paragraphs until the environment attachment is complete.

For example, if you open a notebook when the Graph Studio environment is detached, then the **Compute Environment** slider displays the detacheded environment status until the environment creation job is started at the background:



Then attaching status is displayed until the environment gets attached successfully:



In case the environment creation job fails at the background, then an appropriate error message is displayed. You can then navigate to the Jobs page to view the error details.

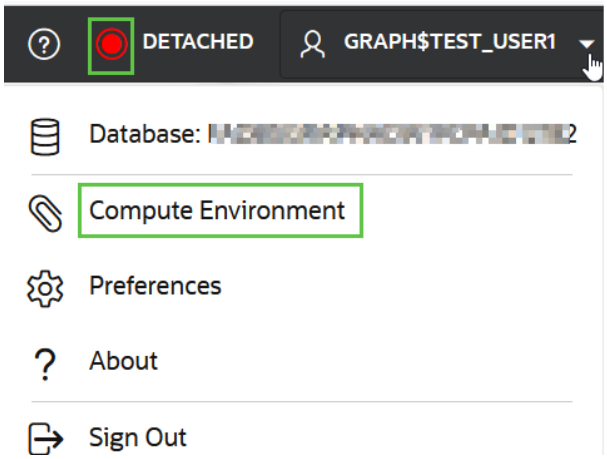
Inspect the Compute Environment

You can inspect the state of your internal compute environment to see if it is attached to Graph Studio.

Additionally, you can also view the memory configuration for the graph server and the supported notebook interpreters.

1. Click on your **username** on the top right drop-down menu of your interface.

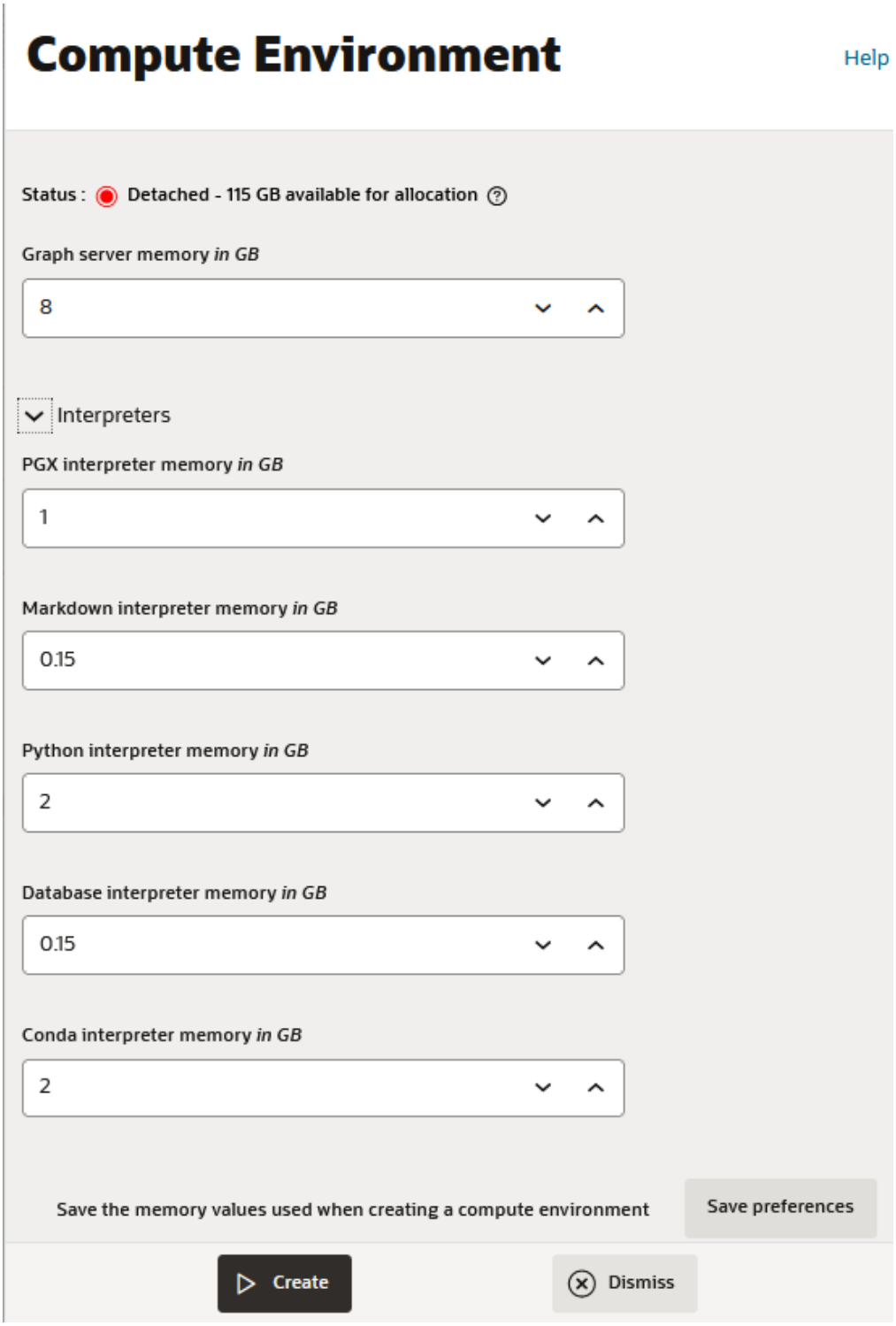
   The drop-down menu appears as shown:



2. Select **Compute Environment** from the drop-down menu.

> **Tip**
>
> You can click the compute environment status indicator in the header to directly open the **Compute Environment** slider.

The **Compute Environment** slider opens as shown:



You can view the following environment details:

- **Status** of your compute environment.
  The compute environment can be available in one of the following states:

State	Description
Attached	Graph Studio is currently attached to a compute environment.

State	Description
Attaching	Graph Studio is currently in the process of attaching to a compute environment.
Detached	Graph Studio is currently not attached to any compute environment
Detaching	Graph Studio is currently in the process of detaching from a compute environment

If the compute environment status is **Attached**, then you can also view the total amount of memory allocated to the environment.

- **Graph server memory** configuration.
- Click **Interpreters** to view the memory configuration for the interpreters.

Manually Manage the Compute Environment

Although Graph Studio can automatically manage the attaching and detaching process of the compute environment in the background, you can also manually manage the environment.

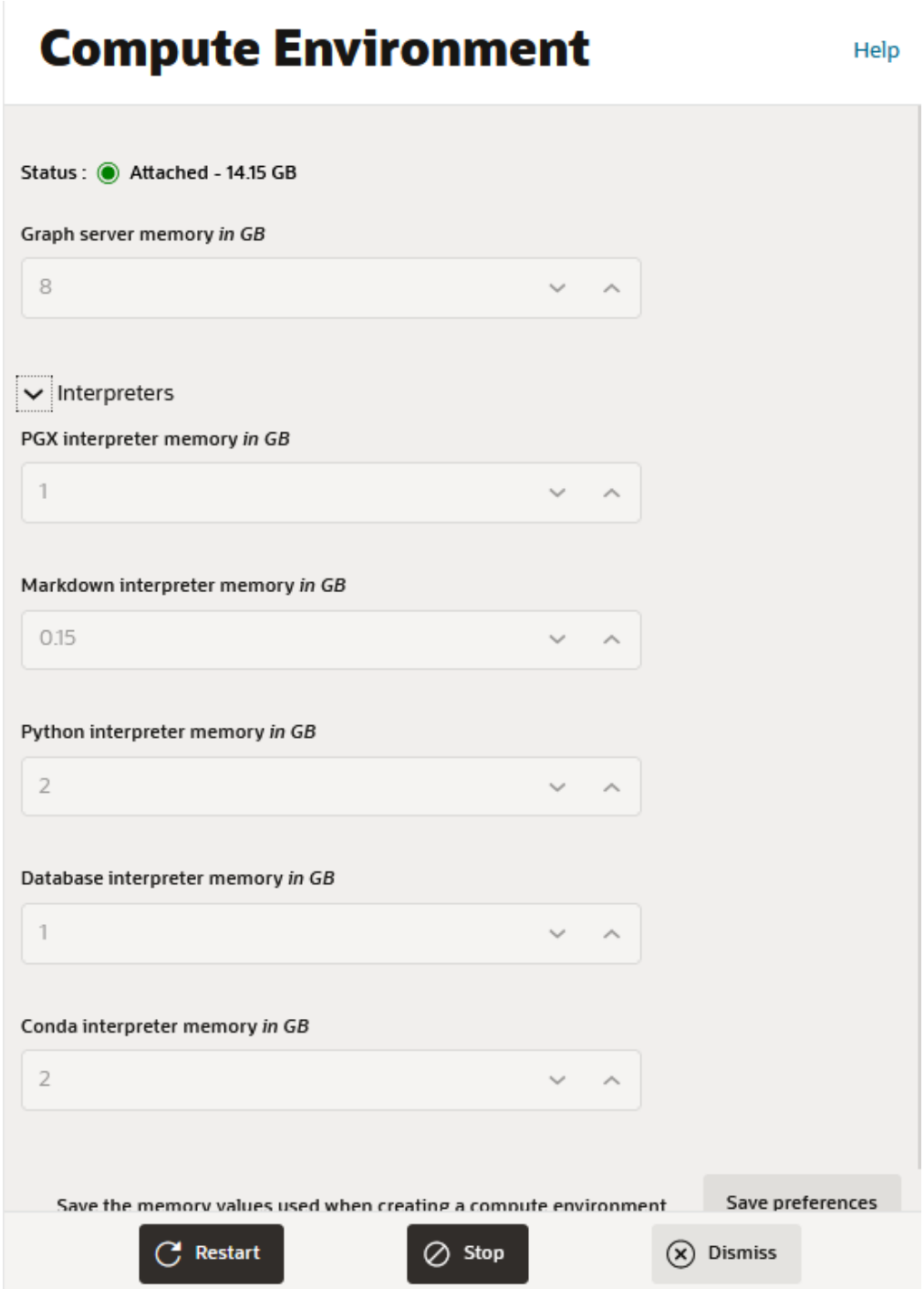
The following lists a few situations which require manual intervention:

- Increase (or decrease) the maximum amount of the graph server (PGX) memory available for analysis and optionally save the memory value as the default graph server (PGX) memory configuration.
- Increase (or decrease) the maximum amount of memory available for notebook interpreters and optionally save the memory values as the default memory configurations for the interpreters.
  For instance, if the result of your PGQL (RDBMS) query contains thousands of long strings, you may have to increase the memory of the **Database interpreter** to avoid out of memory errors.
- Code in a notebook accidentally caused the environment to enter a bad state.
- The environment ran out of memory.

To manually manage the environment:

1. Click on your **username** on the top right drop-down menu of Graph Studio and then select **Compute Environment**.

   The **Compute Environment** slider appears as shown:



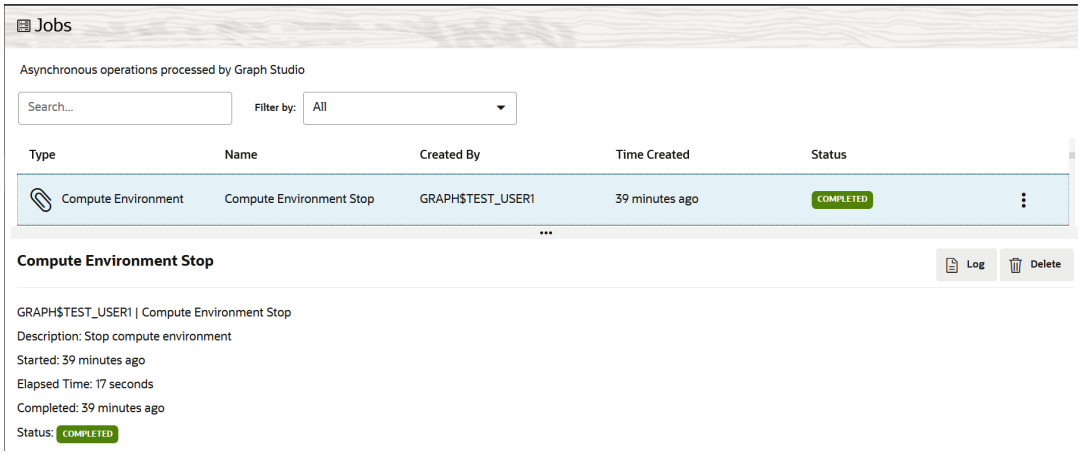
2. Click **Restart** or **Attach** or **Detach** as it may apply.

   The following table describes all the supported manual options to manage the compute environment:

Manual Options	Description
Detach the compute environment	If the compute environment is currently attached, you can detach it by clicking the **Stop** button. This will cause the compute environment to enter the **Detaching** state.

Manual Options	Description
Attach the compute environment	If the compute environment is currently detached, you can: **a.** Select the amount of **Graph server memory** you want to attach to the compute environment. It is important to note that currently Graph Studio does not allow allocation of more than 109 GB of memory for graph analysis per tenancy. In case you get one of the following error messages, although you selected less than 109 GB: • `Not enough memory available` • `The number is too high` then the cause could be one of the following: • Other Autonomous AI Databases in your tenancy currently being attached to the compute environment. • The sum of the memory given to the graph server and all the interpreters has exceeded the maximum memory allocation limit. If you require more memory, please contact Oracle Cloud Support. **b.** Click **Interpreters** to configure memory for the interpreters: • **PGX interpreter memory** Note that this memory configuration applies for the following interpreters as they all share the configured memory space: – Java (PGX) interpreter – PGQL (PGX) interpreter – Custom Algorithm (PGX) interpreter • **Markdown interpreter memory** • **Python interpreter memory** • **Database interpreter memory** Note that this memory configuration applies for the following interpreters as they all share the configured memory space: – SQL interpreter – PGQL (RDBMS) interpreter – SPARQL (RDF) interpreter • **Conda interpreter memory** **c.** Optionally, click **Save preferences** to save the values as the default memory settings. **d.** Click **Create** to attach to the compute environment. **Note** The total amount of memory allocated to the compute environment is the sum of the memory given to the graph server and all the interpreters.
Restart the compute environment	You can detach and attach again in a single operation by clicking the **Restart** button. In this case, Graph Studio will attach to a compute environment with the same amount of memory as the current configuration for the graph server and the interpreters.

3. Monitor the progress of any of the manual operations on the Jobs page.



A

Autonomous AI Database Graph PGX API Limitations

The following features and APIs of the graph server available in our ***on-premises*** offering (Oracle Graph Server and Client) are **not** available in the managed cloud service when being invoked from within `%java-pgx` or `%python-pgx` paragraphs.

Using any of these APIs will result either in errors being returned upon invocation or in not achieving the desired behavior.

See the following for reference information about these on-premises APIs:

- *Oracle Graph Java API Reference for Property Graph*: See Javadoc for more information.
- *Oracle Graph Python API Reference for Property Graph*: See Python API Reference for more information.

Manage the server state

All APIs that PGX offers to manage the server state are not available. This includes most of the methods available on the `ServerInstance` object. The following example lists a few administrative APIs that are not supported:

- Java API
- Python API

Java API

- `ServerInstance#getServerState()`
- `ServerInstance#killSession()`
- `ServerInstance#shutdownEngine()`

Python API

- `ServerInstance.get_server_state()`
- `ServerInstance.kill_session()`
- `ServerInstance.shutdown_engine()`

Instead, use the capabilities available in Graph Studio to manage the execution environment.

Read graphs

APIs to read a graph *directly* from files or any other input sources are not available. For example:

- Java API
- Python API

Java API

- `PgxSession#readGraphWithProperties` and similar methods
- `PgxSession#readGraphFiles` and similar methods

Python API

- `PgxSession.read_graph_with_properties()` and similar methods
- `PgxSession.read_graph_files()` and similar methods

Instead, import the data you want to analyze as a graph into the Autonomous AI Database using any of the available data import capabilities such as DBMS_CLOUD, SQL Developer Web, or Oracle Data Integrator. After the data is in the Autonomous AI Database, use Graph Studio to convert the data into a graph or import it as a graph. Only graphs managed and loaded into memory by Graph Studio can be accessed using PGX APIs.

Grant In-memory Graph Permissions to Other Users

APIs to grant permissions on in-memory graphs to other users are not available For example:

- Java API
- Python API

Java API

- `PgxGraph#grantPermission()` and similar methods

Python API

- `PgxGraph.grant_permission()` and similar methods

Instead, you can share graphs with other users through corresponding GRANT statements in the Autonomous AI Database. You can also conveniently share graphs with other users using the Share capability available in Graph Studio.

Export graphs

APIs to write in-memory graphs to the local file system are not available:

- Java API

- Python API

Java API

```
PgxGraph#store()
```

Python API

```
PgxGraph.store()
```

User defined functions (UDFs)

The ability to define and invoke UDFs is not available.

Changing the execution environment

Modifying the execution environment of the current session as shown in the following example is not supported.

- Java API
- Python API

Java API

```
PgxSession#getExecutionEnvironment()
```

Python API

```
PgxSession.get_execution_environment()
```

B

Submit a Service Request

You can raise a service request with **My Oracle Support**, if you need help to resolve issues when working with Graph Studio in Oracle Autonomous AI Database.

My Oracle Support is a customer portal that offers product services through various support tools and contains a repository of useful information, where you can find solution to your issue. You can raise a service request using this application through one of the following two interfaces:

1. **My Oracle Support**
2. **Cloud Support**

You must meet the following prerequisites to create a service request:

- You must have a Support Identifier which verifies your eligibility for Support services.
- You must have an account at **My Oracle Support**

1. Access **My Oracle Support** at https://support.oracle.com.

   You can choose to create a service request either from **My Oracle Support** interface or from **Cloud Support** interface by using the switch toggle button on the top-right of the window.
2. Perform the following steps to create a service request from **My Oracle Support** interface:
   a. Click **Create Technical SR** on the Service Requests tab.
   b. Enter the **Problem Summary**.
   c. Enter the **Problem Description**.

      > **Note**
      >
      > It is important to provide your **Region**, **Tenancy OCID** and **Database Name** along with your problem details. See Obtain Tenancy Details on how to obtain the tenancy details for your instance.

   d. Enter the **Error Codes**.
   e. Select the **Cloud tab** under "Where is the Problem".
   f. Specify **Autonomous Database on Shared Infrastructure** in the Service Type field.
   g. Select a **Problem Type** and provide the **Support Identifier** details.
   h. Click **Next** until you have provided all the mandatory information.
   i. Click **Submit**.

      Your service request is created.
3. Perform the following steps to create a service request from **Cloud Support** interface:
   a. Click **Create Technical SR** on the Service Requests tab.
   b. Follow through sub-steps **2.f** to **2.i** in the preceding step.

Your service request is created.

C

Known Issues for Graph Studio

You can learn about the issues you may encounter when using Graph Studio and how to work around them.

Syntax error not thrown for a missing closing parenthesis ")" in a Java paragraph in Notebook

Syntax error must be thrown when executing a `%java-pgx` paragraph containing an incomplete Java statement due to a missing closing parenthesis. However, the Java interpreter in notebook, returns a `Successful execution: No result returned` message, which is incorrect. For example:

```
%java-pgx
out.println("This line is problematic";
<small><i>Successful execution: No result returned.</i></small>
```

This is because internally the paragraphs are interpreted through JShell which considers the incomplete command statement to be of multiple lines. Until a command termination using the closing parenthesis is executed, any other execution of `%java-pgx` in the subsequent paragraphs inside the notebook are considered as continuation of the incomplete statement and therefore will produce incorrect results. For example, executing the following paragraph after running the preceding code does not retrieve the graph configurations as expected:

```
%java-pgx
PgxGraph g = session.getGraph("BANK_GRAPH")
<small><i>Successful execution: No result returned.</i></small>
```

Workaround

To work around this problem, you can use one of the following options:

- Restore the notebook environment to the normal state by performing the following steps:
  1. Execute a closing parenthesis statement in a new `%java-pgx` paragraph to mark the termination of the incomplete statement as shown:

     ```
     %java-pgx
     )
     Error:
     ')' expected
     out.println("This line is problematic";
                                            ^
     ```

     Running the code displays the error message.

2. Fix the incorrect statement to include the closing parenthesis and re-execute the statement.

```
%java-pgx
out.println("This line is problematic");
This line is problematic
```

- Restart the environment. See Manually Manage the Compute Environment for more information to restart the environment.

After implementing one of the workaround options, any execution of `%java-pgx` paragraphs in the notebook will produce the desired results. For example:

```
%java-pgx
PgxGraph g = session.getGraph("BANK_GRAPH")

PgxGraph[name=BANK_GRAPH,N=1000,E=5001,created=1628583419041]
```

D

Move PG Objects to PGQL or SQL Property Graph

PG Objects graph type is desupported in Graph Studio. Therefore, you must move to PGQL or SQL property graphs.

Perform the following steps:

1. Navigate to the Notebooks page and open a notebook.
2. Drop the **PG Objects** graph by calling the `OPG_APIS.DROP_PG` method using the SQL interpreter in a notebook paragraph.

   The following example drops the **PG Objects** graph named `pg_graph`.

   ```
   %sql
   begin
     OPG_APIS.DROP_PG('pg_graph');
   end;
   ```

3. Create a **PGQL Property Graph** or **SQL Property Graph**.

   See Create a Property Graph from Existing Relational Tables for more information.