Oracle® Database

Oracle Globally Distributed Database Guide

23c
F46983-01
August 2023

Oracle Database Oracle Globally Distributed Database Guide, 23c

F46983-01

Copyright © 2018, 2023, Oracle and/or its affiliates.

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

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

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

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

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

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

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

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

Contents

Changes in Oracle Database 23c

New Features xvii
Deprecated Features xvii
Desupported Features xvii

1 Oracle Globally Distributed Database Overview

What is a Globally Distributed Database 1-1
About Oracle Globally Distributed Database 1-1
Oracle Globally Distributed Database as Distributed Partitioning 1-2
Benefits of Oracle Globally Distributed Database 1-3
Example Applications using Oracle Globally Distributed Database 1-4
Flexible Deployment Models 1-5
High Availability in Oracle Globally Distributed Database 1-5
Data Distribution Methods 1-6
Client Request Routing 1-6
Query Processing 1-7
High Speed Data Ingest 1-7
Deployment Automation 1-7
Data Migration 1-7
Lifecycle Management 1-8
Federated Sharding 1-8
What's New in Release 23c 1-9
Where To Go From Here 1-13

2 Oracle Globally Distributed Database Architecture and Concepts

Components of the Oracle Globally Distributed Database Architecture 2-1
- Sharded Database and Shards 2-2
- Shard Catalog 2-3
- Shard Director 2-3
- Global Service 2-4
- Management Interfaces for Oracle Globally Distributed Database 2-4

Oracle Globally Distributed Database Schema Objects 2-4
Partitions, Tablespaces, and Chunks 2-5
Tablespace Sets 2-6
Sharded Tables 2-7
Sharded Table Family 2-9
How a Table Family Is Sharded 2-9
Duplicated Tables 2-10
Non-Table Objects Created on All Shards 2-12
Sharded Data Distribution Methods 2-12
System-Managed Sharding 2-13
User-Defined Sharding 2-15
Directory-Based Sharding 2-17
Directory-based Sharding Use Cases 2-18
Directory-Based Sharding Concepts and Architecture 2-18
Creating Sharded Tables in Directory-based Sharding 2-19
Composite Sharding 2-20
Using Subpartitions with Sharding 2-22
Replication in Oracle Globally Distributed Database 2-25
Replication in Oracle Globally Distributed Database 2-25
Using Oracle Data Guard with a Sharded Database 2-25
Using Raft Replication in Oracle Globally Distributed Database 2-30
Configuring and Managing Raft Replication 2-36
Client Application Request Routing 2-37
Query Processing and the Query Coordinator 2-38

3 Security in an Oracle Globally Distributed Database Environment

Using TCPS Protocol and Transport Layer Security 3-1
Using Wallets with Oracle Globally Distributed Database 3-1
Using Application Contexts During Cross-Shard Operations 3-3
Behavior Differences 3-4
Using Transparent Data Encryption 3-4
Creating a Single Encryption Key on All Shards 3-5

4 Oracle Globally Distributed Database Deployment

Introduction to Sharded Database Deployment 4-1
Planning Your Sharded Database Deployment 4-2
Plan the Sharded Database Configuration 4-2
Provision and Configure Hosts and Operating Systems 4-3
Install the Oracle Database Software 4-5

Install the Shard Director Software 4-5
Create the Shard Catalog Database 4-5
Create the Shard Databases 4-9
Validate the Shard Database 4-15
Configure the Oracle Globally Distributed Database Topology 4-16
Create the Shard Catalog 4-17
Add and Start Shard Directors 4-19
Add Shardspaces If Needed 4-19
Add Shardgroups If Needed 4-20
Verify the Sharding Topology 4-20
Add the Shard CDBs 4-21
Add the Shard PDBs 4-22
Add Host Metadata 4-23
Check Free DB_FILES 4-24
Deploy the Configuration 4-24
Create and Start Global Database Services 4-26
Verify Shard Status 4-27
Example Oracle Globally Distributed Database Deployment 4-27
Example Oracle Globally Distributed Database Topology 4-28
Deploy the Example Oracle Globally Distributed Database 4-31

5 Using Oracle Globally Distributed Database in Oracle Cloud Infrastructure

Deploy an Oracle Globally Distributed Database on Kubernetes 5-1
Deploy an Oracle Globally Distributed Database with Terraform 5-1
Deploy an Oracle Globally Distributed Database with Docker 5-1

6 Oracle Globally Distributed Database Schema Design

Schema Design Considerations 6-1
Sharding Keys 6-2
Choosing Sharding Keys 6-2
Primary Key and Foreign Key Constraints 6-5
Enabling Automatic Data Movement on Sharding Key Update 6-5
Creating Schema Objects 6-7
Create an All-Shards User 6-7
Creating a Sharded Table Family 6-7
Designing Schemas With Multiple Table Families 6-10
Creating Sharded Tables 6-12
Tablespace Set Sizing 6-12

Sharded Tables for System-Managed Sharding 6-13
Sharded Tables for User-Defined Sharding 6-14
Sharded Tables for Composite Sharding 6-15
Sharded Tables for Directory-Based Sharding 6-15
Creating Duplicated Tables 6-16
Updating Duplicated Tables and Synchronizing Their Contents 6-17
Setting the Duplicated Table Global Refresh Rate 6-18
Customizing Duplicated Table Refresh Rates 6-18
Refreshing Duplicated Tables On Demand 6-19
Duplicated Table Support and Limitations 6-20
Creating Indexes on Sharded Tables 6-21
Modifying an Oracle Globally Distributed Database Schema 6-22
DDL Processing in an Oracle Globally Distributed Database 6-23
Creating Objects Locally and Globally 6-24
Monitor DDL Processing and Verify Object Creation 6-24
DDL Syntax Extensions for Oracle Globally Distributed Database 6-28
CREATE TABLESPACE SET 6-28
ALTER TABLESPACE SET 6-29
DROP TABLESPACE SET and PURGE TABLESPACE SET 6-29
CREATE TABLE 6-29
ALTER TABLE 6-32
ALTER SESSION 6-32
Running PL/SQL Procedures in an Oracle Globally Distributed Database 6-33
Generating Unique Sequence Numbers Across Shards 6-34
Schema Creation Examples 6-36
Schema for System-Managed Sharding 6-36
Schema for User-Defined Sharding 6-39
Schema for Composite Sharding 6-42
DDL Failure and Recovery Examples 6-45
High Speed Data Ingest with SQL*Loader 6-50

7 Oracle Globally Distributed Database Administration

Managing the Oracle Globally Distributed Database Stack 7-1
Starting Up the Oracle Globally Distributed Database Stack 7-2
Shutting Down the Oracle Globally Distributed Database Stack 7-2
Oracle Globally Distributed Database Users and Roles 7-2
Overview of Users and Roles 7-2
Oracle Globally Distributed Database Roles 7-3
About the GSMUSER Account 7-3
About the GSMROOTUSER Account 7-4

Backing Up and Recovering Oracle Globally Distributed Database 7-4
Backup and Restoration Terminology 7-4
Automated and On-Demand Backups 7-5
Supported Backup Destinations 7-5
Limitations 7-5
Prerequisites to Configuring Centralized Backup and Restore 7-6
Configuring Automated Backups 7-7
Specifying Multiple Recovery Catalogs 7-11
Backup Set Encryption 7-11
Using Oracle Object Storage as a Backup Destination 7-13
Using Recovery Appliance as a Backup Destination 7-15
Using Amazon S3 as a Backup Destination 7-21
Enabling and Disabling Automated Backups 7-22
Backup Job Operation 7-23
Monitoring Backup Status 7-23
Viewing an Existing Backup Configuration 7-24
Running On-Demand Backups 7-26
Viewing Backup Job Status 7-26
Listing Backups 7-29
Validating Backups 7-29
Deleting Backups 7-29
Creating and Listing Global Restore Points 7-30
Restoring Shards From Backup 7-31
Restoring the Shard Catalog from Backup 7-32
Removing Backup Configuration from a Shard 7-33
Running RMAN Commands from GDSCTL 7-33
Error Handling for Automated Backup Operations 7-34
Propagation of Parameter Settings Across Shards 7-35
Patching and Upgrading Oracle Globally Distributed Database 7-35
Patching and Upgrading Oracle Globally Distributed Database 7-35
Upgrading Oracle Globally Distributed Database Components 7-36
Post-Upgrade Steps for Oracle Globally Distributed Database 21c 7-37
Compatibility and Migration from Oracle Database 18c 7-39
Downgrading an Oracle Globally Distributed Database 7-40
Managing Oracle Globally Distributed Database with Enterprise Manager Cloud Control 7-40
Prerequisite: Enable Oracle Globally Distributed Database Metrics 7-41
Prerequisite: Discover the Oracle Globally Distributed Database Topology 7-42
Oracle Globally Distributed Database Management with Oracle Enterprise Manager Cloud Control 7-43
Monitoring an Oracle Globally Distributed Database 7-46
Querying System Objects Across Shards 7-46

Monitoring an Oracle Globally Distributed Database with Enterprise Manager Cloud Control 7-47
Sharded Database Home Page 7-47
Data Distribution and Performance Page 7-50
Monitoring Oracle Globally Distributed Database with GDSCTL 7-54
Shard Management 7-55
About Adding Shards 7-55
Work Flow for Adding Shards 7-55
Removing a Shard From the Pool 7-56
Replacing a Shard 7-56
Converting a Physical Standby to a Snapshot Standby 7-59
Migrating a Non-PDB Shard to a PDB 7-60
Managing Shards with Oracle Enterprise Manager Cloud Control 7-60
Validating a Shard 7-62
Adding Primary Shards 7-62
Adding Standby Shards 7-63
Deploying Shards 7-64
Editing a Shard 7-65
Removing a Shard 7-65
Chunk Management 7-65
Resharding and Hot Spot Elimination 7-65
Moving Chunks 7-67
Updating an In-Process Chunk Move Operation 7-68
Splitting Chunks 7-69
Splitting Chunks into Shardspaces Based on Super Key 7-69
Managing Chunks with Oracle Enterprise Manager Cloud Control 7-71
Moving Chunks with Oracle Enterprise Manager Cloud Control 7-71
Splitting Chunks with Oracle Enterprise Manager Cloud Control 7-72
Shard Director Management 7-73
Creating a Shard Director 7-73
Editing a Shard Director Configuration 7-73
Removing a Shard Director 7-74
Region Management 7-74
Creating a Region 7-74
Editing a Region Configuration 7-75
Removing a Region 7-75
Shardspace Management 7-75
Creating a Shardspace 7-75
Adding a Shardspace to a Composite Sharded Database 7-76
Shardgroup Management 7-77
Creating a Shardgroup 7-78

Services Management 7-78
Creating a Service 7-78

8 Query and DML Processing

How Database Requests are Routed to the Shards 8-1
Routing Queries and DMLs Directly to Shards 8-1
Routing Queries and DMLs by Proxy 8-2
Connecting to the Query Coordinator 8-3
Query Coordinator Operation 8-3
Query Processing for Single-Shard Queries 8-5
Query Processing for Multi-Shard Queries 8-5
Specifying Consistency Levels in a Multi-Shard Query 8-6
Multi-Shard Query Coordinator Availability and Scalability 8-6
Pushing PL/SQL Function Queries to the Shards 8-7
Gathering Optimizer Statistics on Sharded Tables 8-7
Supported Query Constructs and Example Query Shapes 8-9
Queries on Sharded Tables Only 8-10
Queries Involving Both Sharded and Duplicated Tables 8-10
Supported Aggregate Functions 8-12
Queries with User-Defined Types 8-12
Execution Plans for Proxy Routing 8-14
Supported DMLs and Examples 8-16
Simple DMLs Where Only the Target Table is Referenced 8-16
DMLs Referencing Other Tables 8-16
Example Merge Statements 8-17
Limitations in Multi-Shard DML Support 8-18

9 Developing Applications for Oracle Globally Distributed Database

Direct Routing to a Shard 9-1
APIs Supporting Direct Routing 9-2
Oracle JDBC APIs 9-2
Oracle Call Interface 9-3
Oracle Universal Connection Pool APIs 9-4
Oracle Data Provider for .NET APIs 9-7
JDBC Sharding Data Source 9-8

10 Migrating to an Oracle Globally Distributed Database

Migration with Oracle Data Pump 10-1
Schema Migration 10-1

Migrating the Sample Schema 10-3
Migrating Data to a Sharded Database 10-5
Loading the Sample Schema Data 10-7
Migrating Data Without a Sharding Key 10-10
Using External Tables to Load Data into a Sharded Database 10-11
Loading Data into Duplicated Tables 10-12
Loading Data into Sharded Tables 10-13
Using Oracle GoldenGate to Replicate Data Between Sharded and Non-Sharded Databases 10-15
Oracle GoldenGate Replication Prerequisites 10-15
Replicating Data from a Non-Sharded Database to a Sharded Database 10-15

11 Raft Replication Configuration and Management

Enabling Raft Replication 11-1
Specifying Replication Unit Attributes 11-1
Ensuring Replicas Are Not Placed in the Same Rack 11-2
Getting Runtime Information for Replication Units 11-2
Scaling with Raft Replication 11-3
Moving Replication Unit Replicas 11-3
Changing the Replication Unit Leader 11-4
Copying Replication Units 11-4
Moving A Chunk to Another Replication Unit 11-5
Splitting Chunks in Raft Replication 11-5
Getting the Replication Type 11-6
Enable or Disable Reads from Follower Replication Units 11-6
Tuning Flow Control to Mitigate Follower Lag 11-6
Setting Transaction Consensus Timeout 11-8
Viewing Parameter Settings 11-8
Dynamic Performance Views for Raft Replication 11-9
Raft Replication Restrictions 11-9

12 Deploying and Managing a Directory-Based Oracle Globally Distributed Database

Directory-Based Sharding Roadmap 12-1
Creating a Shard Catalog for Directory-Based Sharding 12-2
Creating Tables Sharded by Directory 12-2
Managing Keys in Directory-Based Sharding 12-4
DML Support on Tables Sharded by Directory 12-5
Adding a New Tablespace and Chunks (Partition) in a Shardspace 12-5
Chunk Management in Directory Based Sharding 12-6

Splitting Partitions (Chunks) 12-6
Sharding Key Directory Public View 12-6

13 Using the Sharding Advisor

About Sharding Advisor 13-1
Run Sharding Advisor 13-2
Run Sharding Advisor on a Non-Production System 13-3
Review Sharding Advisor Output 13-4
Choose a Sharding Advisor Recommended Configuration 13-5
Sharding Advisor Usage and Options 13-5
Sharding Advisor Output Tables 13-7
- SHARDINGADVISOR_CONFIGURATIONS Table 13-7
- SHARDINGADVISOR_CONFIGDETAILS Table 13-8
- SHARDINGADVISOR_QUERYTYPES Table 13-8

Sharding Advisor Output Review SQL Examples 13-9
Sharding Advisor Security 13-10

14 Sharding JSON Document Collections

Overview of Sharding JSON Documents 14-1
Preparing the Environment 14-2
Creating an All-Shards User with SODA Privileges 14-2
Choosing a Sharding Key 14-3
Using SODA ID as the Sharding Key 14-4
- Creating a Sharded Table for the JSON Collection 14-4
  - Creating a Sharded Table: System-Managed 14-5
  - Creating a Sharded Table: User-Defined 14-5
- Creating a Mapped SODA Collection on the Sharded Table 14-6
- Code Samples 14-7
  - Java Code Sample 14-7
  - Python Code Sample 14-11

Using a JSON Field as a Sharding Key 14-13
- Creating a Sharded Table for the JSON Collection 14-13
  - Creating a Sharded Table: System-Managed 14-14
  - Creating a Sharded Table: User-Defined 14-14
- Creating a Mapped SODA Collection on the Sharded Table 14-15
- Creating a Trigger to Populate the Sharding Key 14-16
- Code Samples 14-17
  - Java Code Sample 14-17
  - Python Code Sample 14-19

Additional Information About Sharding with SODA 14-21
Performance Tuning 14-21
Scaling Out Shards 14-22

15 Achieving Data Sovereignty with Oracle Globally Distributed Database

Overview of Data Sovereignty 15-1
Benefits of Implementing Data Sovereignty with Oracle Globally Distributed Database 15-1
Implementing Data Sovereignty with Oracle Globally Distributed Database 15-2
Data Sovereignty Use Case 15-3
Overview of the Data Sovereignty Solution 15-3
Deployment Topology for Data Sovereignty 15-5
Configuring the Data Sovereignty Use Case 15-6
Configuring VCN Networks in All Three OCI Regions 15-6
Configuring Remote VCN Peering Between All Three Regions 15-7
Configuring Private DNS for Naming Resolution Between the Regions 15-8
Installing a Global Service Manager in Each Region 15-8
Collecting TNS entries for Shard Catalog and Sharded Databases 15-9
Configuring the Shard Catalog 15-10
Configuring the Shard Databases 15-10
Creating the Oracle Globally Distributed Database 15-11
Implementing a Session-Based Application Context Policy 15-13

16 Troubleshooting

Troubleshooting Tips 16-1
Pre-Deployment Network Validation 16-1
Checking the Sharding Method 16-1
Checking the Replication Type 16-2
Checking the Oracle Data Guard Protection Mode 16-3
Checking Which Shards Are Mapped to a Key 16-3
Checking Shard Operation Mode (Read-Only or Read-Write) 16-4
Checking DDL Text 16-5
Checking Chunk Migration Status 16-5
Checking Table Type (Sharded or Duplicated) 16-6
Checking User Type (Local or ALL_SHARD) 16-7
Identifying Tables Created as Sharded Tablespaces 16-7
Checking if Shard DDL is Enabled or Disabled 16-7
Filtering Data by Sharding Key 16-8
Gathering Optimizer Statistics on Sharded Tables 16-9
Generate HTML SQL Monitor Output for a Query Running from the Shard Catalog 16-11

Tracing and Debug Information 16-11
Enabling Tracing 16-12
Where to Find Alert Logs and Trace Files 16-12
Common Error Patterns and Resolutions 16-13
Shard Director Fails to Start 16-13
Tablespace Set Creation Fails 16-14
Issues Using Deploy Command 16-15
Issues Moving Chunks 16-15
Issues with Oracle Database Vault 16-16
Issue During Deployment of Role-Separated Environment 16-16

17 Oracle Globally Distributed Database Solutions

Combine Existing Non-Sharded Databases into a Federated Sharded Database 17-1
Overview 17-1
About Federated Sharding 17-1
Federated Sharding Schema Requirements 17-2
Sharded and Duplicated Tables in a Federated Sharding Configuration 17-2
Limitations to Federated Sharding 17-2
Federated Sharding Security 17-2
Creating and Deploying a Federated Sharding Configuration 17-2
Create the Federated Sharding Configuration 17-3
Retrieve, Inspect, and Apply the DDLs 17-4
Convert Tables to Duplicated Tables 17-5
Prepare the Shards For Multi-Shard Queries 17-6
Federated Sharding Reference 17-6
SYNC SCHEMA Operations 17-6
Troubleshooting Federated Sharding 17-10
Creating Affinity Between Middle-Tier and Shards 17-11

18 Oracle Globally Distributed Database Reference

Using GDSCTL with Oracle Globally Distributed Database 18-1
GDSCTL Operation 18-1
Starting GDSCTL 18-1
Running GDSCTL Commands Interactively 18-1
Running GDSCTL Batch Operations 18-2
GDSCTL Help Text 18-2
GDSCTL Connections 18-2
GDSCTL Shard Catalog Connections 18-2
GDSCTL Shard Director Connections 18-3

GDSCTL Commands Used with Oracle Globally Distributed Database 18-3
SHARDED_TABLE_FAMILIES 18-5

Preface

Review the following topics to:

- Discover how you can use this document to learn about Oracle Globally Distributed Database
- Get accessibility information for this document
- See a list of related documents that may help you design, develop, deploy, and manage your Oracle Globally Distributed Database environment
- Learn about typographic conventions used in this document

Audience

This document was written with a wide variety of audienaces in mind. System and application architects can use it to evaluate Oracle Globally Distributed Database suitability for their requirements. IT managers can scope out the work needed to implement Oracle Globally Distributed Database for proof of concept and production deployments. Database administrators can find information to help them deploy and maintain a sharded database. Application developers can learn about any code changes for using Oracle Globally Distributed Database. Finally, business analysts can use this document as a guide to figure out costing for an Oracle Globally Distributed Database implementation.

Documentation Accessibility

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

Access to Oracle Support

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

Related Documents

The following publications may be of particular interest to you:

- Oracle Database Install and Upgrade
- *Oracle Database Administrator's Guide*
- *Oracle Data Guard Concepts and Administration*
- *Oracle Data Guard Broker*
- *Oracle Database Global Data Services Concepts and Administration Guide*
- *Oracle Database JDBC Developer's Guide*
- *Oracle Universal Connection Pool Developer's Guide*
- *Oracle Data Provider for .NET Developer's Guide.*

- *Oracle Call Interface Developer's Guide*

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, a value in a list of values, or terms defined in the text.
italic	Italic type indicates emphasis on a particular word or phrase, or book titles.
`monospace`	Monospace type indicates • SQL statements, commands, and code in examples • SQL statements, configuration parameter names, keywords, and commands in the text • URLs, file names, folder or directory names, and paths • Text that appears on the screen, and text that you enter, when shown in combination with computer output
monospace italic	Monospace italic type indicates placeholder variables for which you supply the values.

Changes in Oracle Database 23c

The following are changes in *Oracle Globally Distributed Database Guide* for Oracle Database 23c.

New Features

New features and enhancements for Oracle Globally Distributed Database in Oracle Database 23c are described in What's New in Release 23c

Deprecated Features

The following features are deprecated in Release 23c, and may be desupported in a future release:

- Deprecation of Oracle Data Provider for .NET, Unmanaged Driver

  Oracle Data Provider for .NET (ODP.NET), Unmanaged Driver is deprecated in Oracle Database 23c, and can be desupported in a future release. Oracle recommends existing unmanaged ODP.NET applications migrate to ODP.NET, Managed Driver.

Desupported Features

The following features are desupported in Oracle Database Release 23c:

- Desupport of Service Attribute Value, `SESSION_STATE_CONSISTENCY = STATIC`

  The session attribute values `FAILOVER_TYPE = TRANSACTION` with `SESSION_STATE_CONSISTENCY = STATIC` are no longer a supported service attribute combination.

- Desupport of Oracle GoldenGate Replication for Oracle Globally Distributed Database High Availability

  The use of Oracle GoldenGate Replication for Shard-Level High Availability in Oracle Globally Distributed Database is desupported in Oracle Database 23c.

- Desupport of Oracle Database Extensions for .NET

  Oracle Database Extensions for .NET is desupported. Oracle recommends that you either place .NET code in the middle tier, or use the External Procedures feature, or rewrite the code using PL/SQL or Java.

1

Oracle Globally Distributed Database Overview

Learn about Oracle Globally Distributed Database capabilities and benefits in this high level conceptual discussion.

What is a Globally Distributed Database

A globally distributed database is a method of partitioning data to distribute the computational and storage workload, which helps in achieving hyperscale computing.

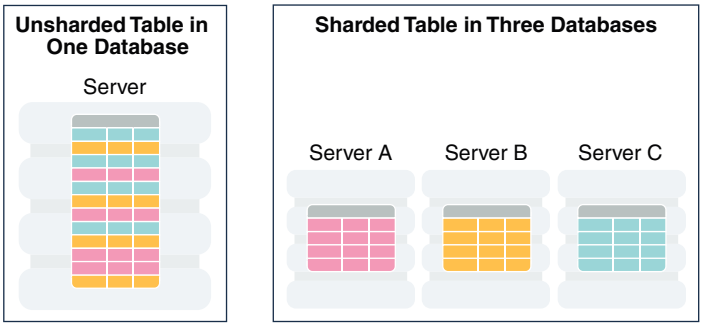
Hyperscale computing is a computing architecture that can scale up or down quickly to meet increased demand on the system. This architecture innovation was originally driven by internet giants that run distributed sites and has been adopted by large-scale cloud providers.

Companies often achieve hyperscale computing using a technology called database sharding, in which they distribute segments of a data set—a shard—across lots of databases on lots of different computers.

Sharding uses a *shared-nothing* architecture in which shards share no hardware or software. All of the shards together make up a single logical database, called a *sharded database*.

From the perspective of the application, a sharded database looks like a single database: the number of shards, and the distribution of data across those shards, are *completely transparent* to database applications. From the perspective of a database administrator, a sharded database consists of multiple databases that can be managed collectively.

Figure 1-1 Distribution of a Table Across Database Shards



About Oracle Globally Distributed Database

Oracle Globally Distributed Database is a feature of Oracle Database that lets you automatically distribute and replicate data across a pool of Oracle databases that share no hardware or software. Oracle Globally Distributed Database provides the best features and capabilities of mature RDBMS and NoSQL databases, as described here.

- SQL language used for object creation, strict data consistency, complex joins, ACID transaction properties, distributed transactions, relational data store, security, encryption, robust performance optimizer, backup and recovery, and patching with Oracle Database
- Oracle innovations and enterprise-level features, including Advanced Security, Automatic Storage Management (ASM), Advanced Compression, partitioning, high-performance storage engine, SMP scalability, Oracle RAC, Exadata, in-memory columnar, online redefinition, JSON document store, and so on
- Sharding-aware Oracle Database tools, such as SQL Developer, Enterprise Manager Cloud Control, Recovery Manager (RMAN), and Data Pump, for sharded database application development and management
- Programmatic interfaces, such as Java Database Connectivity (JDBC), Oracle Call Interface (OCI), Universal Connection Pool (UCP), Oracle Data Provider for .NET (ODP.NET), and PL/SQL, including extensions for sharded application development
- Extreme availability with Oracle Data Guard and Oracle Active Data Guard. Other replication options include Raft replication, which is built into Oracle Globally Distributed Database.
- Support for multi-model data like relational, text, and JSON
- Existing life-cycle management and operational processes can be kept, leveraging in-house and world-wide Oracle database administrator skill sets
- Enterprise-level support
- Extreme scalability and availability of NoSQL databases

Oracle Globally Distributed Database as Distributed Partitioning

Sharding is a database scaling technique based on horizontal partitioning of data across multiple independent physical databases. Each physical database in such a configuration is called a shard.

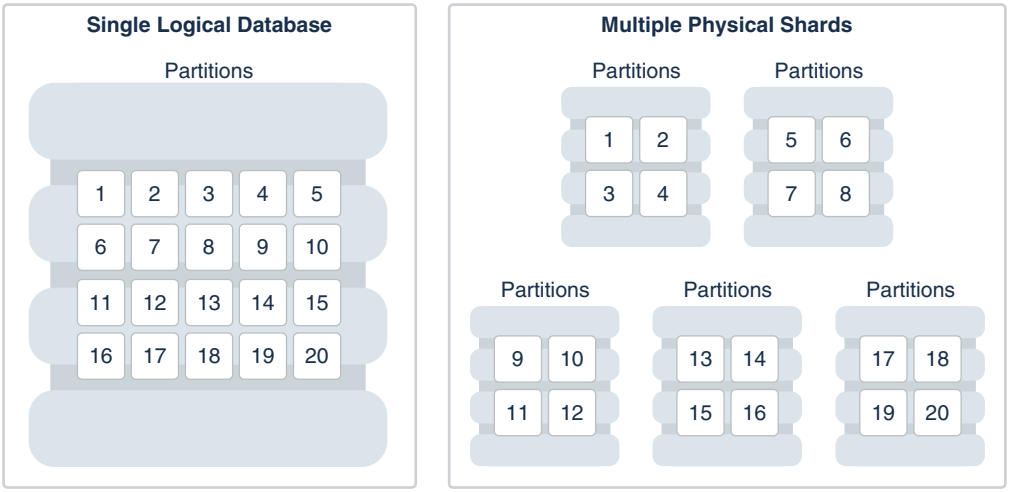
From the perspective of an application, a sharded database in Oracle Globally Distributed Database looks like a single database; the number of shards, and the distribution of data across those shards, are *completely transparent* to the application.

Even though a sharded database looks like a single database to applications and application developers, from the perspective of a database administrator, a sharded database consists of a set of discrete Oracle databases, each of which is a single shard, that can be managed collectively.

A sharded table is partitioned across all shards of the sharded database. Table partitions on each shard are not different from partitions that could be used in an Oracle database that is not sharded.

The following figure shows the difference between partitioning on a single logical database and partitions distributed across multiple shards.

Figure 1-2 Oracle Globally Distributed Database as Distributed Partitioning



Oracle Globally Distributed Database automatically distributes the partitions across shards when you issue the `CREATE SHARDED TABLE` statement, and the distribution of partitions is transparent to applications. The figure above shows the logical view of a sharded table and its physical implementation.

Benefits of Oracle Globally Distributed Database

Oracle Globally Distributed Database provides linear scalability, complete fault isolation, and global data distribution for the most demanding applications.

Key benefits of Oracle Globally Distributed Database include:

- **Linear Scalability**

  The Oracle Globally Distributed Database shared–nothing architecture eliminates performance bottlenecks and provides unlimited scalability. Oracle Globally Distributed Database supports scaling up to 1000 shards.

- **Extreme Availability and Fault Isolation**

  Single points of failure are eliminated because shards do not share resources such as software, CPU, memory, or storage devices. The failure or slow-down of one shard does not affect the performance or availability of other shards.

  Shards are protected by Oracle MAA best practice solutions, such as Oracle Data Guard and Oracle RAC.

  An unplanned outage or planned maintenance of a shard impacts only the availability of the data on that shard, so only the users of that small portion of the data are affected, for example, during a failover brownout.

- **Geographical Distribution of Data**

  Oracle Globally Distributed Database enables you to deploy a global database, where a single logical database could be distributed over multiple geographies. This makes it possible to satisfy data privacy regulatory requirements (Data Sovereignty) as well as allows to store particular data close to its consumers (Data Proximity).

Example Applications using Oracle Globally Distributed Database

Oracle Globally Distributed Database provides benefits for a variety of use cases.

Real Time OLTP

Real time OLTP applications have a very high transaction processing throughput, a large user population, huge amounts of data, and require strict data consistency and management at scale. Some examples include internet-facing consumer applications, financial applications such as mobile payments, large scale SaaS applications such as billing and medical applications. The benefits of using Oracle Globally Distributed Database for such applications include:

- Linear scalability of transactions per second, with response time staying constant as new shards are added to support larger data volume
- Better application SLAs, because planned and unplanned outages on any given shard does not impact the data stored and available on other shards
- Strict data consistency for transactional applications
- Transactions spanning multiple shards
- Support for complex joins, triggers, and stored procedures
- Simplified manageability at scale

Global Applications

Many enterprise applications are global in nature, where the same application serves customers in multiple geographic locations. Such applications typically use a single logical global database which is shared across multiple geographical regions. The benefits of a shared global database include:

- Strict enforcement of data sovereignty, where data privacy regulations require data to stay in a certain geographic location, region, country, or even state.
- Reduction of data replication across locations
- Better application SLAs, because planned and unplanned outages in one region do not impact other regions

Internet of Things and Data Streaming Applications

Typically such applications collect large amounts of data and stream it at a very high speed. Oracle Globally Distributed Database has optimized data stream libraries which use Oracle Database's direct path I/O technology to load data into the sharded database with extremely high speed. Data load requirements for these applications can be in to 100s of millions of records per second. Once the data is loaded directly into the database, it is available for immediate processing with advanced query processing and analytic capabilities.

Machine Learning

Many machine learning applications require training and scoring of models in real time. Model training and scoring for many applications using algorithms like anomaly detection, and clustering is specific to a given entity (for example, a given user's financial transaction patterns or specific device metrics at a certain time of the day).

This kind of data can easily be shared by using a sharding key specific to the user or devices. Additionally, Oracle Database Machine Learning algorithms can be applied directly in the database obviating the need for a separate data pipeline and machine learning processing infrastructure.

Big Data Analytics

When you have terabytes of data, sharding means you don't have to warehouse data to do analytics on it. With up to 1000 shards in capacity, Oracle Globally Distributed Database can turn a relational database into a warehouse-sized data store. With the Federated Sharding solution, multiple database installations in different locations that run the same application can be converted into a federated sharded database so that you can run data analytics without moving the data.

NoSQL Alternative

NoSQL solutions lack major RDBMS features, such as relational schema, SQL, complex data types, online schema changes, multi-core scalability, security, ACID properties, CR for single-shard operations, and so on. With Oracle Globally Distributed Database you get the nearly limitless scaling and sharding you had with NoSQL *and* all of the features and benefits of Oracle Database.

Flexible Deployment Models

The shared-nothing architecture of Oracle Globally Distributed Database lets you keep your data on-premises, in the cloud, or on a hybrid of cloud and on-premises systems. Because the database shards do not share any resources, the shards can exist anywhere on a variety of on-premises and cloud systems.

You can choose to deploy all of the shards on-premises, have them all in the cloud, or you can split them up between cloud and on-premises systems to suit your needs.

Shards can be deployed on all database deployment models such as single instance, Exadata, and Oracle RAC.

High Availability in Oracle Globally Distributed Database

Oracle Globally Distributed Database relies on replication for availability. Replication is automatically configured and deployed when the sharded database is created. Oracle Globally Distributed Database provides various means of replication depending on your needs.

Oracle Data Guard is tightly integrated with Oracle Globally Distributed Database to provide high availability and disaster recovery with strict data consistency and zero data loss. Oracle Data Guard replication maintains one or more synchronized copies (standbys) of a shard (the primary) for high availability and data protection. Standbys can be deployed locally or remotely, and when using Oracle Active Data Guard can also be open for read-only access.

Raft replication is built into Oracle Globally Distributed Database to provide a consensus-based, high-performance, low-overhead availability solution, with distributed replicas and fast failover with zero data loss, while automatically maintaining the replication factor if shards fail. With Raft replication management overhead does not increase with the number of shards. If you are used to NoSQL databases and do not expect to know anything about how replication works, Oracle Globally Distributed Database native replication just works.

Optionally, you can use also **Oracle RAC** for shard-level high availability, complemented by replication, to maintain shard-level data availability in the event of a cluster outage. Each

shard can be deployed on an Oracle RAC cluster to give it instant protection from node failure. For example, each shard could be a two node Oracle RAC cluster.

Data Distribution Methods

Because Oracle Globally Distributed Database is based on table partitioning, all of the sub-partitioning methods provided by Oracle Database are also supported by Oracle Globally Distributed Database. A data distribution *method* controls the placement of the data on the shards. Oracle Globally Distributed Database supports system-managed, user defined, directory-based, or composite sharding methods.

- **System-managed** does not require you to map data to shards. The data is automatically distributed across shards using partitioning by consistent hash. The partitioning algorithm uniformly and randomly distributes data across shards.
- **User-defined** lets you explicitly specify the mapping of data to individual shards. It is used when, because of performance, regulatory, or other reasons, certain data needs to be stored on a particular shard, and the administrator needs to have full control over moving data between shards.
- **Composite** allows you to use two levels of partitioning. First the data is partitioned by range or list and then it is partitioned further by consistent hash.

  In many use cases, especially for data sovereignty and data proximity requirements, the composite method offers the best of both system-managed and user-defined methods, giving you the automation you want and the control over data placement you need.
- **Directory-based** is an enhancement of the user-defined method, whereby the location of data records associated with any sharding key is specified dynamically at runtime based on user preferences. The key location information is stored in a directory, which can hold a large set of key values in the hundreds of thousands.

  You have the freedom to move individual key values from one location to another, or make bulk movements to scale up or down, or for data and load balancing. The location information can include the shard database information and partition information.

Client Request Routing

Oracle Globally Distributed Database supports direct, key-based routing from an application to a shard, routing by proxy with the shard catalog, and routing to middle tiers, such as application containers, web containers, and so on, which are affinitized with shards. Oracle Database client drivers and connection pools are sharding aware.

- **Key-based routing.** Oracle client-side drivers (JDBC, OCI, UCP, ODP.NET) can recognize sharding keys specified in the connection string for high performance data dependent routing. A shard routing cache in the connection layer is used to route database requests directly to the shard where the data resides.
- **Routing by proxy.** Oracle Globally Distributed Database supports routing for queries that do not specify a sharding key, giving any database application the flexibility to run SQL statements, without specifying the shards on which the query should be processed. Proxy routing can handle single-shard queries and multi-shard queries.
- **Middle-tier routing.** In addition to sharding the data tier, you can shard the web tier and application tier, distributing the shards of those middle tiers to service a

particular set of database shards, creating a pattern known as a *swim lane*. A smart router can route client requests based on specific sharding keys to the appropriate swim lane, which in turn establishes connections on its subset of shards.

Query Processing

No changes to query and DML statements are required to support Oracle Globally Distributed Database. Most existing DDL statements will work the same way on a sharded database with the same syntax and semantics as they do on a non-sharded Oracle Database.

In the same way that DDL statements can be processed on all shards in a configuration, so too can certain Oracle-provided PL/SQL procedures.

Oracle Globally Distributed Database also has its own keywords in the SQL DDL statements, which can only be run against a sharded database.

High Speed Data Ingest

SQL*Loader enables direct data loading into the Oracle Globally Distributed Database shards for a high speed data ingest.

SQL*Loader is a bulk loader utility used for moving data from external files into the Oracle database. Its syntax is similar to that of the DB2 load utility, but comes with more options. SQL*Loader supports various load formats, selective loading, and multi-table loads.

SQL*Loader client can automatically start a parallel direct path load for data without dividing the data into separate files and starting multiple clients.

Other benefits include:

- Streaming capability lets you receive data from a large group of clients without blocking
- Group records according to Oracle RAC shard affinity using native UCP
- Optimize CPU allocation while decoupling record processing from I/O
- Fastest insert method for the Oracle Database through Direct Path Insert, bypassing SQL and writing directly in the database files

Deployment Automation

Oracle Globally Distributed Database deployment is highly automated with Terraform, Kubernetes, and Ansible scripts.

The deployment scripts take a simple input file describing your desired deployment topology, and run from a single host to deploy shards to all of the sharded database hosts. Pause, resume, and cleanup operations are included in the scripts in case of errors.

Data Migration

The Oracle Globally Distributed Database Sharding Advisor tool helps with sharded database schema design for migration from a non-sharded to sharded database. Oracle Data Pump is sharding aware and is used to migrate data from a non-sharded Oracle database to a sharded Oracle database.

Sharding Advisor

The Sharding Advisor is a tool provided with Oracle Globally Distributed Database which can help you design an optimal sharded database configuration by analyzing your current database schema and workload, and recommending topology configurations and database schema designs. The Sharding Advisor bases recommendations on key goals such as parallelism (distributing query processing evenly among shards), minimizing cross-shard join operations, and minimizing duplicated data.

Oracle Data Pump

You can load data directly into the shards by running Oracle Data Pump on each shard. This method is very fast because the entire data loading operation can complete within the period of time needed to load the shard with the maximum subset of the entire data set.

Lifecycle Management

The Oracle Globally Distributed Database command-line interface and Oracle Enterprise Manager help you manage your sharded database.

Using the tools provided you can:

- **Provision** new sharded databases with scripts
- **Scale out** as needed by adding more shards online and take advantage of automatic rebalancing
- **Scale in** by moving data and consolidating hardware when loads are low
- **Monitor** performance statistics using Enterprise Manager
- **Back up** for disaster recovery using Cloud Backup Service, RMAN, and Zero Data Loss Recovery Appliance
- **Patches and Upgrades** automated with oPatchAuto in rolling mode

Federated Sharding

Unify multiple existing databases into one Oracle Globally Distributed Database architecture.

Global businesses might have multiple instances of same applications deployed for multiple departments in multiple regions. Federated sharding allows mapping of databases of such applications in to a single federated database and provides the following benefits.

- Queries can be seamlessly processed against a single federated database using multi-shard query coordinator
- Removes the need to replicate data for reporting and analytics purposes
- Tolerance for differences in schema and database versions

What's New in Release 23c

Raft Replication

Raft replication, built right into Oracle Globally Distributed Database, provides built-in replication without requiring configuration of Oracle GoldenGate or Oracle Data Guard. Raft replication uses logical replication with a consensus-based commit protocol, which enables declarative replication configuration and sub-second fail over.

Oracle Globally Distributed Database relies on replication for availability, but using Active Data Guard and GoldenGate requires you to understand, deploy, and maintain the underlying replication technologies, which results in operational overhead, especially with hyperscale deployments.

Raft replication makes replication simple and transparent because replication is built right into Oracle Globally Distributed Database.

See Replication in Oracle Globally Distributed Database for more information.

Directory-Based Sharding

Today, our customers are using three types of sharding methods: System-managed, User-defined , and Composite sharding. Directory-based sharding, which is an enhanced type of user-defined data distribution method, lets you control the placement of the data on the shards.

Directory-based sharding helps you use sharding more efficiently when there is not a large enough set of distinct key values (that is, enough to result in an even distribution of data with system-managed sharding by hashing). Directory-based sharding also provides full control over the mapping of key values to shards.

Directory-based sharding allows you to explicitly associate key value with shards, giving you full control over the mapping of key values to shards.

See Directory-Based Sharding for more information.

Synchronous Duplicated Tables

This feature introduces a new kind of duplicated table that is synchronized on the shards ‘on-commit’ on the shard catalog. The rows in a duplicated table on the shards are synchronized with the rows in the duplicated table on the shard catalog when the active transaction performing DMLs on the duplicated tables in the shard catalog is committed.

This features enables efficient and absolute data consistency and synchronization for duplicated tables, across all shards at all times.

See Creating Duplicated Tables for more information.

Fine-Grained Refresh Rate Control For Duplicated Tables

This feature enables refresh rate control for individual duplicated tables, and it helps optimize the use of resources by customization of refresh rates for individual duplicated tables. .

The new `REFRESH` clause in `CREATE DUPLICATED TABLE` and `ALTER TABLE` allows you to specify a refresh interval for a duplicated table. You can set an interval in seconds, minutes, hours, or you can set the table to only refresh on demand. For example:

```
CREATE DUPLICATED TABLE Products
...
REFRESH INTERVAL 2 MINUTE;

ALTER TABLE Products MODIFY REFRESH ON DEMAND;
```

See Customizing Duplicated Table Refresh Rates for more information.

Sharded Database Coordinated Backup and Restore Enhancements

Coordinated backup and restore functionality has several major enhancements in this release:

- Enhanced error handling and diagnosis for backup jobs (see Error Handling for Automated Backup Operations).
- Automation of shard catalog database restore. Previously the shard catalog had to be restored manually using RMAN. In this release `GDSCTL RESTORE BACKUP` is enhanced to support shard catalog restoration with the option `-shard CATALOG` (see Restoring the Shard Catalog from Backup).
- Using the new `GDSCTL RMAN` command you can submit RMAN commands to a list of shards for execution. RMAN statements can be submitted in the body of the command, or you can reference an RMAN command file. (see Running RMAN Commands from GDSCTL).
- Support for using different RMAN recovery catalogs for different shards is provided by running `GDSCTL CONFIG BACKUP` multiple times to specify different recovery catalogs for specific shards and the shard catalog (see Specifying Multiple Recovery Catalogs).
- Encryption of backup sets with the new `GDSCTL CONFIG BACKUP -encryption` option lets you enable or disable the encryption of backup sets and choose an encryption algorithm (see Backup Set Encryption).
- Support for additional backup destinations:
  - Oracle Object Storage - using existing `GDSCTL CONFIG BACKUP` options you can now send backups to Object Storage (see Using Oracle Object Storage as a Backup Destination).
  - Zero Data Loss Recovery Appliance - `GDSCTL CONFIG BACKUP` is enhanced with several new Recovery Appliance-specific parameters (`-zdlra_*`) to specify Recovery Appliance as the backup destination (see Using Recovery Appliance as a Backup Destination).
  - Amazon S3 - using existing `GDSCTL CONFIG BACKUP` options you can now send backups to Amazon Simple Storage Service (see Using Amazon S3 as a Backup Destination).

Automatic Data Move on Sharding Key Update

Sometimes a sharding key value needs updating. The previous solution was to delete the data associated with the old key value and re-insert it with a new key value. The goal of this enhancement is to allow row movement both within a shard and between shards.

When the sharding key value on a particular row of a sharded table is updated, Now moving the data to a new location is handled by Oracle Globally Distributed Database, whether it is in a different partition on the same shard or on a different shard.

This feature provides you with the flexibility to update the sharding key without worrying about the destination of the records, because this feature allows Oracle Globally Distributed Database to take care of the row movement regardless of the row destination, which could be on a different shard.

Benefits:

- Provides flexibility to update the sharding key value
- Allows transparent data movement between shards

See Enabling Automatic Data Movement on Sharding Key Update for more information.

Move Data Chunks Between Shardspaces

In the composite sharding method, the data can be organized in different shardspaces based on the value of super shard key column values. This enhancement creates a way to split the existing data chunks as per super shard key values into new shardspaces.

In Oracle 21c, for composite sharding, the data should be arranged physically as per super shard keys, possibly in different a shardspace once the super shard key is specified, but this data movement had to be done manually by selecting a huge amount of data from the source shardspace, inserting it into a temporary table, then deleting it from the source shardspace and inserting from the temporary table into the target shardspace. This would be done over database links (`DBLINK`), row by row, and would take a significant amount of time.

Now bulk move of data between shardspaces is supported by running the `split partitionset` command. When there is a need to re-arrange the data based on the super shard key, you split the existing data chunks as per super shard key values into new shardspaces, which triggers the bulk data movement. Oracle Globally Distributed Database 23c splits the data and makes the data available for access for client connections for the maximum possible time while the bulk data movement operation is in progress.

Benefits :

- Automates the splitting and moving data
- Keeps data online during split and move
- Allows bulk data move using transportable tablespace

See Splitting Chunks into Shardspaces Based on Super Key for more information.

Global Partitioned Index Support on Subpartitions

This feature allows a global partitioned index on the sharding key when the sharded table is sub-partitioned, as in a composite sharded database. You can create primary key/unique indexes on sharded tables that are composite partitioned without having to include sub-partition keys.

This feature removes the restriction on the primary key columns when the sharded table is sub-partitioned, as in composite sharding.

See Creating Indexes on Sharded Tables for more information and examples.

PL/SQL Function Cross-Shard Query Support

PL/SQL functions are enhanced with the keyword `SHARD_ENABLE` to allow PL/SQL functions to be referenced in cross-shard queries. With the new keyword, the query optimizer takes the initiative to push the execution of the PL/SQL function to the shards.

This feature significantly improves performance for PL/SQL functions for sharded database environments.

See Pushing PL/SQL Function Queries to the Shards for more information.

See SHARD_ENABLE Clause in *Oracle Database PL/SQL Language Reference* for syntax and usage information.

Pre-Deployment Network Validation

An option to run a series of checks while processing several GDSCTL commands during the configuration and deployment of a sharded database verify that there is no potential environmental issue.

This feature proactively avoids common pitfalls to reduce time taken to complete a sharded database deployment.

See Pre-Deployment Network Validation for more information.

New Partition Set Operations for Composite Sharding

For sharded databases using the composite sharding method, two new `ALTER TABLE` operations enhance partition set maintenance.

Previously, partition set operations did not support specifying tablespace sets for child and reference-partitioned tables that are affected due to add and split partition set operations. `MOVE PARTITIONSET` lets you move a whole partition set from one tablespace set to another, within the same shardspace. `MODIFY PARTITIONSET` lets you add values to the list of values of a given partition set.

These new operations enhance resharding capability. `MOVE PARTITIONSET` gives you the control to move all subpartitions of a given table to another tablespace set, within a given shardspace. You can also specify separate tablespace sets for LOBs and subpartitions. `MODIFY PARTITIONSET` extends the add list values feature of partitions to partition sets.
See ALTER TABLE in *Oracle Database SQL Language Reference* for more information.

Parallel Cross-Shard DML Support

The Oracle Globally Distributed Database query coordinator can run cross-shard updates and inserts in parallel on multiple shards.

This feature improves cross-shard DML performance by running updates and inserts in parallel rather than serially.

Oracle Data Pump Adds Support for Sharding Metadata

Oracle Data Pump supports data migration to Oracle Globally Distributed Database with support for sharded DDL. You can migrate sharding objects to a target database based on source database shard objects.

Oracle Data Pump adds support for sharding DDL in the API `dbms_metadata.get_ddl()`. A new transform parameter, `INCLUDE_SHARDING_CLAUSES`, facilitates this support. If this parameter is set to `true`, and the underlying object contains it, then the `get_ddl()` API returns sharding DDL for `create table`, `sequence`, `tablespace`, and `tablespace set`.

See *Oracle Database Utilities* topics TRANSFORM and Placing Conditions on Transforms, and *Oracle Database PL/SQL Packages and Types Reference* topic SET_TRANSFORM_PARAM and SET_REMAP_PARAM Procedures for details, examples, and reference.

Automatic Parallel Direct Path Load Using SQL*Loader

SQL*Loader enables direct data loading into the database shards for high speed data ingest. SQL*Loader can load data faster and easier into Oracle Database with automatic parallelism and more efficient data storage.

With this release, SQL*Loader client can automatically start a parallel direct path load for data without dividing the data into separate files and starting multiple SQL*Loader clients. Instead of preparing your tables manually for parallel loads and setting the `PARALLEL` parameter, you can perform the same task automatically by running SQL*Loader with just one command, setting the degree of parallelism using the `DEGREE_OF_PARALLELISM` parameter, and setting `DIRECT=TRUE`.

See *Oracle Database Utilities* topics Sharded Automatic Parallel Loading Modes for SQL*Loader and Automatic Parallel Load of Table Data with SQL*Loader for more information.

Where To Go From Here

Planning and deploying a Oracle Globally Distributed Database configuration that best fits your requirements can be a daunting task. The following roadmap can guide you through the process, from initial planning to life cycle management of a sharded database.

- **Learn** about Oracle Globally Distributed Database components, architecture, and how Oracle Globally Distributed Database works in Oracle Globally Distributed Database Architecture and Concepts
- **Plan** your specific sharded database requirements, including both the technical and operational aspects of your IT systems and business processes, as described in Planning Your Sharded Database Deployment
- **Deploy** a sharded database topology configuration, as explained, with examples, in Oracle Globally Distributed Database Deployment
- **Design** a sharded database schema for balanced distribution of data and workload across shards in Oracle Globally Distributed Database Schema Design
- **Develop** a high performance, efficient sharded database application using the concepts and APIs described in Developing Applications for Oracle Globally Distributed Database
- **Migrate** your existing database and application to a sharded database, as explained in Migrating to an Oracle Globally Distributed Database
- **Manage** your sharded database with the procedures described in Oracle Globally Distributed Database Administration

2

Oracle Globally Distributed Database Architecture and Concepts

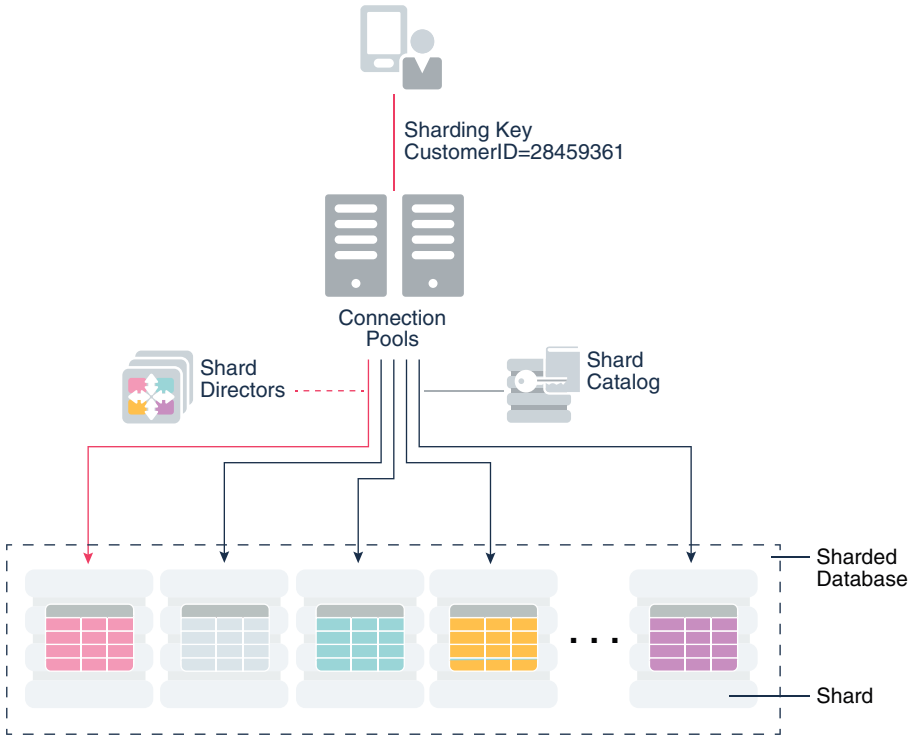
The following topics guide you through the concepts and architecture for Oracle Globally Distributed Database.

- Components of the Oracle Globally Distributed Database Architecture
- Oracle Globally Distributed Database Schema Objects
- Sharded Data Distribution Methods
- Replication in Oracle Globally Distributed Database
- Client Application Request Routing
- Query Processing and the Query Coordinator

Components of the Oracle Globally Distributed Database Architecture

The following figure illustrates the major architectural components of Oracle Globally Distributed Database, which are described in the topics that follow.

Figure 2-1 Oracle Globally Distributed Database Architecture



Sharded Database and Shards

A **sharded database** is a collection of **shards**.

A sharded database is a single logical Oracle Database that is horizontally partitioned across a pool of physical Oracle Databases (shards) that share no hardware or software.

Each shard in the sharded database is an independent Oracle Database instance that hosts subset of a sharded database's data. Shared storage is not required across the shards.

Shards can be hosted anywhere an Oracle database can be hosted. Oracle Globally Distributed Database supports all of the deployment choices for a shard that you would expect with a single instance or clustered Oracle Database, including on-premises, any cloud platform, Oracle Exadata Database Machine, virtual machines, and so on.

Shards can all be placed in one region or can be placed in different regions. A region in the context of Oracle Globally Distributed Database represents a data center or multiple data centers that are in close network proximity.

Shards can be replicated for high availability and disaster recovery with Oracle Globally Distributed Database native replication (Raft-based) or Oracle Data Guard.

Shard Catalog

A **shard catalog** is an Oracle Database that supports automated shard deployment, centralized management of an Oracle Globally Distributed Database sharded database, and multi-shard queries.

A shard catalog serves following purposes

- Serves as an administrative server for entire sharded database
- Stores a gold copy of the database schema
- Manages multi-shard queries with a multi-shard query coordinator
- Stores a gold copy of duplicated table data

The shard catalog is a special-purpose Oracle Database that is a persistent store for sharded database configuration data and plays a key role in centralized management of a sharded database. All configuration changes, such as adding and removing shards and global services, are initiated on the shard catalog. All DDLs in a sharded database are processed by connecting to the shard catalog.

The shard catalog also contains the primary copy of all duplicated tables in a sharded database. The shard catalog uses materialized views to automatically replicate changes to duplicated tables in all shards. The shard catalog database also acts as a query coordinator used to process multi-shard queries and queries that do not specify a sharding key.

Multiple shard catalogs can be deployed for high availability purposes. Using Oracle Data Guard for shard catalog high availability is a recommended best practice.

At run time, unless the application uses key-based queries, the shard catalog is required to direct queries to the shards. Sharding key-based transactions continue to be routed and processed by the sharded database and are unaffected by a catalog outage.

During the brief period required to complete an automatic failover to a standby shard catalog, downtime affects the ability to perform maintenance operations, make schema changes, update duplicated tables, run multi-shard queries, or perform other operations like add shard, move chunks, and so on, which induce topology change.

Shard Director

Shard directors are network listeners that enable high performance connection routing based on a sharding key.

Oracle Database 12*c* introduced the global service manager to route connections based on database role, load, replication lag, and locality. In support of Oracle Globally Distributed Database, global service managers support routing of connections based on data location. A global service manager, in the context of Oracle Globally Distributed Database, is known as a shard director.

A shard director is a specific implementation of a global service manager that acts as a regional listener for clients that connect to a sharded database. The director maintains a current topology map of the sharded database. Based on the sharding key passed during a connection request, the director routes the connections to the appropriate shard.

For a typical sharded database, a set of shard directors are installed on dedicated low-end commodity servers in each region. To achieve high availability and scalability, deploy multiple shard directors. You can deploy up to five shard directors in a given region.

The following are the key capabilities of shard directors:

- Maintain runtime data about sharded database configuration and availability of shards
- Measure network latency between its own and other regions
- Act as a regional listener for clients to connect to a sharded database
- Manage global services
- Perform connection load balancing

Global Service

A **global service** is a database service that is use to access data in an Oracle Globally Distributed Database sharded database.

A global service is an extension to the notion of the traditional database service. All of the properties of traditional database services are supported for global services. For sharded databases, additional properties are set for global services — for example, database role, replication lag tolerance, region affinity between clients and shards, and so on. For a read-write transactional workload, a single global service is created to access data from any primary shard in a sharded database. For highly available shards using Oracle Active Data Guard, a separate read-only global service can be created.

Management Interfaces for Oracle Globally Distributed Database

The GDSCTL command-line utility is used to configure, deploy, monitor, and manage an Oracle Globally Distributed Database sharded database. Oracle Enterprise Manager Cloud Control can also be used for monitoring and management.

Like SQL*Plus, `GDSCTL` is a command-line utility with which you can control all stages of a sharded database's life cycle. You can run `GDSCTL` remotely from a different server or laptop to configure and deploy a sharded database topology, and then montior and manage your sharded database.

`GDSCTL` provides a simple declarative way of specifying the configuration of a sharded database and automating its deployment. Only a few GDSCTL commands are required to create a sharded database.

You can also use Cloud Control for sharded database monitoring and life cycle management if you prefer a graphical user interface. With Cloud Control you can monitor availability and performance, and you can make changes to a sharding configuration, such as add and deploy shards, services, shard directors, and other sharding components.

Oracle Globally Distributed Database Schema Objects

To obtain the benefits of sharding in an Oracle Globally Distributed Database, the schema of a sharded database should be designed in a way that maximizes the number of database requests processed on a single shard.

Partitions, Tablespaces, and Chunks

Distribution of partitions across shards is achieved by creating partitions in tablespaces that reside on different shards.

Each partition of a sharded table is stored in a separate tablespace, making the tablespace the unit of data distribution in a sharded database.

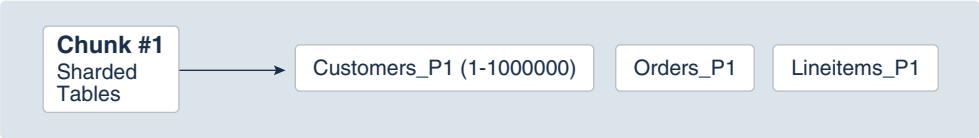
As described in Sharded Table Family, to minimize the number of multi-shard joins, corresponding partitions of all tables in a table family are always stored in the same shard. This is guaranteed when tables in a table family are created in the same set of distributed tablespaces as shown in the syntax examples where tablespace set `ts1` is used for all tables.

However, it is possible to create different tables from a table family in different tablespace sets, for example the Customers table in tablespace set `ts1` and Orders in tablespace set `ts2`. In this case, it must be guaranteed that the tablespace that stores partition 1 of Customers always resides in the same shard as the tablespace that stores partition 1 of Orders.

To support this functionality, a set of corresponding partitions from all of the tables in a table family, called a *chunk*, is formed. A chunk contains a single partition from each table of a table family. This guarantees that related data from different sharded tables can be moved together. In other words, a chunk is the unit of data migration between shards. In system-managed and composite sharding, the number of chunks within each shard is specified when the sharded database is created. In user-defined sharding, the total number of chunks is equal to the number of partitions.

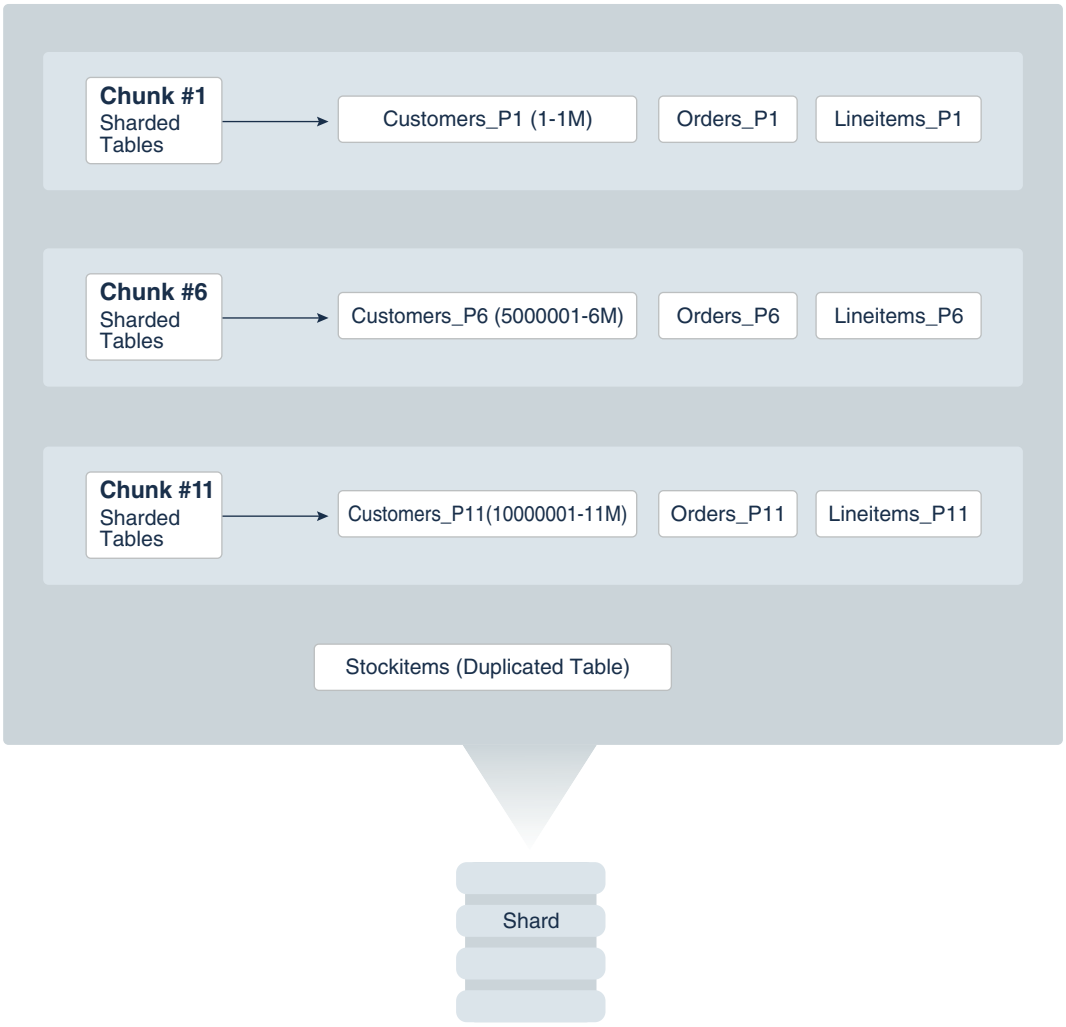
A chunk that contains corresponding partitions from the tables of Cutomers-Orders-LineItems schema is shown in the following figure.

Figure 2-2 Chunk as a Set of Partitions



Each shard contains multiple chunks as shown in the following figure.

Figure 2-3 Contents of a Shard



In addition to sharded tables, a shard can also contain one or more duplicated tables. Duplicated tables cannot be stored in tablespaces that are used for sharded tables.

Tablespace Sets

Oracle Globally Distributed Database creates and manages tablespaces as a unit called a `TABLESPACE SET`.

System-managed and composite sharding use `TABLESPACE SET`, while user-defined sharding uses regular tablespaces.

A tablespace is a logical unit of data distribution in a sharded database. The distribution of partitions across shards is achieved by automatically creating partitions in tablespaces that reside on different shards. To minimize the number of multi-shard joins, the corresponding partitions of related tables are always stored in the same shard. Each partition of a sharded table is stored in a separate tablespace.

The `PARTITIONS AUTO` clause specifies that the number of partitions should be automatically determined. This type of hashing provides more flexibility and efficiency in migrating data between shards, which is important for elastic scalability.

The number of tablespaces created per tablespace set is determined based on the number of chunks that were defined for the shardspace during deployment.

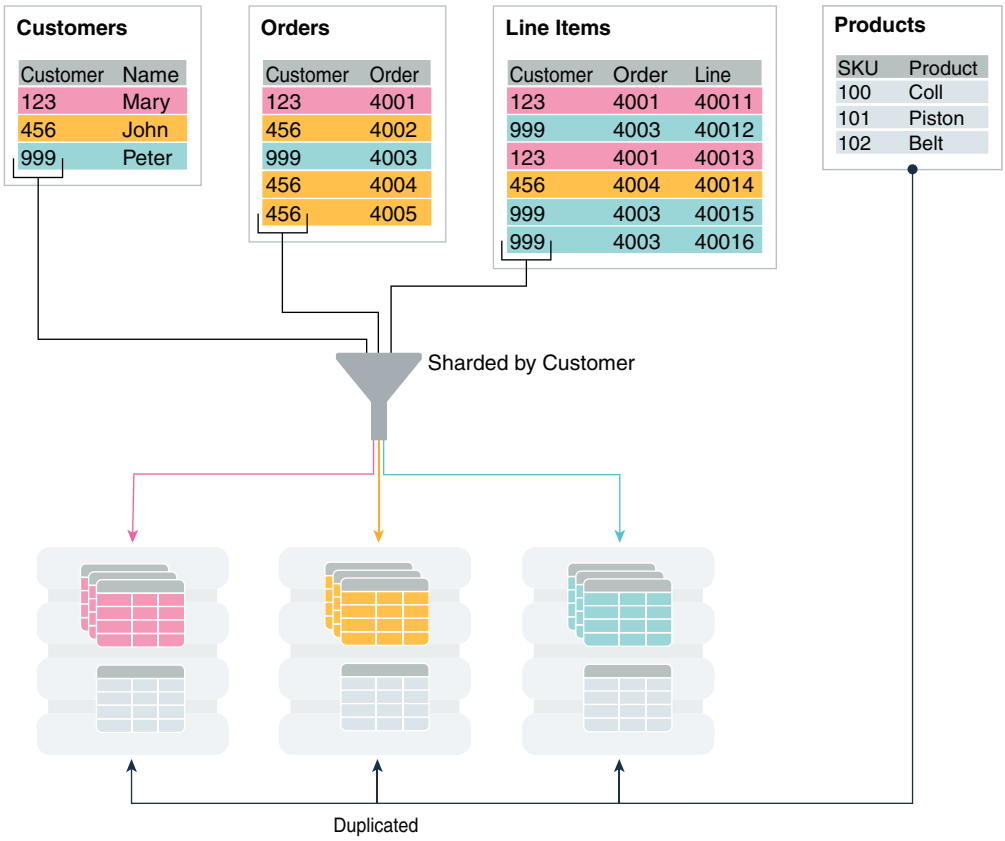
> **Note:**
>
> Only Oracle Managed Files are supported by tablespace sets.
>
> Individual tablespaces cannot be dropped or altered independently of the entire tablespace set.
>
> `TABLESPACE SET` cannot be used with the user-defined sharding method.

Sharded Tables

A database table is split up across the shards, so that each shard contains the table with the same columns, but a *different subset of rows*. A table split up in this manner is called a *sharded table*.

The following figure shows how a set of large tables (referred to as a table family), shown in the one database on the left, can be horizontally partitioned across the three shards shown on the right, so that each shard contains a subset of the data, indicated with red, yellow, and blue rows.

Figure 2-4 Horizontal Partitioning of a Table Across Shards



Partitions are distributed across shards at the tablespace level, based on a sharding key. Examples of keys include customer ID, account number, and country ID. The following data types are supported for the sharding key.

- `NUMBER`
- `INTEGER`
- `SMALLINT`
- `RAW`
- `(N)VARCHAR`
- `(N)VARCHAR2`
- `(N)CHAR`
- `DATE`
- `TIMESTAMP`

Each partition of a sharded table resides in a separate tablespace, and each tablespace is associated with a specific shard. Depending on the sharding method, the association can be established automatically or defined by the administrator.

Even though the partitions of a sharded table reside in multiple shards, to the application, the table looks and behaves exactly the same as a partitioned table in a

single database. SQL statements issued by an application never have to refer to shards or depend on the number of shards and their configuration.

The familiar SQL syntax for table partitioning specifies how rows should be partitioned across shards. For example, the following SQL statement creates a sharded table, horizontally partitioning the table across shards based on the sharding key `cust_id`.

```
CREATE SHARDED TABLE customers
( cust_id     NUMBER NOT NULL
, name        VARCHAR2(50)
, address     VARCHAR2(250)
, region      VARCHAR2(20)
, class       VARCHAR2(3)
, signup      DATE
,CONSTRAINT cust_pk PRIMARY KEY(cust_id)
)
PARTITION BY CONSISTENT HASH (cust_id)
PARTITIONS AUTO
TABLESPACE SET ts1
;
```

The sharded table is partitioned by consistent hash, a special type of hash partitioning commonly used in scalable distributed systems. This technique automatically spreads tablespaces across shards to provide an even distribution of data and workload.

> **Note:**
>
> Global indexes on sharded tables are not supported, but local indexes are supported.

Sharded Table Family

A sharded table family is a set of tables that are sharded in the same way. Often there is a parent-child relationship between database tables with a referential constraint in a child table (foreign key) referring to the primary key of the parent table.

Multiple tables linked by such relationships typically form a tree-like structure where every child has a single parent. A set of such tables is referred to as a table family. A table in a table family that has no parent is called the root table. There can be only one root table in a table family.

How a Table Family Is Sharded

Sharding a table family is illustrated here with the Customers–Orders–LineItems schema.

Before sharding, the tables in the schema may look as shown in the examples below. The three tables have a parent-child relationship, with Customers as the root table.

Customers Table (Root) Before Sharding

```
CustNo    Name       Address        Location  Class
--------- ---------- -------------- --------- ------
123       Brown      100 Main St    us3       Gold
```

```
456       Jones      300 Pine Ave   us1       Silver
999       Smith      453 Cherry St  us2       Bronze
```

Orders Table Before Sharding

```
OrderNo   CustNo   OrderDate
--------- -------- -----------
4001      123      14-FEB-2013
4002      456      09-MAR-2013
4003      456      05-APR-2013
4004      123      27-MAY-2013
4005      999      01-SEP-2013
```

LineItems Table Before Sharding

```
LineNo  OrderNo  CustNo  StockNo    Quantity
------  -------  ------  -------    --------
40011   4001     123     05683022   1
40012   4001     123     45423609   4
40013   4001     123     68584904   1
40021   4002     456     05683022   1
40022   4002     456     45423509   3
40022   4003     456     80345330   16
40041   4004     123     45423509   1
40042   4004     123     68584904   2
40051   4005     999     80345330   12
```

The tables can be sharded by the customer number, `CustNo`, in the Customers table, which is the root. The shard containing data pertaining to customer 123 is shown in the following example tables.

Customers Table Shard With Customer 123 Data

```
CustNo    Name       Address        Location   Class
--------- ---------- -------------- ---------- ------
123       Brown      100 Main St    us3        Gold
```

Orders Table Shard With Customer 123 Data

```
OrderNo   CustNo   OrderDate
--------- -------- -----------
4001      123      14-FEB-2013
4004      123      27-MAY-2013
```

LineItems Table Shard With Customer 123 Data

```
LineNo  OrderNo  CustNo  StockNo    Quantity
------  -------  ------  -------    --------
40011   4001     123     05683022   1
40012   4001     123     45423609   4
40013   4001     123     68584904   1
40041   4004     123     45423509   1
40042   4004     123     68584904   2
```

Duplicated Tables

In Oracle Globally Distributed Database a table with the same contents in each shard is called a *duplicated table*.

A sharded database includes both **sharded** tables that are horizontally partitioned across shards, and **duplicated** tables that are replicated to all shards. Duplicated

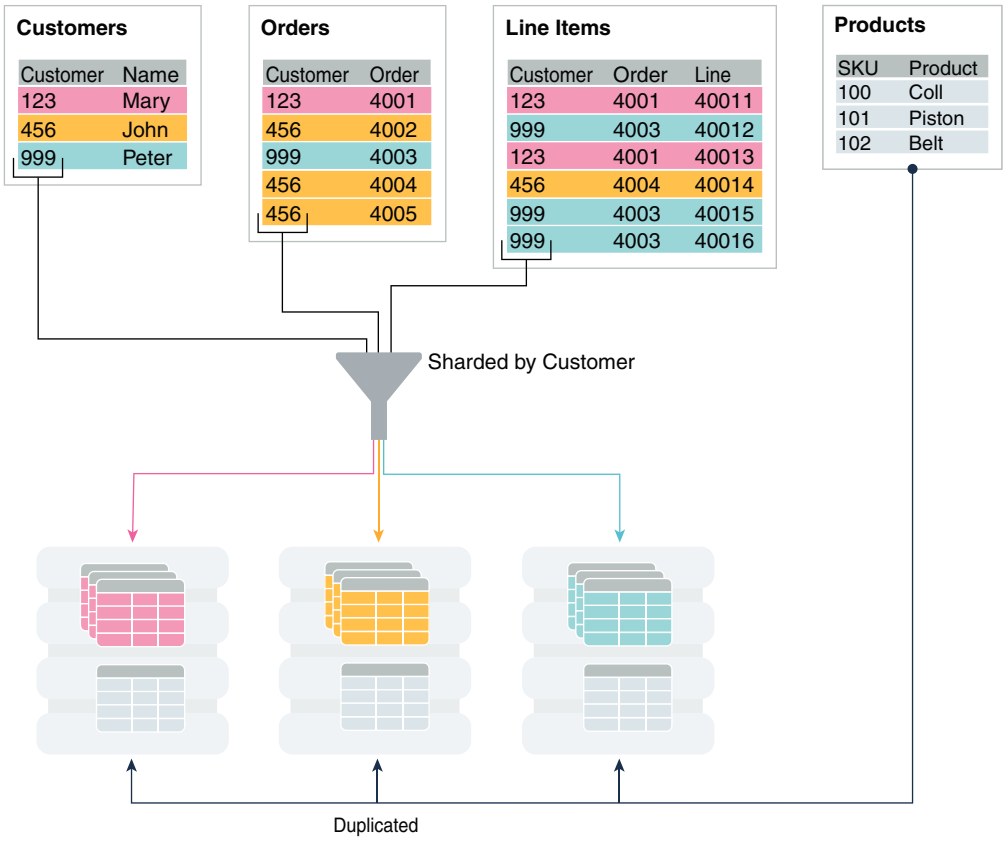
tables are a good choice for relatively small tables that are not updated frequently, and that are often accessed together with sharded tables. For this reason, duplicated tables usually contain reference information, for example, a Stock Items table that is common to each shard.

For many applications, the number of database requests handled by a single shard can be maximized by duplicating read-only or read-mostly tables across all shards. The combination of sharded and duplicated tables enables all transactions associated with a sharding key to be processed by a single shard. This technique enables linear scalability and fault isolation.

As an example of the need for a duplicated table, consider the table family that is described in Sharded Table Family. This database schema might also include a `Products` table which contains data that is shared by all the customers in the shards that were created for this table family, and it cannot be sharded by the customer number. To prevent multi-shard queries during order processing, the entire table can be duplicated on all shards in the sharded database.

The difference between sharded tables (Customers, Orders, and Line Items) and a duplicated table (Products) is shown in the following figure.

Figure 2-5 Sharded Tables and a Duplicated Table



In the figure above, the Customers, Orders, and Line Items tables are all sharded by a Customer ID number into three shards, illustrated by the colors of the rows in each table at the top of the figure, and the corresponding color of the sharded table in each shard. The duplicated table , Products, shown in gray, is replicated to all of the shards in its entirety, shown by the arrow from the table pointing to each of the three shards.

See Creating Duplicated Tables for more information, limitations, and examples.

Non-Table Objects Created on All Shards

In addition to duplicated tables, other schema objects, such as users, roles, views, indexes, synonyms, functions, procedures, and packages, and non-schema database objects, such as tablespaces, tablespace sets, directories, and contexts, can be created on all shards in an Oracle Globally Distributed Database.

Unlike tables, which require an extra keyword in the `CREATE` statement—`SHARDED` or `DUPLICATED`—other objects are created on all shards using existing syntax. The only requirement is that the `SHARD DDL` session property must be enabled.

Note that *automatic* creation on all shards of the following objects is not supported in this release. These objects can be created by connecting to individual shards.

- Cluster
- Control file
- Database link
- Disk group
- Edition
- Flashback archive
- Materialized zone map
- Outline
- Pfile
- Profile
- Restore point
- Rollback segment
- Summary

Materialized views and view logs are supported starting in Oracle Database 18*c*, with the following restrictions:

- Materialized views created on sharded tables remain empty on the catalog database, while the corresponding materialized views on shards contain data from each of the individual shards.
- Only the `REFRESH COMPLETE ON DEMAND USING TRUSTED CONSTRAINTS` option is supported for materialized views on sharded tables.

Sharded Data Distribution Methods

Learn about the methods supported by Oracle Globally Distributed Database to distribute sharded table data (also called the **sharding method**, how to choose a sharding method, and how to use subpartitioning.

The following topics discuss each of the sharding methods supported in Oracle Globally Distributed Database:

- System-Managed Sharding

- User-Defined Sharding
- Directory-Based Sharding
- Composite Sharding

System-Managed Sharding

System-managed sharding is a sharding method which does not require the user to specify mapping of data to shards. Data is automatically distributed across shards using partitioning by consistent hash. The partitioning algorithm evenly and randomly distributes data across shards.

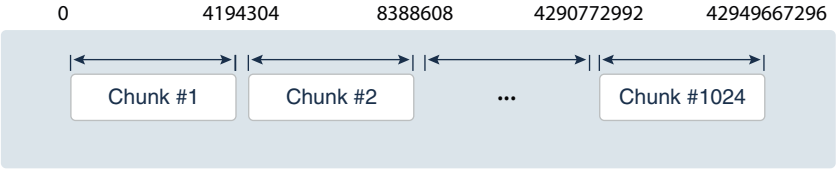
The distribution used in system-managed sharding is intended to eliminate hot spots and provide uniform performance across shards. Oracle Globally Distributed Database automatically maintains the balanced distribution of chunks when shards are added to or removed from a sharded database.

Consistent hash is a partitioning strategy commonly used in scalable distributed systems. It is different from traditional hash partitioning. With traditional hashing, the bucket number is calculated as `HF(key) % N` where HF is a hash function and N is the number of buckets. This approach works fine if N is constant, but requires reshuffling of all data when N changes.

More advanced algorithms, such as linear hashing, do not require rehashing of the entire table to add a hash bucket, but they impose restrictions on the number of buckets, such as the number of buckets can only be a power of 2, and on the order in which the buckets can be split.

The implementation of consistent hashing used in Oracle Globally Distributed Database avoids these limitations by dividing the possible range of values of the hash function (for example. from 0 to 2^{32}) into a set of N adjacent intervals, and assigning each interval to a chunk , as shown in the figure below. In this example, the sharded database contains 1024 chunks, and each chunk gets assigned a range of 2^{22} hash values. Therefore partitioning by consistent hash is essentially partitioning by the range of hash values.

Figure 2-6 Ranges of Hash Values Assigned to Chunks



Assuming that all of the shards have the same computing power, an equal number of chunks is assigned to each shard in the sharded database. For example, if 1024 chunks are created in a sharded database that contains 16 shards, each shard will contain 64 chunks.

In the event of resharding, when shards are added to or removed from a sharded database, some of the chunks are relocated among the shards to maintain an even distribution of chunks across the shards. The contents of the chunks does not change during this process; no rehashing takes place.

When a chunk is split, its range of hash values is divided into two ranges, but nothing needs to be done for the rest of the chunks. Any chunk can be independently split at any time.

All of the components of a sharded database that are involved in directing connection requests to shards maintain a routing table that contains a list of chunks hosted by each shard and ranges of hash values associated with each chunk. To determine where to route a particular database request, the routing algorithm applies the hash function to the provided value of the sharding key, and maps the calculated hash value to the appropriate chunk, and then to a shard that contains the chunk.

The number of chunks in a sharded database with system-managed sharding can be specified when the shard catalog is created. If not specified, the default value, 120 chunks per shard, is used. Once a sharded database is deployed, the number of chunks can only be changed by splitting chunks.

Before creating a sharded table partitioned by consistent hash, a set of tablespaces (one tablespace per chunk) has to be created to store the table partitions. The tablespaces are automatically created by processing the SQL statement, `CREATE TABLESPACE SET`.

All of the tablespaces in a tablespace set have the same physical attributes and can only contain Oracle Managed Files (OMF). In its simplest form, the `CREATE TABLESPACE SET` statement has only one parameter, the name of the tablespace set, for example:

```
CREATE TABLESPACE SET ts1;
```

In this case each tablespace in the set contains a single OMF file with default attributes. To customize tablespace attributes, the `USING TEMPLATE` clause (shown in the example below) is added to the statement. The `USING TEMPLATE` clause specifies attributes that apply to each tablespace in the set.

```
CREATE TABLESPACE SET ts1
USING TEMPLATE
(
 DATAFILE SIZE 10M
 EXTENT MANAGEMENT LOCAL UNIFORM SIZE 256K
 SEGMENT SPACE MANAGEMENT AUTO
 ONLINE
)
;
```

After a tablespace set has been created, a table partitioned by consistent hash can be created with partitions stored in the tablespaces that belong to the set. The `CREATE TABLE` statement might look as follows:

```
CREATE SHARDED TABLE customers
( cust_id     NUMBER NOT NULL
, name        VARCHAR2(50)
, address     VARCHAR2(250)
, location_id VARCHAR2(20)
, class       VARCHAR2(3)
, signup      DATE
, CONSTRAINT cust_pk PRIMARY KEY(cust_id)
)
PARTITION BY CONSISTENT HASH (cust_id)
PARTITIONS AUTO
```

```
TABLESPACE SET ts1
;
```

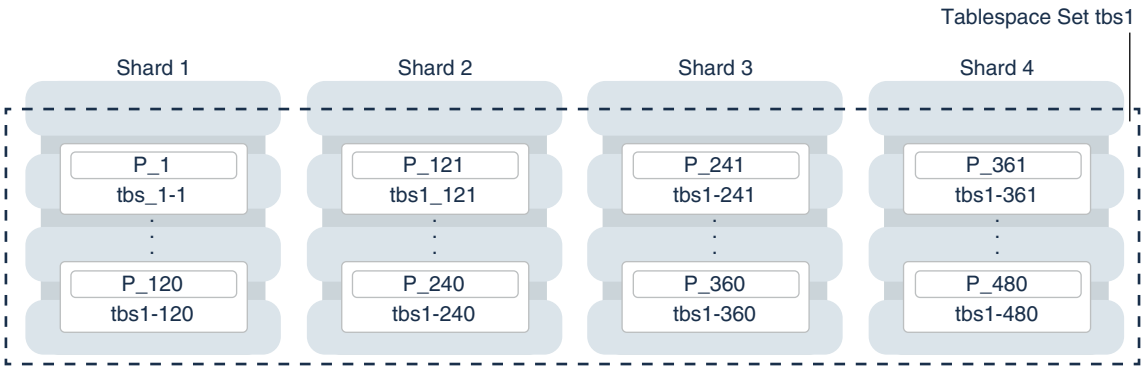
`PARTITIONS AUTO` in this statement means that the number of partitions is automatically set to the number of tablespaces in the tablespace set `ts1` (which is equal to the number of chunks) and each partition will be stored in a separate tablespace.

Each tablespace in a tablespace set belongs to a distinct chunk. In the other words, a chunk can contain only one tablespace from a given tablespace set. However, the same tablespace set can be used for multiple tables that belong to the same table family. In this case, each tablespace in the set will store multiple partitions, one from each table.

Alternatively, each table in a table family can be stored in a separate tablespace set. In this case, a chunk contains multiple tablespaces, one from each tablespace set with each tablespace storing a single partition.

The following figure illustrates the relationship between partitions, tablespaces, and shards for a use case with a single sharded table. In this case, each chunk contains a single tablespace, and each tablespace stores a single partition.

Figure 2-7 System-Managed Sharding



> **Note:**
>
> The sharding method is specified in the `GDSCTL CREATE SHARDCATALOG` command and cannot be changed later.

User-Defined Sharding

User-defined sharding lets you explicitly specify the mapping of data to individual shards. It is used when, because of performance, regulatory, or other reasons, certain data needs to be stored on a particular shard, and the administrator needs to have full control over moving data between shards.

Another advantage of user-defined sharding is that, in case of planned or unplanned outage of a shard, the user knows exactly what data is not available. The disadvantage of user-defined sharding is the need for the database administrator to monitor and maintain balanced distribution of data and workload across shards.

Understanding Shardspaces

A *shardspace* is set of shards that store data that corresponds to a range or list of key values. In user-defined sharding, a shardspace consists of a shard or a set of fully replicated shards. For simplicity, assume that each shardspace consists of a single shard.

Adding Shardspaces to a User-Defined Configuration

Before shards and their CDBs are added to a user-defined sharding configuration, the shardspaces must be created and populated. For example, you can use the following GDSCTL commands:

```
ADD SHARDSPACE -SHARDSPACE east
ADD SHARDSPACE -SHARDSPACE central
ADD SHARDSPACE -SHARDSPACE west
ADD CDB -CONNECT cdb1
ADD CDB -CONNECT cdb2
ADD CDB -CONNECT cdb3
ADD SHARD –CONNECT shard-1 -CDB cdb1 –SHARDSPACE west;
ADD SHARD –CONNECT shard-2 -CDB cdb2 –SHARDSPACE central;
ADD SHARD –CONNECT shard-3 -CDB cdb3 –SHARDSPACE east;
```

Creating Tablespaces for User-Defined Sharding

There is no tablespace set for user-defined sharding. Each tablespace has to be created individually, and explicitly associated with a shardspace.

The following statements can be used to create the tablespaces for each shardspace in the example above.

```
CREATE TABLESPACE tbs1 IN SHARDSPACE west;
CREATE TABLESPACE tbs2 IN SHARDSPACE central;
CREATE TABLESPACE tbs3 IN SHARDSPACE east;
```

Creating Sharded Tables in User-Defined Sharding

With user-defined sharding, a sharded table can be partitioned by range or list. The `CREATE TABLE` syntax for a sharded table is not very different from the syntax for a regular table, except for the requirement that each partition should be stored in a separate tablespace.

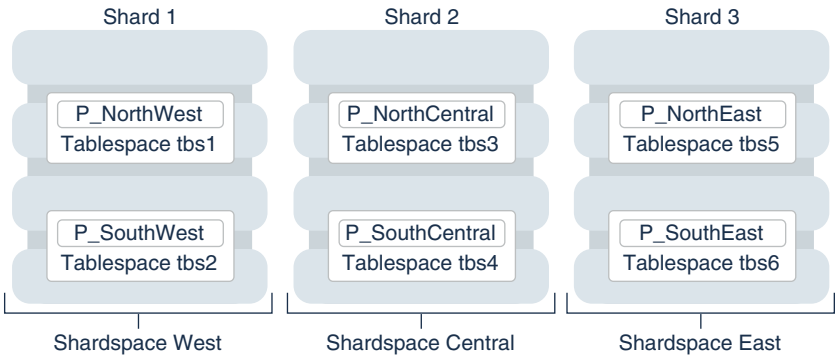
For example:

```
 CREATE SHARDED TABLE accounts
( id             NUMBER
, account_number NUMBER
, customer_id    NUMBER
, branch_id      NUMBER
, state          VARCHAR(2) NOT NULL
, status         VARCHAR2(1)
)
PARTITION BY LIST (state)
( PARTITION p_west VALUES ('OR', 'WA') TABLESPACE ts1
, PARTITION p_central VALUES ('SD', 'WI') TABLESPACE ts2
, PARTITION p_east VALUES ('NY', 'VM', 'NJ') TABLESPACE ts3
```

```
)
;
```

The following figure shows the mapping of partitions to tablespaces, and tablespaces to shards, for the `accounts` table in the previous examples.

Figure 2-8 User-Defined Sharding



Chunk Management in User-Defined Sharding

As with system-managed sharding, tablespaces created for user-defined sharding are assigned to chunks. However, no chunk migration is automatically started when a shard is added to the sharded database. You must run the GDSCTL `MOVE CHUNK` command for each chunk that needs to be migrated.

The total number of chunks is defined by the number of partitions specified in the sharded table. The number of chunks for a given shardspace is the number of partitions assigned to it. The `ALTER TABLE ADD`, `DROP`, `SPLIT`, and `MERGE PARTITION` commands on the sharded table increases or decrease the number of chunks.

The GDSCTL `SPLIT CHUNK` command, which is used to split a chunk in the middle of the hash range for system-managed sharding, is not supported for user-defined sharding. You must use the `ALTER TABLE SPLIT PARTITION` statement to split a chunk.

Replication in User-Defined Sharding

For a user-defined sharded database, two replication schemes are supported:

- Oracle Data Guard
- Oracle Active Data Guard.

User-defined sharding is not supported where Raft replication is used as the replication method.

Directory-Based Sharding

Directory-based sharding allows you to explicitly associate key value with shards dynamically at run time, which gives you fine-grained control over mapping of key values to shards

Compare this with system-managed sharding, which can result in an uneven data distribution, especially when there is a relatively large number of distinct key values (tens to

hundreds of thousands), yet often not large enough for hash-based assignments to achieve uniform data distribution.

Also, compare this with regular user-defined sharding, which is best suited for a small number of static key values that can be specified during schema creation time.

Directory-based Sharding Use Cases

The following use cases illustrate when it would be advantageous to use the directory-based data distribution method in your sharded database.

System sharding results in uneven data distribution

Directory-based sharding can be beneficial when system sharding results in uneven data distribution as the number of distinct key values are not large enough

A typical use case is a B2B application that manages data for a large number of business customer accounts, in the scale of tens of thousands of such accounts.

An example is a dealership application, which hosts and manages data for many dealers. The number of dealerships is in the tens of thousands, which is not large enough to result in even distribution of data with hashing. What's more, the amount of data for different dealerships can be drastically different: some dealers are large operations while others are much smaller, so it is not desirable that we treat them all the same way as in system sharding. There may also be a need to designate different resources/locations for the different dealerships based on application-specific criteria.

Grouping certain key values together into the same location or chunk

Directory-based sharding is useful when you need to group certain key values together into the same location or chunk for affinity purposes, and when needed this group can be moved together in an efficient manner

An example is a social network application, where grouping together customers who often exchange messages on the same shard minimizes the cross-shard traffic. The grouping must be preserved during re-sharding when data is moved between shards. On the other hand, if a member of a group starts communicating more with members of another group, their data must be moved to the appropriate group with minimal impact on the application.

Implement custom policy-based sharding

Directory-based sharding can be used to implement custom policy-based sharding, such as round-robin, random, least data, and so on.

Directory-Based Sharding Concepts and Architecture

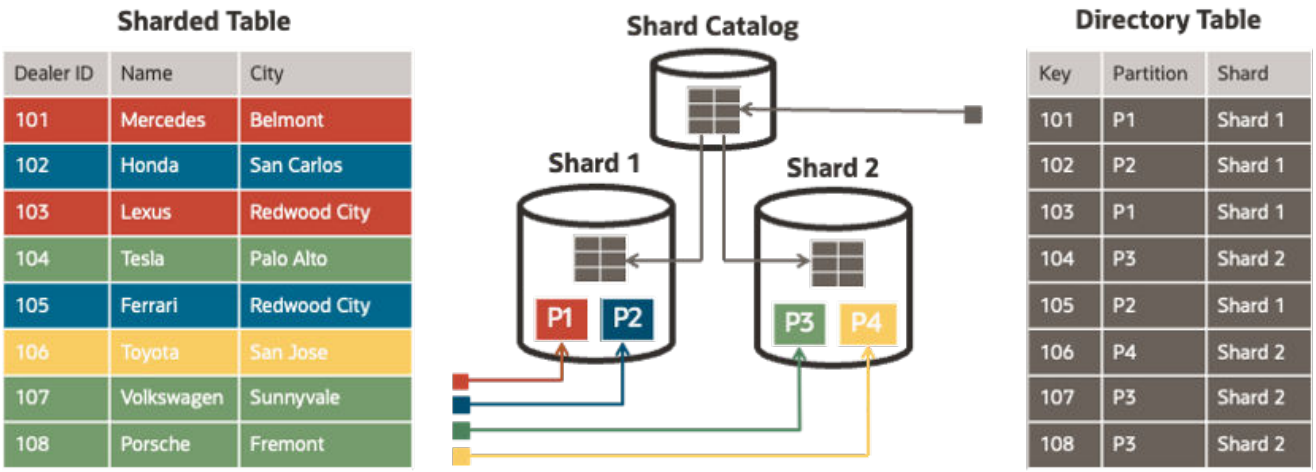
The following are key concepts for understanding directory-based sharding.

- Mapping of key values to partitions and shards is stored in a directory table.
- Directory table is automatically created in the shard catalog and shards when a table sharded by directory is created.
- Shard director (GSM) and client-side connection pools cache the directory for routing purposes. Key values in caches are encrypted.

- Directory is automatically updated when rows are inserted into or deleted from the sharded table for inserts with an auto-assignment rule enabled. Deletes do not auto-delete the mapping in the directory.
- Sharded table contains a virtual column with partition information, which is used for partition pruning.

The following figure shows the key components of directory-based sharding: the **directory table** is hosted on the shard catalog, and is duplicated to all of the shards. The sharded tables are distributed across different shards based on the key/partition mappings in the directory table.

Figure 2-9 Directory-based sharding architecture



Key insert and update operations are performed on the shard catalog, and synchronously duplicated to the shards at commit time.

Client pools fetch the key to chunk/shard mappings from each shard the same way as in other sharding methods. They also subscribe to FAN events that notify them about new key mappings or deletions.

Because directory-based sharding is an enhancement of the user-defined sharding method, see User-Defined Sharding for information about the user-defined sharding method and some examples.

Creating Sharded Tables in Directory-based Sharding

Directory-based sharded tables are created using `PARTITION BY DIRECTORY` in the `CREATE SHARDED TABLE` statement.

For example:

```
CREATE SHARDED TABLE customers
( id             NUMBER NOT NULL
, name           VARCHAR2(30)
, address        VARCHAR2(30)
, status         VARCHAR2(1)
,
CONSTRAINT cust_pk PRIMARY KEY(id)
```

```
)
PARTITION BY DIRECTORY (id)
( PARTITION p1 TABLESPACE tbs1,
  PARTITION p2 TABLESPACE tbs2,
  PARTITION p3 TABLESPACE tbs3…);
```

Note:

- Unlike in user-defined sharding, key values are not specified for the partitions in the CREATE TABLE statement.
- The directory table is automatically created during root table creation. The definition of the directory table is:

  <shard_user_schema>.<root_table>$SDIR

For information about creating objects, deploying, and managing a directory-based sharded database, see Deploying and Managing a Directory-Based Oracle Globally Distributed Database.

Composite Sharding

The composite sharding method allows you to create multiple shardspaces for different subsets of data in a table partitioned by consistent hash. A *shardspace* is set of shards that store data that corresponds to a range or list of key values.

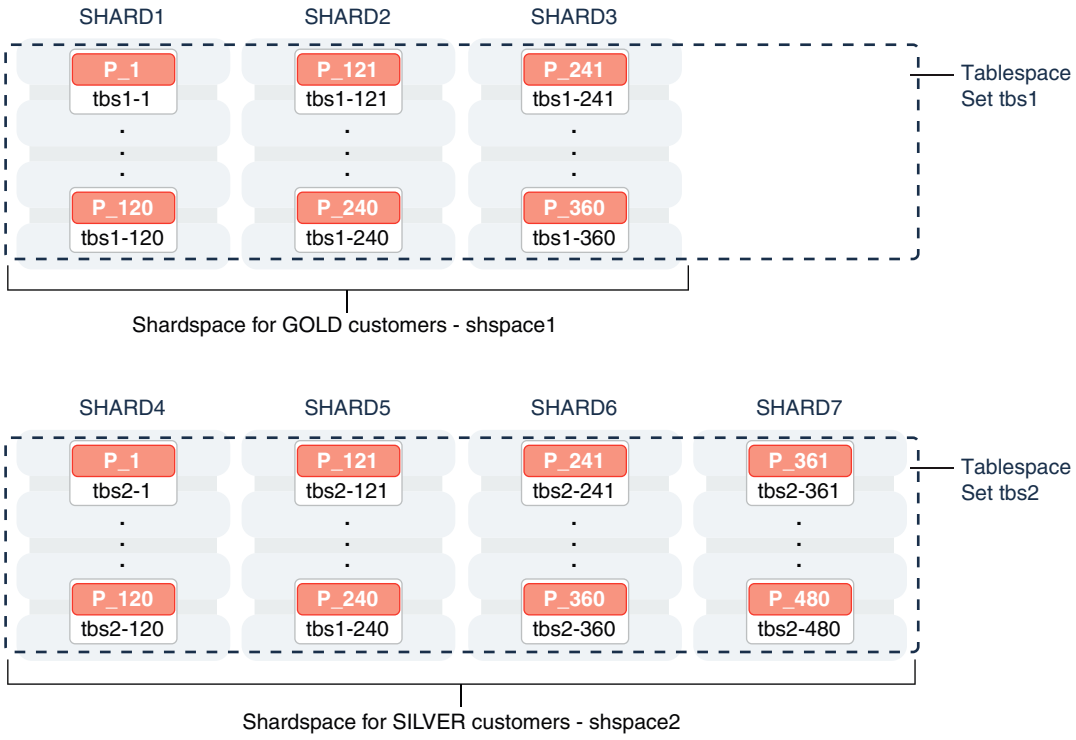
System-managed sharding uses partitioning by consistent hash to randomly distribute data across shards. This provides better load balancing compared to user-defined sharding that uses partitioning by range or list. However, system-managed sharding does not give the user any control on assignment of data to shards.

When sharding by consistent hash on a primary key, there is often a requirement to differentiate subsets of data within a sharded database in order to store them in different geographic locations, allocate to them different hardware resources, or configure high availability and disaster recovery differently. Usually this differentiation is done based on the value of another (non-primary) column, for example, customer location or a class of service.

Composite sharding is a combination of user-defined and system-managed sharding which, when required, provides benefits of both methods. With composite sharding, data is first partitioned by list or range across multiple shardspaces, and then further partitioned by consistent hash across multiple shards in each shardspace. The two levels of sharding make it possible to automatically maintain balanced distribution of data across shards in each shardspace, and, at the same time, partition data across shardspaces.

For example, suppose you want to allocate three shards hosted on faster servers to “gold” customers and four shards hosted on slower machines to “silver” customers. Within each set of shards, customers have to be distributed using partitioning by consistent hash on customer ID.

Figure 2-10 Composite Sharding



Two shardspaces need to be created for such a configuration. For example, you can use the following GDSCTL commands.

```
ADD SHARDSPACE -SHARDSPACE shspace1;
ADD SHARDSPACE -SHARDSPACE shspace2;

ADD CDB -CONNECT cdb1
ADD CDB -CONNECT cdb2
....
ADD CDB -CONNECT cdb7

ADD SHARD -CONNECT shard1 -CDB cdb1 -SHARDSPACE shspace1;
ADD SHARD -CONNECT shard2 -CDB cdb2 -SHARDSPACE shspace1;
ADD SHARD -CONNECT shard3 -CDB cdb3 -SHARDSPACE shspace1;
ADD SHARD -CONNECT shard4 -CDB cdb4 -SHARDSPACE shspace2;
ADD SHARD -CONNECT shard5 -CDB cdb5 -SHARDSPACE shspace2;
ADD SHARD -CONNECT shard6 -CDB cdb6 -SHARDSPACE shspace2;
ADD SHARD -CONNECT shard7 -CDB cdb7 -SHARDSPACE shspace2;
```

With composite sharding, as with the other sharding methods, tablespaces are used to specify the mapping of partitions to shards. To place subsets of data in a sharded table into different shardspaces, a separate tablespace set must be created in each shardspace as shown in the following example.

```
CREATE TABLESPACE SET tbs1 IN SHARDSPACE shspace1;
CREATE TABLESPACE SET tbs2 IN SHARDSPACE shspace2;
```

To store user-defined subsets of data in different tablespaces, Oracle Globally Distributed Database provides syntax to group partitions into sets and associate each set of partitions with a tablespace set. Support for partition sets can be considered a logical equivalent of a higher level of partitioning which is implemented on top of partitioning by consistent hash.

The statement in the following example partitions a sharded table into two partition sets: gold and silver, based on class of service. Each partition set is stored in a separate tablespace. Then data in each partition set is further partitioned by consistent hash on customer ID.

```
CREATE SHARDED TABLE customers
( cust_id NUMBER NOT NULL
, name VARCHAR2(50)
, address VARCHAR2(250)
, location_id VARCHAR2(20)
, class VARCHAR2(3)
, signup_date DATE
, CONSTRAINT cust_pk PRIMARY KEY(cust_id, class)
)
PARTITIONSET BY LIST (class)
  PARTITION BY CONSISTENT HASH (cust_id)
  PARTITIONS AUTO
(PARTITIONSET gold VALUES ('gld') TABLESPACE SET tbs1,
 PARTITIONSET silver VALUES ('slv') TABLESPACE SET tbs2)
;
```

> **Note:**
>
> The sharding method is specified in the `GDSCTL CREATE SHARDCATALOG` command and cannot be changed later.

Using Subpartitions with Sharding

Because Oracle Globally Distributed Database is based on table partitioning, all of the subpartitioning methods provided by Oracle Database are also supported for sharding.

Subpartitioning splits each partition into smaller parts and may be beneficial for efficient parallel processing within a shard, especially in the case of sharding by range or list when the number of partitions per shard may be small.

From a manageability perspective, subpartitioning makes it possible to support the tiered storage approach by putting subpartitions into separate tablespaces and moving them between storage tiers. Migration of subpartitions between storage tiers can be done without sacrificing the scalability and availability benefits of sharding and the ability to perform partition pruning and partition-wise joins on a primary key.

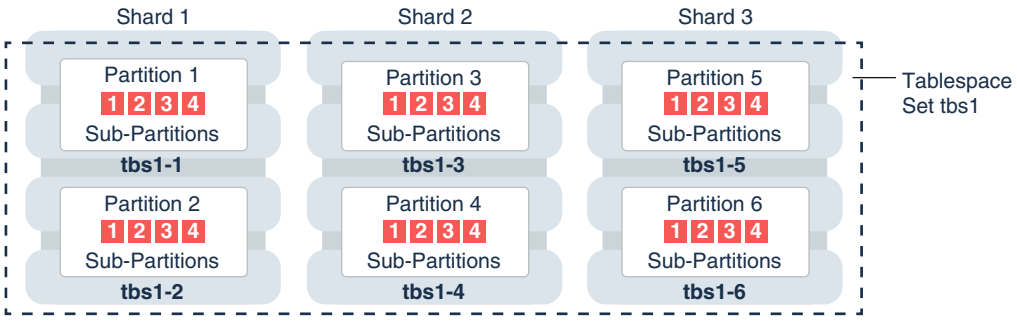
The following example shows system-managed sharding by consistent hash combined with subpartitioning by range.

```
CREATE SHARDED TABLE customers
( cust_id     NUMBER NOT NULL
, name        VARCHAR2(50)
```

```
, address     VARCHAR2(250)
, location_id VARCHAR2(20)
, class       VARCHAR2(3)
, signup_date DATE
, CONSTRAINT cust_pk PRIMARY KEY(cust_id, signup_date)
)
TABLESPACE SET ts1
PARTITION BY CONSISTENT HASH (cust_id)
SUBPARTITION BY RANGE (signup_date)
SUBPARTITION TEMPLATE
( SUBPARTITION per1 VALUES LESS THAN (TO_DATE('01/01/2000','DD/MM/YYYY')),
  SUBPARTITION per2 VALUES LESS THAN (TO_DATE('01/01/2010','DD/MM/YYYY')),
  SUBPARTITION per3 VALUES LESS THAN (TO_DATE('01/01/2020','DD/MM/YYYY')),
  SUBPARTITION future VALUES LESS THAN (MAXVALUE)
)
PARTITIONS AUTO
;
```

The following figure offers a graphical view of the table created by this statement.

Figure 2-11 Subpartitions Stored in the Tablespace of the Parent Partition



In this example each subpartition is stored in the parent partition's tablespace. Because subpartitioning is done by date, it makes more sense to store subpartitions in separate tablespaces to provide the ability to archive older data or move it to a read-only storage. The appropriate syntax is shown here.

```
CREATE SHARDED TABLE customers
( cust_id     NUMBER NOT NULL
, name        VARCHAR2(50)
, address     VARCHAR2(250)
, location_id VARCHAR2(20)
, class       VARCHAR2(3)
, signup_date DATE NOT NULL
, CONSTRAINT cust_pk PRIMARY KEY(cust_id, signup_date)
)
PARTITION BY CONSISTENT HASH (cust_id)
SUBPARTITION BY RANGE(signup_date)
SUBPARTITION TEMPLATE
( SUBPARTITION per1 VALUES LESS THAN (TO_DATE('01/01/2000','DD/MM/YYYY'))
      TABLESPACE SET ts1,
```

```
  SUBPARTITION per2 VALUES LESS THAN (TO_DATE('01/01/2010','DD/MM/
YYYY'))
       TABLESPACE SET ts2,
  SUBPARTITION per3 VALUES LESS THAN (TO_DATE('01/01/2020','DD/MM/
YYYY'))
       TABLESPACE SET ts3,
  SUBPARTITION future VALUES LESS THAN (MAXVALUE)
       TABLESPACE SET ts4
)
PARTITIONS AUTO
;
```

Note that in the case of a database that is not sharded, when tablespaces are specified in the subpartition template it means that subpartition N from every partition is stored in the same tablespace. This is different in case of sharding when subpartitions that belong to the different partitions must be stored in separate tablespaces so that they can be moved in the event of resharding.

Subpartitioning can be used with composite sharding, too. In this case data in a table is organized in three levels: partition sets, partitions, and subpartitions. Examples of the three levels of data organization are shown below.

Specifying subpartition templates per partitionset is not supported to ensure that there is uniformity in the number and bounds of subpartitions across partitionsets. If you need to specify tablespaces for subpartitions per partitionset, you can use the `SUBPARTITIONS STORE IN` clause.

```
CREATE SHARDED TABLE customers
( cust_id     NUMBER NOT NULL
, name        VARCHAR2(50)
, address     VARCHAR2(250)
, location_id VARCHAR2(20)
, class       VARCHAR2(3) NOT NULL
, signup_date DATE NOT NULL
, CONSTRAINT cust_pk PRIMARY KEY(cust_id, class, signup_date)
)
PARTITIONSET BY LIST (class)
PARTITION BY CONSISTENT HASH (cust_id)
SUBPARTITION BY RANGE (signup_date)
  SUBPARTITION TEMPLATE /* applies to both SHARDSPACEs */
  ( SUBPARTITION per1 VALUES LESS THAN (TO_DATE('01/01/2000','DD/MM/
YYYY'))
  , SUBPARTITION per2 VALUES LESS THAN (TO_DATE('01/01/2010','DD/MM/
YYYY'))
  , SUBPARTITION per3 VALUES LESS THAN (TO_DATE('01/01/2020','DD/MM/
YYYY'))
  , SUBPARTITION future VALUES LESS THAN (MAXVALUE)
)
PARTITIONS AUTO
(
  PARTITIONSET gold   VALUES ('gld') TABLESPACE SET tbs1
 subpartitions store in(tbs1)
, PARTITIONSET silver VALUES ('slv') TABLESPACE SET tbs2
 subpartitions store in(tbs2)
```

```
)
;
```

Replication in Oracle Globally Distributed Database

Topics:

- Replication in Oracle Globally Distributed Database
- Using Oracle Data Guard with a Sharded Database
- Using Raft Replication in Oracle Globally Distributed Database

Replication in Oracle Globally Distributed Database

In addition to it's own built-in, Raft replication feature, Oracle Globally Distributed Database is tightly integrated with Oracle Data Guard.

Replication provides high availability, disaster recovery, and additional scalability for reads. A unit of replication can be a shard, a part of a shard, or a group of shards.

Replication topology in a sharded database is declaratively specified using GDSCTL command syntax. You can choose either Oracle Data Guard or Raft replication to replicate your data. Oracle Globally Distributed Database automatically deploys the specified replication topology to the procured systems, and enables data replication.

Shard-level Replication

The availability of a sharded database is not affected by an outage or slowdown of one or more shards. Oracle Active Data Guard replication is used to provide individual shard-level high availability. Replication is automatically configured and deployed when the sharded database is created.

Optionally, you can use Oracle RAC for shard-level high availability, complemented by replication, to maintain shard-level data availability in the event of a cluster outage. Oracle Globally Distributed Database automatically fails over database connections from a shard to its replica in the event of an unplanned outage.

Raft Replication

Instead of replication at the shard level, the Raft replication feature in Oracle Globally Distributed Database creates smaller replication units and distributes them automatically to handle chunk assignment, chunk movement, workload distribution, and balancing upon scaling (addition or removal of shards), including planned or unplanned shard availability changes.

Unlike Data Guard replication, Raft replication does not need to be reconfigured when shards are added or removed, and replicas do not need to be actively managed.

Using Oracle Data Guard with a Sharded Database

Oracle Data Guard replication maintains one or more synchronized copies (standbys) of a shard (the primary) for high availability and data protection. Standbys may be deployed locally or remotely, and when using Oracle Active Data Guard can also be open for read-only access.

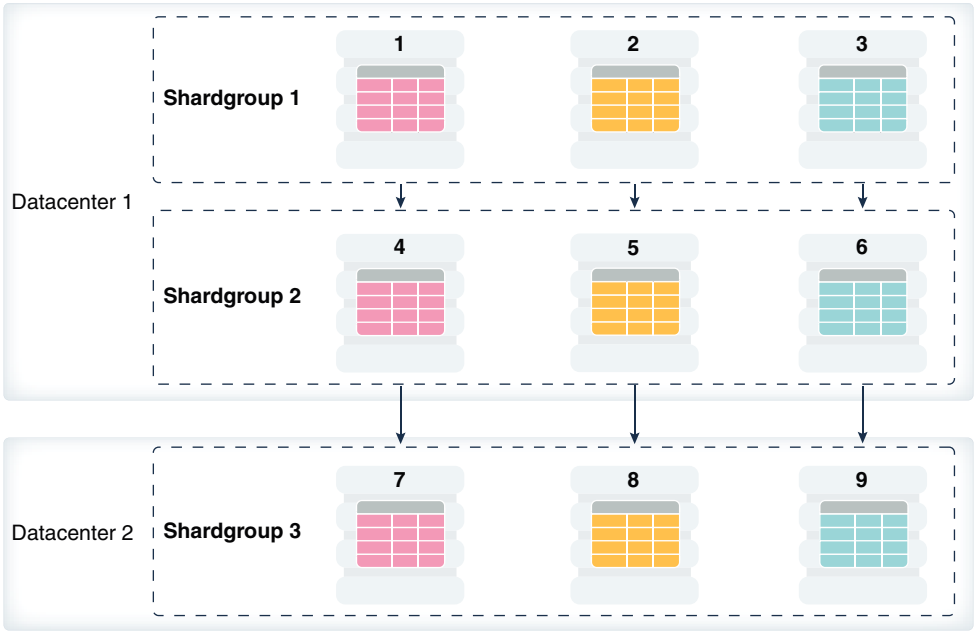
Oracle Data Guard can be used as the replication technology for sharded databases using the system-managed, user-defined, or composite method of sharding.

Using Oracle Data Guard with a System-Managed Sharded Database

In system-managed and composite sharding, the logical unit of replication is a group of shards called a *shardgroup*. In system-managed sharding, a shardgroup contains all of the data stored in the sharded database. The data is sharded by consistent hash across shards that make up the shardgroup. Shards that belong to a shardgroup are usually located in the same data center. An entire shardgroup can be fully replicated to one or more shardgroups in the same or different data centers.

The following figure illustrates how Data Guard replication is used with system-managed sharding. In the example in the figure there is a primary shardgroup, Shardgroup 1, and two standby shardgroups, Shardgroup 2 and Shardgroup 3. Shardgroup 1 consists of Data Guard primary databases (shards 1-3). Shardgroup 2 consists of local standby databases (shards 4-6) which are located in the same data center and configured for synchronous replication. And Shardgroup 3 consists of remote standbys (shards 7-9) located in a different data center and configured for asynchronous replication. Oracle Active Data Guard is enabled in this configuration, so each standby is open read-only.

Figure 2-12 System-Managed Sharding with Data Guard Replication



The concept of shardgroup as a logical unit of replication hides from the user the implementation details of replication. With Data Guard, replication is done at the shard (database) level. The sharded database in the figure above consists of three sets of replicated shards: {1, 4, 7}, {2, 5, 8} and {3, 6, 9}. Each set of replicated shards is managed as a Data Guard Broker configuration with fast-start failover (FSFO) enabled.

To deploy replication, specify the properties of the shardgroups (region, role, and so on) and add shards to them. Oracle Globally Distributed Database automatically

configures Data Guard and starts an FSFO observer for each set of replicated shards. It also provides load balancing of the read-only workload, role based global services and replication lag, and locality based routing.

For high availability, Data Guard standby shards can be placed in the same region where the primary shards are placed. For disaster recovery, the standby shards can be located in another region.

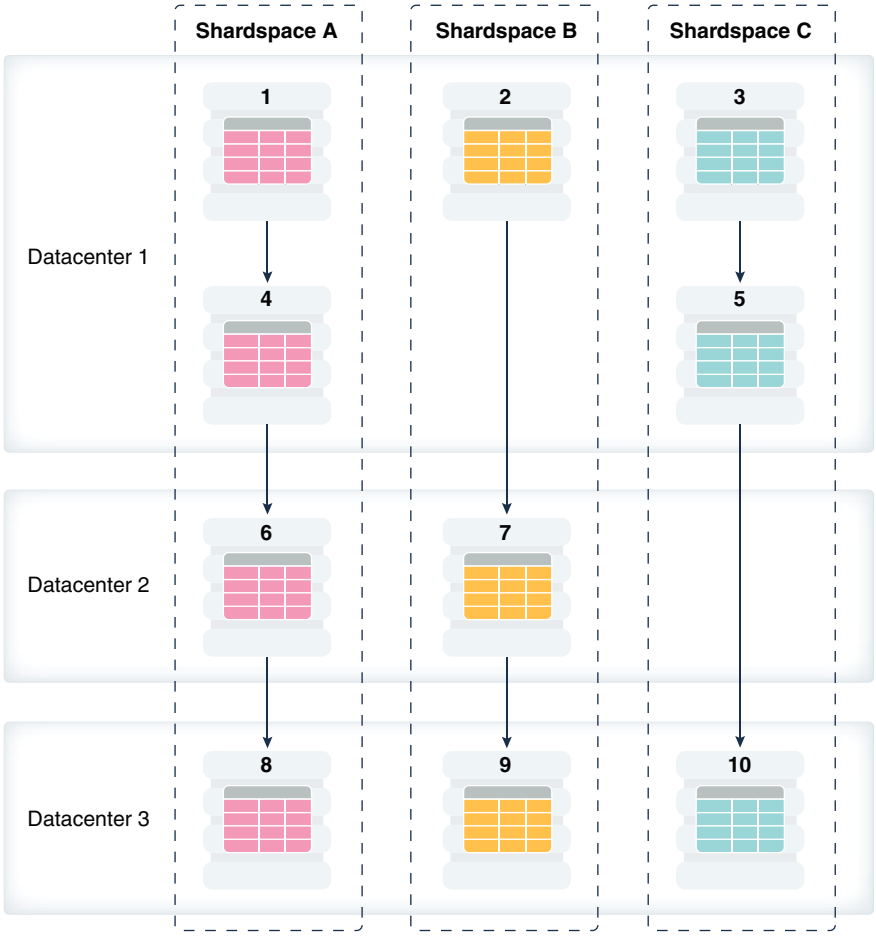
Run the following GDSCTL commands to deploy the example configuration shown in the figure above.

```
CREATE SHARDCATALOG -database host00:1521:shardcat -region dc1,dc2

ADD GSM -gsm gsm1 -listener 1571 -catalog host00:1521:shardcat -region dc1
ADD GSM -gsm gsm2 -listener 1571 -catalog host00:1521:shardcat -region dc2
START GSM -gsm gsm1
START GSM -gsm gsm2

ADD SHARDGROUP -shardgroup shardgroup1 -region dc1 -deploy_as primary
ADD SHARDGROUP -shardgroup shardgroup2 -region dc1 -deploy_as active_standby
ADD SHARDGROUP -shardgroup shardgroup3 -region dc2 -deploy_as active_standby

ADD CDB -connect cdb1
ADD CDB -connect cdb2
...
ADD CDB -connect cdb9

ADD SHARD -connect shard1 -CDB cdb1 -shardgroup shardgroup1
ADD SHARD -connect shard2 -CDB cdb2 -shardgroup shardgroup2
...
ADD SHARD -connect shard9 -CDB cdb9 -shardgroup shardgroup3

DEPLOY
```

Using Oracle Data Guard with a User-Defined Sharded Database

With user-defined sharding the logical (and physical) unit of replication is a shard. Shards are not combined into shardgroups. Each shard and its replicas make up a *shardspace* which corresponds to a single Data Guard Broker configuration. Replication can be configured individually for each shardspace. Shardspaces can have different numbers of standbys which can be located in different data centers. An example of user-defined sharding with Data Guard replication is shown in the following figure.

Figure 2-13 User-Defined Sharding with Data Guard Replication



Run the following GDSCTL commands to deploy the example configuration shown in the figure above.

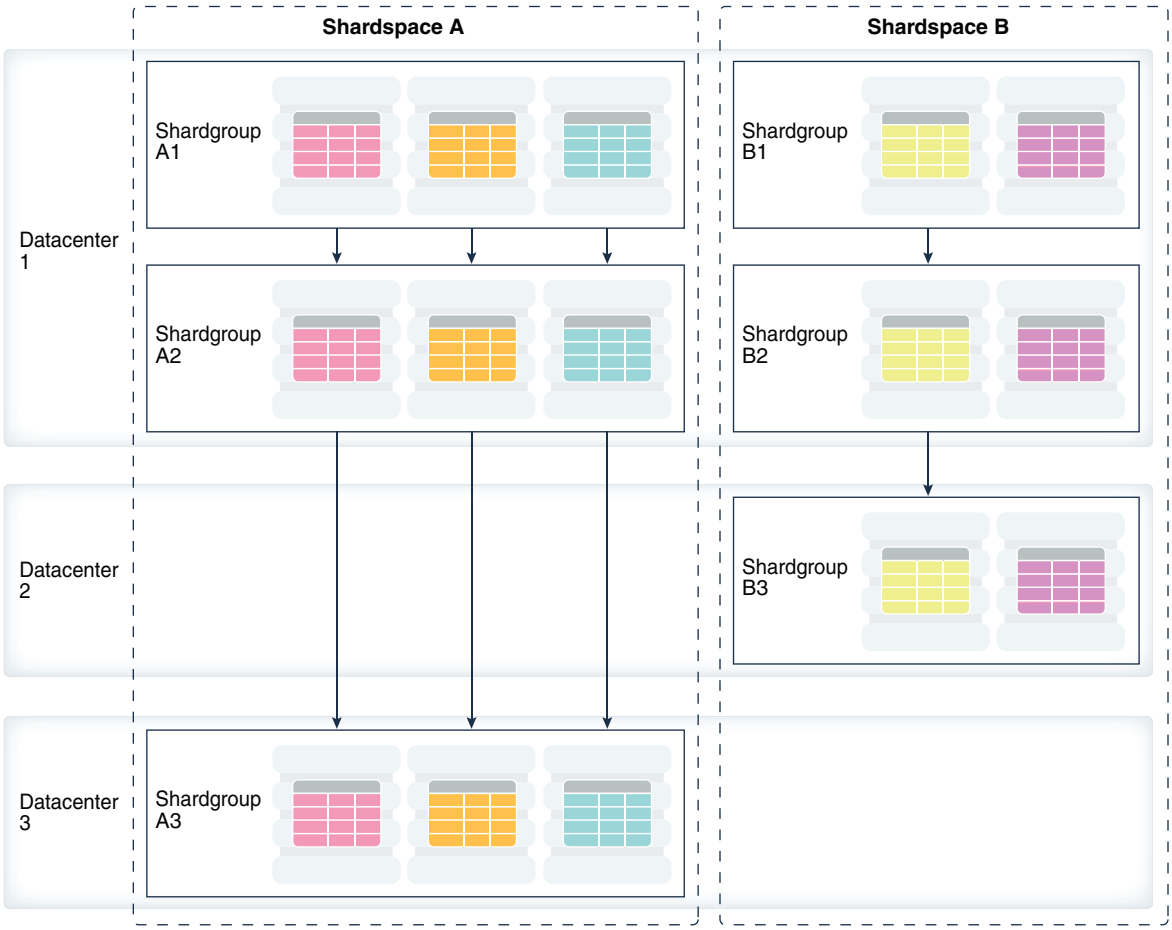
```
CREATE SHARDCATALOG -sharding user -database host00:1521:cat -region
dc1,dc2,dc3

ADD GSM -gsm gsm1 -listener 1571 -catalog host00:1521:cat -region dc1
ADD GSM -gsm gsm2 -listener 1571 -catalog host00:1521:cat -region dc2
ADD GSM -gsm gsm3 -listener 1571 -catalog host00:1521:cat -region dc3
START GSM -gsm gsm1
START GSM -gsm gsm2
START GSM -gsm gsm3

ADD SHARDSPACE -shardspace shardspace_a
ADD SHARDSPACE -shardspace shardspace_b
ADD SHARDSPACE -shardspace shardspace_c

ADD CDB -connect cdb1
ADD CDB -connect cdb2
...
ADD CDB -connect cdb10
```

```
ADD SHARD -connect shard1 -CDB cdb1 -shardspace shardspace_a
ADD SHARD -connect shard2 -CDB cdb2 -shardspace shardspace_b
...
ADD SHARD -connect shard10 -CDB cdb10 -shardspace shardspace_c

DEPLOY
```

Using Oracle Data Guard with a Composite Sharded Database

In composite sharding, similar to user-defined sharding, a sharded database consists of multiple shardspaces. However, each shardspace, instead of replicated shards, contains replicated shardgroups.

Figure 2-14 Composite Sharding with Data Guard Replication



Run the following GDSCTL commands to deploy the example configuration shown in the figure above.

```
CREATE SHARDCATALOG -sharding composite -database host00:1521:cat -region
dc1,dc2,dc3

ADD GSM -gsm gsm1 -listener 1571 -catalog host00:1521:cat -region dc1
```

```
ADD GSM -gsm gsm2 -listener 1571 -catalog host00:1521:cat -region dc2
ADD GSM -gsm gsm3 -listener 1571 -catalog host00:1521:cat -region dc3
START GSM -gsm gsm1
START GSM -gsm gsm2
START GSM -gsm gsm3

ADD SHARDSPACE -shardspace shardspace_a
ADD SHARDSPACE -shardspace shardspace_b

ADD SHARDGROUP -shardgroup shardgroup_a1 -shardspace shardspace_a -
region dc1
-deploy_as primary
ADD SHARDGROUP -shardgroup shardgroup_a2 -shardspace shardspace_a -
region dc1
-deploy_as active_standby
ADD SHARDGROUP -shardgroup shardgroup_a3 -shardspace shardspace_a -
region dc3
-deploy_as active_standby
ADD SHARDGROUP -shardgroup shardgroup_b1 -shardspace shardspace_b -
region dc1
-deploy_as primary
ADD SHARDGROUP -shardgroup shardgroup_b2 -shardspace shardspace_b -
region dc1
-deploy_as active_standby
ADD SHARDGROUP -shardgroup shardgroup_b3 -shardspace shardspace_b -
region dc2
-deploy_as active_standby

ADD CDB -connect cdb1
ADD CDB -connect cdb2
...

ADD SHARD -connect shard1 -cdb cdb1 -shardgroup shardgroup_a1
ADD SHARD -connect shard2 -cdb cdb2 -shardgroup shardgroup_a2
...

DEPLOY
```

Considerations

If a single-instance primary fails over to its standby, unlike Oracle RAC, you must intervene to reinstate the old primary by starting the database in mount state. The broker will then automatically complete the reinstatement.

Using Raft Replication in Oracle Globally Distributed Database

Oracle Globally Distributed Database provides built-in fault tolerance with **Raft replication**, a capability that integrates data replication with transaction execution in a sharded database.

Raft replication enables fast automatic failover with zero data loss. If all shards are in the same data center, it is possible to achieve sub-second failover. Raft replication is active/active; each shard can process reads and writes for a subset of data. This capability provides a uniform configuration with no primary or standby shards.

Raft replication is integrated and transparent to applications. There is no need to configure and manage Oracle Data Guard or Oracle GoldenGate to achieve high availability. Raft replication automatically reconfigures replication in case of shard host failures or when shards are added or removed from the sharded database.

When Raft replication is enabled, a sharded database contains multiple replication units. A replication unit (RU) is a set of chunks that have the same replication topology. Each RU has multiple replicas placed on different shards. The Raft consensus protocol is used to maintain consistency between the replicas in case of failures, network partitioning, message loss, or delay.

Replication Unit

When Raft replication is enabled, a sharded database contains multiple **replication units**. A replication unit (RU) is a set of chunks that have the same replication topology. Each RU has three replicas placed on different shards. The Raft consensus protocol is used to maintain consistency between the replicas in case of failures, network partitioning, message loss, or delay.

Each shard contains replicas from multiple RUs. Some of these replicas are leaders and some are followers. Raft replication tries to maintain a balanced distribution of leaders and followers across shards. By default each shard is a leader for two RUs and is a follower for four other RUs. This makes all shards active and provides optimal utilization of hardware resources.

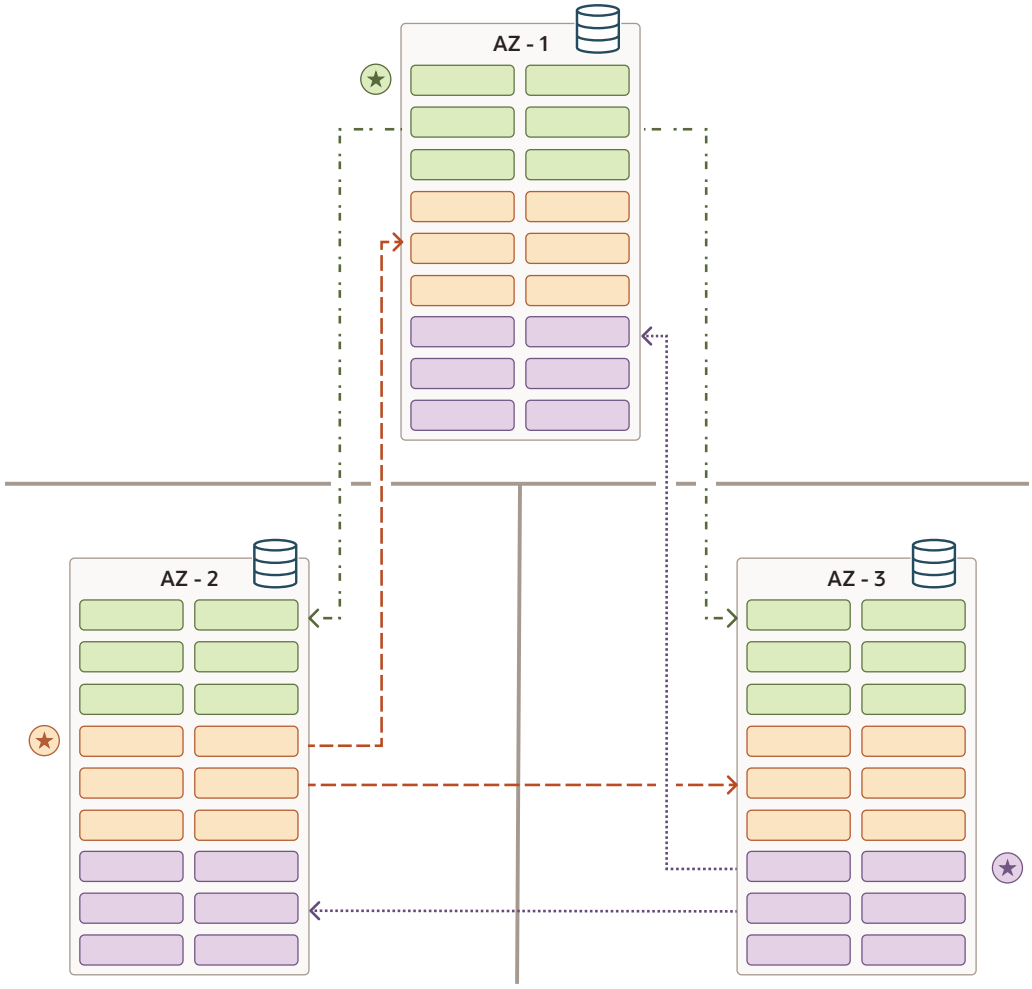
In Oracle Globally Distributed Database, an RU is a set of chunks, as shown in the image below.

The diagram above illustrates the relationship among shards, chunk sets, and chunks. A *shard* contains a set of *chunks*. A *chunk* is a set of table partitions in a given table family. A chunk is a unit of resharding (data movement across shards). A set of chunks which have the same replication topology is called *chunk set*.

Raft Group

Each replication unit contains exactly one chunk set and has a **leader** and a set of **followers**, and these members form a *raft group*. The leader and its followers for a replication unit contain replicas of the same chunk set in different shards as shown below. A shard can be the leader for some replication units and a follower for other replication units.

All DMLs for a particular subset of data are executed in the leader first, and then are replicated to its followers.



In the image above, a leader in each shard, indicated by the set of chunks in one color with a star next to it, points to a follower (of the same color) on each of the two other shards.

Replication Factor

The **replication factor** (RF) determines the number of participants in a RAFT group. This number includes the leader and its followers.

The RU needs a majority of replicas available for write.

- RF = 3: tolerates one replica failure
- RF = 5: tolerates two replica failures

In Oracle Globally Distributed Database , the replication factor is specified for the entire sharded database, that is all replication units in the database have the same RF. The number of followers is limited to two, thus the replication factor is three.

Raft Log

Each RU is associated with a set of *Raft logs* and OS processes that maintain the logs and replicate changes from the leader to followers. This allows multiple RUs to operate independently and in parallel within a single shard and across multiple shards. It also makes it possible to scale the replication up and down by changing the number of RUs.

Changes to data made by a DML are recorded in the Raft log. A commit record is also recorded at the end of each user transaction. Raft logs are maintained independently from redo logs and contain logical changes to rows. Logical replication reduces failover time because followers are open to incoming transactions and can quickly become the leader.

The Raft protocol guarantees that followers receive log records in the same order they are generated by the leader. A user transaction is committed on the leader as soon as half of the followers acknowledge the receipt of the commit record and writes it to the Raft log.

Transactions

On a busy system, multiple commits are acknowledged at the same time. The synchronous propagation of transaction commit records provides zero data loss. The application of DML change records to followers, however, is done asynchronously to minimize the impact on transaction latency.

Leader Election Process

Per Raft protocol, if followers do not receive data or heartbeat from the leader for a specified period of time, then a new leader election process begins.

The default heartbeat interval is 150 milliseconds, with randomized election timeouts (up to 150 milliseconds) to prevent multiple shards from triggering elections at the same time, leading to split votes.

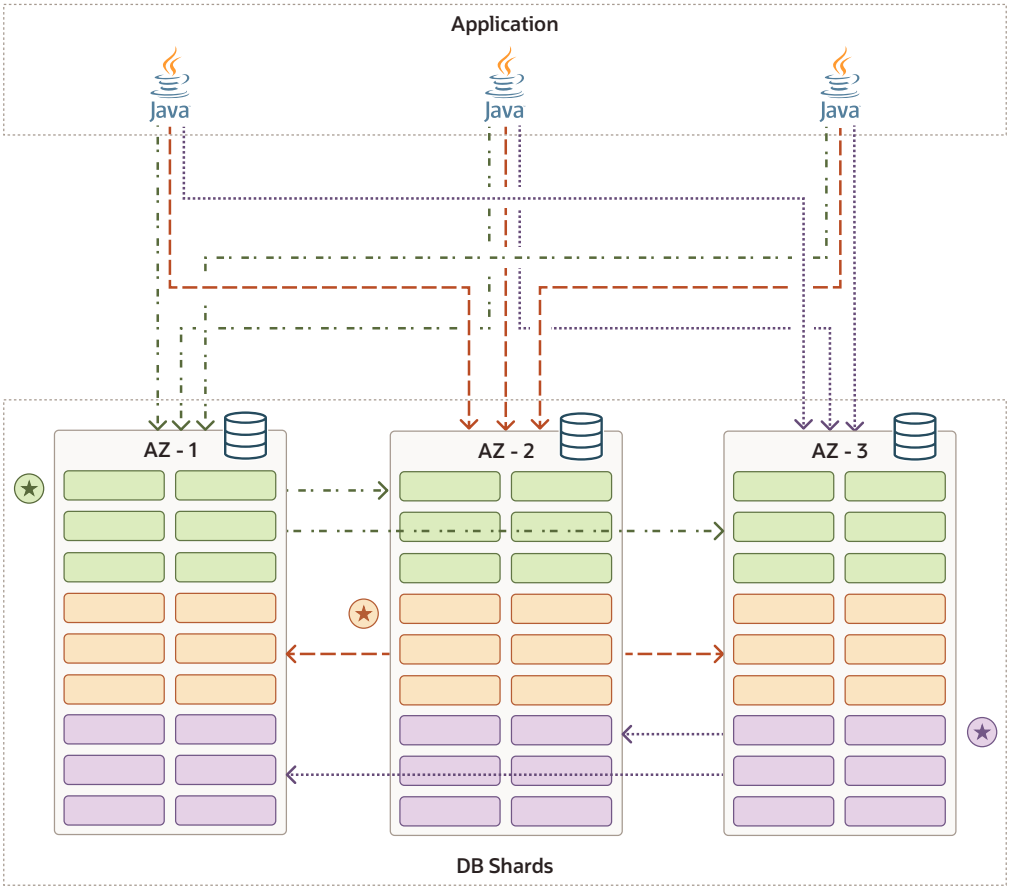
Node Failure

Node failure and recovery are handled in an automated way with minimal impact on the application.

The failover time is sub-3 seconds with less than 10 millisecond network latencies between Availability Zones. This includes failure detection, shard failover, change of leadership, application reconnecting to new leader, and continuing business transactions as before.

The impact of the failure on the application can further be abstracted by configuring retries in JDBC driver and end customer experience will be that a particular request took longer rather than getting an error.

The following is an illustration of a sharded database with all three shards in a healthy state. Applications requests are able to reach all three shards, and replication between the leaders and followers is ongoing between the shards.



Leader Node Failure

When the leader for a replication unit becomes unavailable, followers will initiate a new leader election process using the Raft protocol.

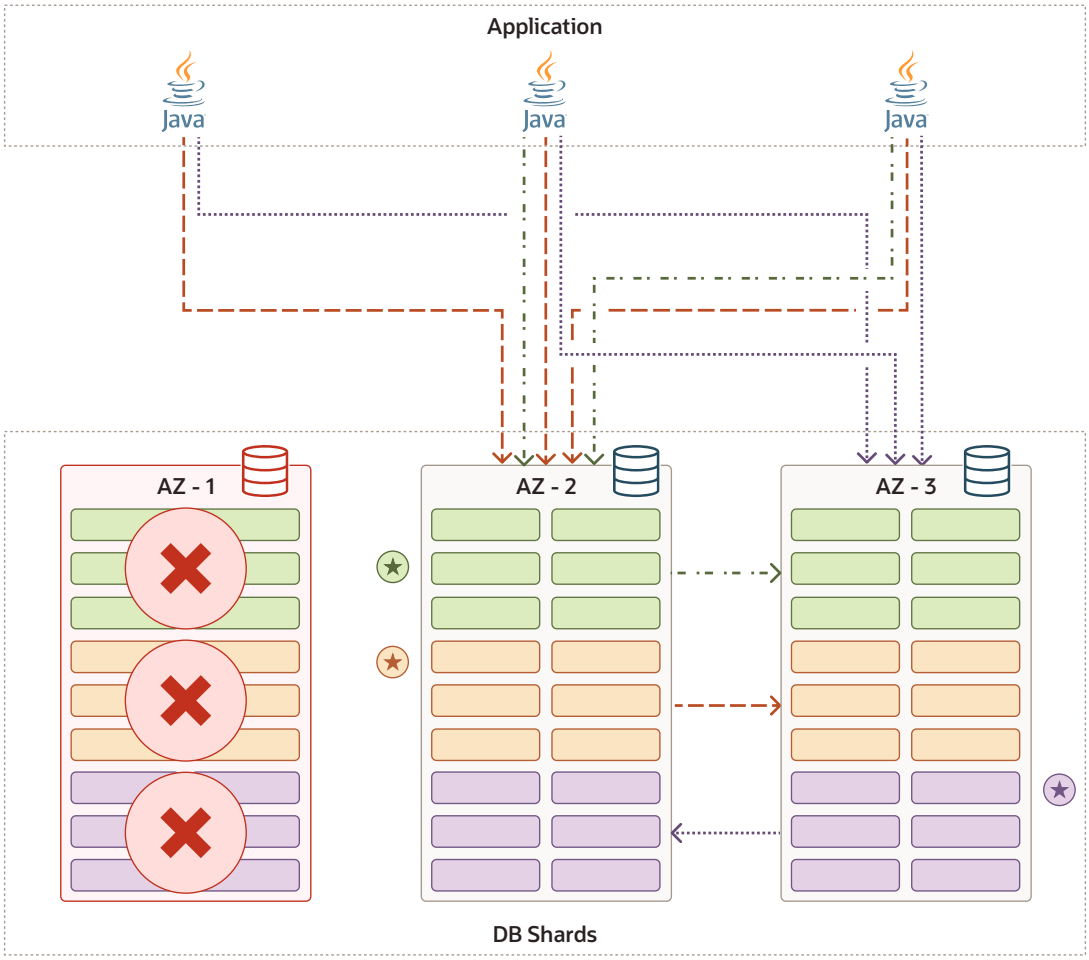
As long as a majority of the nodes (quorum) are still healthy, the Raft protocol will ensure that a new leader is elected from the available nodes.

When one of the followers succeeds in becoming the new leader, proactive notifications are sent from the shard to the client driver of leadership change. The client driver starts routing the request to the new leader shard. Routing clients (such as UCP) are notified using ONS notifications to update their shard and chunk mapping, ensuring that they route traffic to the leader.

During this failover and reconnection period, the application could be configured to wait and retry with the retry interval and retry counts settings at the JDBC driver configuration. These are very similar to the present RAC instance failover configuration.

Upon connecting to new leader, the application will continue to function as before.

The following diagram shows that the first shard failed, and that a new leader for the replication unit whose leader was once on that first shard has been replaced by a new leader in the second shard.



Failback

When the original leader comes back online after a failure, it first tries to identify the current leader and attempts to rejoin the cluster as a follower. Once the failed shard rejoins the cluster, it asks the leader for logs based on its current index in order to sync up with the leader.

Raft log retention is configurable using the *_shard_size_per_raft_logfile_gb* parameter. Once it synchronizes, it will continue to stay as a follower.

Leadership rebalancing can be done by calling the API `SWITCHOVER RU -REBALANCE`, which could also be scripted if needed.

If there are not enough Raft logs available on the present leader, the follower will have to be repopulated from one of the good followers using data copy API (`COPY RU`).

Follower Node Failure

If a follower node becomes unavailable, the leader's attempts to replicate the Raft log to that follower will fail.

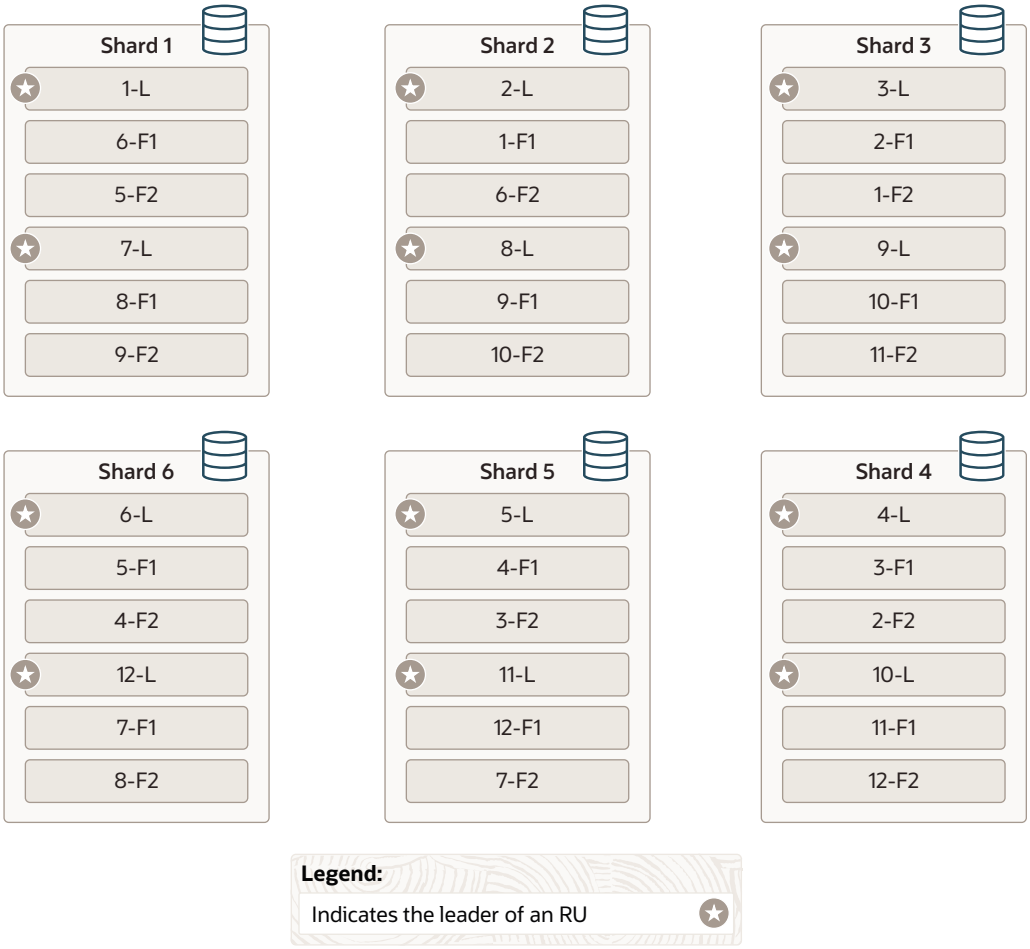
The leader will attempt to reach the failed follower indefinitely until the failed follower rejoins or a new follower replaces it.

If needed, a new follower will have to be created and added to the cluster, and its data needs to be synchronized from a good follower as explained above.

Example Raft Replication Deployment

The following diagram shows a simple sharded database Raft replication deployment with 6 shards, 12 replication units, and replication factor = 3.

Each shard has two leaders and 4 followers. Each member is labeled with an RU number and the suffix indicates whether it is a leader (-L) or follower (-Fn). The leaders are also indicated by a star. In this configuration, two shards can take over the load of a failed shard.



To configure this deployment, at the shard catalog creation step, you specify `GDSCTL CREATE SHARDCATALOG -repl` **`NATIVE`**. You can specify the replication factor (`-repfactor`) in the `GDSCTL CREATE SHARDCATALOG` or `ADD SHARDGROUP` commands. Similar to the specification of chunks, you specify the number of replication units in `GDSCTL CREATE SHARDCATALOG` or `ADD SHARDSPACE` commands.

Configuring and Managing Raft Replication

Oracle Globally Distributed Database provides commands and options in the GDSCTL CLI to enable and manage Raft replication in a system-managed sharded database.

See Raft Replication Configuration and Management for topics about configuring and managing Raft replication.

> **Note:**
>
> Raft replication is not supported for user-defined and composite data distribution methods in Oracle Globally Distributed Database.

Client Application Request Routing

To route a client application request directly to a shard, you connect to the shard using the Oracle drivers and provide a sharding key with the request.

About Sharding Keys

All database requests that require high performance and fault isolation must only access data associated with a single value of the sharding key. The application must provide the sharding key when establishing a database connection. If this is the case, the request is routed directly to the appropriate shard.

Multiple requests can be processed in the same session as long as they all are related to the same sharding key. Such transactions typically access 10s or 100s of rows. Examples of single-shard transactions include order entry, lookup and update of a customer's billing record, and lookup and update of a subscriber's documents.

Database requests that must access data associated with multiple values of the sharding key, or for which the value of the sharding key is unknown, must be issued from the query coordinator which orchestrates parallel processing of the query across multiple shards.

About Oracle Connection Drivers

At run time, connection pools act as shard directors by routing database requests across pooled connections. Oracle Database supports connection-pooling in data access drivers such as OCI, JDBC, and ODP.NET. These drivers can recognize sharding keys specified as part of a connection request. Similarly, the Oracle Universal Connection Pool (UCP) for JDBC clients can recognize sharding keys specified in a connection URL. Oracle UCP also enables non-Oracle application clients such as Apache Tomcat and WebSphere to work with Oracle Globally Distributed Database.

Oracle clients use UCP cache routing information to directly route a database request to the appropriate shard, based on the sharding keys provided by the application. Such data-dependent routing of database requests eliminates an extra network hop, decreasing the transactional latency for high volume applications.

Routing information is cached during an initial connection to a shard, which is established using a shard director. Subsequent database requests for sharding keys within the cached range are routed directly to the shard, bypassing the shard director.

Like UCP, a shard director can process a sharding key specified in a connect string and cache routing information. However, UCP routes database requests using an already established connection, while a shard director routes connection requests to a shard. The routing cache automatically refreshes when a shard becomes unavailable or changes occur to the sharding topology. For high-performance, data-dependent routing, Oracle recommends using a connection pool when accessing data in the sharded database.

Separate connection pools must be used for direct routing and routing requests through the query coordinator. For direct routing, separate global services must be created for read-write and read-only workloads. This is true only if Data Guard replication is used. For proxy routing, use the `GDS$CATALOG` service on the shard catalog database.

Query Processing and the Query Coordinator

The **query coordinator** is part of the shard catalog. The query coordinator provides query processing support for the sharded database. With its access to the sharded database topology metadata in the shard catalog, there are three general cases in which the query coordinator plays an important part.

1. Single Shard Queries with No Sharding Key

   If a sharding key is not passed from the application, the query coordinator figures out which shard contains the data required by the query and sends the query there for processing.

2. Multi-Shard Queries

   The query coordinator can also assist with queries that need data from more than one shard, called **multi-shard queries**, for example `SELECT COUNT(*) FROM Customer`.

3. Aggregate Queries

   The query coordinator handles aggregate queries typically used in reporting, such as aggregates on sales data.

In every case, the query coordinator's SQL compiler identifies the relevant shards automatically and coordinates the query processing across all of the participating shards.

In a single-shard query scenario, the entire query is processed on the single participating shard, and the query coordinator just passes processed rows back to the client.

For a multi-shard query the SQL compiler analyzes and rewrites the query into query fragments that are sent and processed by the participating shards. The queries are rewritten so that most of the query processing is done on the participating shards and then aggregated by the coordinator.

The query coordinator uses Oracle Database's parallel query engine to optimize and push multi-shard queries in parallel to the shards. Each shard processes the query on the subset of data that it has. Then the results are returned back to the query coordinator, which sends them back to the client.

In essence, the shards act as compute nodes for the queries issued by the query coordinator. Because the computation is pushed to the data, there is reduced movement of data between shards and the coordinator. This arrangement also enables the effective use of resources by offloading processing from the query coordinator on to the shards as much as possible.

Specifying Consistency Levels

You can specify different consistency levels for multi-shard queries. For example, you might want some queries to avoid the cost of SCN synchronization across shards, and these shards could be globally distributed. Another use case is when you use standbys for replication and slightly stale data is acceptable for multi-shard queries, as the results could be fetched from the primary and its standbys. A multi-shard query must maintain global read consistency (CR) by issuing the query at the highest common SCN across all the shards.

High Availability and Performance

It is highly recommended that the query coordinator be protected with Oracle Data Guard in Maximum Availability protection mode (zero data loss failover) with fast-start failover enabled. The query coordinator may optionally be Oracle RAC-enabled for additional availability and scalability. To improve the scalability and availability of multi-shard query workloads, Oracle Active Data Guard standby shard catalog databases in read-only mode can act as multi-shard query coordinators.

In aggregation use cases and issuing SQL without a sharding key, you will experience a reduced level of performance compared with direct, key-based, routing.

3

Security in an Oracle Globally Distributed Database Environment

Using TCPS Protocol and Transport Layer Security

To secure the communication between the various Oracle Globally Distributed Database components in a distributed environment, Oracle recommends that you use Oracle Database Native Network Encryption or the TCPS protocol and Transport Layer Security (TLS) for all connections to, and between, the shard catalog and shards. For information about this security feature, see

Configuring Oracle Database Native Network Encryption and Data Integrity

Configuring Secure Sockets Layer Authentication.

Using Wallets with Oracle Globally Distributed Database

Beginning with Oracle Database 21c, Oracle wallets created for an Oracle Globally Distributed Database sharded database are an important part of any deployment. All primary databases and their replicas within the sharded database configuration must have a sharding-specific wallet file present to ensure proper operation.

These wallets are created during the deployment of a sharded database and enable encrypted data to be sent between the shard catalog and individual shards. The process by which the wallets are created establishes a trust relationship between the different components of a sharded database deployment and prevents unauthorized operations from occurring on a shard.

The wallets themselves are created on the shard catalog and any shard catalog replicas when the GDSCTL command `CREATE SHARDCATALOG` is issued, and the wallets are created on the shards when the GDSCTL command `DEPLOY` is issued.

After a successful deployment, the wallet files contain information needed for shard catalogs and shards to connect to one another to perform operations such as DDL processing, user context propagation, and the passing of other sensitive data. The information stored in the wallet includes sharding-specific encryption and decryption keys, connect strings, and encrypted passwords. Any command issued from GDSCTL or SQL*Plus which changes this data will automatically cause the wallet to be updated with the new information.

Compatibility and Migration from Oracle Database 19c

For existing Oracle Globally Distributed Database configurations which are being upgraded from a previous Oracle Database release, perform the steps in Post-Upgrade Steps for Oracle Globally Distributed Database 21c after the database upgrade.

Locating the Wallet

The location of the wallet files is under the directory specified by the `wallet_root` database initialization parameter. If `wallet_root` is not set before issuing `CREATE SHARDCATALOG` or

DEPLOY, then wallet_root is set to $ORACLE_BASE/admin/*db_unique_name* on the shard catalog or shards, respectively.

For example, assume the following after logging into the shard catalog or into a shard.

```
SQL> select guid from v$pdbs where con_id =
sys_context('userenv','con_id');

GUID
--------------------------------
C23E7C78D5B77D50E0537517C40ACE4A

SQL> select value from v$parameter where name='wallet_root';

VALUE
------------------------------------------------------------------------
---------
your-path-to-keystore
```

Given these values, the sharding-specific wallet file name is *your-path-to-keystore*/C23E7C78D5B77D50E0537517C40ACE4A/shard/cwallet.sso.

Wallets on Shard Catalog Replicas

If a standby database is created as a replica of the shard catalog, the shard wallet for the catalog must be manually copied from the primary shard catalog wallet. Find the location of the primary wallet using the above method, and make a copy to the correct location on the standby shard catalog database.

> **Note:**
>
> The value of wallet_root may be different on the standby shard catalog, and may not be set. Remember to set the value of wallet_root before copying the wallet to the standby location
>
> The wallet only exists on the primary shard catalog after the GDSCTL create shardcatalog command is run. If a standby shard catalog database is created before running create shardcatalog, then first run create shardcatalog to create the shard wallet on the primary shard catalog, then copy the wallet to the standby shard catalog.
>
> The shard catalog database also requires a wallet for CDB$ROOT. When copying and backing up shard wallets for a shard catalog database, you should also copy the shard wallet for CDB$ROOT regardless of which PDB is being used for the shard catalog.

Wallet Life Cycle Management

Once a sharded database has been deployed, it is crucial that the shard wallet is maintained throughout the life cycle of the shard catalog, the shards, and their replicas. Specifically, the shard wallet should be included in all backup and restore operations for each database, just as if it were a database data file.

Likewise, if a PDB is cloned, relocated, or otherwise moved, then the shard wallet should accompany the PDB to its new location. Note that in the case of PDB cloning specifically, the GUID for the PDB changes during the cloning operation, and therefore the path to the wallet will change as described above.

Updating a Wallet

If the shard wallet becomes lost, out of date. or is no longer accessible, a newly populated wallet can be created using the following GDSCTL command:

```
gdsctl sync database -database shard_name
```

Attempting to perform certain operations when the wallet is not present, or its contents are out of date, results in one or more of the following errors.

```
ORA-03873: unable to encrypt DDL statement with error ...

ORA-03874: unable to encrypt GSMUSER password with error ...

ORA-03876: error ... when attempting to generate a temporary key to add new
shards

ORA-03894: "Failed to send keys to shard %s with error ...."

ORA-03896: Unable to load the sharding wallet successfully.

ORA-00600: internal error code, arguments: [gwsec_get_latest_key]
```

Using Application Contexts During Cross-Shard Operations

The ability to use several Oracle security features such as Virtual Private Database (VPD), Unified Auditing, and Oracle Label Security (OLS) typically depend upon the use of session-level application contexts.

Before Oracle 21c, any cross-shard operations such as cross-shard queries or DMLs initiated by the shard catalog would not send session-level application context values to the affected shards. Therefore, features that depended on the context values being passed from the shard catalog session to the shards were not supported in a sharded environment.

Starting with Oracle 21c, any database session-based application context values set before a cross-shard query or DML are sent securely to all shards involved in the operation. This is how features such as VPD, auditing, and OLS are supported in a sharding environment.

For example, if a user connects to the shard catalog or a query coordinator from SQL*Plus and calls the `DBMS_SESSION.SET_CONTEXT` procedure to set a context value, then that value is sent to any shards involved in subsequent cross-shard operations initiated from the SQL*Plus session on the shard catalog. Calling the `SYS_CONTEXT` function on the shard will return the value originally set on the shard catalog as you would expect.

Note the following limitations when you attempt to use application contexts for cross-shard operations:

- The maximum length of a context value is 1968 bytes, as opposed to 4000 bytes in non-sharded environments.
- The maximum length of a context attribute name is 32 bytes, as opposed to 128 bytes in non-sharded environments.
- Only database session-based contexts initialized locally are currently supported.

- All of the shards in the configuration must be Oracle Database 21c or later releases for the context value to be passed during cross-shard operations.

For more information see Using Application Contexts to Retrieve User Information.

Behavior Differences

Some behavior that you would expect from a typical Oracle Database is modified in the context of a database sharded with Oracle Globally Distributed Database.

In general, database limits on a per-user or per-schema basis are not aggregated across all databases in the Oracle Globally Distributed Database sharded database, but only apply on a per-database level.

From an application perspective, a sharded database acts a single, logical database in most respects. However, a Oracle Globally Distributed Database sharded database itself consists of several independent, loosely-coupled Oracle Database instances acting as shard catalogs, query coordinators, and shards. As a result, some behavior that you would expect from a typical Oracle Database is modified in the context of a database sharded with Oracle Globally Distributed Database.

For example, if a sharded user is created and a user profile is assigned to the user with the SQL statement CREATE PROFILE, the limits set in the profile do not apply to the sharded database as a whole. Rather, they apply to each database that is a part of the larger, virtual sharded database.

Therefore, if you set the maximum number of failed login attempts to 20 for a sharded user, that limit does not apply to the entire sharded database but rather applies to each individual database in the configuration. If 20 failed attempts are reached when logging into a particular shard, then those failures do not count against the limits on the other shards or the shard catalog.

Using Transparent Data Encryption

Oracle Globally Distributed Database supports Transparent Data Encryption (TDE), but to successfully move chunks in a sharded database with TDE enabled, all of the shards must share and use the same encryption key for the encrypted tablespaces.

A sharded database consists of multiple independent databases and a shard catalog database. For TDE to work properly certain restrictions apply, especially when data is moved between shards. For chunk movement between shards to work when data is encrypted, you must ensure that all of the shards use the same encryption key.

There are two ways to accomplish this:

- Create and export an encryption key from the shard catalog, and then import and activate the key on all of the shards individually.
- Store the wallet in a shared location and have the shard catalog and all of the shards use the same wallet.

The following TDE statements are automatically propagated to shards when run on the shard catalog with shard DDL enabled:

- ALTER SYSTEM SET ENCRYPTION WALLET [OPEN|CLOSE] IDENTIFIED BY *password*
- ALTER SYSTEM SET ENCRYPTION KEY

- ADMINISTER KEY MANAGEMENT SET KEYSTORE [OPEN|CLOSE] IDENTIFIED BY *password*
- ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY *password*
- ADMINISTER KEY MANAGEMENT USE KEY IDENTIFIED BY *password*
- ADMINISTER KEY MANAGEMENT CREATE KEYSTORE IDENTIFIED BY *password*

Limitations

The following limitations apply to using TDE with Oracle Globally Distributed Database.

- For `GDSCTL MOVE CHUNK` to work, all of the shard database hosts must be on the same platform.
- `MOVE CHUNK` cannot use compression during data transfer, which may impact performance.
- Only encryption on the tablespace level is supported. Encryption on specific columns is not supported.

For more information about TDE see Introduction to Transparent Data Encryption

Creating a Single Encryption Key on All Shards

To propagate a single encryption key to all of the databases in the sharded database configuration, you must create a master encryption key on the shard catalog, then use wallet export, followed by wallet import onto the shards, and activate the keys.

This procedure assumes that the keystore password and wallet directory path are the same for the shard catalog and all of the shards. If you require different passwords and directory paths, all of the commands should be issued individually on each shard and the shard catalog with shard DDL disabled, using the shard's own password and path.

These steps should be done before any data encryption is performed.

1. Create an encryption key on the shard catalog.

   With shard DDL enabled, issue the following statements.

   ```
   ADMINISTER KEY MANAGEMENT CREATE KEYSTORE wallet_directory_path
    IDENTIFIED BY keystore_password;
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY
   keystore_password;
   ```

   The value for *keystore_password* should be the same if you prefer to issue wallet open and close commands centrally from the shard catalog.

   The wallet directory path should match the `WALLET_ROOT` in the corresponding initialization parameter file.

2. With shard DDL disabled, issue the following statement to activate the encryption key.

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
    IDENTIFIED BY keystore_password;
   ADMINISTER KEY MANAGEMENT USE KEY master_key_id
    IDENTIFIED BY keystore_password WITH BACKUP;
   ```

All of the shards and the shard catalog database now have the same encryption key activated and ready to use for data encryption. On the shard catalog, you can issue TDE DDLs (with shard DDL enabled), such as:

- Create encrypted tablespaces and tablespace sets.
- Create sharded tables using encrypted tablespaces.
- Create sharded tables containing encrypted columns (with limitations).

3. Validate that the key IDs on all of the shards match the ID on the shard catalog.

```
SELECT KEY_ID  FROM V$ENCRYPTION_KEYS
WHERE ACTIVATION_TIME =
 (SELECT MAX(ACTIVATION_TIME) FROM V$ENCRYPTION_KEYS
  WHERE ACTIVATING_DBID = (SELECT DBID FROM V$DATABASE));
```

4. `ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY keystore_password WITH BACKUP;`

   An encryption key is created and activated in the shard catalog database's wallet.

   If you issue this statement with DDL enabled, it will also create encryption keys in each of the shards' wallets, which are different keys from that of the shard catalog. For data movement to work, you cannot use different encryption keys on each shard.

5. Get the master key ID from the shard catalog keystore.

```
SELECT KEY_ID  FROM V$ENCRYPTION_KEYS
WHERE ACTIVATION_TIME =
 (SELECT MAX(ACTIVATION_TIME) FROM V$ENCRYPTION_KEYS
  WHERE ACTIVATING_DBID = (SELECT DBID FROM V$DATABASE));
```

6. With shard DDL disabled, export the catalog wallet containing the encryption key.

```
ADMINISTER KEY MANAGEMENT EXPORT ENCRYPTION KEYS WITH SECRET
secret_phrase TO
 wallet_export_file IDENTIFIED BY keystore_password;
```

7. Physically copy the wallet file to each of the shard hosts, into their corresponding wallet export file location, or put the wallet file on a shared disk to which all of the shards have access.
8. With shard DDL disabled, log on to each shard and import the wallet containing the key.

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY
keystore_password;
ADMINISTER KEY MANAGEMENT IMPORT ENCRYPTION KEYS WITH SECRET
secret_phrase FROM
 wallet_export_file IDENTIFIED BY keystore_password WITH BACKUP;
```

9. Restart the shard databases.
10. Activate the key on all of the shards on the shard catalog with shard DDL enabled.

4

Oracle Globally Distributed Database Deployment

Create and configure a sharded database, beginning with host provisioning, and continuing through software configuration, database setup, sharding metadata creation, and schema creation. This process is known as *deployment*.

Introduction to Sharded Database Deployment

Oracle Globally Distributed Database provides the capability to automatically deploy the sharded database, which includes both the shards and the replicas.

The sharded database administrator defines the topology (regions, shard hosts, replication technology) and invokes the `DEPLOY` command with a declarative specification using the `GDSCTL` command-line interface.

Before You Begin

There are many different configurations and topologies that can be used for a sharded database. Depending on your application's particular architecture and system requirements, you may have several choices from which to choose when designing your system. Familiarize yourself with Planning Your Sharded Database Deployment before proceeding with deployment.

Sharded Database Deployment Road map

At a high level, the deployment steps are:

1. Set up the components.
   - Provision and configure the hosts that will be needed for the sharding configuration and topology selected (see Provision and Configure Hosts and Operating Systems).
   - Install Oracle Database software on the selected catalog and shard nodes (see Install the Oracle Database Software).
   - Install global service manager (GSM) software on the shard director nodes (see Install the Shard Director Software).
2. Create databases needed to store the sharding metadata and the application data.
   - Create a database that will become the shard catalog along with any desired replicas for disaster recovery (DR) and high availability (HA) (see Create the Shard Catalog Database).
   - Create databases that will become the shards in the configuration including any standby databases needed for DR and HA (see Create the Shard Databases).
3. Specify the sharding topology using some or all the following commands from the `GDSCTL` command line utility, among others (see Configure the Oracle Globally Distributed Database Topology).
   - `CREATE SHARDCATALOG`

- `ADD GSM`
- `START GSM`
- `ADD SHARDGROUP`
- `ADD SHARD`
- `ADD INVITEDNODE`

4. Run `DEPLOY` to deploy the sharding topology configuration (see Deploy the Configuration).
5. Add the global services needed to access any shard in the sharded database (see Create and Start Global Database Services).
6. Verify the status of each shard (see Verify Shard Status).

When the sharded database configuration deployment is complete and successful, you can create the sharded schema objects needed for your application. See Oracle Globally Distributed Database Schema Objects.

The topics that follow describe each of the deployment tasks in more detail along with specific requirements for various components in the system. These topics can act as a reference for the set up and configuration of each particular step in the process. However, by themselves, they will not produce a fully functional sharding configuration since they do not implement a complete sharding scenario, but only provide the requirements for each step.

Example Oracle Globally Distributed Database Deployment walks you through a specific deployment scenario of a representative reference configurations. This section provides examples of every command needed to produce a fully functional sharded databases once all the steps are completed.

Planning Your Sharded Database Deployment

Many decisions need to be made when planning your sharded database deployment including the sharded database topology, replication method, and the sharding methodology.

Your particular sharded database may employ a variety of Oracle software components such as Oracle Data Guard and Oracle Real Application Clusters (Oracle RAC) along with different data distribution methods including system-managed (automatic), user-defined, and composite sharding.

Depending on which sharding method you choose (system, user-defined, or composite sharding), you can further refine your topology planning with decisions about considerations such as the number of chunks, shardgroups or shardspaces, regions, standbys, and open as opposed to mounted databases, and so on.

See Oracle Globally Distributed Database Architecture and Concepts for information pertaining to these topology options.

Plan the Sharded Database Configuration

To plan your Oracle Globally Distributed Database configuration you need an understanding of the objects that make up a sharded database configuration, so that you can best configure and deploy them to meet your requirements.

The sharded database configuration consists of the data distribution (sharding) method, replication (high availability) technology, the default number of chunks to be present in the sharded database, the location and number of shard directors, the numbers of shardgroups, shardspaces, regions, and shards in the sharded database, and the global services that will be used to connect to the sharded database.

- **Sharding method** - To decide which sharding methodology works best for your application, see Sharded Data Distribution Methods for a full discussion.
- **Topology** - To learn about each sharded database component you need, see Components of the Oracle Globally Distributed Database Architecture for a full discussion.
- **Replication** - To decide on a replication strategy, see Replication in Oracle Globally Distributed Database.

Oracle Database Global Data Services Architecture

Because the Oracle Globally Distributed Database feature is built on the Oracle Database Global Data Services feature, to plan your topology you might benefit from an understanding of the Global Data Services architecture. See Introduction to Global Data Services for conceptual information about Global Data Services.

Provision and Configure Hosts and Operating Systems

Before you install any software, review these hardware, network, and operating system requirements for Oracle Globally Distributed Database.

- **Oracle Database Enterprise Edition** is required when running an Oracle Globally Distributed Database.
- Hardware and operating system requirements for **shards** are the same as those for Oracle Database. See your Oracle Database installation documentation for these requirements.
- Hardware and operating system requirements for the **shard catalog and shard directors** are the same as those for the Global Data Services catalog and global service manager. See Oracle Database Global Data Services Concepts and Administration Guide for these requirements.
- **Network** Low Latency GigE is strongly recommended
- **Port communication** requirements are as follows.
  - Each and every shard must be able to reach each and every shard director's listener and ONS ports. The shard director listener ports and the ONS ports must also be opened to the application/client tier, all of the shards, the shard catalog, and all other shard directors.

    The default listener port of the shard director is 1522, and the default ONS ports on most platforms are 6123 for the local ONS and 6234 for remote ONS.
  - Each and every shard must be able to reach the TNS Listener port (default 1521) of the shard catalog (both primary and standbys).
  - The TNS Listener port of each shard must be opened to all shard directors and the shard catalog.
  - All of the port numbers listed above are modifiable during the deployment configuration. However, the port numbers to be used must be known before setting up the host software.

- **Host name resolution** must be successful between all of the shard catalog, shards, and shard director hosts. Operating system commands such as 'ping' must succeed from a given host to any other host when specifying any host names provided during sharded database configuration commands.

Number and Sizing of Host Systems

Depending on your specific configuration, the hosts that are needed may include the following:

- **Shard catalog host.** The shard catalog host runs the Oracle Database that serves as the shard catalog. This database contains a small amount of sharding topology metadata and any duplicated tables that are created for your application. In addition, the shard catalog acts as a multi-shard query coordinator for cross-shard queries and services connections for applications that have not been written to be sharding-aware. In general, the transaction workload and size of this database are not particularly large.
- **Shard catalog database standbys (replicas).** At least one more host to contain a replica or standby of the primary shard catalog database is recommended. This host is necessary in case of a failure of the primary catalog host.

  In addition, while acting as a standby database, this host can also be configured to be a query coordinator for cross-shard queries. To improve the scalability and availability of multi-shard query workloads, Oracle Active Data Guard standby shard catalog databases in read-only mode can act as multi-shard query coordinators. See Multi-Shard Query Coordinator Availability and Scalability.
- **Shard director host.** The shard director (global service manager) software can reside on a separate host, or it can be co-located on the same host as the shard catalog. This component of the sharding system is comprised of a network listener and several background processes used to monitor and configure a sharded configuration. If it is co-located on the same host as the catalog database, the shard director must be installed in a separate Oracle Home from the catalog database, because the installation package is different than the one used for Oracle Database.
- **Multiple shard directors.** For high-availability purposes, it is recommended that you have more than one shard director running in a sharded system. Any additional shard directors can run on their own hosts or on the hosts running the standby shard catalog databases.
- **Shards.** In addition to the above hosts, each shard that is configured in the system should also run on its own separate host. The hosts and their configurations chosen for this task should be sized in the same way as a typical Oracle Database host depending on how much load is put on each particular shard.
- **Shard standbys (replicas).** Again, for high-availability and disaster recovery purposes, use Oracle Data Guard and replicas created for all sharded data. Additional hosts will be needed to run these replica or standby databases.

When the number of hosts and capacity requirements for each host have been determined, provision your hardware resources as appropriate for your environment using whatever methodologies you choose.

Before installing any software, you must confirm that the hosts can communicate with each other though the ports as described above. Because a sharding configuration is inherently a distributed system, it is crucial that this connectivity between and among all of the hosts is confirmed before moving on to the next steps in the deployment

process. Failure to set up port access correctly will lead to failures in subsequent commands.

Install the Oracle Database Software

Install Oracle Database on each system that will host the shard catalog, a database shard, or their replicas.

Aside from the requirement that the shard catalog and all of the shards in an Oracle Globally Distributed Database configuration require Oracle Database Enterprise Edition, there are no other special installation considerations needed for sharding as long as the installation is successful and all post-install scripts have been run successfully.

See your platform's installation guide at https://docs.oracle.com/en/database/oracle/oracle-database/ for information about configuring operating system users.

Install the Shard Director Software

Install the global service manager software on each system that you want to host a shard director.

Note that this software installation is distinct from an Oracle Database installation. If you choose to co-locate the shard director software on the same host as the shard catalog database, it must be installed in a separate Oracle Home.

See *Oracle Database Global Data Services Concepts and Administration Guide* for information about installing the global service manager software.

Create the Shard Catalog Database

Use the following information and guidelines to create the shard catalog database.

The shard catalog database contains a small amount of sharding topology metadata and also contains all the duplicated tables that will be created for use by your sharded application. The shard catalog database also acts as a query coordinator to run cross-shard queries that select and aggregate data from more than one shard.

From a sharding perspective, the way in which you create or provision the catalog database is irrelevant. The database can be created with the Database Configuration Assistant (DBCA), manually using SQL*Plus, or provisioned from cloud infrastructure tools.

As long as you have a running Oracle Database Enterprise Edition instance on the shard catalog host with the following characteristics, it can used as the shard catalog.

- Create a pluggable database (PDB) for use as the shard catalog database. Using the root container (`CDB$ROOT`) of a container database (CDB) as the shard catalog database is not supported.
- Your shard catalog database must use a server parameter file (`SPFILE`). This is required because the sharding infrastructure uses internal database parameters to store configuration metadata, and that data needs to persist across database startup and shutdown operations.

```
$ sqlplus / as sysdba

SQL> show parameter spfile
```

```
NAME     TYPE      VALUE
-------- --------- ------------------------------------
spfile   string    /u01/app/oracle/dbs/spfilecat.ora
```

- The database character set and national character set must be the same, because it is used for all of the shard databases. This means that the character set chosen must contain all possible characters that will be inserted into the shard catalog or any of the shards.

  This requirement arises from the fact that Oracle Data Pump is used internally to move transportable tablespaces from one shard to another during sharding `MOVE CHUNK` commands. A requirement of that mechanism is that character sets must match on the source and destination.

  ```
  $ sqlplus / as sysdba

  SQL> alter session set container=catalog_pdb_name;
  SQL> select * from nls_database_parameters
    2  where parameter like '%CHARACTERSET';

  PARAMETER                                VALUE
  ---------------------------------------- --------------------
  NLS_NCHAR_CHARACTERSET                   AL16UTF16
  NLS_CHARACTERSET                         WE8DEC
  ```

- Because the shard catalog database can run multi-shard queries which connect to shards over database links, the `OPEN_LINKS` and `OPEN_LINKS_PER_INSTANCE` database initialization parameter values must be greater than or equal to the number of shards that will be part of the sharded database configuration.

  ```
  $ sqlplus / as sysdba

  SQL> alter session set container=catalog_pdb_name;
  SQL> show parameter open_links

  NAME                                 TYPE        VALUE
  ------------------------------------ ----------- ------------
  open_links                           integer     20
  open_links_per_instance              integer     20
  ```

- Set the `DB_FILES` database initialization parameter greater than or equal to the total number of chunks and/or tablespaces in the system.

  Each data chunk in a sharding configuration is implemented as a tablespace partition and resides in its own operating system data file. As a result, the `DB_FILES` database initialization parameter must be greater than or equal to the total number of chunks (as specified on the `CREATE SHARDCATALOG` or `ADD SHARDSPACE` commands) and/or tablespaces in the system.

  ```
  $ sqlplus / as sysdba

  SQL> alter session set container=catalog_pdb_name;
  SQL> show parameter db_files

  NAME                                 TYPE        VALUE
  ```

```
---------------------------------- ----------- ------------
db_files                           integer     1024
```

- To support Oracle Managed Files, which is used by the sharding chunk management infrastructure, the `DB_CREATE_FILE_DEST` database parameter must be set to a valid value.

  This location is used during chunk movement operations (for example `MOVE CHUNK` or automatic rebalancing) to store the transportable tablespaces holding the chunk data. In addition, files described in *Oracle Database Administrator's Guide*, "Using Oracle Managed Files," are also stored in this location as is customary for any Oracle database using Oracle Managed Files.

  ```
  $ sqlplus / as sysdba

  SQL> alter session set container=catalog_pdb_name;
  SQL> show parameter db_create_file_dest

  NAME                 TYPE      VALUE
  -------------------- --------- ------------------------------
  db_create_file_dest  string    /u01/app/oracle/oradata
  ```

- If a standby catalog database will be part of the sharding configuration, the `STANDBY_FILE_MANAGEMENT` database parameter should be set to in order to automatically create new database files on any standby catalog databases.

  If this parameter is set to `MANUAL` (which is the default), then new database files created during `CREATE TABLESPACE` commands, for example, will not be created on the standby. This will cause data unavailability and application errors if the standby ever becomes a primary database.

  ```
  $ sqlplus / as sysdba

  SQL> alter session set container=catalog_pdb_name;
  SQL> show parameter standby_file_management

  NAME TYPE VALUE
  ---------------------------------- ----------- ------------
  standby_file_management stirng AUTO
  ```

- An Oracle-provided user account named `GSMCATUSER` must be unlocked and assigned a password inside the PDB designated for the shard catalog. This account is used by the shard director processes to connect to the shard catalog database and perform administrative tasks in response to sharding commands.

  Note that `GSMCATUSER` is a common user in the container database. As a result, its password is the same for `CDB$ROOT` and all PDBs in the CDB. If multiple PDBs in a single CDB are to be used as catalog databases for different sharding configurations, they will all share the same `GSMCATUSER` password which can be a security concern. To avoid this potential security concern, configure a separate CDB to host each shard catalog. Each CDB should contain only a single shard catalog PDB so that no other PDBs in the CDB can share the common `GSMCATUSER` password. In this way, multiple shard catalogs can be configured across several CDBs, each having different `GSMCATUSER` passwords.

The password you specify is used later during sharding topology creation in any `ADD GSM` commands that are issued. It never needs to be specified again because the shard director stores it securely in an Oracle Wallet and decrypts it only when necessary.

The `MODIFY GSM` command can be used to update the stored password if it is later changed on the shard catalog database.

```
$ sqlplus / as sysdba

SQL> alter user gsmcatuser account unlock;

User altered.

SQL> alter user gsmcatuser identified by gsmcatuser_password;

User altered.

SQL> alter session set container=catalog_pdb_name;
SQL> alter user gsmcatuser account unlock;

User altered.
```

- A shard catalog administrator account must be created, assigned a password, and granted privileges inside the PDB designated as the shard catalog.

  This account is the administrator account for the sharding metadata in the shard catalog database. It is used to access the shard catalog using the `GDSCTL` utility when an administrator needs to makes changes to the sharded database topology or perform other administrative tasks.

  `GDSCTL` connects as this user to the shard catalog database when `GDSCTL` commands are run. The user name and password specified are used later in the `CREATE SHARDCATALOG` command. As with the `GSMCATUSER` account above, the user name and password are stored securely in an Oracle Wallet for later use. The stored credentials can be updated by issuing an explicit `CONNECT` command from `GDSCTL` to reset the values in the wallet.

  ```
  $ sqlplus / as sysdba

  SQL> alter session set container=catalog_pdb_name;
  SQL> create user mysdbadmin identified by mysdbadmin_password;

  User created.

  SQL> grant gsmadmin_role to mysdbadmin;

  Grant succeeded.
  ```

- Set up and run an Oracle Net TNS Listener at your chosen port (default is 1521) that can service incoming connection requests for the shard catalog PDB.

  The TNS Listener can be created and configured in whatever way you wish. Depending on how the database was created, it may be necessary to explicitly create a database service that can allow for direct connection requests to the PDB without the need to use `ALTER SESSION SET CONTAINER`.

To validate that the listener is configured correctly, do the following using your newly created *mysdbadmin* account above and an appropriate connect string. Running `LSNRCTL SERVICES` lists all services currently available using the listener.

```
$ sqlplus mysdbadmin/mysdbadmin_password@catalog_connect_string

SQL> show con_name

CON_NAME
------------------------
catalog_pdb_name
```

Once you confirm connectivity, make note of the *catalog_connect_string* above. It is used later in the configuration process in the `GDSCTL CREATE SHARDCATALOG` command. Typically, it will be of the form *host*:*port*/*service_name* (for example, `cathost.example.com:1521/catalog_pdb.example.com`).

After all of the above requirements have been met, the newly created database can now be the target of a `GDSCTL CREATE SHARDCATALOG` command.

For high availability and disaster recovery purposes, it is highly recommended that you also create one or more standby shard catalog databases. From a sharding perspective, as long as the above requirements are also met on the standby databases, and all changes to the primary shard catalog database are consistently applied to the standbys, there are no further sharding-specific configuration steps required.

Create the Shard Databases

The databases that will be used as shards should be created on their respective hosts.

As with the shard catalog database, the way in which you create or provision the shard databases is irrelevant from a sharding perspective. The database can be created with the Database Configuration Assistant (DBCA), manually using SQL*Plus, or provisioned from Oracle Cloud Infrastructure tools.

As long as you have a running Oracle Database Enterprise Edition instance on each shard host that meets the following requirements, it can be used as a shard.

Unlock `GSMROOTUSER`

An Oracle-provided user account named `GSMROOTUSER` must be unlocked and assigned a password inside `CDB$ROOT` of the database designated for a shard. In addition, this user must be granted the `SYSDG` and `SYSBACKUP` system privileges.

The `GSMROOTUSER` account is used by `GDSCTL` and the shard director processes to connect to the shard database to perform administrative tasks in response to sharding commands. The password specified is used by `GDSCTL` during sharding topology creation in any `ADD CDB` commands that are issued. It is also be used by the shard director during the `DEPLOY` command to configure Oracle Data Guard (as necessary) on the shard databases. It never needs to be specified again by the user, because `GDSCTL` and the shard director store it securely in an Oracle Wallet and decrypt it only when necessary. The `MODIFY CDB` command can be used to update the stored password if it is later changed on the shard database.

```
$ sqlplus / as sysdba
```

```
SQL> alter user gsmrootuser account unlock;

User altered.

SQL> alter user gsmrootuser identified by gsmrootuser_password;

User altered.

SQL> grant SYSDG, SYSBACKUP to gsmrootuser;

Grant succeeded.
```

Unlock GSMUSER

An Oracle-provided user account named GSMUSER must be unlocked and assigned a password inside the PDB designated as the shard database. In addition, this user must be granted the SYSDG and SYSBACKUP system privileges.

Note that GSMUSER is a common user in the container database. As a result, its password is the same for CDB$ROOT and all PDBs in the CDB, which can be a security concern. To avoid this, host only one shard PDB per CDB, and do not unlock the GSMUSER account in any other PDBs.

This account is used by the shard director processes to connect to the shard database and perform administrative tasks in response to sharding commands. The password specified is used later during sharding topology creation in any ADD SHARD commands that are issued. The password never needs to be specified again because the shard director stores it securely in an Oracle Wallet and only decrypts it when necessary. You can update the stored password using the MODIFY SHARD command if the password is later changed on the shard database.

```
$ sqlplus / as sysdba

SQL> alter user gsmuser account unlock;

User altered.

SQL> alter user gsmuser identified by gsmuser_password;

User altered.

SQL> alter session set container=shard_pdb_name;
SQL> alter user gsmuser account unlock;

User altered.

SQL> grant SYSDG, SYSBACKUP to gsmuser;

Grant succeeded.
```

Create a PDB

Create a pluggable database (PDB) for use as the shard database. Using the root container (CDB$ROOT) of a container database (CDB) as a shard is not supported.

Verify SPFILE Exists

Your shard database must use a server parameter file (SPFILE).

The SPFILE is required because the sharding infrastructure uses internal database parameters to store configuration metadata, and that data must persist through database startup and shutdown operations.

```
$ sqlplus / as sysdba

SQL> alter session set container=shard_pdb_name;
SQL> show parameter spfile

NAME     TYPE      VALUE
-------- --------- ------------------------------------
spfile   string    /u01/app/oracle/dbs/spfileshard.ora
```

Calculate and Set DB_FILES Appropriately

Set the DB_FILES database initialization parameter greater than or equal to the total number of chunks and/or tablespace sets required in the sharded database.

Each data chunk in a sharding configuration is implemented as a tablespace partition and resides in its own operating system data file. As a result, the DB_FILES database initialization parameter must be greater than or equal to the total number of chunks (as specified in the CREATE SHARDCATALOG or ADD SHARDSPACE commands) and/or tablespace sets in the system.

Note that the number of chunks present on a shard in a Raft replication scenario is the total of all chunks that the shard is either leader or follower for.

To calculate the number of database files created for sharding objects on a given shard:

Sharded database files required = (Number of CREATE TABLESPACE SET SQL statements executed using SHARD DDL) * (Number of chunks present on the shard + 1)

DB_FILES must be set to at least the number of files used by sharding (above) **PLUS** non-sharding database files (system, sysaux, and so on) **PLUS** any extra needed by generic RDBMS code (5); therefore:

DB_FILES required in each shard = (Number of sharded database files required, as calculated above) + Number of default database files(6) + 5

Check Character Sets

The database character set and national character set of the shard database must be the same as that used for the shard catalog database and all other shard databases. This means that the character set you choose must contain all possible characters that will be inserted into the shard catalog or any of the shards.

This requirement arises from the fact that Oracle Data Pump is used internally to move transportable tablespaces from one shard to another during sharding MOVE CHUNK commands. A requirement of that mechanism is that character sets must match on the source and destination.

```
$ sqlplus / as sysdba

SQL> alter session set container=shard_pdb_name;
```

```
SQL> select * from nls_database_parameters
  2  where parameter like '%CHARACTERSET';

PARAMETER                                VALUE
---------------------------------------- ----------------------
NLS_NCHAR_CHARACTERSET                   AL16UTF16
NLS_CHARACTERSET                         WE8DEC
```

Set COMPATIBLE to 12.2.0 or Higher

The COMPATIBLE initialization parameter must be set to at least 12.2.0.

```
$ sqlplus / as sysdba

SQL> alter session set container=shard_pdb_name;
SQL> show parameter compatible

NAME                 TYPE        VALUE
-------------------- ----------- ------------------
compatible           string      21.0.0
```

Set DB_CREATE_FILE_DEST

To support Oracle Managed Files, used by the sharding chunk management infrastructure, the DB_CREATE_FILE_DEST database parameter must be set to a valid value.

This location is used during chunk movement operations (for example MOVE CHUNK or automatic rebalancing) to store the transportable tablespaces holding the chunk data. In addition, files described in *Oracle Database Administrator's Guide*, "Using Oracle Managed Files," are also stored in this location as is customary for any Oracle database using Oracle Managed Files.

```
$ sqlplus / as sysdba

SQL> alter session set container=shard_pdb_name;
SQL> show parameter db_create_file_dest

NAME                 TYPE      VALUE
-------------------- --------- ------------------------------
db_create_file_dest  string    /u01/app/oracle/oradata
```

Create DATA_PUMP_DIR

A directory object named DATA_PUMP_DIR must be created and accessible in the PDB from the GSMADMIN_INTERNAL account.

GSMADMIN_INTERNAL is an Oracle-supplied account that owns all of the sharding metadata tables and PL/SQL packages. It should remain locked and is never used to login interactively. It's only purpose is to own and control access to the sharding metadata and PL/SQL.

```
$ sqlplus / as sysdba
```

```
SQL> create or replace directory DATA_PUMP_DIR as '/u01/app/oracle/oradata';

Directory created.

SQL> alter session set container=shard_pdb_name;
SQL> grant read, write on directory DATA_PUMP_DIR to gsmadmin_internal;

Grant succeeded.
```

Set DB_FILE_NAME_CONVERT

To support file movement from shard to shard, the DB_FILE_NAME_CONVERT database parameter must be set to a valid value. This location is used when standby databases are in use, as is typical with non-sharded databases, and the location can also be used during chunk movement operations. For regular file system locations, it is recommended that this parameter end with a trailing slash (/).

```
$ sqlplus / as sysdba

SQL> alter session set container=shard_pdb_name;
SQL> show parameter db_file_name_convert

NAME TYPE VALUE
---------------------- --------- ------------------------------
db_file_name_convert   string    /dbs/SHARD1/, /dbs/SHARD1S/
```

Set Up Oracle Net TNS Listener

Set up and run an Oracle Net TNS Listener at your chosen port (default is 1521) that can service incoming connection requests for the shard PDB.

The TNS Listener can be created and configured in whatever way you wish. Depending on how the database was created, it may be necessary to explicitly create a database service that can allow for direct connection requests to the PDB without the need to use ALTER SESSION SET CONTAINER.

To validate that the listener is configured correctly, run the following command using your newly unlocked GSMUSER account and an appropriate connect string. Running LSNRCTL SERVICES lists all services currently available using the listener.

```
$ sqlplus gsmuser/gsmuser_password@shard_connect_string

SQL> show con_name

CON_NAME
-----------------------
shard_pdb_name
```

Once you confirm connectivity, make note of the *shard_connect_string* above. It is used later in the configuration process in the GDSCTL ADD SHARD command. Typically, the connect string is in the form *host*:*port*/*service_name* (for example, shardhost.example.com:1521/shard_pdb.example.com).

If standby shard databases will be used:

Enable Flashback Database

Enable Flashback Database if your sharded database will use standby shard databases.

```
$ sqlplus / as sysdba

SQL> alter session set container=shard_pdb_name;
SQL> select flashback_on from v$database;

FLASHBACK_ON
------------------
YES
```

Enable `FORCE LOGGING`

`FORCE LOGGING` mode must be enabled if your shard database will use standby shard databases.

```
$ sqlplus / as sysdba

SQL> alter session set container=shard_pdb_name;
SQL> select force_logging from v$database;

FORCE_LOGGING
---------------------------------------
YES
```

Set `STANDBY_FILE_MANAGEMENT`

If a standby shard databases will be part of the sharding configuration, the `STANDBY_FILE_MANAGEMENT` database parameter should be set to `AUTO` to automatically create new database files on any standby shard databases.

If this parameter is set to `MANUAL` (which is the default), then new database files created during `CREATE TABLESPACE` commands, for example, will not be created on the standby. This will cause data unavailability and application errors if the standby ever becomes a primary database.

```
$ sqlplus / as sysdba

SQL> alter session set container=shard_pdb_name;
SQL> show parameter standby_file_management

NAME TYPE VALUE
------------------------------------ ----------- ------------
standby_file_management string AUTO
```

If Raft replication will be used:

Set the following database initialization parameters:

- `FILESYSTEMIO_OPTIONS=setall` - enables asynchronous I/O

- `UNDO_RETENTION=900` - this is the default, and it is automatically tuned, but it is recommend that you do not explicitly set this parameter to a very low value.

Validate the Shard Database

To validate that all of the shard database requirements have been met, you can run an Oracle-supplied procedure, `validateShard`, that inspects the shard database and reports any issues encountered. This procedure is read-only and makes no changes to the database configuration.

The `validateShard` procedure can and should be run against primary, mounted (unopened) standby, and Active Data Guard standby databases that are part of the sharded database configuration. You can run `validateShard` multiple times and at any time during the sharded database life cycle, including after upgrades and patching.

To run the `validateShard` package, do the following:

```
$ sqlplus / as sysdba

SQL> alter session set container=shard_pdb_name;
SQL> set serveroutput on
SQL> execute dbms_gsm_fix.validateShard
```

This procedure will produce output similar to the following:

```
INFO: Data Guard shard validation requested.
INFO: Database role is PRIMARY.
INFO: Database name is SHARD1.
INFO: Database unique name is shard1.
INFO: Database ID is 4183411430.
INFO: Database open mode is READ WRITE.
INFO: Database in archivelog mode.
INFO: Flashback is on.
INFO: Force logging is on.
INFO: Database platform is Linux x86 64-bit.
INFO: Database character set is WE8DEC. This value must match the character
set of the catalog database.
INFO: 'compatible' initialization parameter validated successfully.
INFO: Database is a multitenant container database.
INFO: Current container is SHARD1_PDB1.
INFO: Database is using a server parameter file (spfile).
INFO: db_create_file_dest set to: '/u01/app/oracle/dbs'
INFO: db_recovery_file_dest set to: '/u01/app/oracle/dbs'
INFO: db_files=1000. Must be greater than the number of chunks and/or
tablespaces to be created in the shard.
INFO: dg_broker_start set to TRUE.
INFO: remote_login_passwordfile set to EXCLUSIVE.
INFO: db_file_name_convert set to: '/dbs/SHARD1/, /dbs/SHARD1S/'
INFO: GSMUSER account validated successfully.
INFO: DATA_PUMP_DIR is '/u01/app/oracle/dbs/
9830571348DFEBA8E0537517C40AF64B'.
```

All output lines marked `INFO` are for informational purposes and should be validated as correct for your configuration.

All output lines marked `ERROR` must be fixed before moving on to the next deployment steps. These issues will cause errors for certain sharding operations if they are not resolved.

All output lines marked `WARNING` may or may not be applicable for your configuration. For example, if standby databases will not be used for this particular deployment, then any warnings related to standby databases or recovery can be ignored. This is especially true for non-production, proof-of-concept, or application development deployments. Review all warnings and resolve as necessary.

Once all of the above steps have been completed, the newly created database can now be the target of a `GDSCTL ADD SHARD` command.

For high availability and disaster recovery purposes, it is highly recommended that you also create one or more standby shard databases. From a sharding perspective, as long as the above requirements are also met on the standby databases, and all changes to the primary shard database are applied to the standbys, the standby database only needs to be added to the sharding configuration with an `ADD SHARD` command.

Configure the Oracle Globally Distributed Database Topology

After the databases for the shard catalog and all of the shards are configured, along with corresponding TNS listeners, you can add the sharding metadata to the shard catalog database using `GDSCTL`. The sharding metadata describes the topology used for the sharded database.

The sharded database topology consists of the sharding method, replication (high availability) technology, the default number of chunks to be present in the sharded database, the location and number of shard directors, the numbers of shardgroups, shardspaces, regions, and shards in the sharded database, and the global services that will be used to connect to the sharded database.

Keep the Global Data Services Control Utility (GDSCTL) Command Reference in the *Oracle Database Global Data Services Concepts and Administration Guide* on hand for information about usage and options for the `GDSCTL` commands used in the configuration procedures.

Follow the procedures listed below, in order, to complete your sharded database topology configuration.

Run the commands from a shard director host, because the `GDSCTL` command line interface is installed there as part of the shard director (global service manager) installation.

Create the Shard Catalog

Use the `GDSCTL CREATE SHARDCATALOG` command to create metadata describing the sharded database topology in the shard catalog database.

Note that once you run `CREATE SHARDCATALOG`, and the rest of the sharding metadata has been created, there are several metadata properties that cannot be modified without recreating the entire sharded database from scratch. These include the sharding method (system-managed, user-defined, composite), replication technology (Oracle Data Guard or Raft replication), default number of chunks in the shardspace, and others. Make sure that you consult the `GDSCTL` reference documentation for the complete list of possible command options and their defaults.

Shard Catalog Connect String

When you run the `CREATE SHARDCATALOG` command, `GDSCTL` connects to the shard catalog database with the user name and connect string specified.

If your shard catalog database has an associated standby database for high availability or disaster recovery purposes, the connection string, *catalog_connect_string* in the examples that follow, should specify all primary and standby databases. If you don't include the standby databases in the connect string, then the shard director processes will not be able to connect to the standby if the primary shard catalog is unavailable.

Note that *catalog_connect_string* should specify the PDB for the shard catalog database, not the `CDB$ROOT`.

The following is a simple `tnsnames.ora` entry.

```
CATALOG_CONNECT_STRING=
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = tcp)(HOST = primary_catalog)(PORT = 1521))
      (ADDRESS = (PROTOCOL = tcp)(HOST = standby_catalog)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = catpdb.example.com)
    )
  )
```

Creating the Shard Catalog

Run `CREATE SHARDCATALOG` with the settings appropriate for your planned sharding topology.

System-Managed Sharding Method

In the following example, the sharded database metadata is created for a system-managed sharding configuration with two regions named `region1` and `region2`. Because system-managed is the default sharding method, it does not need to be specified with the `-sharding` parameter.

```
GDSCTL> create shardcatalog -database catalog_connect_string
 -user mysdbadmin/mysdbadmin_password -repl DG -region region1,region2
```

Note also that if `-shardspace` is not specified, a default shardspace named `shardspaceora` is created. If `-region` is not specified, the default region named `regionora` is created. If the single default region is created along with the default shardspace, then a default shardgroup named `shardspaceora_regionora` is also created in the shardspace.

For replication (`-repl`) with system-managed sharding, you can choose either Oracle Data Guard (`DG`) or Raft replication (`native`).

Composite Sharding Method

The following example shows you how to create shard catalog metadata for a composite sharded database with Data Guard replication in `MaxAvailability` protection mode, 60 chunks per shardspace, and two shardspaces.

```
GDSCTL> create shardcatalog -database catalog_connect_string
 -user mysdbadmin/mysdbadmin_password -sharding composite -chunks 60
 -protectmode maxavailability -shardspace shardspace1,shardspace2
```

User-Defined Sharding Method

The next example shows you how to create shard catalog metadata for a user-defined sharded database with Data Guard replication.

```
GDSCTL> create shardcatalog -database catalog_connect_string
 -user mysdbadmin/mysdbadmin_password -sharding user
 -protectmode maxperformance
```

Consult the `GDSCTL` documentation or run `GDSCTL HELP CREATE SHARDCATALOG` for more details about the command usage.

Replication Settings

Oracle Data Guard can be used with any sharding method, and is configured in the `CREATE SHARDCATALOG` command with `-repl` **`DG`**.

Raft replication requires a bit more planning, but it is also enabled in `CREATE SHARDCATALOG` command with `-repl` **`native`**. See Raft Replication Configuration and Management for additional configurable attributes.

Future Connections to the Shard Catalog

`GDSCTL` stores the credentials for the shard catalog administrator in a wallet on the local host. However, for subsequent `GDSCTL` sessions on other hosts, it may be necessary to explicitly connect to the shard catalog in order to perform administrative tasks by running the `GDSCTL CONNECT` command, as shown here.

```
GDSCTL> connect mysdbadmin/mysdbadmin_password@catalog_connect_string
```

Add and Start Shard Directors

Add to the configuration the shard directors, which will monitor the sharding system and run background tasks in response to GDSCTL commands and other events, and start them.

The following commands must be run on the host where the shard director processes are to run. This can be the shard catalog host or a dedicated host for the shard director processes.

1. Add and start a shard director (GSM), as shown in the following example.

   ```
   GDSCTL> connect mysdbadmin/mysdbadmin_password@catalog_connect_string
   GDSCTL> add gsm -gsm sharddirector1 -catalog catalog_connect_string -pwd
   gsmcatuser_password
   GDSCTL> start gsm -gsm sharddirector1
   ```

   The value for the -gsm parameter is the name that you will be using to reference this shard director in later GDSCTL commands. The values for the -catalog and -pwd parameters should be the same used when you created the shard catalog database.

   Use the -listener, -localons, and -remoteons parameters as described in the GDSCTL reference to override the default port numbers of 1522, 6123, and 6234, respectively. Always confirm that the port numbers to be used, whether default or user-specified, are available on the host and do not conflict with other running software or Oracle listeners.

2. Repeat the ADD GSM and START GSM commands for any additional shard directors on each shard director host.

   Replace the shard director name (that is, sharddirector1 in the example) with an appropriate value for each shard director.

   If more than one shard director is used, then multiple regions must have been created for them in the CREATE SHARDCATALOG command, or you can add them later by running ADD REGION.

   Specify a region for each shard director with the -region parameter on each ADD GSM command, as shown here.

   ```
   GDSCTL> add gsm -gsm sharddirector2 -catalog catalog_connect_string -pwd
   gsmcatuser_password -region dc2
   ```

For later GDSCTL sessions, you might need to explicitly specify the shard director to be administered. If an error message is shown referencing the default GSMORA shard director, run GDSCTL SET GSM before continuing, as shown here.

```
GDSCTL> set gsm -gsm sharddirector1
```

Add Shardspaces If Needed

If you are using composite or user-defined sharding, and you need to add more shardspaces to complete your desired sharding topology, use the ADD SHARDSPACE command to add additional shardspaces.

- Run `ADD SHARDSPACE` as shown here.

```
GDSCTL> add shardspace -shardspace shardspace2
```

By default, the `ADD SHARDSPACE` command inherits the `-chunks` and `-protectmode` values that you used in the `CREATE SHARDCATALOG` command. You can specify, on a per-shardspace basis,the number of chunks and the Data Guard protection mode by using the `-chunks` and `-protectmode` parameters with `ADD SHARDSPACE`.

Add Shardgroups If Needed

If your sharded database topology uses the system-managed or composite sharding method, you can add any necessary additional shardgroups for your application.

Each shardspace must contain at least one primary shardgroup and may contain any number or type of standby shardgroups. Shardgroups are not used in the user-defined sharding method.

- Run `ADD SHARDGROUP` to add shardgroups to the configuration.

```
GDSCTL> add shardgroup -shardgroup shardgroup_primary -shardspace
shardspace1
 -deploy_as primary -region region1
GDSCTL> add shardgroup -shardgroup shardgroup_standby -shardspace
shardspace1
 -deploy_as active_standby -region region2
```

Note that when you run `ADD SHARDGROUP` you can specify one of three types of shardgroups: `primary`, `standby` (mounted, not open), and `active_standby` (open, available for queries) using the `-deploy_as` parameter (the default is `standby`).

Any shards subsequently added to the shardgroup must be opened in the mode corresponding to the `-deploy_as` setting for the shardgroup. For example, read-write for primary shardgroups, mounted for standby shardgroups, or read-only with apply for active standby shardgroups.

After shards are deployed, their current mode is monitored by the shard directors and communicated to the shard catalog such that it is possible and expected that shards of different open modes may be in the same shardgroup, depending upon subsequent switchover or failover operations.

Verify the Sharding Topology

Before adding information about your shard databases to the catalog, verify that your sharding topology is correct before proceeding by using the various `GDSCTL CONFIG` commands.

Once shards are added and deployed, it is no longer possible to change much of the shard catalog metadata, so validating your configuration is an important task at this point.

- Run `GDSCTL CONFIG` to view overall configuration information.

```
GDSCTL> config

Regions
------------------------
region1
region2

GSMs
------------------------
sharddirector1
sharddirector2

Sharded Database
------------------------
orasdb

Databases
------------------------

Shard Groups
------------------------
shardgroup_primary
shardgroup_standby

Shard spaces
------------------------
shardspaceora

Services
------------------------

GDSCTL pending requests
------------------------
Command                   Object                  Status
-------                   ------                  ------

Global properties
------------------------
Name: oradbcloud
Master GSM: sharddirector1
DDL sequence #: 0
```

You can use the various `GDSCTL CONFIG` commands to display more information about shardspaces, shardgroups, and other shard catalog objects. For a complete list of `GDSCTL CONFIG` command variants, see the `GDSCTL` reference documentation or run `GDSCTL HELP`.

Add the Shard CDBs

Add the CDBs containing the shard PDBs to the sharding configuration with the `ADD CDB` command.

1. Run the `ADD CDB` command as shown here.

   ```
   GDSCTL> add cdb -connect cdb_connect_string -pwd
   gsmrootuser_password
   ```

   This command causes `GDSCTL` to connect to `GSMROOTUSER`/*gsmrootuser_password@cdb_connect_string* as `SYSDG` to validate settings and to retrieve the `DB_UNIQUE_NAME` of the CDB, which will become the CDB name in the shard catalog.
2. Repeat the `ADD CDB` command for all of the CDBs that contain a shard PDB in the configuration.
3. When all of the CDBs are added, run `GDSCTL CONFIG CDB` to display a list of CDBs in the catalog.

   ```
   GDSCTL> config cdb
   ```

Add the Shard PDBs

Use the `ADD SHARD` command to add the shard PDB information to the shard catalog, then verify it with the `CONFIG SHARD` command.

1. Run `ADD SHARD` with the usage appropriate to your sharding method, as shown in the following examples.

   For **system-managed** or **composite** sharding, run `ADD SHARD` with the parameters shown here.

   ```
   GDSCTL> add shard -connect shard_connect_string -pwd
   gsmuser_password
   -shardgroup shardgroup_name -cdb cdb_name
   ```

   For **user-defined** sharding, the command usage is slightly different.

   ```
   GDSCTL> add shard -connect shard_connect_string -pwd
   gsmuser_password
   -shardspace shardspace_name -deploy_as db_mode -cdb cdb_name
   ```

   The `-cdb` parameter specifies the name of the CDB in which the shard PDB exists, `-shardgroup` or `-shardspace` specifies the location of the shard in your sharding topology, and `-deploy_as` specifies the open mode (`primary`, `standby`, `active_standby`) of the shard.

   > **Note:**
   >
   > It is highly recommended that you set `server=dedicated` in the connect string.

When you run `ADD SHARD`, `GDSCTL` connects to `GSMUSER`/*gsmuser_password@shard_connect_string* as SYSDG to validate the settings on the shard, re-runs `dbms_gsm_fix.validateShard` to check for errors, and constructs the shard name using the convention *db_unique_name_of_CDB_PDB_name* (for example `cdb1_pdb1`).

Finally, the metadata that describes the shard is added to the shard catalog.

2. Run `GDSCTL CONFIG SHARD` to view the shard metadata on the shard catalog.

```
GDSCTL> config shard
Name      Shard Group          Status    State    Region    Availability
--------- -------------------  ------    -----    ------    ------------
cdb1_pdb1 shardgroup_primary   U         none     region1   -
cdb2_pdb1 shardgroup_standby   U         none     region2   -
cdb3_pdb2 shardgroup_primary   U         none     region1   -
cdb4_pdb2 shardgroup_standby   U         none     region2   -
```

Note that the value for Status is `U` for “undeployed”, and State and Availability are `none` and – until the `DEPLOY` command is successfully run.

Add Host Metadata

Add all of the host names and IP addresses of your shard hosts to the shard catalog.

As part of the deployment process, the shard director contacts the shards and directs them to register with the shard director’s TNS listener process. This listener process only accepts incoming registration requests from trusted sources and will reject registration requests from unknown hosts.

If your shard hosts have multiple host names or network interfaces assigned to them, it is possible that the incoming registration request to the shard director may come from a host that was not automatically added during `ADD SHARD`. In this case, the registration request is rejected and the shard will not deploy correctly. The visible symptom of this problem will be that `CONFIG SHARD` shows `PENDING` for the shard’s Availability after `DEPLOY` has completed.

To avoid this issue, use the `GDSCTL ADD INVITEDNODE` command to manually add all host names and IP addresses of your shard hosts to the shard catalog metadata.

1. View a list of trusted hosts.

   By default, the `ADD SHARD` command adds the default host name of the shard host to the shard catalog metadata, so that any registration requests from that host to the shard director will be accepted. You can view the list of trusted hosts by running the `GDSCTL CONFIG VNCR` command.

```
GDSCTL> config vncr
```

2. Ping from all of the hosts in the configuration to verify successful host name resolution.

   Any hosts listed in the `CONFIG VNCR` output must be reachable by name from all of the other hosts in the topology. Use the `ping` command from the shard, shard catalog, and shard director hosts to verify that hostname resolution succeeds for all of the host names listed.

   To resolve any issues, use operating system commands or settings to ensure that all of the host names can be resolved.

3. Run the `REMOVE INVITEDNODE` command to manually remove any host names that are not necessary and cannot be resolved from all of the hosts.
4. Run the `ADD INVITEDNODE` command to manually add all host names and IP addresses of your shard hosts to the shard catalog metadata.

```
GDSCTL> add invitednode 127.0.0.1
```

Check Free DB_FILES

Verify that there are enough free data files in each shard to make sure there is enough capacity to create the number of chunks and tablespace sets you need for the sharded database.

To check free `DB_FILES` and parameter setting:

```
SQL> select count(*) from v$datafile;

  COUNT(*)
----------
XxxXX

SQL> show parameter db_files

NAME                                 TYPE     VALUE
------------------------------------ -----------
------------------------------
db_files                             integer     200
```

Formulas to calculate the `DB_FILES` setting can be found in Create the Shard Databases.

Deploy the Configuration

When the sharded database topology has been fully configured with `GDSCTL` commands, run the `GDSCTL DEPLOY` command to deploy the Oracle Globally Distributed Database configuration.

When you run the `GDSCTL DEPLOY` command the output looks like the following.

```
GDSCTL> deploy
deploy: examining configuration...
deploy: requesting Data Guard configuration on shards via GSM
deploy: shards configured successfully
The operation completed successfully
```

What Happens During Deployment

As you can see, when you run `DEPLOY` several things happen.

- GDSCTL calls a PL/SQL procedure on the shard catalog that examines the sharded database topology configuration to determine if there are any undeployed shards present that are able to be deployed.

- For shards that need to be deployed, the shard catalog sends requests to the shard director to update database parameters on the shards, populate topology metadata on the shard, and direct the shard to register with the shard director.
- If Oracle Data Guard replication is in use, and standby databases are present to deploy, then the shard director calls PL/SQL APIs on the primary shards to create a Data Guard configuration, or to validate an existing configuration on the primary and standby sets. Fast Start Failover functionality is enabled on all of the shards and, in addition, the shard director starts a Data Guard observer process on its host to monitor the Data Guard configuration.
- If new shards are being added to an existing sharded database that already contains deployed shards (called an incremental deployment), then any DDL statements that have been run previously are run on the new shards to ensure that the application schemas are identical across all of the shards.
- Finally, in the case of an incremental deployment on a sharded database using system-managed or composite sharding methods, automatic chunk movement is scheduled in the background, which is intended to balance the number of chunks distributed among the shards now in the configuration. This process can be monitored using the `GDSCTL CONFIG CHUNKS` command after the `DEPLOY` command returns control to `GDSCTL`.

What Does a Successful Deployment Look Like?

Following a successful deployment, the output from `CONFIG SHARD` should look similar to the following, if Data Guard active standby shards are in use.

```
GDSCTL> config shard
Name      Shard Group          Status   State     Region   Availability
--------- -------------------  -------  --------  -------  ------------
cdb1_pdb1 shardgroup_primary   Ok       Deployed  region1  ONLINE
cdb2_pdb1 shardgroup_standby   Ok       Deployed  region2  READ ONLY
cdb3_pdb2 shardgroup_primary   Ok       Deployed  region1  ONLINE
cdb4_pdb2 shardgroup_standby   Ok       Deployed  region2  READ ONLY
```

If mounted, non-open standbys are in use, the output will be similar to the following, because the shard director is unable to log in to check the status of a mounted database.

```
GDSCTL> config shard
Name      Shard Group          Status        State     Region   Availability
--------- -------------------  ------------- --------  -------  ------------
cdb1_pdb1 shardgroup_primary   Ok            Deployed  region1  ONLINE
cdb2_pdb1 shardgroup_standby   Uninitialized Deployed  region2  -
cdb3_pdb2 shardgroup_primary   Ok            Deployed  region1  ONLINE
cdb4_pdb2 shardgroup_standby   Uninitialized Deployed  region2  -
```

What To Do If Something Is Not Right

If any shards are showing an availability of `PENDING`, confirm that all steps related to `ADD INVITEDNODE` and `CONFIG VNCR` from the topology configuration were completed. If not, complete them now and run `GDSCTL SYNC DATABASE -database` *`shard_name`* to complete shard deployment.

Create and Start Global Database Services

After the shards are successfully deployed, and the correct status has been confirmed, create and start global database services on the shards to service incoming connection requests from your application.

As an example, the commands in the following examples create read-write services on the primary shards in the configuration and read-only services on the standby shards. These service names can then be used in connect strings from your application to appropriately route requests to the correct shards.

Example 4-1 Add and start a global service that runs on all of the primary shards

The following commands create and start a global service named `oltp_rw_srvc` that a client can use to connect to the sharded database. The `oltp_rw_srvc` service runs read/write transactions on the primary shards.

```
GDSCTL> add service -service oltp_rw_srvc -role primary
GDSCTL> start service -service oltp_rw_srvc
```

Example 4-2 Add and start a global service for the read-only workload to run on the standby shards

The `oltp_ro_srvc` global service is created and started to run read-only workloads on the standby shards. This assumes that the standby shards are Oracle Active Data Guard standby shards which are open for read-only access. Mounted, non-open standbys cannot service read-only connections, and exist for disaster recovery and high availability purposes only.

```
GDSCTL> add service -service oltp_ro_srvc -role physical_standby
GDSCTL> start service -service oltp_ro_srvc
```

Example 4-3 Verify the status of the global services

```
GDSCTL> config service

Name         Network name                    Pool    Started Preferred
all
----         ------------                    ----    -------
-------------
oltp_rw_srvc oltp_rw_srvc.orasdb.oracdbcloud orasdb  Yes     Yes
oltp_ro_srvc oltp_ro_srvc.orasdb.oracdbcloud orasdb  Yes     Yes


GDSCTL> status service
Service "oltp_rw_srvc.orasdb.oradbcloud" has 2 instance(s). Affinity:
ANYWHERE
   Instance "orasdb%1", name: "cdb1_pdb1", db: "cdb1_pdb1", region:
"region1", status: ready.
   Instance "orasdb%21", name: "cdb3_pdb2", db: "cdb3_pdb2", region:
"region1", status: ready.
Service "oltp_ro_srvc.orasdb.oradbcloud" has 2 instance(s). Affinity:
```

```
ANYWHERE
   Instance "orasdb%11", name: "cdb2_pdb1", db: "cdb2_pdb1", region:
"region2", status: ready.
   Instance "orasdb%31", name: "cdb4_pdb2", db: "cdb4_pdb2", region:
"region2", status: ready.
```

Verify Shard Status

Once you complete the `DEPLOY` step in your Oracle Globally Distributed Database configuration deployment, verify the detailed status of a shard

- Run `GDSCTL CONFIG SHARD` to see the detailed status of each shard.

```
GDSCTL> config shard -shard cdb1_pdb1
Name: cdb1_pdb1
Shard Group: shardgroup_primary
Status: Ok
State: Deployed
Region: region1
Connection string:shard_connect_string
SCAN address:
ONS remote port: 0
Disk Threshold, ms: 20
CPU Threshold, %: 75
Version: 23.0.0.0
Failed DDL:
DDL Error: ---
Management error:
Failed DDL id:
Availability: ONLINE
Rack:


Supported services
------------------------
Name Preferred Status
---- --------- ------
oltp_ro_srvc Yes Enabled
oltp_rw_srvc Yes Enabled
```

Example Oracle Globally Distributed Database Deployment

This example explains how to deploy a typical system-managed Oracle Globally Distributed Database with multiple replicas, using Oracle Data Guard for high availability.

To deploy a system-managed sharded database you create shardgroups and shards, create and configure the databases to be used as shards, run the `DEPLOY` command, and create role-based global services.

You are not required to map data to shards in the system-managed sharding method, because the data is automatically distributed across shards using partitioning by consistent hash. The partitioning algorithm evenly and randomly distributes data across shards. For

more conceptual information about the system-managed sharding method, see System-Managed Sharding.

Example Oracle Globally Distributed Database Topology

Consider the following system-managed Oracle Globally Distributed Database configuration, where shardgroup sg1 contains the primary shards, while shardgroups sg2 and sg3 contain standby replicas.

In addition, let's assume that the replicas in shardgroup sg2 are Oracle Active Data Guard standbys (that is, databases open for read-only access), while the replicas in shardgroup sg3 are mounted databases that have not been opened.

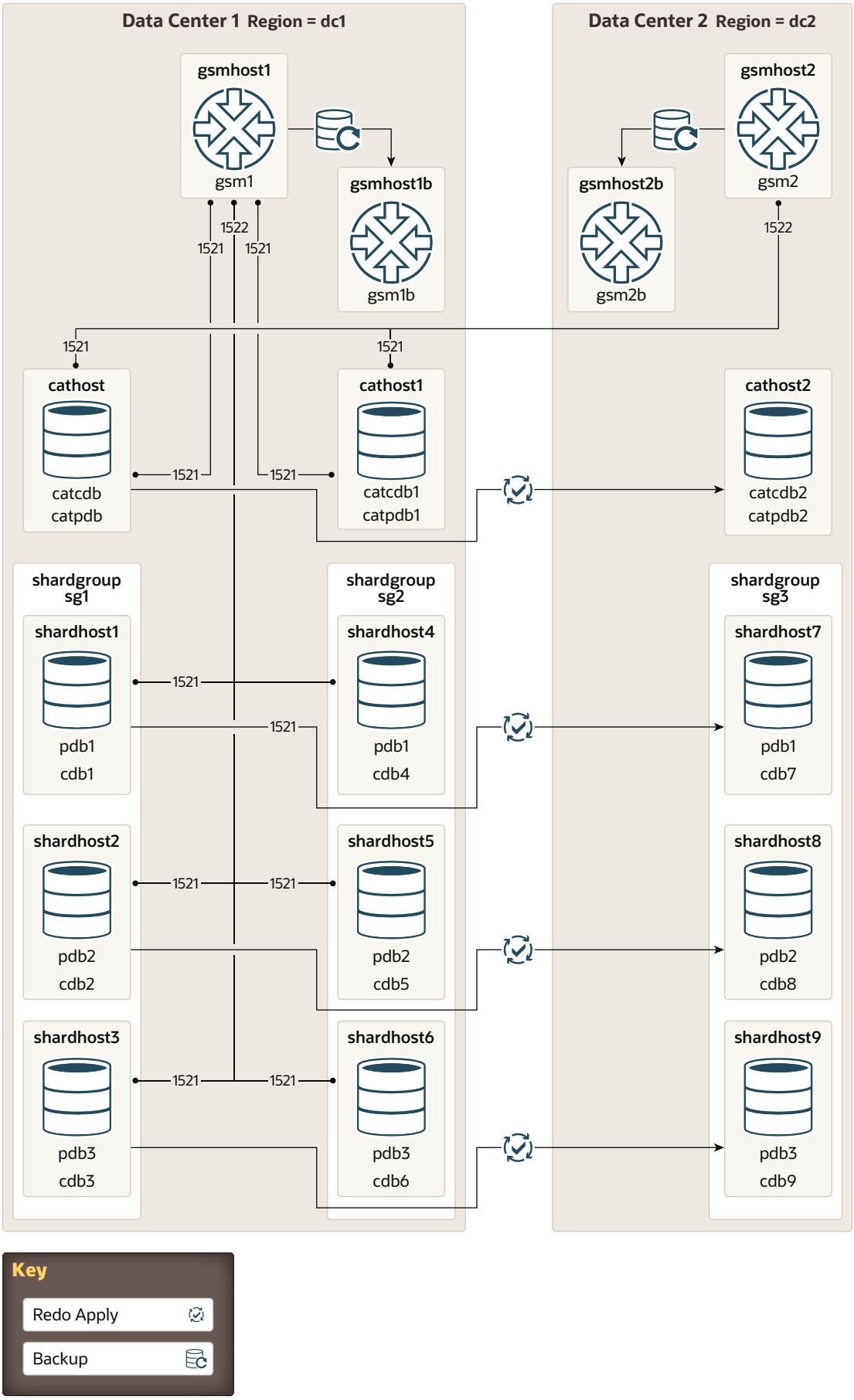

Data Center 1 Region = dc1
Data Center 2 Region = dc2
gsmhost1
gsm1
gsmhost1b
gsm1b
gsmhost2
gsm2
gsmhost2b
gsm2b
1522
1521
1521
1522
1521
1521
cathost
catcdb
catpdb
cathost1
catcdb1
catpdb1
cathost2
catcdb2
catpdb2
1521
1521
shardgroup sg1
shardgroup sg2
shardgroup sg3
shardhost1
pdb1
cdb1
shardhost4
pdb1
cdb4
shardhost7
pdb1
cdb7
shardhost2
pdb2
cdb2
shardhost5
pdb2
cdb5
shardhost8
pdb2
cdb8
shardhost3
pdb3
cdb3
shardhost6
pdb3
cdb6
shardhost9
pdb3
cdb9
Key
Redo Apply
Backup

Table 4-1 Example System-Managed Topology Host Names

Topology Object	Description
Shard Catalog Database	Every sharded database topology requires a shard catalog. In our example, the shard catalog database has 2 standbys, one in each data center. Primary • Data center = 1 • Host name = cathost • DB_UNIQUE_NAME = catcdb • PDB name = catpdb • Connect service name = catpdb Active Standby • Data center = 1 • Host name = cathost1 Standby • Data center = 2 • Host name = cathost2
Regions	Because there are two data centers involved in this configuration, there are two corresponding regions created in the shard catalog database. Data center 1 • Region name = dc1 Data center 2 • Region name = dc2
Shard Directors (global service managers)	Each region requires a shard director running on a host within that data center. Data center 1 • Shard director host name = gsmhost1 • Shard director name = gsm1 Data center 2 • Shard director hast name = gsmhost2 • Shard director name = gsm2
Shardgroups	Data center 1 • sg1 • sg2 Data center 2 • sg3
Shards	• Host names = shardhost1, …, shardhost9 • DB_UNIQUE_NAME = cdb1, …, cdb9 • PDB names = pdb1, pdb2, pdb3 PDB names on standby replicas are the same as the PDB names on their corresponding primaries

Deploy the Example Oracle Globally Distributed Database

Do the following steps to deploy the example system-managed sharded database with multiple replicas, using Oracle Data Guard for high availability.

1. Provision and configure the following hosts: cathost, cathost1, cathost2, gsmhost1, gsmhost2, and hosts shardhost1 through shardhost9.

   See Provision and Configure Hosts and Operating Systems for details.

2. Install the Oracle Database software on the following hosts: cathost, cathost1, cathost2, and shardhost1 through shardhost9.

   See Install the Oracle Database Software for details.

3. Install the shard director software on hosts gsmhost1 and gsmhost2.

   See Install the Shard Director Software for details.

4. Create the shard catalog database and start an Oracle TNS Listener on cathost.

   Additionally, create standby replicas of the catalog on cathost1 and cathost2, and verify that changes made to the primary catalog are applied on these standbys.

   See Create the Shard Catalog Database for details.

5. Create the 3 primary databases that will contain the sharded data on hosts shardhost1, shardhost2 and shardhost3.

   Create the corresponding replicas, located and named as listed here.

   - shardhost1 (cdb1/pdb1) replicas on shardhost4 (cdb4) and shardhost7 (cdb7)
   - shardhost2 (cdb2/pdb2) replicas on shardhost5 (cdb5) and shardhost8 (cdb8)
   - shardhost3 (cdb3/pdb3) replicas on shardhost6 (cdb6) and shardhost9 (cdb9)

   The db_unique_name of the 9 container databases (CDB) should be cdb1 through cdb9, in which the PDB names should be pdb1, pdb2 and pdb3 on the three primaries and their replicas.

   The service names for the CDBs should be cdb1 through cdb9, which the service names for the PDB shards are pdb1, pdb2, and pdb3.

   See Create the Shard Databases for details.

6. Assuming that all port numbers are the defaults, to configure the sharded database topology, issue the following `GDSCTL` commands, replacing domains and passwords with the appropriate values.

   a. On host gsmhost1, run the following commands in `GDSCTL`.

   ```
   create shardcatalog -database cathost.example.com:1521/
   catpdb.example.com -user mydbsadmin/mydbsadmin_password -region
   dc1,dc2


   add gsm -gsm gsm1 -region dc1 -catalog cathost.example.com:1521/
   catpdb.example.com -pwd gsmcatuser_password
   start gsm -gsm gsm1
   ```

   See Create the Shard Catalog and Add and Start Shard Directors for details.

b. On host gsmhost2, run the following commands in GDSCTL.

```
connect mydbsadmin/mydbsadmin_password@cathost.example.com:1521/
catpdb.example.com
add gsm -gsm gsm2 -region dc2 -catalog cathost.example.com:1521/
catpdb.example.com -pwd gsmcatuser_password
start gsm -gsm gsm2
```

See Add and Start Shard Directors for details.

c. Back on host gsmhost1, run the following from GDSCTL to complete the sharded database setup.

```
add shardgroup -shardgroup sg1 -deploy_as primary -region dc1
add shardgroup -shardgroup sg2 -deploy_as active_standby -region
dc1
add shardgroup -shardgroup sg3 -deploy_as standby -region dc2
add cdb -connect shardhost1.example.com:1521/cdb1.example.com -
pwd gsmrootuser_password
add cdb -connect shardhost2.example.com:1521/cdb2.example.com -
pwd gsmrootuser_password
```

Repeat the ADD CDB command for shardhost3 through shardhost9 and cdb3 through cdb9, then run the following commands.

```
add shard -connect shardhost1.example.com:1521/pdb1.example.com -
pwd gsmuser_password -shardgroup sg1 -cdb cdb1
add shard -connect shardhost2.example.com:1521/pdb2.example.com -
pwd gsmuser_password -shardgroup sg1 -cdb cdb2
add shard -connect shardhost3.example.com:1521/pdb3.example.com -
pwd gsmuser_password -shardgroup sg1 -cdb cdb3
add shard -connect shardhost4.example.com:1521/pdb1.example.com -
pwd gsmuser_password -shardgroup sg2 -cdb cdb4
add shard -connect shardhost5.example.com:1521/pdb2.example.com -
pwd gsmuser_password -shardgroup sg2 -cdb cdb5
add shard -connect shardhost6.example.com:1521/pdb3.example.com -
pwd gsmuser_password -shardgroup sg2 -cdb cdb6
add shard -connect shardhost7.example.com:1521/pdb1.example.com -
pwd gsmuser_password -shardgroup sg3 -cdb cdb7
add shard -connect shardhost8.example.com:1521/pdb2.example.com -
pwd gsmuser_password -shardgroup sg3 -cdb cdb8
add shard -connect shardhost9.example.com:1521/pdb3.example.com -
pwd gsmuser_password -shardgroup sg3 -cdb cdb9
```

See Add Shardgroups If Needed, Add the Shard CDBs, and Add the Shard PDBs for details.

d. Use the CONFIG VNCR and ADD INVITEDNODE commands to validate that all of the VNCR entries are valid and sufficient for a successful deployment.

See Add Host Metadata for details.

e. Run DEPLOY from GDSCTL to complete the configuration of the sharded database.

See Deploy the Configuration for details.

f. Add and start services for read-write and read-only access to the sharded database.

```
add service -service oltp_rw_srvc -role primary
start service -service oltp_rw_srvc
add service -service oltp_ro_srvc -role physical_standby
start service -service oltp_ro_srvc
```

See Create and Start Global Database Services for details.

7. You can use the `GDSCL CONFIG`, `CONFIG SHARD`, and `CONFIG SERVICE` commands to validate that all of the shards and services are online and running.

See Verify Shard Status for details.

5

Using Oracle Globally Distributed Database in Oracle Cloud Infrastructure

Tooling for Oracle Globally Distributed Database includes Terraform, Kubernetes, and Docker scripts to automate and further simplify the sharded database deployment operations.

Deploy an Oracle Globally Distributed Database on Kubernetes

Automate the provisioning of sharded databases on Oracle Kubernetes Engine (OKE) using Oracle Cloud Infrastructure Ansible Modules and Helm/Chart.

To deploy Oracle Globally Distributed Database on OKE, Oracle Cloud Infrastructure Ansible Modules create compute resources, configure the network, and create block storage volumes by using YAML files passed to Ansible playbooks.

Find the instructions and downloads for sharded database deployment on Kubernetes at https://github.com/oracle/db-sharding/tree/master/oke-based-sharding-deployment.

Deploy an Oracle Globally Distributed Database with Terraform

Tooling for Oracle Globally Distributed Database includes Terraform modules and scripts to automate your sharded database deployment on both Oracle Cloud Infrastructure and on-premises systems.

The Terraform modules and scripts create and configure a complete sharded database infrastructure, including shard directors, shard catalogs, and shards. The scripts also provide the option to deploy standby shards and shard catalogs using Oracle Data Guard for replication to provide high availability and disaster recovery of the sharded data.

As part of the set-up process, you install the Terraform binary, download the Oracle Globally Distributed Database shard director installation package, and for on-premises deployments, you download the Oracle Database installation files.

Find the instructions and downloads for Terraform-based sharded database deployment for your target systems at the following locations.

- **Oracle Cloud Infrastructure** https://github.com/oracle/db-sharding/tree/master/deployment-with-terraform/sdb-terraform-oci.
- **On-Premises** https://github.com/oracle/db-sharding/tree/master/deployment-with-terraform/sdb-terraform-onprem

Deploy an Oracle Globally Distributed Database with Docker

Oracle Globally Distributed Database provides sample Docker build files to facilitate sharded database installation, configuration, and environment setup for DevOps users.

In this process you install and configure the Docker engine, create global service manager (shard director) and Oracle Database images, create a network bridge, create containers for

the Oracle Globally Distributed Database objects and shard director, and deploy the containers.

Find the instructions and downloads for sharded database deployment with Docker at https://github.com/oracle/db-sharding/tree/master/docker-based-sharding-deployment.

6

Oracle Globally Distributed Database Schema Design

To obtain the benefits of Oracle Globally Distributed Database, the schema should be designed in a way that maximizes the number of database requests processed on a single shard.

- Schema Design Considerations
- Sharding Keys
- Creating Schema Objects
- Creating Indexes on Sharded Tables
- Modifying an Oracle Globally Distributed Database Schema
- DDL Processing in an Oracle Globally Distributed Database
- Running PL/SQL Procedures in an Oracle Globally Distributed Database
- Pushing PL/SQL Function Queries to the Shards
- Generating Unique Sequence Numbers Across Shards
- Schema Creation Examples
- DDL Failure and Recovery Examples

Schema Design Considerations

Design of the Oracle Globally Distributed Database schema has a big impact on performance and scalability. An improperly designed schema can lead to unbalanced distribution of data and workload across shards and large percentage of multi-shard operations.

The data model should be a hierarchical tree structure with a single root table. Oracle Globally Distributed Database supports any number of levels within the hierarchy.

To obtain the benefits of sharding, the schema of a sharded database should be designed in a way that maximizes the number of database requests processed on a single shard.

A sharded database schema consists of a sharded table family and duplicated tables with the following characteristics.

Sharded table family

- A set of tables which are equi-partitioned by the sharding key.
  - Related data is always stored and moved together.
  - Joins and integrity constraint checks are done within a shard.
- The sharding method and key are based on the application's requirements.
- The sharding key must be included in the primary key.

Duplicated tables

- Non-sharded tables which are replicated to all shards.
- Usually contain common reference data.
- Can be read and updated on each shard.

Planning a Sharded Database Schema Design

Once the sharded database is populated with data, it is impossible to change many attributes of the schema, such as whether a table is sharded or duplicated, sharding key, and so on. Therefore, the following points should be carefully considered before deploying a sharded database.

- Which tables should be sharded?
- Which tables should be duplicated?
- Which sharded table should be the root table?
- What method should be used to link other tables to the root table?
- Which sharding method should be used?
- Which sharding key should be used?
- Which super sharding key should be used (if the sharding method is composite)?

Sharding Keys

Tips for choosing sharding keys, information about constraints rules, and enabling data movement between shards on sharding key updates.

Choosing Sharding Keys

Sharded table partitions are distributed across shards at the tablespace level, based on a sharding key. Examples of keys include customer ID, account number, and country ID.

Sharding keys must adhere to the following characteristics.

- The sharding key should be very stable; its value should almost never change.
- The sharding key must be present in all of the sharded tables. This allows the creation of a family of equi-partitioned tables based on the sharding key.
- Joins between tables in a table family should be performed using the sharding key.

Sharding Keys for System-Managed Sharded Databases

For the system-managed sharding method, the sharding key must be based on a column that has high cardinality; the number of unique values in this column must be much bigger than the number of shards. Customer ID, for example, is a good candidate for the sharding key, while a United States state name is not.

A sharding key can be a single column or multiple columns. When multiple columns are present, the hash of the columns are concatenated to form the sharding key.

The following examples create a sharded table called Customers and specify that columns `cust_id` and `name` form the sharding keys for the table.

```
CREATE SHARDED TABLE customers
(cust_id     NUMBER NOT NULL
```

```
, name        VARCHAR2(50)
, address     VARCHAR2(250)
, region      VARCHAR2(20)
, class       VARCHAR2(3)
, signup      DATE,
CONSTRAINT cust_pk PRIMARY KEY(cust_id, name))
PARTITION BY CONSISTENT HASH (cust_id,name)
PARTITIONS AUTO
TABLESPACE SET ts1;


CREATE SHARDED TABLE Orders
( OrderNo   NUMBER NOT NULL
, CustNo    NUMBER NOT NULL
, Name      VARCHAR2(50) NOT NULL
, OrderDate DATE
, CONSTRAINT OrderPK PRIMARY KEY (CustNo, Name, OrderNo)
, CONSTRAINT CustFK  FOREIGN KEY (CustNo, Name) REFERENCES
Customers(Cust_ID, Name)
)
PARTITION BY REFERENCE (CustFK);
```

Sharding Keys for Composite Sharded Databases

Composite sharding enables two levels of sharding - one by list or range and another by consistent hash. This is accomplished by the application providing two keys: a super sharding key and a sharding key.

Composite sharding does not support multi-column `LIST` partitionsets, as shown here.

```
CREATE SHARDED TABLE customers (
cust_id     NUMBER NOT NULL,
Name    VARCHAR2(50) NOT NULL,
class VARCHAR2(3) NOT NULL ,
class2 number not null,
CONSTRAINT cust_pk PRIMARY KEY(cust_id,name,class))
PARTITIONSET BY LIST (class, class2)
PARTITION BY CONSISTENT HASH (cust_id,name)
PARTITIONS AUTO (
PARTITIONSET silver VALUES (('SLV',1),('BRZ',2)) TABLESPACE SET ts1
PARTITIONSET gold   VALUES (('GLD',3),('OTH',4)) TABLESPACE SET ts2);

PARTITION BY CONSISTENT HASH (cust_id,name)
*
ERROR at line 8:
ORA-02514: list PARTITIONSET method expects a single partitioning column
```

Multi-column `RANGE` partitionsets are supported, as shown below.

```
CREATE SHARDED TABLE customers (
cust_id     NUMBER NOT NULL,
Name    VARCHAR2(50) NOT NULL,
class number NOT NULL ,
```

```
class2 number not null,
CONSTRAINT cust_pk PRIMARY KEY(cust_id,name,class))
PARTITIONSET BY RANGE (class, class2)
PARTITION BY CONSISTENT HASH (cust_id,name)
PARTITIONS AUTO (
PARTITIONSET silver VALUES LESS THAN (10,100) TABLESPACE SET ts1,
PARTITIONSET gold   VALUES LESS THAN (20,200) TABLESPACE SET ts2);

Table created.
```

In both of the above cases, the sharding key (not the partitionset key) can be multi-column.

Sharding Keys for User-Defined Sharded Databases

For partition by list in user-defined sharding, Oracle Globally Distributed Database expects a single sharding key column. An error is thrown when multiple columns are specified for a list-partitioned sharded table.

```
CREATE SHARDED TABLE accounts
( id             NUMBER
, account_number NUMBER
, customer_id    NUMBER
, branch_id      NUMBER
, state          VARCHAR(2) NOT NULL
, state2         VARCHAR(2) NOT NULL
, status         VARCHAR2(1)
)
PARTITION BY LIST (state,state2)
( PARTITION p_northwest VALUES ('OR', 'WA') TABLESPACE ts1
, PARTITION p_southwest VALUES ('AZ', 'UT', 'NM') TABLESPACE ts2
, PARTITION p_northcentral VALUES ('SD', 'WI') TABLESPACE ts3
, PARTITION p_southcentral VALUES ('OK', 'TX') TABLESPACE ts4
, PARTITION p_northeast VALUES ('NY', 'VM', 'NJ') TABLESPACE ts5
, PARTITION p_southeast VALUES ('FL', 'GA') TABLESPACE ts6
);

ERROR at line 1:
ORA-03813: list partition method expects a single partitioning column
in
user-defined sharding
```

For a range-partitioned sharded table, you can specify multiple columns as sharding key columns.

```
CREATE SHARDED TABLE accounts
( id             NUMBER
, account_number NUMBER
, customer_id    NUMBER
, branch_id      NUMBER
, state          NUMBER NOT NULL
, state2         NUMBER NOT NULL
, status         VARCHAR2(1)
```

```
)
PARTITION BY RANGE (state, state2)
( PARTITION p_northwest VALUES LESS THAN(10, 100) TABLESPACE ts1
, PARTITION p_southwest VALUES LESS THAN(20,200) TABLESPACE ts2);

Table created.
```

But in both cases, the sharding key (not the partitionset key) can be multi-column.

Sharding Key Type Support

The following data types are supported for the sharding key.

- NUMBER
- INTEGER
- SMALLINT
- RAW
- (N)VARCHAR
- (N)VARCHAR2
- (N)CHAR
- DATE
- TIMESTAMP

Primary Key and Foreign Key Constraints

In a Oracle Globally Distributed Database environment, the primary key constraints and foreign key constraints are controlled by the following rules.

- For primary keys, there are unique constraints and unique indexes on sharded tables; the column list must contain the sharding key columns. In earlier Oracle releases the restriction was that the sharding key must be a prefix of such columns, but this rule is now more relaxed.
- Foreign keys from one sharded table to another sharded table also must contain the sharding key. This is automatically enforced because a foreign key refers to either the primary key or unique columns of the referenced table.
- Foreign keys on sharded tables must be within the same table family. This is required because different table families have different sharding key columns.
- Foreign keys in sharded tables referencing local tables are not allowed.
- Foreign keys in sharded tables referencing duplicated tables are not allowed.
- Foreign keys in duplicated table referencing sharded tables are not allowed.

Enabling Automatic Data Movement on Sharding Key Update

You can update the sharding key for any particular record directly on the shard where the data is located, using a normal `SQL UPDATE` statement. Oracle Globally Distributed Database moves the data to the correct shard automatically, as a distributed transaction in the background.

> **Note:**
>
> It is recommended that you commit your previous work and start a new transaction before you update a sharding key, because a distributed transaction has a higher risk than local transaction in the case of a remote machine or network failure. To avoid race condition, the rows being updated are locked before starting the insertion-deletion operation.

Use Case

Sometimes a sharding key value for a particular record must be updated. For example, employee location can be one of the sharding keys. When employees move from one country to another, their country value must be updated, in which case the data must be moved to the shard mapped to the new key value.

What happens when I update a sharding key?

When a record's sharding key value is updated in a sharded table, the record could end up in three possible locations after the update, depending on which shard the new value is mapped to.

1. The row stays within the same partition.

2. The row is moved to a different partition in the same shard.

3. The row is moved to a different shard.

When the sharding key value on a particular row of a sharded table is updated, Oracle Globally Distributed Database handles moving the data to a new location, whether it is in a different partition on the same shard or on a different shard.

Support for Sharded Table Family

A *table family* is a parent-child relationship between database tables. Multiple tables linked by such relationships typically form a tree-like hierarchy where every child has a single parent. A table family can be defined using reference partitioning or the `PARENT` clause.

Support for automatic row movement in a table family which was created with the `PARENT` clause, requires primary key-foreign key constraints to be added between the parent and child table.

In a table family which was created with reference partitioning, the primary key-foreign key constraints between the parent and child table already exist, so automatic row movement is supported and no extra step is needed.

Enable and Disable Automatic Data Movement

This operation uses the `ROW MOVEMENT` clause on the sharded table in the database.

```
ALTER TABLE tablename ENABLE ROW MOVEMENT;
```

```
ALTER TABLE tablename DISABLE ROW MOVEMENT;
```

`ROW MOVEMENT` can also be specified on `CREATE TABLE`.

When `ROW MOVEMENT` is enabled and there is an update to the sharding key value, the data is transparently moved between shards.

For details about the `ROW MOVEMENT` clause, see row_movement_clause in *Oracle Database SQL Language Reference*.

Enable Automatic Data Movement in a Table Family

In the case of a table family, `ROW MOVEMENT` must be enabled on the child table first, and then on the parent table.

In the example below, `accounts` is the root table, `orders` is a child of `accounts`, `lineitems` is a child of `orders`.

```
ALTER TABLE lineitems ENABLE ROW MOVEMENT;

ALTER TABLE orders ENABLE ROW MOVEMENT;

ALTER TABLE accounts ENABLE ROW MOVEMENT;
```

Creating Schema Objects

The following topics show you how to create the schema objects in your Oracle Globally Distributed Database.

Refer back to Oracle Globally Distributed Database Schema Objects for conceptual information about these objects.

Create an All-Shards User

Local users that only exist in the shard catalog database do not have the privileges to create schema objects in the Oracle Globally Distributed Database. The first step of creating the sharded database schema is to create an **all-shards** user.

Create an all-shards user by connecting to the shard catalog database as a privileged user, enabling `SHARD DDL`, and running the `CREATE USER` command. When the all-shards user connects to the shard catalog database, the `SHARD DDL` mode is enabled by default.

> **Note:**
>
> Local users can create non-schema sharded database objects, such as tablespaces, directories, and contexts, if they enable `SHARD DDL` mode; however, they cannot create schema objects, such as tables, views, indexes, functions, procedures, and so on.
>
> Sharded objects cannot have any dependency on local objects. For example, you cannot create an all-shard view on a local table.
>
> You cannot grant `SYS` privileges to sharded users using sharded DDL. You must log in to each shard and grant the privilege to the account manually on that shard.

Creating a Sharded Table Family

Create a sharded table family with the `SQL CREATE TABLE` statement. You can specify parent-child relationships between tables using reference partitioning or equi-partitioning.

Use Reference Partitioning to Specify Parent-Child Relationships Between Tables

The recommended way to create a sharded table family is to specify parent-child relationships between tables using reference partitioning.

Partitioning by reference simplifies the syntax since the partitioning scheme is only specified for the root table. Also, partition management operations that are performed on the root table are automatically propagated to its descendents. For example, when adding a partition to the root table, a new partition is created on all its descendents.

The appropriate `CREATE TABLE` statements for Customers–Orders–LineItems schema using a system-managed sharding methodology are shown below. The first statement creates the root table of the table family, Customers.

```
CREATE SHARDED TABLE Customers
( CustNo      NUMBER NOT NULL
, Name        VARCHAR2(50)
, Address     VARCHAR2(250)
, CONSTRAINT RootPK PRIMARY KEY(CustNo)
)
PARTITION BY CONSISTENT HASH (CustNo)
PARTITIONS AUTO
TABLESPACE SET ts1
;
```

The following two statements create the Orders and LineItems tables, which are a child and grandchild of the Customers table.

```
CREATE SHARDED TABLE Orders
( OrderNo   NUMBER NOT NULL
, CustNo    NUMBER NOT NULL
, OrderDate DATE
, CONSTRAINT OrderPK PRIMARY KEY (CustNo, OrderNo)
, CONSTRAINT CustFK  FOREIGN KEY (CustNo) REFERENCES Customers(CustNo)
)
PARTITION BY REFERENCE (CustFK)
;


CREATE SHARDED TABLE LineItems
( CustNo    NUMBER NOT NULL
, LineNo    NUMBER(2) NOT NULL
, OrderNo   NUMBER(5) NOT NULL
, StockNo   NUMBER(4)
, Quantity  NUMBER(2)
, CONSTRAINT LinePK  PRIMARY KEY (CustNo, OrderNo, LineNo)
, CONSTRAINT LineFK  FOREIGN KEY (CustNo, OrderNo) REFERENCES
Orders(CustNo, OrderNo)
)
PARTITION BY REFERENCE (LineFK)
;
```

In the example statements above, corresponding partitions of all tables in the family are stored in the same tablespace set, TS1. However, it is possible to specify separate tablespace sets for each table.

Note that in the example statements above, the partitioning column `CustNo` used as the sharding key is present in all three tables. This is despite the fact that reference partitioning, in general, allows a child table to be equi-partitioned with the parent table without having to duplicate the key columns in the child table. The reason for this is that reference partitioning requires a primary key in a parent table because the primary key must be specified in the foreign key constraint of a child table used to link the child to its parent. However, a primary key on a sharded table must be the same as, or contain, the sharding key. This makes it possible to enforce global uniqueness of a primary key without coordination with other shards, a critical requirement for linear scalability.

To summarize, the use of reference-partitioned tables in a sharded database requires adhering to the following rules:

- A primary key on a sharded table must either be the same as the sharding key, or contain the sharding key. This is required to enforce global uniqueness of a primary key without coordination with other shards.
- Reference partitioning requires a primary key in a parent table, because the primary key must be specified in the foreign key constraint of a child table to link the child to its parent. It is also possible to have a foreign key constraint when the parent table has just `UNIQUE` constraint, but no `PRIMARY KEY`. The sharding key must also be `NOT NULL`.

  For example, to link the LineItems (child) table to the Orders (parent) table, you need a primary key in the Orders table. The second rule implies that the primary key in the Orders table contains the `CustNo` value. (This is an existing partitioning rule not specific to Oracle Globally Distributed Database.)

Use Equi-Partitioning to Specify Parent-Child Relationships Between Tables

In some cases it is impossible or undesirable to create primary and foreign key constraints that are required for reference partitioning. For such cases, specifying parent-child relationships in a table family requires that all tables are explicitly equi-partitioned. Each child table is created with the `PARENT` clause in `CREATE SHARDED TABLE` that contains the name of its parent. An example of the syntax is shown below.

```
 CREATE SHARDED TABLE Customers
( CustNo      NUMBER NOT NULL
, Name        VARCHAR2(50)
, Address     VARCHAR2(250)
, region      VARCHAR2(20)
, class       VARCHAR2(3)
, signup      DATE
)
PARTITION BY CONSISTENT HASH (CustNo)
PARTITIONS AUTO
TABLESPACE SET ts1
;

CREATE SHARDED TABLE Orders
( OrderNo   NUMBER
, CustNo    NUMBER NOT NULL
, OrderDate DATE
)
PARENT Customers
PARTITION BY CONSISTENT HASH (CustNo)
PARTITIONS AUTO
TABLESPACE SET ts1
```

```
;

CREATE SHARDED TABLE LineItems
( LineNo    NUMBER
, OrderNo   NUMBER
, CustNo    NUMBER NOT NULL
, StockNo   NUMBER
, Quantity  NUMBER
)
PARENT Customers
PARTITION BY CONSISTENT HASH (CustNo)
PARTITIONS AUTO
TABLESPACE SET ts1
;
```

Because the partitioning scheme is fully specified in all of the `CREATE SHARDED TABLE` statements, any table can be independently subpartitioned. This is not permitted with reference partitioning where subpartitions can only be specified for the root table and the subpartitioning scheme is the same for all tables in a table family.

Note that this method only supports two-level table families, that is, all children must have the same parent and grandchildren cannot exist. This is not a limitation as long as the partitioning column from the parent table exists in all of the child tables.

> **See Also:**
>
> *Oracle Database VLDB and Partitioning Guide* for information about reference partitioning

Designing Schemas With Multiple Table Families

An Oracle Globally Distributed Database schema can have multiple table families, where all of the data from different table families reside in the same chunks, which contain partitions from different table families sharing the same hash key range.

> **Note:**
>
> Multiple table families are supported in system-managed sharded databases only. Composite and user-defined sharded databases only support one table family.

To create a new table family, create a root sharded table and specify tablespace sets that are not used by existing tablespace families. Each table family is identified by its root table. Tables in the different table families should not be related to each other.

Each table family should have its own sharding key definition, while the same restriction on having the same sharding key columns in child tables still holds true within each table family. This means that all tables from different table families are sharded the same way with consistent hash into the same number of chunks, with each chunk containing data from all the table families.

Design your table families such that queries between different table-families are minimal and only carried out on the sharding coordinator, as many such joins will have an effect on performance

The following example shows you how to create multiple table families using the `PARENT` clause with a system-managed sharding methodology (`PARTITION BY CONSISTENT HASH`).

```
CREATE SHARDED TABLE Customers <=== Table Family #1
( CustId NUMBER NOT NULL
, Name VARCHAR2(50)
, Address VARCHAR2(250)
, region VARCHAR2(20)
, class VARCHAR2(3)
, signup DATE
)
PARTITION BY CONSISTENT HASH (CustId)
PARTITIONS AUTO
TABLESPACE SET ts1
;

CREATE SHARDED TABLE Orders
( OrderNo NUMBER
, CustId NUMBER
, OrderDate DATE
)
PARENT Customers
PARTITION BY CONSISTENT HASH (CustId)
PARTITIONS AUTO
TABLESPACE SET ts1
;

CREATE SHARDED TABLE LineItems
( LineNo NUMBER
, OrderNo NUMBER
, CustId NUMBER
, StockNo NUMBER
, Quantity NUMBER
)
)
PARENT Customers
PARTITION BY CONSISTENT HASH (CustId)
PARTITIONS AUTO
TABLESPACE SET ts1
;

CREATE SHARDED TABLE Products <=== Table Family #2
( ProdId NUMBER NOT NULL,
  CONSTRAINT pk_products PRIMARY KEY (ProdId)
)
PARTITION BY CONSISTENT HASH (ProdId)
PARTITIONS AUTO
TABLESPACE SET ts_2
;
```

> **Note:**
>
> ORA-3850 is thrown if you attempt to use a tablespace set for a table family, but that tablespace set is already in use by an existing table family.
>
> Joins across table families may not be efficient, and if you have many such joins, or if they are performance-critical, you should use duplicated tables instead of multiple table families.

Associating Global Services With Multiple Table Families

Each table family should be associated with a different global service. Applications from different table families each have their own connection pool and service, and use their own sharding key for routing to the correct shard.

When you create the first root table (that is, the first table family) all of the existing global services are automatically associated with it. You can use the `GDSCTL MODIFY SERVICE` command to change the services associated with a table family after more table families are created, as shown in this example.

```
GDSCTL> MODIFY SERVICE -GDSPOOL shdpool -TABLE_FAMILY sales.customer -
SERVICE sales
```

Creating Sharded Tables

A sharded table is a table that is partitioned into smaller and more manageable pieces among multiple databases, called shards.

The following topics guide your decisions and provide instructions for creating sharded tables:

- Tablespace Set Sizing
- Sharded Tables for System-Managed Sharding
- Sharded Tables for User-Defined Sharding
- Sharded Tables for Composite Sharding
- Sharded Tables for Directory-Based Sharding

Tablespace Set Sizing

When you create a tablespace set on the shard catalog, you must make sure you have enough space for the tablespaces created on the shard catalog and on each of the shards.

This is especially important in a metered usage environment.

For example, with a shard catalog and three shards in the configuration, you issue the following statements.

```
ALTER SESSION ENABLE SHARD DDL;
CREATE TABLESPACE SET TSP_SET_1 IN SHARDSPACE SHSPC_1 USING TEMPLATE
 (DATAFILE SIZE 100M  AUTOEXTEND ON NEXT 1M MAXSIZE UNLIMITED);
```

For example, assuming a default of 120 chunks per shard, the command creates the following objects in these configurations:

- System sharding:

  120 tablespaces (for 120 chunks on each shard)

  + 1 tablespace in the shard catalog

  = 120 + 1 tablespace in each shard, with initial tables space 100M

- Raft replication:

  360 tablespaces (for 360 chunks on each shard, because there are 3 chunks created for each replication unit)

  + 1 tablespace in the shard catalog

  = 360 + 1 tablespace in each shard, with initial tables space 100M

If the required amount of storage is not planned for, this can lead to a failed DDL, and that will require significant effort to recover from.

To prevent this issue, you must set the database initialization parameter `DB_FILES` greater than or equal to the total number of chunks and/or tablespace sets required in the shard. Find a formula for calculating `DB_FILES` in Create the Shard Databases.

Also, note that all tablespaces in a tablespace set are **bigfile tablespaces**. A bigfile tablespace is a tablespace with a single, but potentially very large (up to 4G blocks) data file. See Bigfile Tablespaces in *Oracle Database Administrator's Guide* for details.

Sharded Tables for System-Managed Sharding

In a system-managed sharded database, data is automatically distributed across the shards using partitioning by consistent hash.

Before creating a sharded table, create a tablespace set with `CREATE TABLESPACE SET` to store the table partitions.

```
CREATE TABLESPACE SET ts1;
```

If you need to customize the tablespace attributes, add the `USING TEMPLATE` clause to `CREATE TABLESPACE SET` as shown in this example.

```
CREATE TABLESPACE SET ts1
USING TEMPLATE
( DATAFILE SIZE 10M
  EXTENT MANAGEMENT LOCAL UNIFORM SIZE 256K
  SEGMENT SPACE MANAGEMENT AUTO
  ONLINE
)
;
```

You create a sharded table with `CREATE SHARDED TABLE`, horizontally partitioning the table across the shards based on the sharding key `cust_id`.

```
CREATE SHARDED TABLE customers
( cust_id     NUMBER NOT NULL
```

```
, name        VARCHAR2(50)
, address     VARCHAR2(250)
, region      VARCHAR2(20)
, class       VARCHAR2(3)
, signup      DATE
CONSTRAINT cust_pk PRIMARY KEY(cust_id)
)
PARTITION BY CONSISTENT HASH (cust_id)
PARTITIONS AUTO
TABLESPACE SET ts1
;
```

A system-managed sharded table is partitioned by consistent hash, by specifying `PARTITION BY CONSISTENT HASH (`*primary_key_column*`)`.

The `PARTITIONS AUTO` clause specifies that the number of partitions is automatically set to the number of tablespaces in the tablespace set `ts1`, and each partition is stored in a separate tablespace.

Sharded Tables for User-Defined Sharding

In a user-defined sharded database, you explicitly map data to individual shards. A sharded table in a user-defined sharded database can be partitioned by range or list.

You do not create tablespace sets for user-defined sharded tables; however, you must create each tablespace individually and explicitly associate it with a shardspace deployed in the sharded database configuration, as shown here.

```
CREATE TABLESPACE tbs1 IN SHARDSPACE west;
CREATE TABLESPACE tbs2 IN SHARDSPACE central;
CREATE TABLESPACE tbs3 IN SHARDSPACE east;
```

When you create the sharded table, you define the partitions with the ranges or lists of data to be stored in each tablespace, as shown in the following example.

```
CREATE SHARDED TABLE accounts
( id             NUMBER
, account_number NUMBER
, customer_id    NUMBER
, branch_id      NUMBER
, state          VARCHAR(2) NOT NULL
, status         VARCHAR2(1)
)
PARTITION BY LIST (state)
( PARTITION p_west VALUES ('OR', 'WA') TABLESPACE ts1
, PARTITION p_central VALUES ('SD', 'WI') TABLESPACE ts2
, PARTITION p_east VALUES ('NY', 'VM', 'NJ') TABLESPACE ts3
)
;
```

Sharded Tables for Composite Sharding

The sharded database using the composite sharding method allows you to partition subsets of data that correspond to a range or list of key values in a table partitioned by consistent hash.

With composite sharding, as with the other sharding methods, tablespaces are used to specify the mapping of partitions to shards. To partition subsets of data in a sharded table, a separate tablespace set must be created for each shardspace deployed in the sharded database configuration as shown in the following example.

```
CREATE TABLESPACE SET tbs1 IN SHARDSPACE shspace1;
CREATE TABLESPACE SET tbs2 IN SHARDSPACE shspace2;
```

The statement in the following example partitions a sharded table into two partition sets: gold and silver, based on class of service. Each partition set is stored in a separate tablespace. Then data in each partition set is further partitioned by consistent hash on customer ID.

```
CREATE SHARDED TABLE customers
( cust_id NUMBER NOT NULL
, name VARCHAR2(50)
, address VARCHAR2(250)
, location_id VARCHAR2(20)
, class VARCHAR2(3)
, signup_date DATE
, CONSTRAINT cust_pk PRIMARY KEY(cust_id, class)
)
PARTITIONSET BY LIST (class)
  PARTITION BY CONSISTENT HASH (cust_id)
  PARTITIONS AUTO
(PARTITIONSET gold VALUES ('gld') TABLESPACE SET tbs1,
 PARTITIONSET silver VALUES ('slv') TABLESPACE SET tbs2)
;
```

Sharded Tables for Directory-Based Sharding

Create directory-based sharded tables using `PARTITION BY DIRECTORY` in the `CREATE SHARDED TABLE` statement.

For example:

```
CREATE SHARDED TABLE customers
( id             NUMBER NOT NULL
, name           VARCHAR2(30)
, address        VARCHAR2(30)
, status         VARCHAR2(1)
,
CONSTRAINT cust_pk PRIMARY KEY(id)
)
PARTITION BY DIRECTORY (id)
( PARTITION p1 TABLESPACE tbs1,
  PARTITION p2 TABLESPACE tbs2,
```

```
PARTITION p3 TABLESPACE tbs3…);
```

Note:

- Unlike in user-defined sharding, key values are not specified for the partitions in the `CREATE TABLE` statement.
- The directory table is automatically created during root table creation. The definition of the directory table is:

  <shard_user_schema>.<root_table>`$SDIR`
- If a child table is created with `PARENT` clause in a different schema from the root table, an additional privilege is required for the child table's schema owner. (This is only for directory-based sharding and is not required for regular user-defined sharding.)

  This is because there is a foreign key constraint on the child table to the directory table's sharding key columns, to ensure that no rows can be inserted into the child table without the sharding key value being present in the directory mapping. As a consequence, the child table's schema needs a reference privilege on the directory table's sharding key columns.

  See "Granting References" in Creating Tables Sharded by Directory for the workaround.

Creating Duplicated Tables

The number of database requests handled by a single shard can be maximized by duplicating read-only or read-mostly tables across all shards.

Duplicated tables are a good choice for relatively small tables that are not updated frequently, and that are often accessed together with sharded tables. See Duplicated Tables for more detailed concepts and a diagram.

Types of Duplicated Tables

There are three types of duplicated tables you can create in a sharded database:

- Duplicated tables that refresh at a "refresh interval" set for the table. (see Setting the Duplicated Table Global Refresh Rate and Customizing Duplicated Table Refresh Rates)
- Duplicated tables that refresh on demand. These tables don't refresh until you explicitly attempt to refresh them. (see Refreshing Duplicated Tables On Demand)
- Duplicated tables that refresh on commit. These are called **synchronous duplicated tables**. (See example below)

With the first two types of duplicated table, you can connect to any shard and update a duplicated table directly on the shard. Then the update is **asynchronously** propagated to all other shards.

A *synchronous* duplicated table is a duplicated table that is synchronized on the shards ‘on-commit’ on the shard catalog. The rows in a duplicated table on the shards

are automatically synchronized with the rows in the duplicated table on the shard catalog when the active transaction performing DMLs on the duplicated tables is committed.

See Updating Duplicated Tables and Synchronizing Their Contents for more details.

Creating a Duplicated Table

A duplicated table, Products, can be created using the following statement:

```
CREATE DUPLICATED TABLE Products
( StockNo     NUMBER PRIMARY KEY
, Description VARCHAR2(20)
, Price       NUMBER(6,2))
;
```

Creating a Synchronous Duplicated Table

To create this same table as a synchronous duplicated table, use the `SYNCHRONOUS` keyword in the statement, as shown here:

```
CREATE DUPLICATED TABLE Products
( StockNo     NUMBER PRIMARY KEY
, Description VARCHAR2(20)
, Price       NUMBER(6,2))
SYNCHRONOUS;
```

See CREATE TABLE in *Oracle Database SQL Language Reference* for more information about the `DUPLICATED` clause.

Updating Duplicated Tables and Synchronizing Their Contents

Oracle Globally Distributed Database synchronizes the contents of duplicated tables using Materialized View Replication.

A duplicated table on each shard is represented by a materialized view. The primary table for the materialized views is located in the shard catalog. The `CREATE DUPLICATED TABLE` statement automatically creates the primary table, materialized views, and other objects required for materialized view replication.

Synchronous Duplicated Tables

Synchronous duplicated tables refresh automatically on commit from the shard catalog.

When the active transaction is committed on the duplicated tables created with `SYNCHRONOUS` in the shard catalog, a multi-shard DML is initiated for any synchronous duplicated tables that were updated with DMLs. To minimize the impact on performance of this commit, these synchronization DMLs are performed in parallel.

> **Note:**
>
> All shards must be up and running for a synchronous duplicated table DML to get refreshed on the shards “on-commit” on the shard catalog.

Non-Synchronous Duplicated Tables

For duplicated tables that are not created with `SYNCHRONOUS`, you can connect to any shard and update a duplicated table directly on the shard. What happens after that depends on whether you have set up automated refresh.

The materialized views on all of the shards can be refreshed with one of the two options:

- Automatic refresh at a configurable frequency per table
- On-demand refresh by running a stored procedure

For automatic refresh, to get better refresh performance, you can also use a stored procedure interface to create materialized view refresh groups.

On a refresh, the update is first propagated over a database link from the shard to the primary table on the shard catalog. Then the update is asynchronously propagated to all other shards as a result of a materialized view refresh.

Setting the Duplicated Table Global Refresh Rate

You can set a global refresh rate for all duplicated tables.

By default duplicated tables are refreshed every 60 seconds. The example below shows increasing the refresh interval to 100 seconds by setting the database parameter `shrd_dupl_table_refresh_rate`.

```
SQL> show parameter refresh

NAME                                 TYPE        VALUE
------------------------------------ -----------
------------------------------
shrd_dupl_table_refresh_rate         integer     60

SQL> alter system set shrd_dupl_table_refresh_rate=100 scope=both;

System altered.

SQL> show parameter refresh

NAME                                 TYPE        VALUE
------------------------------------ -----------
------------------------------
shrd_dupl_table_refresh_rate         integer     100
```

Customizing Duplicated Table Refresh Rates

You can set a finer grained refresh rate for individual duplicated tables.

Table-level refresh rates can be initially set with `CREATE TABLE`, and can be updated using `ALTER TABLE`.

The `REFRESH` clause syntax allows you to specify a refresh interval in seconds, minutes, hours, or you can set the table to only refresh on demand.

```
[REFRESH INTERVAL refresh_rate [SECOND|MINUTE|HOUR] | REFRESH ON
DEMAND]
```

For example, to create a duplicated table with customized refresh rate of two minutes:

```
CREATE DUPLICATED TABLE Products
( StockNo NUMBER PRIMARY KEY
, Description VARCHAR2(20)
, Price NUMBER(6,2))
REFRESH INTERVAL 2 MINUTE;
```

(To set on demand refresh, see Refreshing Duplicated Tables On Demand.)

To alter the duplicated table with a customized refresh rate of one hour:

```
ALTER TABLE table_name MODIFY REFRESH INTERVAL 1 HOUR;
```

If `DEFAULT` is specified, the value set in database parameter `shrd_dupl_table_refresh_rate` is used.

```
ALTER TABLE table_name MODIFY REFRESH INTERVAL DEFAULT;
```

Refreshing Duplicated Tables On Demand

You can set duplicated tables to be refreshed on demand rather than at a refresh interval.

When configured with the `REFRESH ON DEMAND` clause, duplicated tables are not automatically refreshed. You need to manually refresh these tables.

Setting On-Demand Refresh

To create a duplicated table that you can refresh on demand:

```
CREATE DUPLICATED TABLE Products
( StockNo NUMBER PRIMARY KEY
, Description VARCHAR2(20)
, Price NUMBER(6,2))
REFRESH ON DEMAND;
```

To update a duplicated table refresh method to on-demand refresh:

```
ALTER TABLE table_name MODIFY REFRESH ON DEMAND;
```

Refreshing the Duplicated Table On Demand

To refresh the tables created with the `ON DEMAND` clause, a utility procedure is provided which can be run on the shard catalog.

```
exec sys.refreshDuplicatedTable(table_name);
```

Here *table_name* can optionally be qualified with *schema_name*, so it would be *schema_name.table_name*.

Alternatively, you can refresh duplicated table materialized views directly on shards using the `DBMS_MVIEW.REFRESH` procedure.

Duplicated Table Support and Limitations

Keep the following considerations in mind when designing your schema with duplicated tables.

The following are supported for duplicated tables:

- `ALTER TABLE ADD/DROP CONSTRAINT` (one constraint at a time)
- `ALTER TABLE ADD/DROP PRIMARY KEY`

The following are not supported for duplicated tables.

- `NOLOGGING`
- `ALTER TABLE RENAME COLUMN`
- `PARTITION BY REFERENCE`
- `PARTITION BY SYSTEM`
- `CLUSTERED TABLE`
- Non-final `UDT` or `NESTED TABLE`
- `VECTOR ENCODE`
- `LONG DATATYPE`
- `INVISIBLE` columns
- Encrypted columns
- Information Lifecycle Management (ILM) policies
- Running `CREATE TABLE AS SELECT` (CTAS) in parallel
- Foreign keys: Sharded tables can reference duplicated tables (under an event). Foreign key constraints between duplicated tables and sharded tables are generally not allowed, with the exception that in user-defined sharding, you can create `DISABLE NOVALIDATE` foreign key constraints between sharded and duplicated tables.

> **Note:**
>
> A race condition is possible when a transaction run on a shard tries to update a row which was deleted on the shard catalog. In this case, an error is returned and the transaction on the shard is rolled back.
>
> The following use cases are not supported when updating duplicated tables on a shard.
>
> - Updating a LOB or a data type not supported by database links
> - Updating or deleting of a row inserted by the same transaction

Creating Indexes on Sharded Tables

You can create local indexes on sharded tables. You can also create a global partitioned index on the sharding key when the sharded table is sub-partitioned

Local Indexes

Unique local indexes on sharded tables must contain the sharding key.

The following example creates a local index named `id1` for the `id` column of the `account` table.

```
CREATE INDEX id1 ON account (id) LOCAL;
```

The following example creates a local unique index named `id2` for the `id` and `state` columns of the `account` table.

```
CREATE UNIQUE INDEX id2 ON account (id, state) LOCAL;
```

Global Indexes on Subpartitions

Global indexes on most sharded tables are not allowed because they can compromise the performance of online chunk movement. However, you can create a primary key/unique indexes on sharded tables that are composite partitioned without having to include sub-partition keys.

The following `CREATE INDEX` syntax is used to create a global index on a composite partitioned sharded table.

```
CREATE [UNIQUE] INDEX index_name ON table_name (col1, col2 ...)
[TABLESPACE SET tsset]
PARTITIONED AS TABLE;
```

For example, the following statement creates a composite sharded table with a primary key.

```
CREATE SHARDED TABLE customers
( cust_id     NUMBER NOT NULL
, name        VARCHAR2(50)
, address     VARCHAR2(250)
, location_id VARCHAR2(20)
, class       VARCHAR2(3)
, signup_date DATE
, CONSTRAINT cust_pk PRIMARY KEY(cust_id)
)
TABLESPACE SET ts1
PARTITION BY CONSISTENT HASH (cust_id)
SUBPARTITION BY RANGE (signup_date)
SUBPARTITION TEMPLATE
( SUBPARTITION per1 VALUES LESS THAN (TO_DATE('01/01/2000','DD/MM/YYYY')),
  SUBPARTITION per2 VALUES LESS THAN (TO_DATE('01/01/2010','DD/MM/YYYY')),
  SUBPARTITION per3 VALUES LESS THAN (TO_DATE('01/01/2020','DD/MM/YYYY')),
  SUBPARTITION future VALUES LESS THAN (MAXVALUE))
```

```
PARTITIONS AUTO
;
```

The following statement shows the creation of a `PARTITIONED AS TABLE` index.

```
CREATE UNIQUE INDEX custid_idx
ON customers(cust_id)
TABLESPACE SET tsidx1
PARTITIONED AS TABLE;
```

SPLIT CHUNK Handling

The global partitioned index is split automatically when splitting the underlying table partition when chunks are split.

A chunk split is in fact a series of split partition operations. For a global index, the default behavior during partition split is to invalidate the entire index. For the sharded table index, because each index partition is equi-partitioned with the table, the sharded database can issue corresponding split index operations automatically.

MOVE CHUNK Handling

Rather than invalidate the entire index, for a sharded table index, the sharded database invalidates only the index partition affected and rebuilds indexes after all exchanges are done.

Modifying an Oracle Globally Distributed Database Schema

When making changes to duplicated tables or sharded tables in an Oracle Globally Distributed Database, these changes should be done from the shard catalog database.

Before running any DDL operations on a sharded database, enable sharded DDL with

```
ALTER SESSION ENABLE SHARD DDL;
```

This statement ensures that the DDL changes will be propagated to each shard in the sharded database.

The DDL changes that are propagated are commands that are defined as “schema related,” which include operations such as `ALTER TABLE`. There are other operations that are propagated to each shard, such as the `CREATE, ALTER, DROP` user commands for simplified user management, and `TABLESPACE` operations to simplify the creation of tablespaces on multiple shards.

`GRANT` and `REVOKE` operations can be done from the shard catalog and are propagated to each shard, providing you have enabled shard DDL for the session. If more granular control is needed you can issue the command directly on each shard.

Operations such as DBMS package calls or similar operations are not propagated. For example, operations gathering statistics on the shard catalog are not propagated to each shard.

If you perform an operation that requires a lock on a table, such as adding a not null column, it is important to remember that each shard needs to obtain the lock on the

table in order to perform the DDL operation. Oracle's best practices for applying DDL in a single instance apply to sharded environments.

Multi-shard queries, which are processed on the shard catalog, issue remote queries across database connections on each shard. In this case it is important to ensure that the user has the appropriate privileges on each of the shards, whether or not the query will return data from that shard.

> **See Also:**
>
> *Oracle Database SQL Language Reference* for information about operations used with duplicated tables and sharded tables

DDL Processing in an Oracle Globally Distributed Database

To create a schema in an Oracle Globally Distributed Database, you must issue DDL commands on the shard catalog database, which validates the DDLs and processes them locally before they are processed on the shards.

The shard catalog database contains local copies of all of the objects that exist in the sharded database, and serves as the primary copy of the sharded database schema. If the shard catalog validation and processing of DDLs are successful, the DDLs are automatically propagated to all of the shards and applied in the order in which they were issued on the shard catalog.

If a shard is down or not accessible during DDL propagation, the shard catalog keeps track of DDLs that could not be applied to the shard, and then applies them when the shard is back up. When the shard comes back online, all of the DDLs that have been processed in the sharded database are applied in the same order to the shard before it becomes accessible to clients.

When a new shard is added to a sharded database, all of the DDLs that have been processed in the sharded database are applied in the same order to the shard before it becomes accessible to clients.

There are two ways you can issue DDLs in a sharded database.

- Use the `GDSCTL SQL` command.

  When you issue a DDL with the `GDSCTL SQL` command, as shown in the following example, `GDSCTL` waits until all of the shards have finished processing the DDL and returns the status.

  ```
  GDSCTL> sql "create tablespace set tbsset"
  ```

- Connect to the shard catalog database using SQL*Plus using the `GDS$CATALOG.`*`sdbname`* service.

  When you issue a DDL command on the shard catalog database, it returns the status when it finishes processing locally, but the propagation of the DDL to all of the shards happens in the background asynchronously.

  ```
  SQL> create tablespace set tbsset;
  ```

> **Note:**
>
> Using the SYS account to process shard DDL is not recommended; create a privileged account for this purpose.

For information about DDL syntax extensions for Oracle Globally Distributed Database, see DDL Syntax Extensions for Oracle Globally Distributed Database.

Creating Objects Locally and Globally

Objects created using GDSCTL creates global Oracle Globally Distributed Database objects; however, you can create local or global objects by connecting to the shard catalog with SQL*Plus.

When a DDL to create an object is issued using the GDSCTL `sql` command, the object is created on all of the shards. A primary copy of the object is also created in the shard catalog database. An object that exists on all shards, and the shard catalog database, is called a *sharded database object*.

When connecting to the shard catalog using SQL*Plus, two types of objects can be created: sharded database objects and local objects. *Local objects* are traditional objects that exist only in the shard catalog. Local objects can be used for administrative purposes, or they can be used by multi-shard queries originated from the shard catalog database, to generate and store a report, for example.

Sharded objects cannot have any dependency on local objects. For example, you cannot create an all-shard view on a local table.

The type of object (sharded database or local) that is created in a SQL*Plus session depends on whether the `SHARD DDL` mode is enabled in the session. This mode is enabled by default on the shard catalog database for the all-shards user, which is a user that exists on all of the shards and the shard catalog database. All of the objects created while `SHARD DDL` is enabled in a session are sharded database objects.

To enable `SHARD DDL` in the session, the all-shards user must run

```
ALTER SESSION ENABLE SHARD DDL
```

All of the objects created while `SHARD DDL` is disabled are local objects. To create a local object, the all-shards user must first run

```
ALTER SESSION DISABLE SHARD DDL
```

See ALTER SESSION for more information about the `SHARD DDL` session parameter.

Monitor DDL Processing and Verify Object Creation

You can monitor DDL processing using GDSCTL and SQL, to verify that the DDLs are propagated to all of the shards.

Monitor DDL Processing

You can check the status of the DDL propagation to the shards by using the GDSCTL `show ddl` and `config shard` commands.

This check is mandatory when a DDL is run using SQL*Plus on the shard catalog, because SQL*Plus does not return the DDL status on all of the shards.

The `show ddl` command output might be truncated. You can run `SELECT ddl_text FROM gsmadmin_internal.ddl_requests` on the shard catalog to see the full text of the statements.

Run the following command from the shard director host.

```
GDSCTL> show ddl
id    DDL Text                                  Failed shards
--    --------                                  -------------
5     grant connect, resource to app_schema
6     grant dba to app_schema
7     grant execute on dbms_crypto to app_s...
8     CREATE TABLESPACE SET  TSP_SET_1 usin...
9     CREATE TABLESPACE products_tsp datafi...
10    CREATE SHARDED TABLE Customers (   Cu...
11    CREATE SHARDED TABLE Orders (   Order...
12    CREATE SEQUENCE Orders_Seq;
13    CREATE SHARDED TABLE LineItems (   Or...
14    CREATE MATERIALIZED VIEW "APP_SCHEMA"...
```

Run the `config shard` command on each shard in your configuration, as shown here, and note the Last Failed DDL line in the command output.

```
GDSCTL> config shard -shard sh1
Name: sh1
Shard Group: primary_shardgroup
Status: Ok
State: Deployed
Region: region1
Connection string: shard_host_1:1521/sh1_host:dedicated
SCAN address:
ONS remote port: 0
Disk Threshold, ms: 20
CPU Threshold, %: 75
Version: 18.0.0.0
Last Failed DDL:
DDL Error: ---
Failed DDL id:
Availability: ONLINE

Supported services
------------------------
Name                                          Preferred Status
----                                          --------- ------
oltp_ro_srvc                                  Yes       Enabled
oltp_rw_srvc                                  Yes       Enabled
```

Verify Tablespace Set Creation

Verify that the tablespaces of the tablespace set you created for the sharded table family, and the tablespaces you created for the duplicated tables, are created on all of the shards.

The number of tablespaces in the tablespace set, shown below as C001TSP_SET_1 through C006TSP_SET_1, is based on the number of chunks specified in the GDSCTL `create shardcatalog` command when the sharded database configuration was deployed.

The duplicated Products tablespace is shown below as PRODUCTS_TSP.

Run `SELECT TABLESPACE_NAME` on all of the shards in your configuration, as shown here.

```
$ sqlplus / as sysdba

SQL> select TABLESPACE_NAME, BYTES/1024/1024 MB from sys.dba_data_files
 order by tablespace_name;

TABLESPACE_NAME             MB
------------------------ ----------
C001TSP_SET_1           100
C002TSP_SET_1           100
C003TSP_SET_1                 100
C004TSP_SET_1                 100
C005TSP_SET_1                 100
C006TSP_SET_1                 100
PRODUCTS_TSP            100
SYSAUX                  650
SYSTEM                  890
SYS_SHARD_TS                  100
TSP_SET_1                      100

TABLESPACE_NAME              MB
------------------------ ----------
UNDOTBS1                       105
USERS                                5

13 rows selected.
```

Verify Chunk Creation and Distribution

Verify that the chunks and chunk tablespaces were created on all of the shards.

Run the GDSCTL `config chunks` command as shown here, and note the ranges of chunk IDs on each shard.

```
GDSCTL> config chunks
Chunks
------------------------
Database                       From      To
--------                       ----      --
sh1                            1         6
sh2                            1         6
```

```
sh3                            7         12
sh4                            7         12
```

Run the following SQL statements on each of the shards in your configuration, as shown here.

```
SQL> show parameter db_unique_name

NAME             TYPE        VALUE
---------------- ----------- ------------------------------
db_unique_name   string      sh1

SQL> select table_name, partition_name, tablespace_name
 from dba_tab_partitions
 where tablespace_name like 'C%TSP_SET_1'
 order by tablespace_name;

TABLE_NAME       PARTITION_NAME   TABLESPACE_NAME
---------------- ---------------- --------------------
ORDERS           CUSTOMERS_P1     C001TSP_SET_1
CUSTOMERS        CUSTOMERS_P1     C001TSP_SET_1
LINEITEMS        CUSTOMERS_P1     C001TSP_SET_1
CUSTOMERS        CUSTOMERS_P2     C002TSP_SET_1
LINEITEMS        CUSTOMERS_P2     C002TSP_SET_1
ORDERS           CUSTOMERS_P2     C002TSP_SET_1
CUSTOMERS        CUSTOMERS_P3     C003TSP_SET_1
ORDERS           CUSTOMERS_P3     C003TSP_SET_1
LINEITEMS        CUSTOMERS_P3     C003TSP_SET_1
ORDERS           CUSTOMERS_P4     C004TSP_SET_1
CUSTOMERS        CUSTOMERS_P4     C004TSP_SET_1

TABLE_NAME       PARTITION_NAME   TABLESPACE_NAME
---------------- ---------------- --------------------
LINEITEMS        CUSTOMERS_P4     C004TSP_SET_1
CUSTOMERS        CUSTOMERS_P5     C005TSP_SET_1
LINEITEMS        CUSTOMERS_P5     C005TSP_SET_1
ORDERS           CUSTOMERS_P5     C005TSP_SET_1
CUSTOMERS        CUSTOMERS_P6     C006TSP_SET_1
LINEITEMS        CUSTOMERS_P6     C006TSP_SET_1
ORDERS           CUSTOMERS_P6     C006TSP_SET_1
18 rows selected.
```

Connect to the shard catalog database and verify that the chunks are uniformly distributed, as shown here.

```
$ sqlplus / as sysdba

SQL> SELECT a.name Shard, COUNT(b.chunk_number) Number_of_Chunks
  FROM gsmadmin_internal.database a, gsmadmin_internal.chunk_loc b
  WHERE a.database_num=b.database_num
  GROUP BY a.name
  ORDER BY a.name;

SHARD                  NUMBER_OF_CHUNKS
```

```
------------------------------ ----------------
sh1                            6
sh2                            6
sh3                            6
sh4                            6
```

Verify Table Creation

To verify that the sharded and duplicated tables were created, log in as the application schema user on the shard catalog database and each of the shards and query the tables on a database shard, as shown below with the example app_schema user.

```
$ sqlplus app_schema/app_schema_password
Connected.

SQL> select table_name from user_tables;

TABLE_NAME
--------------------------------------------------------------------------------
CUSTOMERS
ORDERS
LINEITEMS
PRODUCTS

4 rows selected.
```

DDL Syntax Extensions for Oracle Globally Distributed Database

Oracle Globally Distributed Database includes SQL DDL statements with syntax that can only be run against a sharded database.

Changes to query and DML statements are not required to support Oracle Globally Distributed Database, and the changes to the DDL statements are very limited. Most existing DDL statements will work the same way on a sharded database, with the same syntax and semantics, as they do on a non-sharded database.

CREATE TABLESPACE SET

This statement creates a tablespace set that can be used as a logical storage unit for one or more sharded tables and indexes. A tablespace set consists of multiple Oracle tablespaces distributed across shards in a shardspace.

The `CREATE TABLESPACE SET` statement is intended specifically for sharded databases. Its syntax is similar to `CREATE TABLESPACE`.

```
CREATE TABLESPACE SET tablespace_set
       [IN SHARDSPACE shardspace]
              [USING TEMPLATE (
    { MINIMUM EXTENT size_clause
    | BLOCKSIZE integer [ K ]
    | logging_clause
    | FORCE LOGGING
    | ENCRYPTION tablespace_encryption_spec
    | DEFAULT [ table_compression ] storage_clause
```

```
  | { ONLINE | OFFLINE }
  | extent_management_clause
  | segment_management_clause
  | flashback_mode_clause
  }...
 )];
```

Note that in system-managed sharding there is only one default shardspace in the sharded database. The number of tablespaces in a tablespace set is determined automatically and is equal to the number of chunks in the corresponding shardspace.

All tablespaces in a tablespace set are bigfile tablespaces and have the same properties. The properties are specified in the `USING TEMPLATE` clause and they describe the properties of one single tablespace in the tablespace set. This clause is the same as `permanent_tablespace_clause` for a typical tablespace, with the exception that a data file name cannot be specified in the `datafile_tempfile_spec` clause. The data file name for each tablespace in a tablespace set is generated automatically.

Note that a tablespace set can only consist of permanent tablespaces, there is no system, undo, or temporary tablespace set. Also, note that in the example below the total data file size of the tablespace set is `100mx`*N* (where *N* is the number of tablespaces in the tablespace set).

Example

```
CREATE TABLESPACE SET TSP_SET_1 IN SHARDSPACE sgr1
USING TEMPLATE
( DATAFILE SIZE 100m
  EXTEND MANAGEMENT LOCAL
  SEGMENT SPACE MANAGEMENT AUTO
);
```

ALTER TABLESPACE SET

This statement alters a tablespace set that can be used as a logical storage unit for one or more sharded tables and indexes.

The `SHARDSPACE` property of a tablespace set cannot be modified. All other attributes of a tablespace set can be altered just as for a regular permanent tablespace. Because tablespaces in a tablespace set are bigfile, the `ADD DATAFILE` and `DROP DATAFILE` clauses are not supported.

DROP TABLESPACE SET and PURGE TABLESPACE SET

These statements drop or purge a tablespace set, which can be used as a logical storage unit for one or more sharded tables and indexes.

The syntax and semantics for these statements are similar to `DROP` and `PURGE TABLESPACE` statements.

CREATE TABLE

The `CREATE TABLE` statement has been extended to create sharded and duplicated tables, and specify a table family.

Syntax

```
CREATE [ { GLOBAL TEMPORARY | SHARDED | DUPLICATED} ]
         TABLE [ schema. ] table
      { relational_table | object_table | XMLType_table }
          [ PARENT [ schema. ] table ] ;
```

The following parts of the `CREATE TABLE` statement are intended to support sharded databases

- The `SHARDED` and `DUPLICATED` keywords indicate that the table content is either partitioned across shards or duplicated on all shards respectively. The `DUPLICATED` keyword is the only syntax change to create duplicated tables. All other changes described below apply only to sharded tables.
- The `PARENT` clause links a sharded table to the root table of its table family.
- In system and composite sharding, to create a sharded table, `TABLESPACE SET` is used instead of `TABLESPACE`. All clauses that contain `TABLESPACE` are extended to contain `TABLESPACE SET`.
- Three clauses: `consistent_hash_partitions`, `consistent_hash_with_subpartitions`, and `partition_set_clause` in the `table_partitioning_clauses`.

```
table_partitioning_clauses ::=
{range_partitions
| hash_partitions
| list_partitions
| composite_range_partitions
| composite_hash_partitions
| composite_list_partitions
| reference_partitioning
| system_partitioning
| consistent_hash_partitions
| consistent_hash_with_subpartitions
| partition_set_clause
}
```

Example

```
CREATE SHARDED TABLE customers
( cust_id     NUMBER NOT NULL
, name        VARCHAR2(50)
, address     VARCHAR2(250)
, location_id VARCHAR2(20)
, class       VARCHAR2(3)
, signup_date DATE
,
CONSTRAINT cust_pk PRIMARY KEY(cust_id, class)
)
PARTITIONSET BY LIST (class)
PARTITION BY CONSISTENT HASH (cust_id)
PARTITIONS AUTO
(PARTITIONSET gold   VALUES ('gld') TABLESPACE SET ts2,
```

```
 PARTITIONSET silver VALUES ('slv') TABLESPACE SET ts1)
;
```

Example of consistent_hash_with_subpartitions

```
CREATE SHARDED TABLE Customers
       ( "custi_id" NUMBER NOT NULL
       , name VARCHAR2(50)
       , class VARCHAR2(3) NOT NULL
       , signup_date DATE
       ,
       CONSTRAINT cust_pk PRIMARY KEY("custi_id",name,signup_date,class)
       )
       PARTITIONSET BY LIST (class)
       PARTITION BY CONSISTENT HASH ("custi_id",name)
       SUBPARTITION BY RANGE (signup_date)
       SUBPARTITION TEMPLATE
          ( SUBPARTITION per1 VALUES LESS THAN (TO_DATE('01/01/2000','DD/MM/
YYYY'))
          , SUBPARTITION per2 VALUES LESS THAN (TO_DATE('01/01/2010','DD/MM/
YYYY'))
          , SUBPARTITION per3 VALUES LESS THAN (TO_DATE('01/01/2020','DD/MM/
YYYY'))
          , SUBPARTITION future VALUES LESS THAN (MAXVALUE))
       PARTITIONS AUTO
       (
         PARTITIONSET "gold" VALUES ('Gld','BRZ') TABLESPACE SET ts1
SUBPARTITIONS STORE IN(TBS1,TBS2,TBS3,TBS4)
       , PARTITIONSET "silver" VALUES ('Slv','OTH') TABLESPACE SET ts2
SUBPARTITIONS STORE IN(TBS5,TBS6,TBS7,TBS8)
       ) ;
```

Limitations

Limitations for sharded tables in the current release:

- There is no default tablespace set for sharded tables.
- A temporary table cannot be sharded or duplicated.
- Index-organized sharded tables are not supported.
- A sharded table cannot contain a nested table column or an identity column.
- A primary key constraint defined on a sharded table must contain the sharding column(s). A foreign key constraint on a column of a sharded table referencing a duplicated table column is not supported.
- System partitioning and interval range partitioning are not supported for sharded tables. Specification of individual hash partitions is not supported for partitioning by consistent hash.
- A column in a sharded table used in `PARTITION BY` or `PARTITIONSET BY` clauses cannot be a virtual column.

Duplicated tables in the current release are not supported with the following:

- System and reference partitioned tables

- Non-final abstract types
- Maximum number of columns without primary key is 999
- The `nologging` option
- XMLType column in a duplicated table cannot be used in non-ASSM tablespace

See CREATE TABLE for more information about using the clauses supporting sharded databases.

ALTER TABLE

The `ALTER TABLE` statement is extended to modify sharded and duplicated tables.

There are limitations on using `ALTER TABLE` with a sharded database.

The following options are not supported for a sharded table in a system-managed or composite sharded database:

- Rename
- All operations on individual partitions and subpartitions
- All partition-related operations on the shard, except `TRUNCATE` partition, `UNUSABLE LOCAL INDEXES`, and `REBUILD UNUSABLE LOCAL INDEXES`

The following are not supported for duplicated tables:

- Data types: Non-final abstract types
- Column options: vector encode, invisible column, nested tables
- Clustered table
- External table
- ILM policy
- `PARENT` clause
- Flashback table operation
- System and Reference partitioning
- Enable `NOLOGGING` option
- Drop duplicated table materialized view log
- Drop duplicated table materialized views on shards
- Alter materialized views (of duplicated tables) on shards

ALTER SESSION

The `ALTER SESSION` statement is extended to support sharded databases.

The session-level `SHARD DDL` parameter sets the scope for DDLs issued against the shard catalog database.

```
ALTER SESSION { ENABLE | DISABLE } SHARD DDL;
```

When `SHARD DDL` is enabled, all DDLs issued in the session are executed on the shard catalog and all shards. When `SHARD DDL` is disabled, a DDL is executed only against

the shard catalog database. `SHARD DDL` is enabled by default for a sharded database user (the user that exists on all shards and the catalog). To create a sharded database user, the `SHARD DDL` parameter must be enabled before running `CREATE USER`.

Running PL/SQL Procedures in an Oracle Globally Distributed Database

In the same way that DDL statements can be run on all shards in an Oracle Globally Distributed Database configuration, so too can certain Oracle-provided PL/SQL procedures.

These specific procedure calls behave as if they were sharded DDL statements, in that they are propagated to all shards, tracked by the catalog, and run whenever a new shard is added to a configuration.

All of the following procedures can act as if they were a sharded DDL statement.

- Any procedure in the `DBMS_FGA` package
- Any procedure in the `DBMS_RLS` package
- The following procedures from the `DBMS_STATS` package:
  - `GATHER_INDEX_STATS`
  - `GATHER_TABLE_STATS`
  - `GATHER_SCHEMA_STATS`
  - `GATHER_DATABASE_STATS`
  - `GATHER_SYSTEM_STATS`
- The following procedures from the `DBMS_GOLDENGATE_ADM` package:
  - `ADD_AUTO_CDR`
  - `ADD_AUTO_CDR_COLUMN_GROUP`
  - `ADD_AUTO_CDR_DELTA_RES`
  - `ALTER_AUTO_CDR`
  - `ALTER_AUTO_CDR_COLUMN_GROUP`
  - `PURGE_TOMBSTONES`
  - `REMOVE_AUTO_CDR`
  - `REMOVE_AUTO_CDR_COLUMN_GROUP`
  - `REMOVE_AUTO_CDR_DELTA_RES`

To run one of the procedures in the same way as sharded DDL statements, do the following steps.

1. Connect to the shard catalog database using SQL*Plus as a database user with the `gsm_pooladmin_role`.
2. Enable sharding DDL using `ALTER SESSION ENABLE SHARD DDL`.
3. Run the target procedure using a sharding-specific PL/SQL procedure named `SYS.EXEC_SHARD_PLSQL`.

This procedure takes a single CLOB argument, which is a character string specifying a fully qualified procedure name and its arguments. Note that running the target procedure without using `EXEC_SHARD_PLSQL` causes the procedure to only be run on the shard catalog, and it is not propagated to all of the shards. Running the procedure without specifying the fully qualified name (for example, `SYS.DBMS_RLS.ADD_POLICY`) will result in an error.

For example, to run `DBMS_RLS.ADD_POLICY` on all shards, do the following from SQL*Plus after enabling `SHARD DLL`.

```
exec
sys.exec_shard_plsql('sys.dbms_rls.add_policy(object_schema
   =>
          ''testuser1'',

          object_name     => ''DEPARTMENTS'',

          policy_name     => ''dept_vpd_pol'',

          function_schema => ''testuser1'',

          policy_function => ''authorized_emps'',

          statement_types => ''INSERT, UPDATE, DELETE, SELECT, INDEX'',

          update_check    => TRUE)'

          ) ;
```

Take careful note of the need for double single-quotes inside the target procedure call specification, because the call specification itself is a string parameter to `EXEC_SHARD_PLSQL`.

If the target procedure runs correctly on the shard catalog database, it is queued for processing on all of the currently deployed shards. Any error in running the target procedure on the shard catalog is returned to the SQL*Plus session. Errors while running on the shards can be tracked in the same way they are for DDLs.

Generating Unique Sequence Numbers Across Shards

You can generate globally unique sequence numbers across shards for non-primary key columns, and it is handled by the Oracle Globally Distributed Database.

You may need to generate unique IDs for non-primary key columns, for example order_id, when the customer_id is the sharding key. For this case among others, this feature lets you generate unique sequence numbers across shards, while not requiring you to manage the global uniqueness of a given non-primary key column in your application.

This functionality is supported by the `SHARDED SEQUENCE` object. A sharded sequence is created on the shard catalog but has an instance on each shard. Each instance generates monotonically increasing numbers that belong to a range which does not overlap with ranges used on other shards. Therefore, every generated number is globally unique.

A sharded sequence can be used, for example, to generate a unique order number for a table sharded by a customer ID. An application that establishes a connection to a shard using the customer ID as a key can use a local instance of the sharded sequence to generate a globally unique order number.

Note that the number generated by a sharded sequence cannot be immediately used as a sharding key for a new row being inserted into this shard, because the key value may belong to another shard and the insert will result in an error. To insert a new row, the application should first generate a value of the sharding key and then use it to connect to the appropriate shard. A typical way to generate a new value of the sharding key would be use a regular (non-sharded) sequence on the shard catalog.

If a single sharding key generator becomes a bottleneck, a sharded sequence can be used for this purpose. In this case, an application should connect to a random shard (using the global service without specifying the sharding key), get a unique key value from a sharded sequence, and then connect to the appropriate shard using the key value.

To support this feature, the `SEQUENCE` object clauses, `SHARD` and `NOSHARD`, are included in the `SEQUENCE` object DDL syntax, as shown in the following `CREATE` statement syntax.

```
CREATE | ALTER SEQUENCE [ schema. ]sequence
   [ { INCREMENT BY | START WITH } integer
   | { MAXVALUE integer | NOMAXVALUE }
   | { MINVALUE integer | NOMINVALUE }
   | { CYCLE | NOCYCLE }
   | { CACHE integer | NOCACHE }
   | { ORDER | NOORDER }
   | { SCALE {EXTEND | NOEXTEND} | NOSCALE}
   | { SHARD {EXTEND | NOEXTEND} | NOSHARD}
   ]
```

`NOSHARD` is the default for a sequence. If the `SHARD` clause is specified, this property is registered in the sequence object's dictionary table, and is shown using the `DBA_SEQUENCES`, `USER_SEQUENCES`, and `ALL_SEQUENCES` views.

When `SHARD` is specified, the `EXTEND` and `NOEXTEND` clauses define the behavior of a sharded sequence. When `EXTEND` is specified, the generated sequence values are all of length *(x+y)*, where *x* is the length of a `SHARD` offset of size 4 (corresponding to the width of the maximum number of shards, that is, 1000) affixed at beginning of the sequence values, and *y* is the maximum number of digits in the sequence `MAXVALUE`/`MINVALUE`.

The default setting for the `SHARD` clause is `NOEXTEND`. With the `NOEXTEND` setting, the generated sequence values are at most as wide as the maximum number of digits in the sequence `MAXVALUE`/`MINVALUE`. This setting is useful for integration with existing applications where sequences are used to populate fixed width columns. On invocation of `NEXTVAL` on a sequence with `SHARD NOEXTEND` specified, a user error is thrown if the generated value requires more digits of representation than the sequence's `MAXVALUE`/`MINVALUE`.

If the `SCALE` clause is also specified with the `SHARD` clause, the sequence generates scalable values within a shard for multiple instances and sessions, which are globally unique. When `EXTEND` is specified with both the `SHARD` and `SCALE` keywords, the generated sequence values are all of length *(x+y+z)*, where *x* is the length of a prepended `SHARD` offset of size 4, *y* is the length of the scalable offset (default 6), and *z* is the maximum number of digits in the sequence `MAXVALUE`/`MINVALUE`.

> **Note:**
>
> When using the `SHARD` clause, do not specify `ORDER` on the sequence. Using `SHARD` generates globally unordered values. If `ORDER` is required, create the sequences locally on each node.
>
> The `SHARD` keyword will work in conjunction with `CACHE` and `NOCACHE` modes of operation.

> **See Also:**
>
> *Oracle Database SQL Language Reference*

Schema Creation Examples

The following examples show the steps you would take to create a schema for an Oracle Globally Distributed Database using the system-managed, user-defined, and composite sharding methods.

- Schema for System-Managed Sharding
- Schema for User-Defined Sharding
- Schema for Composite Sharding

Schema for System-Managed Sharding

Create the tablespace set, sharded tables, and duplicated tables for an Oracle Globally Distributed Database that uses the system-managed sharding method.

1. Connect to the shard catalog database, create the application schema user, and grant privileges and roles to the user.

   In this example, the application schema user is called app_schema.

```
$ sqlplus / as sysdba

SQL> alter session enable shard ddl;
SQL> create user app_schema identified by app_schema_password;
SQL> grant all privileges to app_schema;
SQL> grant gsmadmin_role to app_schema;
SQL> grant select_catalog_role to app_schema;
SQL> grant connect, resource to app_schema;
SQL> grant dba to app_schema;
SQL> grant execute on dbms_crypto to app_schema;
```

2. Create a tablespace set for the sharded tables.

```
SQL> CREATE TABLESPACE SET TSP_SET_1 using template
 (datafile size 100m autoextend on next 10M maxsize unlimited
  extent management local segment space management auto);
```

3. If you use LOBs in a column, you can specify a tablespace set for the LOBs.

```
SQL> CREATE TABLESPACE SET LOBTS1;
```

> **Note:**
>
> Tablespace sets for LOBS cannot be specified at the subpartitition level in system-managed sharding.

4. Create a tablespace for the duplicated tables.

   In this example the duplicated table is the Products table in the sample Customers-Orders-Products schema.

```
SQL> CREATE TABLESPACE products_tsp datafile size 100m
 autoextend on next 10M maxsize unlimited
 extent management local uniform size 1m;
```

5. Create a sharded table for the root table.

   In this example, the root table is the Customers table in the sample Customers-Orders-Products schema.

```
SQL> CONNECT app_schema/app_schema_password
SQL> CREATE SHARDED TABLE Customers
  (
    CustId      VARCHAR2(60) NOT NULL,
    FirstName   VARCHAR2(60),
    LastName    VARCHAR2(60),
    Class       VARCHAR2(10),
    Geo         VARCHAR2(8),
    CustProfile VARCHAR2(4000),
    Passwd      RAW(60),
    CONSTRAINT pk_customers PRIMARY KEY (CustId),
    CONSTRAINT json_customers CHECK (CustProfile IS JSON)
  ) TABLESPACE SET TSP_SET_1
  PARTITION BY CONSISTENT HASH (CustId) PARTITIONS AUTO;
```

> **Note:**
>
> If any columns contain LOBs, you can include the tablespace set in the parent table creation statement, as shown here.
>
> ```
> SQL> CREATE SHARDED TABLE Customers
>   (
>     CustId      VARCHAR2(60) NOT NULL,
>     FirstName   VARCHAR2(60),
>     LastName    VARCHAR2(60),
>     Class       VARCHAR2(10),
>     Geo         VARCHAR2(8),
>     CustProfile VARCHAR2(4000),
>     Passwd      RAW(60),
>     image       BLOB,
>     CONSTRAINT pk_customers PRIMARY KEY (CustId),
>     CONSTRAINT json_customers CHECK (CustProfile IS JSON)
>   ) TABLESPACE SET TSP_SET_1
>     LOB(image) store as (TABLESPACE SET LOBTS1)
>   PARTITION BY CONSISTENT HASH (CustId) PARTITIONS AUTO;
> ```

6. Create a sharded table for the other tables in the table family.

   In this example, sharded tables are created for the Orders and LineItems tables in the sample Customers-Orders-Products schema.

   The Orders sharded table is created first:

   ```
   SQL> CREATE SHARDED TABLE Orders
     (
       OrderId     INTEGER NOT NULL,
       CustId      VARCHAR2(60) NOT NULL,
       OrderDate   TIMESTAMP NOT NULL,
       SumTotal    NUMBER(19,4),
       Status      CHAR(4),
       CONSTRAINT  pk_orders PRIMARY KEY (CustId, OrderId),
       CONSTRAINT  fk_orders_parent FOREIGN KEY (CustId)
       REFERENCES Customers ON DELETE CASCADE
     ) PARTITION BY REFERENCE (fk_orders_parent);
   ```

   Create the sequence used for the OrderId column.

   ```
   SQL> CREATE SEQUENCE Orders_Seq;
   ```

   Create a sharded table for LineItems

   ```
   SQL> CREATE SHARDED TABLE LineItems
     (
       OrderId     INTEGER NOT NULL,
       CustId      VARCHAR2(60) NOT NULL,
       ProductId   INTEGER NOT NULL,
       Price       NUMBER(19,4),
   ```

```
    Qty         NUMBER,
    CONSTRAINT  pk_items PRIMARY KEY (CustId, OrderId, ProductId),
    CONSTRAINT  fk_items_parent FOREIGN KEY (CustId, OrderId)
    REFERENCES Orders ON DELETE CASCADE
  ) PARTITION BY REFERENCE (fk_items_parent);
```

7. Create any required duplicated tables.

   In this example, the Products table is a duplicated object.

```
SQL> CREATE DUPLICATED TABLE Products
  (
    ProductId  INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    Name       VARCHAR2(128),
    DescrUri   VARCHAR2(128),
    LastPrice  NUMBER(19,4)
  ) TABLESPACE products_tsp;
```

Next you should monitor the DDL processing and verify that the tablespace sets, tables, and chunks were correctly created on all of the shards.

Schema for User-Defined Sharding

Create the schema user, tablespace set, sharded tables, and duplicated tables for an Oracle Globally Distributed Database that uses the user-defined sharding method.

1. Connect to the shard catalog database, create the application schema user, and grant privileges and roles to the user.

   In this example, the application schema user is called app_schema.

```
$ sqlplus / as sysdba

SQL> alter session enable shard ddl;
SQL> create user app_schema identified by app_schema_password;
SQL> grant all privileges to app_schema;
SQL> grant gsmadmin_role to app_schema;
SQL> grant select_catalog_role to app_schema;
SQL> grant connect, resource to app_schema;
SQL> grant dba to app_schema;
SQL> grant execute on dbms_crypto to app_schema;
```

2. Create tablespaces for the sharded tables.

```
SQL> CREATE TABLESPACE ck1_tsp DATAFILE SIZE 100M autoextend on next 10M
maxsize
unlimited extent management local segment space management auto in
 shardspace shspace1;

SQL> CREATE TABLESPACE ck2_tsp DATAFILE SIZE 100M autoextend on next 10M
maxsize
unlimited extent management local segment space management auto in
 shardspace shspace2;
```

3. If you use LOBs in any columns, you can specify tablespaces for the LOBs.

```
SQL> CREATE TABLESPACE lobts1 ... in shardspace shspace1;

SQL> CREATE TABLESPACE lobts2 ... in shardspace shspace2;
```

4. Create a tablespace for the duplicated tables.

   In this example the duplicated table is the Products table in the sample Customers-Orders-Products schema.

```
SQL> CREATE TABLESPACE products_tsp datafile size 100m autoextend
 on next 10M maxsize unlimited extent management local uniform size
1m;
```

5. Create a sharded table for the root table.

   In this example, the root table is the Customers table in the sample Customers-Orders-Products schema.

```
SQL> CONNECT app_schema/app_schema_password

SQL> ALTER SESSION ENABLE SHARD DDL;

SQL> CREATE SHARDED TABLE Customers
  (
    CustId      VARCHAR2(60) NOT NULL,
    CustProfile VARCHAR2(4000),
    Passwd      RAW(60),
    CONSTRAINT pk_customers PRIMARY KEY (CustId),
    CONSTRAINT json_customers CHECK (CustProfile IS JSON)
  ) PARTITION BY RANGE (CustId)
  ( PARTITION ck1 values less than ('m') tablespace ck1_tsp,
    PARTITION ck2 values less than (MAXVALUE) tablespace ck2_tsp
  );
```

> **Note:**
>
> If any columns in the sharded tables contain LOBs, the CREATE SHARDED TABLE statement can include the LOB tablespaces, as shown here.

```
SQL> CREATE SHARDED TABLE Customers
  (
    CustId      VARCHAR2(60) NOT NULL,
    CustProfile VARCHAR2(4000),
    Passwd      RAW(60),
    image       BLOB,
    CONSTRAINT pk_customers PRIMARY KEY (CustId),
    CONSTRAINT json_customers CHECK (CustProfile IS JSON)
  ) PARTITION BY RANGE (CustId)
  ( PARTITION ck1 values less than ('m') tablespace ck1_tsp
     lob(image) store as (tablespace lobts1),
    PARTITION ck2 values less than (MAXVALUE) tablespace ck2_tsp
     lob(image) store as (tablespace lobts2)
  );
```

6. Create a sharded table for the other tables in the table family.

   In this example, sharded tables are created for the Orders and LineItems tables in the sample Customers-Orders-Products schema.

   The Orders sharded table is created first:

```
SQL> CREATE SHARDED TABLE Orders
  (
    OrderId     INTEGER NOT NULL,
    CustId      VARCHAR2(60) NOT NULL,
    OrderDate   TIMESTAMP NOT NULL,
    SumTotal    NUMBER(19,4),
    Status      CHAR(4),
    CONSTRAINT  pk_orders PRIMARY KEY (CustId, OrderId),
    CONSTRAINT  fk_orders_parent FOREIGN KEY (CustId)
    REFERENCES Customers ON DELETE CASCADE
  ) PARTITION BY REFERENCE (fk_orders_parent);
```

   Create the sequence used for the OrderId column.

```
SQL> CREATE SEQUENCE Orders_Seq;
```

   Create a sharded table for LineItems

```
SQL> CREATE SHARDED TABLE LineItems
  (
    OrderId     INTEGER NOT NULL,
    CustId      VARCHAR2(60) NOT NULL,
    ProductId   INTEGER NOT NULL,
    Price       NUMBER(19,4),
    Qty         NUMBER,
```

```
    CONSTRAINT  pk_items PRIMARY KEY (CustId, OrderId, ProductId),
    CONSTRAINT  fk_items_parent FOREIGN KEY (CustId, OrderId)
    REFERENCES Orders ON DELETE CASCADE
  ) PARTITION BY REFERENCE (fk_items_parent);
```

7. Create any required duplicated tables.

   In this example, the Products table is a duplicated object.

```
SQL> CREATE DUPLICATED TABLE Products
  (
    ProductId  INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    Name       VARCHAR2(128),
    DescrUri   VARCHAR2(128),
    LastPrice  NUMBER(19,4)
  ) TABLESPACE products_tsp;
```

Next you should monitor the DDL processing and verify that the tablespace sets, tables, and chunks were correctly created on all of the shards.

Schema for Composite Sharding

Create the schema user, tablespace set, sharded tables, and duplicated tables for an Oracle Globally Distributed Database that uses the composite sharding method.

1. Connect to the shard catalog host, and set the ORACLE_SID to the shard catalog name.
2. Connect to the shard catalog database, create the application schema user, and grant privileges and roles to the user.

   In this example, the application schema user is called app_schema.

```
$ sqlplus / as sysdba

SQL> connect / as sysdba
SQL> alter session enable shard ddl;
SQL> create user app_schema identified by app_schema_password;
SQL> grant connect, resource, alter session to app_schema;
SQL> grant execute on dbms_crypto to app_schema;
SQL> grant create table, create procedure, create tablespace,
 create materialized view to app_schema;
SQL> grant unlimited tablespace to app_schema;
SQL> grant select_catalog_role to app_schema;
SQL> grant all privileges to app_schema;
SQL> grant gsmadmin_role to app_schema;
SQL> grant dba to app_schema;
```

3. Create tablespace sets for the sharded tables.

```
SQL> CREATE TABLESPACE SET
  TSP_SET_1 in shardspace cust_america using template
  (datafile size 100m autoextend on next 10M maxsize
   unlimited extent management
   local segment space management auto );
```

```
SQL> CREATE TABLESPACE SET
  TSP_SET_2 in shardspace cust_europe using template
  (datafile size 100m autoextend on next 10M maxsize
   unlimited extent management
   local segment space management auto );
```

4. If you use LOBs in any columns, you can specify tablespace sets for the LOBs.

```
SQL> CREATE TABLESPACE SET LOBTS1 in shardspace cust_america ... ;

SQL> CREATE TABLESPACE SET LOBTS2 in shardspace cust_europe ... ;
```

> **Note:**
>
> Tablespace sets for LOBs cannot be specified at the subpartitition level in composite sharding.

5. Create a tablespace for the duplicated tables.

   In this example the duplicated table is the Products table in the sample Customers-Orders-Products schema.

```
CREATE TABLESPACE products_tsp datafile size 100m autoextend on next 10M
 maxsize unlimited extent management local uniform size 1m;
```

6. Create a sharded table for the root table.

   In this example, the root table is the Customers table in the sample Customers-Orders-Products schema.

```
connect app_schema/app_schema_password
alter session enable shard ddl;

CREATE SHARDED TABLE Customers
(
  CustId      VARCHAR2(60) NOT NULL,
  FirstName   VARCHAR2(60),
  LastName    VARCHAR2(60),
  Class       VARCHAR2(10),
  Geo         VARCHAR2(8),
  CustProfile VARCHAR2(4000),
  Passwd      RAW(60),
  CONSTRAINT pk_customers PRIMARY KEY (CustId),
  CONSTRAINT json_customers CHECK (CustProfile IS JSON)
) partitionset by list(GEO)
partition by consistent hash(CustId)
partitions auto
(partitionset america values ('AMERICA') tablespace set tsp_set_1,
partitionset europe values ('EUROPE') tablespace set tsp_set_2
);
```

> **Note:**
>
> If any columns in the sharded tables contain LOBs, the CREATE SHARDED TABLE statement can include the LOB tablespace set, as shown here.

```
CREATE SHARDED TABLE Customers
(
  CustId      VARCHAR2(60)  NOT NULL,
  FirstName   VARCHAR2(60),
  LastName    VARCHAR2(60),
  Class       VARCHAR2(10),
  Geo         VARCHAR2(8)   NOT NULL,
  CustProfile VARCHAR2(4000),
  Passwd      RAW(60),
  image       BLOB,
  CONSTRAINT pk_customers PRIMARY KEY (CustId),
  CONSTRAINT json_customers CHECK (CustProfile IS JSON)
) partitionset by list(GEO)
partition by consistent hash(CustId)
partitions auto
(partitionset america values ('AMERICA') tablespace set
tsp_set_1
 lob(image) store as (tablespace set lobts1),
partitionset europe values ('EUROPE') tablespace set
tsp_set_2
 lob(image) store as (tablespace set lobts2));
```

7. Create a sharded table for the other tables in the table family.

   In this example, sharded tables are created for the Orders and LineItems tables in the sample Customers-Orders-Products schema.

   Create the sequence used for the OrderId column.

```
CREATE SEQUENCE Orders_Seq;
```

   The Orders sharded table is created first:

```
CREATE SHARDED TABLE Orders
(
  OrderId     INTEGER NOT NULL,
  CustId      VARCHAR2(60) NOT NULL,
  OrderDate   TIMESTAMP NOT NULL,
  SumTotal    NUMBER(19,4),
  Status      CHAR(4),
  constraint  pk_orders primary key (CustId, OrderId),
  constraint  fk_orders_parent foreign key (CustId)
    references Customers on delete cascade
) partition by reference (fk_orders_parent);
```

Create a sharded table for LineItems

```
CREATE SHARDED TABLE LineItems
(
  OrderId     INTEGER NOT NULL,
  CustId      VARCHAR2(60) NOT NULL,
  ProductId   INTEGER NOT NULL,
  Price       NUMBER(19,4),
  Qty         NUMBER,
  constraint  pk_items primary key (CustId, OrderId, ProductId),
  constraint  fk_items_parent foreign key (CustId, OrderId)
    references Orders on delete cascade
) partition by reference (fk_items_parent);
```

8. Create any required duplicated tables.

   In this example, the Products table is a duplicated object.

```
CREATE DUPLICATED TABLE Products
(
  ProductId  INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  Name       VARCHAR2(128),
  DescrUri   VARCHAR2(128),
  LastPrice  NUMBER(19,4)
) tablespace products_tsp;
```

Next you should monitor the DDL processing and verify that the tablespace sets, tables, and chunks were correctly created on all of the shards.

DDL Failure and Recovery Examples

The following examples demonstrate the steps to issue a DDL, monitor its status, and what to do when errors are encountered.

When a DDL fails on a shard, all further DDLs on that shard are blocked until the failure is resolved and the GDSCTL `recover shard` command is run.

Note that you must have `GSM_ADMIN` privileges to run these GDSCTL commands.

The following examples demonstrate the case when a DDL is issued using SQL*Plus, but the same status checking and corrective actions apply when using the `GDSCTL SQL` command.

Example 6-1 A DDL processing error on the shard catalog

In this example the user makes a typo in the `CREATE USER` command.

```
SQL> alter session enable shard ddl;
Session altered.

SQL> CREATE USER example_user IDENTRIFIED BY out_standing1;
CREATE USER example_user IDENTRIFIED BY out_Standing1
                         *
ERROR at line 1:
ORA-00922: missing or invalid option
```

The DDL fails to run on the shard catalog and, as expected, the GDSCTL `show ddl` command shows that no DDL was run on any of the shards:

```
GDSCTL> show ddl
id      DDL Text                          Failed shards
--      --------                          -------------
```

Then the user repeats the command with the correct spelling. Note that there is no need to run `alter session enable shard ddl` again because the same session is used.

```
SQL> CREATE USER example_user IDENTIFIED BY out_Standing1;
User created.
```

Now `show ddl` shows that the DDL has been successfully run on the shard catalog database and it did not fail on any shards that are online.

```
GDSCTL> show ddl
id      DDL Text                                      Failed shards
--      --------                                      -------------
1       create user example_user identified by *****
```

> **Note:**
>
> For any shard that is down at the time of the DDL processing, the DDL is automatically applied when the shard is back up.

Example 6-2 Recovery from an error on a shard by issuing a corrective action on that shard

In this example, the user attempts to create a tablespace set for system-managed sharded tables. But the data file directory on one of the shards is not writable, so the DDL is successfully run on the catalog, but fails on the shard.

```
SQL> connect example_user/out_Standing1
Connected

SQL> create tablespace set tbsset;
Tablespace created.
```

Note that there is no need to run `alter session enable shard ddl` because the user example_user was created as the sharded database user and shard ddl is enabled by default.

Check status using GDSCTL `show ddl`:

```
GDSCTL> show ddl
id      DDL Text                                       Failed shards
--      --------                                       -------------
```

```
1       create user example_user identified by *****
2       create tablespace set tbsset                  shard01
```

The command output shows that the DDL failed on the shard shard01. Run the GDSCTL `config shard` command to get detailed information:

```
GDSCTL> config shard -shard shard01

Conversion = ':'Name: shard01
Shard Group: dbs1
Status: Ok
State: Deployed
Region: east
Connection string: (DESCRIPTION=(ADDRESS=(HOST=shard01-host)(PORT=1521)
(PROTOCOL=tcp))
(CONNECT_DATA=(SID=shard01)))
SCAN address:
ONS remote port: 0
Disk Threshold, ms: 20
CPU Threshold, %: 75
Version: 18.0.0.0
Failed DDL: create tablespace set tbsset
DDL Error: ORA-02585: create tablepsace set failure, one of its tablespaces
not created
ORA-01119: error in creating database file \'/ade/b/3667445372/oracle/
rdbms/dbs/
SHARD01/datafile/o1_mf_tbsset_%u_.dbf\'
ORA-27040: file create error, unable to create file
Linux-x86_64 Error: 13: Permission denied
Additional information: 1 \(ngsmoci_execute\)
Failed DDL id: 2
Availability: ONLINE
```

The text beginning with “Failed DDL:” indicates the problem. To resolve it, the user must log in to the shard database host and make the directory writable.

Display the permissions on the directory:

```
cd $ORACLE_HOME/rdbms/dbs
 ls -l ../ | grep dbs
dr-xr-xr-x  4 oracle dba    102400 Jul 20 15:41 dbs/
```

Change the directory to writable:

```
chmod +w .
ls -l ../ | grep dbs
drwxrwxr-x  4 oracle dba    102400 Jul 20 15:41 dbs/
```

Go back to the GDSCTL console and issue the `recover shard` command:

```
GDSCTL> recover shard -shard shard01
```

Check the status again:

```
GDSCTL> show ddl
id      DDL Text                                        Failed shards
--      --------                                        -------------
1       create user example_user identified by *****
2       create tablespace set tbsset

GDSCTL> config shard -shard shard01

Conversion = ':'Name: shard01
Shard Group: dbs1
Status: Ok
State: Deployed
Region: east
Connection string: (DESCRIPTION=(ADDRESS=(HOST=shard01-host)(PORT=1521)
(PROTOCOL=tcp))
(CONNECT_DATA=(SID=shard01)))
SCAN address:
ONS remote port: 0
Disk Threshold, ms: 20
CPU Threshold, %: 75
Version: 18.0.0.0
Last Failed DDL:
DDL Error: ---
DDL id:
Availability: ONLINE
```

As shown above, the failed DDL error no longer appears.

Example 6-3 Recovery from an error on a shard by issuing a corrective action on all other shards

In this example, the user attempts to create another tablespace set, tbs_set, but the DDL fails on a shard because there is already an existing local tablespace with the same name.

On the shard catalog:

```
SQL> create tablespace set tbs_set;
Tablespace created.
```

Check status using the GDSCTL `show ddl` command:

```
GDSCTL> show ddl
id      DDL Text                                        Failed shards
--      --------                                        -------------
1       create user example_user identified by *****
2       create tablespace set tbsset
3       create tablespace set tbs_set                   shard01

GDSCTL> config shard -shard shard01
Conversion = ':'Name: shard01
……
```

```
Failed DDL: create tablespace set tbs_set
DDL Error: ORA-02585: create tablespace set failure, one of its tablespaces
not created
ORA-01543: tablespace \'TBS_SET\' already exists \(ngsmoci_execute\)
```

A solution to this problem is to login to shard01 as a local database administrator, drop the tablespace TBS_SET, and then run GDSCTL `recover shard -shard shard01`. But suppose you want to keep this tablespace, and instead choose to drop the newly created tablespace set that has the name conflict and create another tablespace set with a different name, such as tbsset2. The following example shows how to do that on the shard catalog:

```
SQL> drop tablespace set tbs_set;
SQL> create tablespace set tbs_set2;
```

Check status using GDSCTL:

```
GDSCTL> show ddl
id      DDL Text                                  Failed shards
--      --------                                  -------------
1       create user example_user identified by *****
2       create tablespace set tbsset
3       create tablespace set tbs_set             shard01
4       drop tablespace set tbs_set
5       create tablespace set tbsset2
```

You can see that DDLs 4 and 5 are not attempted on shard01 because DDL 3 failed there. To make this shard consistent with the shard catalog, you must run the GDSCTL `recover shard` command. However, it does not make sense to run DDL 3 on this shard because it will fail again and you actually do not want to create tablespace set tbs_set anymore. To skip DDL 3 run `recover shard` with the `-ignore_first` option:

```
GDSCTL> recover shard -shard shard01 -ignore_first
GSM Errors: dbs1 shard01:ORA-00959: tablespace \'TBS_SET\' does not exist
 (ngsmoci_execute)

GDSCTL> show ddl
id      DDL Text                                  Failed shards
--      --------                                  -------------
1       create user sidney identified by *****
2       create tablespace set tbsset
3       create tablespace set tbs_set
4       drop tablespace set tbs_set               shard01
5       create tablespace set tbsset2
```

There is no failure with DDL 3 this time because it was skipped. However, the next DDL (4 - drop tablespace set tbs_set) was applied and resulted in the error because the tablespace set to be dropped does not exist on the shard.

Because the `-ignore_first` option only skips the first DDL, you need to run `recover shard` again to skip the `drop` statement as well:

```
GDSCTL> recover shard -shard shard01 -ignore_first

GDSCTL> show ddl
id      DDL Text                                 Failed shards
--      --------                                 -------------
1       create user sidney identified by *****
2       create tablespace set tbsset
3       create tablespace set tbs_set
4       drop tablespace set tbs_set
5       create tablespace set tbsset2
```

Note that there are no longer any failures shown, and all of the DDLs were applied successfully on the shards.

When `recover shard` is run with the `-ignore_first` option, the failed DDL is marked to be ignored during incremental deployment. Therefore, DDL numbers 3 and 4 are skipped when a new shard is added to the sharded database, and only DDL numbers 1, 2, and 5 are applied.

High Speed Data Ingest with SQL*Loader

SQL*Loader is a bulk loader utility used for moving data from external files into the Oracle database.

Its syntax is similar to that of the DB2 load utility, but comes with more options. SQL*Loader supports various load formats, selective loading, and multi-table loads. Other benefits include:

- Streaming capability lets you receive data from a large group of clients without blocking
- Group records according to Oracle RAC shard affinity using native UCP
- Optimize CPU allocation while decoupling record processing from I/O
- Fastest insert method for the Oracle Database through Direct Path Insert, bypassing SQL and writing directly in the database files

Automatic Parallel Direct Path Load Using SQL*Loader

SQL*Loader enables direct data loading into the database shards for a high speed data ingest. SQL*Loader can load data faster and easier into Oracle Database with automatic parallelism and more efficient data storage.

In Oracle Database 23c, SQL*Loader client can automatically start a parallel direct path load for data without dividing the data into separate files and starting multiple SQL*Loader clients. This feature prevents fragmentation into many small data extents. The data doesn't need to be resident on the database server. Cloud users can employ this feature to load data in parallel without having to move data on to the cloud system if there is sufficient network bandwidth.

See *Oracle Database Utilities* topics Automatic Parallel Load of Table Data with SQL*Loader and Sharded Automatic Parallel Loading Modes for SQL*Loader for more information.

7

Oracle Globally Distributed Database Administration

Oracle Globally Distributed Database provides tools and some automation for the administration of a sharded database.

> **Note:**
>
> A multitenant container database is the only supported architecture in Oracle Database 21c and later releases. While the documentation is being revised, legacy terminology may persist. In most cases, "database" and "non-CDB" refer to a CDB or PDB, depending on context. In some contexts, such as upgrades, "non-CDB" refers to a non-CDB from a previous release.

The following topics describe the aspects of Oracle Globally Distributed Database administration in detail:

- Managing the Oracle Globally Distributed Database Stack
- Oracle Globally Distributed Database Users and Roles
- Backing Up and Recovering Oracle Globally Distributed Database
- Propagation of Parameter Settings Across Shards
- Patching and Upgrading Oracle Globally Distributed Database
- Managing Oracle Globally Distributed Database with Enterprise Manager Cloud Control
- Monitoring an Oracle Globally Distributed Database
- Shard Management
- Chunk Management
- Shard Director Management
- Region Management
- Shardspace Management
- Shardgroup Management
- Services Management

Managing the Oracle Globally Distributed Database Stack

Follow these recommended sequences for startup and shutdown of components in the Oracle Globally Distributed Database configuration.

Starting Up the Oracle Globally Distributed Database Stack

The following is the recommended startup sequence of the Oracle Globally Distributed Database stack:

- Start the shard catalog database and local listener.
- Start the shard directors (GSMs).
- Start up the shard databases and local listeners.
- Start the global services.
- Start the connection pools and clients.

Shutting Down the Oracle Globally Distributed Database Stack

The following is the recommended shutdown sequence of the Oracle Globally Distributed Database stack:

- Shut down the connection pools and clients.
- Stop the global services.
- Shut down the shard databases and local listeners.
- Stop the shard directors (GSMs).
- Stop the shard catalog database and local listener.

Oracle Globally Distributed Database Users and Roles

Here you will learn about the management of database users and roles specific to Oracle Globally Distributed Database.

Overview of Users and Roles

In Oracle Globally Distributed Database some types of users require certain roles and privileges.

For sharded databases there are three kinds of users:

- Sharded database/GSM administrator - Grant this user the `GSMADMIN_ROLE` role. This role should be granted to one, or only a few accounts, that require elevated privileges to do administrative tasks. This role has a number of powerful privileges, including `ALTER SYSTEM`.
- Sharded database schema owner - Grant this user the `SHARDED_SCHEMA_OWNER` role. This role should be granted only to accounts which own a sharded database schema. The role only has enough privileges to allow the account to manage their own schema for various sharded operations, for example, "select any table" would not be a privilege this role has.
- Regular sharded database user - This type of user includes any account which has been created under `ENABLE SHARD DDL`; these users have no special privileges or roles except those needed to run a sharded application. The database administrator decides which privileges these accounts need, and grants them individually to the account.

Oracle Globally Distributed Database Roles

Oracle Globally Distributed Database provides a set of predefined database roles to help in sharded database administration.

Most of the Oracle Globally Distributed Database roles don't have many privileges, but they do have execute rights on certain Oracle-delivered procedures and packages which allow them to perform administrative tasks.

Predefined Role	Description
GSMADMIN_ROLE	Should be granted to Oracle Globally Distributed Database administrators, so that they can administer the Oracle Globally Distributed Database configuration
SHARDED_SCHEMA_OWNER	Provides privileges for Oracle Globally Distributed Database schema owners to perform administrative tasks on their own schema
GSMCATUSER_ROLE	Granted only the Oracle delivered account GSMCATUSER for internal use
GSMROOTUSER_ROLE	Granted only to Oracle delivered account GSMROOTUSER for internal use
GSMUSER_ROLE	Granted only to Oracle delivered account GSMUSER for internal use

For more information about database roles, see Predefined Roles in an Oracle Database Installation.

About the GSMUSER Account

The GSMUSER account is used by GDSCTL and shard directors (global service managers) to connect to databases in an Oracle Globally Distributed Database configuration.

This account need to be unlocked for both the CDB and PDB.

GSMUSER exists by default on any Oracle database. In an Oracle Globally Distributed Database configuration, the account is used to connect to shards instead of pool databases, and it must be granted both the SYSDG and SYSBACKUP system privileges after the account has been unlocked.

The password given to the GSMUSER account is used in the gdsctl add shard command. Failure to grant SYSDG and SYSBACKUP to GSMUSER on a new shard causes gdsctl add shard to fail with an ORA-1031: insufficient privileges error.

> **See Also:**
>
> add shard in *Global Data Services Concepts and Administration Guide*

About the GSMROOTUSER Account

GSMROOTUSER is a database account specific to Oracle Globally Distributed Database that is only used when pluggable database (PDB) shards are present. The account is used by GDSCTL and global service managers to connect to the root container of container databases (CDBs) to perform administrative tasks.

If PDB shards are not in use, the GSMROOTUSER user should not by unlocked nor assigned a password on any database. However, in sharded configurations containing PDB shards, GSMROOTUSER must be unlocked and granted the SYSDG and SYSBACKUP privileges before a successful gdsctl add cdb command can be run. The password for the GSMROOTUSER account can be changed after deployment if desired using the alter user SQL command in the root container of the CDB in combination with the gdsctl modify cdb -pwd command.

> **See Also:**
>
> add cdb in *Global Data Services Concepts and Administration Guide*

Backing Up and Recovering Oracle Globally Distributed Database

The GDSCTL utility lets you define a backup policy for an Oracle Globally Distributed Database and restore one or more shards, or the entire Oracle Globally Distributed Database, to the same point in time. Configured backups are run automatically, and you can define a schedule to run backups during off-peak hours.

GDSCTL commands in Oracle Globally Distributed Database enable and simplify the centralized management of backup policies for a sharded database, using Oracle MAA best practices. You can create a backup schedule using an incremental scheme that leverages the Oracle Job Scheduler. Oracle Recovery Manager (RMAN) performs the actual backup and restore operations.

Using the GDSCTL centralized backup and restore operations, you can

- Configure backups. See Configuring Automated Backups
- Monitor backup status. See Monitoring Backup Status
- List backups. See Listing Backups
- Validate backups. See Validating Backups
- Restore from backups. See Restoring Shards From Backup and Restoring the Shard Catalog from Backup

Backup and Restoration Terminology

The following are some terms you will encounter in the Oracle Globally Distributed Database backup and restore procedures.

- **Target database** - A database RMAN is to back up.

- **Global SCN** - A common point in time for all target databases for which a restore of the entire sharded database is supported. A restore point is taken at this global SCN, and the restore point is the point to which the sharded database (including the shard catalog) can be restored.

  Note that you are not prohibited from restoring the shard catalog or a specific shard to an arbitrary point in time. However, doing so may put that target in an inconsistent state with the rest of the sharded database and you may need to take corrective action outside of the restore operation.
- **Incremental backup** - Captures block-level changes to a database made after a previous incremental backup.
- **Level 0 incremental backup (level 0 backup)** - The incremental backup strategy starting point, which backs up blocks in the database. This backup is identical in content to a full backup; however, unlike a full backup, the level 0 backup is considered a part of the incremental backup strategy.
- **Level 1 incremental backup (level 1 backup)** - A level 1 incremental backup contains only blocks changed after a previous incremental backup. If no level 0 backup exists in either the current or parent database incarnation and you run a level 1 backup, then RMAN takes a level 0 backup automatically. A level 1 incremental backup can be either cumulative or differential.

Automated and On-Demand Backups

There are two type of backups in Oracle Globally Distributed Database: **automated** backups and **on-demand** backups.

- **Automated backups** are started by Oracle Scheduler jobs based on the job schedules, and they run in the background on the database servers.
- **On-demand backups** are started by users from `GDSCTL`.

  Internally, the on-demand backups are also started by Oracle Scheduler jobs on the database servers. The jobs are created on-fly when the on-demand backup commands are issued. They are temporary jobs and automatically dropped after the backups have finished.

See also: Scheduling Jobs with Oracle Scheduler

Supported Backup Destinations

The following are supported backup destinations for Oracle Globally Distributed Database.

- **Common disk/directory structure** (NFS mount) which can be located anywhere, including the shard catalog database host.
- **Zero Data Loss Recovery Appliance** (advantage is continuous backup, can leverage Data Guard broker to manage and monitor redo transport)
- **Oracle Object Storage**
- **Amazon S3** (Amazon Simple Storage Service)

Limitations

Note the following limitations for Oracle Globally Distributed Database backup and restore using GDSCTL.

- Microsoft Windows is not supported.
- You must provide for backup of Clusterware Repository if Clusterware is deployed

Prerequisites to Configuring Centralized Backup and Restore

Before configuring backup for an Oracle Globally Distributed Database, make sure the following prerequisites are met.

- Create one or more recovery catalogs in a dedicated database. If Recovery Appliance is used for the backup, the recovery catalog in the Recovery Appliance will be used.

  Before you can backup or restore a sharded database using `GDSCTL`, you must have access to a recovery catalog created in a dedicated database. This recovery catalog serves as a centralized RMAN repository for the shard catalog database and all of the shard databases.

  Note the following:

  – The version of the recovery catalog schema in the recovery catalog database must be compatible with the sharded database version because RMAN has compatibility requirements for the RMAN client, the target databases, and the recovery catalog schema. For more information, see *Oracle Database Backup and Recovery Reference*, cross-referenced below.

  – The recovery catalog must not share a host database with the shard catalog because the shard catalog database is one of the target databases in the sharded database backup configuration, and RMAN does not allow the recovery catalog to reside in a target database.

  – It is recommended that you back up the recovery catalog backup periodically, following appropriate best practices.

  – The shard catalog database and all of the shard databases can be configured to use the same recovery catalog or split between various recovery catalogs.

- Configure backup destinations for the shard catalog database and all of the shard databases.

  The backup destination types are either DISK or system backup to tape. The supported DISK destinations are NFS and Oracle ASM file systems.

  Note that, if you are using Oracle Object Storage, Recovery Appliance, or Amazon S3 as a backup destionation, RMAN treats them as tape internally. A spcial backup module needs to be installed on the target database host.

  System backup to tape destinations require additional software modules to be installed on the database host. They must be properly configured to work with RMAN.

  Using Recovery Appliance as the sharded database backup destination is special case. Recovery Appliance comes with a built-in recovery catalog. Because a database can only be registered as a target database with a single recovery catalog, when configuring the sharded database backup with a Recovery Appliance as the backup destination, the built-in recovery catalog is used for the sharded database backup and restore.

  The shard catalog database and all of the shard databases must be configured to use the same Recovery Appliance as the backup destination.

If the shard catalog database or the shard databases are in Data Guard configurations, you can choose to back up either the primary or standby databases.

See also:

- Using Oracle Object Storage as a Backup Destination
- Using Recovery Appliance as a Backup Destination
- Using Amazon S3 as a Backup Destination

- RMAN connects to the target databases as specific internal users to do database backup and restore with the exception of the shard catalog.

  For the shard catalog, a common user in the CDB hosting the shard catalog PDB must be provided at the time when sharded database backup is configured. This user must be granted the `SYSDG` and `SYSBACKUP` privileges. If the CDB is configured to use local undo for its PDBs, the `SYSBACKUP` privilege must also be granted commonly.

  For the shard databases, the internal CDB common user, `GSMROOTUSER`, is used. This user must be unlocked in the shard CDB root databases and granted the `SYSBACKUP` privilege in addition to other privileges that the sharded database requires for `GSMROOUSER`. If the CDB is configured to use local undo for its PDBs, the `SYSBACKUP` privilege must be granted commonly to `GSMROOTUSER`, meaning the `CONTAINER=ALL` clause must be used when granting the `SYSBACKUP` privilege.
- All of the `GDSCTL` commands for sharded database backup and restore operations require the shard catalog database to be open. If the shard catalog database itself must be restored, you must manually restore it.
- You are responsible for offloading backups to tape or other long-term storage media and following the appropriate data retention best practices.

> **Note:**
>
> See RMAN Compatibility in *Oracle Database Backup and Recovery Reference*

Configuring Automated Backups

Use the `GDSCTL CONFIG BACKUP` command to configure automated backups.

You should connect to a shard director (GSM) host to run the `GDSCTL` backup commands. If the commands are run from elsewhere, you must explicitly connect to the shard catalog database using the `GDSCTL CONNECT` command.

When you run the `GDSCTL` backup configuration, you can provide the following inputs.

- A list of databases

  The databases are the shard catalog database and shard databases. Backup configuration requires that the primary databases of the specified databases be open for read and write, but the standby databases can be mounted or open.

  If a database is in a Data Guard configuration when it is configured for backup, all of the databases in the Data Guard configuration are configured for backup. For a shard in Data Guard configuration, you must provide the backup destinations and start times for the primary and all of the standby shards.

This is different for the shard catalog database. The shard catalog database and all the shard catalog standby databases will share a backup destination and a start time.

- Connect strings to the recovery catalog databases

  For the connect string you need a user account with privileges for RMAN, such as `RECOVERY_CATALOG_OWNER` role.

- RMAN backup destination parameters

  These parameters include backup device and channel configurations. Different backup destinations can be used for different shards.

  Please note the following:

  – Backup destinations for shards in Data Guard configuration must be properly defined to ensure that the backups created from standby databases can be used to restore the primary database and conversely. See "Using RMAN to Back Up and Restore Files" in *Oracle Data Guard Concepts and Administration* for Data Guard RMAN support.

  – The same destination specified for the shard catalog database is used as the backup destination for the shard catalog standby databases.

  – For system backup to tape devices, the media managers for the specific system backup to tape devices are needed for RMAN to create channels to read and write data to the devices. The media manager must be installed and properly configured.

  See also:

  – Using Oracle Object Storage as a Backup Destination

  – Using Recovery Appliance as a Backup Destination

  – Using Amazon S3 as a Backup Destination

- Backup target type

  Backup target type defines whether the backups for the shard catalog database and shards should be done at the primary or one of the standby databases. It can be either `PRIMARY` or `STANDBY`. The default backup target type is `STANDBY`. For the shard catalog database or shards that are not in Data Guard configurations, the backups will be done on the shard catalog database or the shards themselves even when the backup target type is `STANDBY`.

- Backup retention policy

  The backup retention policy specifies a database recovery window for the backups. It is specified as a number of days.

  Obsolete backups are not deleted automatically, but a `GDSCTL` command is provided for you to manually delete them.

- Backup schedule

  Backup schedules specify the automated backup start time and repeat intervals for the level 0 and level 1 incremental backups. Different automated backup start times can be used for the shard catalog database and individual shards.

  The time is a local time in the time zone in which the shard catalog database or shard is located. The backup repeat intervals for the level 0 and level 1 incremental backups are the same for the shard catalog database and all the shards in the sharded database,

- CDB root database connect string for the shard catalog database

  The provided user account must have common `SYSBACKUP` privilege in the provided CDB.

When no parameters are provided for the `CONFIG BACKUP` command, `GDSCTL` displays the current sharded database backup configuration. If the backup has not been configured yet when the command is used to show the backup configuration, it displays that the backup is not configured.

To configure a backup, run `GDSCTL CONFIG BACKUP` as shown in the following example. For complete syntax, command options, and usage notes, run `HELP CONFIG BACKUP`.

The following example configures a backup channel of type DISK for the shard catalog database, two parallel channels of type DISK for each of the shards (shard spaces `dbs1` and `dbs2` are used in the shard list), the backup retention window is set to 14 days, the level 0 and level 1 incremental backup repeat intervals are set to 7 and 1 day, and the backup start time is set to 12:00 AM, leaving the incremental backup type the default `DIFFERENTIAL`, and the backup target type the default `STANDBY`.

```
GDSCTL> config backup -rccatalog rccatalog_connect_string
-destination "CATALOG::configure channel device type disk format '/tmp/rman/
backups/%d_%U'"
-destination "dbs1,dbs2:configure device type disk parallelism 2:configure
channel 1 device type disk format '/tmp/rman/backups/1/%U';configure channel
2 device type disk format '/tmp/rman/backups/2/%U'"
-starttime ALL:00:00 -retention 14 -frequency 7,1 -catpwd
gsmcatuser_password -cdb catcdb_connect_string;
```

Once GDSCTL has the input it displays output similar to the following, pertaining to the current status of the configuration operation.

```
Configuring backup for database "sales_catalog" ...

Updating wallet ...
The operation completed successfully

Configuring RMAN ...
new RMAN configuration parameters:
CONFIGURE CHANNEL DEVICE TYPE DISK FORMAT '/tmp/rman/backups/%d_%u';
new RMAN configuration parameters are successfully stored
starting full resync of recovery catalog
full resync complete

new RMAN configuration parameters:
CONFIGURE BACKUP OPTIMIZATION ON;
new RMAN configuration parameters are successfully stored
starting full resync of recovery catalog
full resync complete
...

Creating RMAN backup scripts ...
replaced global script full_backup
replaced global script incremental_backup
...
Creating backup scheduler jobs ...
```

```
The operation completed successfully


Creating restore point creation job ...
The operation completed successfully

Configuring backup for database "sales_east" ...

Updating wallet ...
The operation completed successfully

Configuring RMAN ...
new RMAN configuration parameters:
CONFIGURE DEVICE TYPE DISK PARALLELISM 2 BACKUP TYPE TO BACKUPSET;
new RMAN configuration parameters are successfully stored
starting full resync of recovery catalog
full resync complete

new RMAN configuration parameters:
CONFIGURE CHANNEL 1 DEVICE TYPE DISK FORMAT '/tmp/rman/backups/1/%u';
new RMAN configuration parameters are successfully stored
starting full resync of recovery catalog
full resync complete
...

Configuring backup for database "sales_west" ...
Updating wallet ...
The operation completed successfully

Configuring RMAN ...
...

Recovery Manager complete.
```

As shown in the `CONFIG BACKUP` command output above, `GDSCTL` does the following steps.

1. GDSCTL updates the shard wallets.

   The updated wallets will contain:

   - Connect string and authentication credentials to the RMAN catalog database.
   - Connect string and authentication credentials to the RMAN TARGET database.
   - Automated backup target type and start time.

2. GDSCTL sets up the RMAN backup environment for the database.

   This includes the following tasks.

   - Registering the database as a target in the recovery catalog.
   - Setting up backup channels.
   - Setting up backup retention policies.
   - Enabling control file and server parameter file auto-backup.

- Enabling block change tracking for all the target databases.

3. On the shard catalog, GDSCTL creates global RMAN backup scripts for level 0 and level 1 incremental backups.
4. On the shard catalog, GDSCTL creates a global restore point creation job.
5. On the shard catalog and each of the primary databases, GDSCTL
   - Creates DBMS Scheduler database backup jobs for level 0 and level 1 incremental backups
   - Schedules the jobs based on the backup repeat intervals you configure.

Specifying Multiple Recovery Catalogs

When configuring backups, you can specify different recovery catalogs for different shards and the shard catalog by running GDSCTL `CONFIG BACKUP` multiple times with different recovery catalogs and different shards or the shard catalog specified in each run of the command.

For example:

```
CONFIG BACKUP -shard shard_east -rccatalog rcadmin/rman@rc_east
CONFIG BACKUP -shard shard_west -rccatalog rcadmin/rman@rc_west
```

After running the above two commands, the shard “shard_east” will use “rc_east” as the recovery catalog, while “shard_west” will use “rc_west”, for all following manual and automatic backup jobs, and `LIST/DELETE/VALIDATE/RESTORE BACKUP` commands.

If the parameter `-rccatalog` is not provided, then the recovery catalog credentials and connect identifier specified previously will be used. For example, after running the above two commands, if you issue the following command without parameter `-rccatalog`:

```
CONFIG BACKUP -shard catalog ...
```

Then the shard catalog will use “rc_west” as the recovery catalog.

Note that if running `CONFIG BACKUP` against a sharded database for the first time but without `-rccatalog`, the command will do nothing but print an error message.

Backup Set Encryption

Backup sets to be stored in the Oracle Object Store must be encrypted. Set up encryption on all shards and shard catalog in a sharded database using the following procedure.

Note that you can use encryption even if you are not using Oracle Object Store as a backup destination. This may be preferred when there is sensitive data in the database or if you are using third-party cloud storage such as Amazon S3.

Task 1: Configure TDE Encryption

To enable the encryption of backup sets, a user having `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege needs to perform the following actions on all container databases (CDBs) at the root level for the shards and the shard catalog:

1. Configure the keystore location and type by setting the `WALLET_ROOT` and `TDE_CONFIGURATION` initialization parameters.

   a. First check if `WALLET_ROOT` points to an existing file location.

      If necessary, you can use the following SQL*Plus statement to change it.

      ```
      ALTER SYSTEM SET WALLET_ROOT=wallet-root-location SCOPE=BOTH
      ```

   b. Specify the keystore type.

      ```
      ALTER SYSTEM SET TDE_CONFIGURATION="KEYSTORE_CONFIGURATION=FILE"
      SCOPE=BOTH;
      ```

   c. Restart the database.

   Note the following:

   - The `WALLET_ROOT` parameter specifies the top directory for many different software keystores (such as TDE, Oracle Enterprise User Security (EUS), TLS). For TDE, the directory for automated discovery is `WALLET_ROOT/tde`.
   - In releases older than 19c, the `SQLNET.ENCRYPTION_WALLET_LOCATION` parameter was used to define the keystore directory location. This parameter has since been deprecated. Oracle recommends that you use the `WALLET_ROOT` static initialization parameter and `TDE_CONFIGURATION` dynamic initialization parameter instead.
   - If the value of parameter `TDE_CONFIGURATION` is `KEYSTORE_CONFIGURATION=FILE`, software keystore will be used. Two alternative keystore type values are `KEYSTORE_CONFIGURATION=OKV|HSM` for Oracle Key Vault or hardware security module keystores.

2. Create a password-protected software keystore.

   ```
   ADMINISTER KEY MANAGEMENT CREATE KEYSTORE 'wallet-root-location/tde'
    IDENTIFIED BY keystore_password;
   ```

   Note that *wallet-root-location* should be the same value as the initialization parameter `WALLET_ROOT`.

3. Open the software keystore.

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
    IDENTIFIED BY keystore_password CONTAINER=ALL;
   ```

4. Set the TDE encryption master key.

   ```
   ADMINISTER KEY MANAGEMENT SET KEY
    IDENTIFIED BY keystore_password WITH BACKUP CONTAINER=ALL;
   ```

See also: Configuring a Software Keystore in *Oracle Database Advanced Security Guide*

Task 2: Enable Encryption

To enable the encryption of backup sets, specify an encryption algorithm in the `GDSCTL CONFIG BACKUP` command using option `-encryption`. For example:

```
GDSCTL> CONFIG BACKUP … -encryption shard-list:encryption-algorithm
```

You can get a list of supported algorithms from column `ALGORITHM_NAME` in `V$RMAN_ENCRYPTION_ALGORITHMS`. When an algorithm is given, the `CONFIG BACKUP` command invokes the following RMAN scripts after registering a backup target database in the recovery catalog:

```
CONFIGURE ENCRYPTION FOR DATABASE ON;
CONFIGURE ENCRYPTION ALGORITHM TO encryption_algorithm;
```

> **Note:**
>
> You must configure Transparent Data Encryption on the shards and shard catalog where encryption is needed. Otherwise, manual and automatic backup jobs, or `VALIDATE/RESTORE BACKUP` commands will fail.

Disabling Backup Set Encryption

To disable encryption, you can set the `CONFIG BACKUP -encryption` option to `OFF`.

When `OFF` is specified, the encryption on the shards and/or shard catalog in *shard-list* is disabled. For example:

```
GDSCTL> CONFIG BACKUP … -encryption shard-list:OFF
```

Then the `GDSCTL` invokes the following RMAN script:

```
CONFIGURE ENCRYPTION FOR DATABASE OFF;
```

> **Note:**
>
> When the encryption is disabled, you cannot restore the database with encrypted backup sets.

Using Oracle Object Storage as a Backup Destination

Oracle Object Storage is a convenient location on Oracle Cloud Infrastructure where you can store files. Oracle Recovery Manager (RMAN) has a media library that lets you set Object Storage as a backup destination for sharded databases through the GDSCTL `CONFIG BACKUP` command.

To use Oracle Object Store as a repository for sharded database backups, complete the following prerequisites, and specify the `CONFIG BACKUP` destination as described below.

Prerequisites for Oracle Object Store Backup Destination

1. Configure Transparent Data Encryption (TDE) on all backup target databases, including the shard catalog and the shards where backup destinations will be the Object Storage as described in Backup Set Encryption.

   This includes:

   - Configure the keystore location and type by setting initialization parameters `WALLET_ROOT` and `TDE_CONFIGURATION`.
   - Create the software keystore used by TDE.

2. Enable the encryption of backup sets as described in Backup Set Encryption.
3. Get the following Oracle Cloud related accounts and IDs:
   - An Oracle Cloud account with access to Oracle Cloud Infrastructure Object Storage
   - Oracle Cloud Infrastructure API signing keys, tenant OCID, and user OCID
4. Create an Oracle Object Storage bucket to use as the backup destination..
5. Install Oracle Cloud Infrastructure Backup Modules on the shard catalog and all shards.

   During the installation specify:

   - The credentials for Oracle Cloud console - Note that the installer will put these credentials in a wallet.
   - Bucket name and endpoint URL
   - Oracle Cloud Infrastructure API signing keys, tenant OCID, and user OCID

   See Installing the Oracle Database Cloud Backup Module for OCI in *Using Oracle Database Backup Cloud Service* for more details.

Set the Backup Destination in the Sharded Database Backup Configuration

In the following example, Object Storage is set as the backup destination of shardspace dbs1.

```
GDSCTL> config backup
 -shard dbs1
 -rccatalog rccatalog_connect_string
 -catpwd password
 -cdb catcdb_connect_string
 -encryption dbs1:AES256
 -destination dbs1:"CONFIGURE DEFAULT DEVICE TYPE TO SBT":"CONFIGURE
CHANNEL DEVICE TYPE SBT
 PARMS='SBT_LIBRARY=/orclhome/lib/libopc.so,SBT_PARMS=(OPC_PFILE=/
orclhome/dbs/opct1.ora)'"
```

As shown above, Object Storage requires the following `CONFIG BACKUP` parameter settings in addition to the standard options:

- Enable encryption on the shardspace with option `-encryption`. To support the Oracle Object Storage as a backup destination, the backup sets must be encrypted.

- Specify the following in option `-destination` using RMAN device configuration and channel configuration statements.
  - `SBT_TAPE` as the backup default device
  - SBT library file and OPC configuration file locations in the backup channel parameters, using the following template:

```
CONFIGURE CHANNEL DEVICE TYPE sbt
   PARMS='SBT_LIBRARY=location-of-SBT-library-for-backup-module,
   SBT_PARMS=(OPC_PFILE=location-of-configuration-file)';
```

Using Recovery Appliance as a Backup Destination

Zero Data Loss Recovery Appliance (Recovery Appliance), configured for real-time redo transport, directly transports redo data from the protected database and stores it on the Recovery Appliance. This reduces the window of potential data loss that exists between successive archived log backups.

To use Recovery Appliance as a repository for sharded database backups, complete the following prerequisites and specify the `CONFIG BACKUP` destination as described below.

Prerequisites

1. Copy the shared library `libra.so` from the Recovery Appliance Backup Module to the hosts where the shard catalog and each shard are located.

   You can find this library file in the `$ORACLE_HOME/lib` directory of the Recovery Appliance.

2. (Required only if configuring real-time redo transport) Create a redo transport user at the root level of the shard catalog and shards and grant `CREATE SESSION` and `SYSOPER` privileges to this user.

```
create user c##rman1 identified by rman1;
grant create session, sysoper to rman1;
```

   Note that

   - This user must be a common user on the shard catalog or shard.
   - This user should have the same user name and password as the Recovery Appliance virtual private catalog user assigned to the shard catalog or shards.
   - If in a Data Guard configuration, you only create this user on the primary database, then the user is propagated to the standby databases.

This waives the requirements to install the Recovery Appliance Backup Module on the protected databases. Moreover, you do not need to enroll the protected database with `DBMS_RA` package. The `CONFIG BACKUP` command automates the setup.

Collect Required Information

- The location of the shared library, libra.so, on the shard catalog and each shard
- Connect string to the Recovery Appliance catalog
- The user name and password of a catalog owner to use in RMAN command `CONNECT CATALOG`

- The user name and password of a virtual private catalog (VPC) account of the Recovery Appliance

  HTTP digest access authentication must be enabled for this user.

  For example:

  ```
  create user vpc_user1 identified by vpc;
  grant create session to vpc_user1 identified by vpc;
  # enable HTTP digest access authentication
  alter user vpc_user1 identified by vpc digest enable;
  ```

- The name of a protection policy created on the Recovery Appliance
- The amount of disk space on Recovery Appliance reserved for the shard catalog and each shard
- (Required only if configuring real-time redo transport) The location of the auto login wallets used by shard catalog or shards for real-time redo transport. The `GDSCTL CONFIG BACKUP` command creates the wallet in this location.

Set the Backup Destination in the Sharded Database Backup Configuration

The following example shows how to issue GDSCTL command `CONFIG BACKUP` to set up a backup configuration using the Recovery Appliance “ra_east” as the backup destination of a sharded database:

```
GDSCTL> config backup
 -shard ALL
 -cdb catcdb_connect_string
 -zdlra_catalog username@ra_east
 -zdlra_vpc CATALOG:catalog_vpc_username
 -zdlra_vpc SH1:shard_vpc_username
 -zdlra_policy CATALOG:catalog_protection_policy_name
 -zdlra_policy SH1:shard_protection_policy_name
 -zdlra_space CATALOG:20G
 -zdlra_space SH1:2T
 -zdlra_lib_dir CATALOG:?/lib
 -zdlra_lib_dir SH1:/u01/app/oracle/product/19.0.0/dbhome_1/lib
```

> **Note:**
>
> The `-destination` and `-rccatalog` parameters are not used for configuring Recovery Appliance as the backup destination.

The `-zdlra_*` parameters used in the example are explained below:

Parameter	Description/Usage Notes/Examples
`-zdlra_catalog` *`username[/password]@connect-string`*	This parameter is mutually exclusive with `-rccatalog`. If specified, it indicates that all listed shards and/or shard catalog in parameter `-shard` will use the specified Recovery Appliance as the backup destination. Provide the user name and connect string of the Recovery Appliance administrator that should have RASYS privilege. For example: `-zdlra_catalog cat_username@ra_east` Note: • The password will be prompted if not specified. • The user name specified in this parameter is the administrator of the Recovery Appliance that should have RASYS privilege. It is different from the virtual private catalog user provided in parameter `-zdlra_vpc`.

`-zdlra_vpc(ALL|CATALOG|`*`shard-list`*`):`*`username[/password]`*	Mandatory if `-zdlra_catalog` is specified.
	It specifies the user of a Recovery Appliance VPC (virtual private catalog) user for a specified shard, shardgroup, shardspace, or region.
	For example:
	`-zdlra_vpc CATALOG:catalog_vpc_username`
	`-zdlra_vpc SH1:shard_vpc_username`
	Note:
	• This parameter can appear multiple times on one command line to allow different VPC users for different shards.
	• The shard list specifies a comma-separated list of shard identifiers. They can be shardspaces, shardgroups, regions, or shard names.
	• The password will be prompted if not specified.
	This user is different from the Recovery Appliance administrator (RASYS user) provided in the parameter `-zdlra_catalog`.

Parameter	Description/Usage Notes/Examples

`-zdlra_policy (ALL|CATALOG|`*shard-list*`):`*protection-policy-name*	Mandatory if `-zdlra_catalog` is specified.
	It specifies the name of a protection policy defined in Recovery Appliance.
	For example:
	`-zdlra_policy CATALOG:brzone_dev`
	`-zdlra_policy SH1:silver_dev`
	Note:
	• This parameter can appear multiple times on one command line to allow different protection policies for different shards.
	• The shard list specifies a comma-separated list of shard identifiers. They can be shardspaces, shardgroups, regions, or shard names.

`-zdlra_space (ALL|CATALOG|`*shard-list*`):(0-9)+(K|M|G|T|P|E|Z|Y)`	Mandatory if `-zdlra_catalog` is specified.
	Specifies the amount of disk space allocated on the Recovery Appliance for each shard and shard catalog.
	For example:
	`-zdlra_space CATALOG:20G`
	`-zdlra_space SH1:2T`
	Note:
	• This parameter can appear multiple times on one command line in case the space requirements for different shards are different.
	• The shard list specifies a comma-separated list of shard identifiers. They can be shardspaces, shardgroups, regions, or shard names.
	• The letter at the end is a unit specifier in the following list:
	K: Kilobytes
	M: Megabytes
	G: Gigabytes
	T: Terabytes
	P: Petabytes
	E: Exabytes
	Z: Zettabytes
	Y: Yottabytes

Parameter	Description/Usage Notes/Examples

`-zdlra_lib_dir (ALL|CATALOG|`*shard-list*`):`*library-location*	Mandatory if `-zdlra_catalog` is specified.
	Specifies the location of the shared library, `libra.so`, for the Recovery Appliance backup module. This location should be accessible by the specified shard or shard catalog.
	For example:
	`-zdlra_lib_dir CATALOG:?/lib`
	`-zdlra_lib_dir SH1:/u01/app/oracle/product/23.1.0/dbhome_1/lib`
	Note:
	• This parameter can appear multiple times on one command line in case the library locations on different shards are different.
	• You can use a question mark (?) to represent ORACLE_HOME environment variable. The "at" sign (@) stands for ORACLE_SID environment variable. This can help simplify the setup if the Oracle installation folders for different shards are different.
	For example, if the library file location is in the folder `$ORACLE_HOME/lib/`, then the user can use the following parameter to specify the library file location for all shards:
	`-zdlra_lib_dir ALL:?/lib`

Parameter	Description/Usage Notes/Examples

`-zdlra_redo_wallet (ALL|CATALOG|`*`shard-list`*`):`*`wallet-location`*	Optional.
	Specifies the location of the auto-login wallet used by the redo transport layer to log into Recovery Appliance.
	Once specified, `CONFIG BACKUP` command will configure real-time redo transport for the databases specified in parameter `-shard` and create an auto-login wallet for each protected database.
	For example:
	`-zdlra_redo_wallet CATALOG:?/wallet`
	`-zdlra_redo_wallet SH1:/u01/app/oracle/product/23.1.0/dbhome_1/wallet`
	Note:
	• This parameter can appear multiple times on one command line in case the wallet locations on different shards are different.
	• You can use question mark (?) and "at" sign (@) to represent ORACLE_HOME and ORACLE_SID environment variables respectively. This can help simplify the setup if the Oracle installation folders for different shards are different.
	• The following entry will be added into the auto-login wallet: (*`connect-string-specified-in-parameter-zdlracatalog`*, *`VPC-username`*, *`VPC-user-password`*)
	where VPC user name and password are from parameter `-zdlra_vpc`. As explained in the prerequisites, you must create a redo transport user on the protected database having the same user name as VPC user.

See Global Data Services Control Utility (GDSCTL) Command Reference in *Global Data Services Concepts and Administration Guide* for details about other `CONFIG BACKUP` parameters.

Real-Time Redo Transport Post-Configuration Requirements

If the real-time redo transport is required, perform the following actions after running `CONFIG BACKUP`:

- Add the location of wallets used by redo transport to `sqlnet.ora` and set `SQLNET.WALLET_OVERRIDE` to `TRUE`.

```
WALLET_LOCATION=
  (SOURCE=(METHOD=FILE)(METHOD_DATA=
  (DIRECTORY= wallet_location)))
SQLNET.WALLET_OVERRIDE=TRUE
```

- Reboot the primary databases of the shard catalog and shards, so the changes to `sqlnet.ora` can take effect.

- In the shard catalog and shards (primary and standby databases for all), change the initialization parameter `REDO_TRANSPORT_USER` to the redo transport user (which uses the same user name as the virtual private catalog (VPC) user).

  For example:

  ```
  alter system set "redo_transport_user"=" c##rman1";
  ```

- You can run a manual backup once, or wait until the first automatic backup runs, then the real-time redo transport is active.

  Once active, the following query on Recovery Appliance should return `TRUE`:

  ```
  select NZDL_ACTIVE
  from DBMS_RA. RA_DATABASE
  where DB_UNIQUE_NAME='DB_UNIQUE_NAME-of-the-protected-database'
  ```

Using Amazon S3 as a Backup Destination

Amazon S3 (Amazon Simple Storage Service) is a cloud-based storage service. Oracle's Secure Backup Cloud Module can store backup sets on Amazon S3 storage using Oracle Recovery Manager (RMAN).

To use Amazon S3 as a repository for sharded database backups, complete the following prerequisites and specify the `CONFIG BACKUP` destination as described below.

Prerequisites for Amazon S3 Backup Destination

The Oracle Secure Backup Cloud Module must be installed on the shard catalog and the shards where backup sets will be sent to Amazon S3 cloud storage.

During the installation, you need to provide some parameters, including, but not limited to:

- (Mandatory) `AWSID`, the access key ID for Amazon S3 cloud storage service
- (Mandatory) `AWSKey`, the password for the above ID
- (Optional) `awsEndpoint`, the host name where the backup sets will be sent
- (Optional) `awsPort`, the HTTP/HTTPS connection port number
- (Optional) `location`, the Amazon S3 location
- (Optional) `walletDir`, a directory for Oracle wallet to store Amazon S3 credentials, and proxy information if applicable
- (Optional) `configFile`, a configuration file that contains the parameters used by RMAN, including the Oracle wallet directory.
- (Optional) `libDir`, a directory to store the downloaded Oracle Secure Backup Cloud Module library

During the installation, you will create a wallet to store Amazon S3 credentials and proxy information if applicable, create a configuration file, and download Oracle Secure Backup Cloud Module library.

For detailed information about the installation of backup module, See Using Oracle Secure Backup Cloud Module on Amazon S3 in *Oracle Database Backup and Recovery Reference*.

It is also recommended that you encrypt backup sets destined for Amazon S3 storage. See Backup Set Encryption for details.

Set the Backup Destination in the Sharded Database Backup Configuration

The following example shows how to issue GDSCTL command `CONFIG BACKUP` to set up a backup configuration using the Amazon S3 as the backup destination of shardspace DBS1, and enabling backup set encryption.

```
GDSCTL> config backup
 -shard dbs1
 -rccatalog rccatalog_connect_string
 -catpwd password
 -cdb catcdb_connect_string
 -encryption dbs1:AES256
 -destination dbs1: "CONFIGURE DEFAULT DEVICE TYPE TO SBT":
"CONFIGURE CHANNEL DEVICE TYPE 'SBT_TAPE'
PARMS 'SBT_LIBRARY=/u01/app/oracle/product/19.0.0/dbhome_1/lib/
libosbws.so
ENV=(OSB_WS_PFILE=/u01/app/oracle/product/19.0.0/dbhome_1/dbs/
osbwssales.ora)'"
```

To use Amazon S3 as a backup destination, you must provide the following information when writing channel config statement in option `-destination`.

- The path to shared library libra.so
- The location of the configuration file used by RMAN
  The template for the `CONFIGURE CHANNEL` command is:

  ```
  CONFIGURE CHANNEL DEVICE TYPE 'SBT_TAPE'
     PARMS 'SBT_LIBRARY=secure-backup-library-location
     ENV=(OSB_WS_PFILE=configuration-file-location)'
  ```

  For more information on RMAN channel settings related to Amazon S3, See Configuring RMAN SBT Channels for Recovery Appliance in *Zero Data Loss Recovery Appliance Protected Database Configuration Guide*.

Enabling and Disabling Automated Backups

You can enable or disable backups on all shards, or specific shards, shardspaces, or shardgroups.

All backup jobs are initially disabled. They can be enabled by running the `GDSCTL ENABLE BACKUP` command.

```
GDSCTL> ENABLE BACKUP
```

When not specified, `ENABLE BACKUP` enables the backup on all shards. You can optionally list specific shards, shardspaces, or shardgroups on which to enable the backup.

```
GDSCTL> ENABLE BACKUP -shard dbs1
```

The `DISABLE BACKUP` command disables an enabled backup.

```
GDSCTL> DISABLE BACKUP -shard dbs1
```

Backup Job Operation

Once configured and enabled, backup jobs run on the primary shard catalog database and the primary shards as scheduled.

After a backup job is configured, it is initially disabled. You must enable a backup job for it to run as scheduled. Use the GDSCTL commands `ENABLE BACKUP` and `DISABLE BACKUP` to enable or disable the jobs.

Backup jobs are scheduled based on the backup repeat intervals you configure for the level 0 and level 1 incremental backups, and the backup start time for the shard catalog database and the shards.

Two separate jobs are created for level 0 and level 1 incremental backups. The names of the jobs are `AUTOMATED_SDB_LEVEL0_BACKUP_JOB` and `AUTOMATED_SDB_LEVEL1_BACKUP_JOB`. Full logging is enabled for both jobs.

When running, the backup jobs find the configured backup target type (`PRIMARY` or `STANDBY`), figure out the correct target databases based on the backup target type, and then launch RMAN to back up the target databases. RMAN uses the shard wallets updated during the backup configuration for database connection authentication.

Note that sharded database chunk moves do not delay automated backups.

Monitoring Backup Status

There are a few different ways to monitor the status of automated and on-demand backup jobs.

Monitoring an Automated Backup Job

Because full logging is enabled for the automated backup jobs, DBMS Scheduler writes job processing details in the job log and views. The Scheduler job log and views are your basic resources and starting point for monitoring the automated backups. Note that although the DBMS Scheduler makes a list of job state change events available for email notification subscription. This capability is not used for sharded database automated backups.

You can use the GDSCTL command `LIST BACKUP` to view the backups and find out whether backups are created at the configured backup job repeat intervals.

Automated backups are not delayed by chunk movement in the sharded database, so the backup creation times should be close to the configured backup repeat intervals and the backup start time.

Monitoring an On-Demand Backup Job

Internally, on-demand backup jobs are also started by DBMS Scheduler jobs on the database servers. The names of the temporary jobs are prefixed with tag `MANUAL_BACKUP_JOB_`. On-demand backups always run in the same session that GDSCTL uses to communicate with the database server. Failures from the job are sent directly to the client.

Using DBMS Scheduler Jobs Views

The automated backup jobs only run on the primary shard catalog database and the primary shards. To check the backup job details for a specific target database, connect to the database, or its primary database if the database is in a Data Guard configuration, using SQL*PLUS, and query the DBMS Scheduler views `*_SCHEDULER_JOB_LOG` and `*_SCHEDULER_JOB_RUN_DETAILS` based on the job names.

The names of the two automated backup jobs are `AUTOMATED_SDB_LEVEL0_BACKUP_JOB` and `AUTOMATED_SDB_LEVEL1_BACKUP_JOB`.

You can also use the GDSCTL command `STATUS BACKUP` to retrieve the job state and run details from these views. See Viewing Backup Job Status for more information about running `STATUS BACKUP`.

The job views only contain high level information about the job. For job failure diagnosis, you can find more details about the job in the RDBMS trace files by grepping the job names.

If no errors are found in the job, but still no backups have been created, you can find the PIDs of the processes that the jobs have created to run RMAN for the backups in the trace files, and then look up useful information in the trace files associated with the PIDs.

Using Backup Command Output

This option is only available for on-demand backups.

When you start on-demand backups with `GDSCTL RUN BACKUP`, you can specify the `-sync` command option. This forces all backup tasks to run in the foreground, and the output from the internally launched RMAN on the database servers is displayed in the GDSCTL console.

The downside of running the backup tasks in the foreground is that the tasks will be run in sequence, therefore the whole backup will take more time to complete.

See the GDSCTL reference in *Oracle Database Global Data Services Concepts and Administration Guide* for detailed command syntax and options.

Viewing an Existing Backup Configuration

When `GDSCTL CONFIG BACKUP` is not provided with any parameters, it shows the current backup configuration.

Because the parameters `-destination` and `-starttime` can appear more than once in `CONFIG BACKUP` command line for different shards and backup configuration can be done more than once, multiple items could be listed in each of the Backup destinations and Backup start times sections. The items are listed in the same order as they are specified in the CONFIG BACKUP command line and the order the command is repeatedly run.

To view an existing backup configuration, run `CONFIG BACKUP`, as shown here.

```
GDSCTL> CONFIG BACKUP;
```

If a sharded database backup has not been configured yet, the command output will indicate it. Otherwise the output looks like the following:

```
GDSCTL> config backup
Recovery catalog database user: rcadmin
Recovery catalog database connect descriptor:
(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=den02qxr)(PORT=1521))
(CONNECT_DATA=(SERVICE_NAME=cdb6_pdb1.example.com)))
Catalog database root container user: gsm_admin
Catalog database root container connect descriptor:
(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=den02qxr)(PORT=1521))
(CONNECT_DATA=(SERVICE_NAME=v1908.example.com)))
Backup retention policy in days: 14
Level 0 incremental backup repeat interval in minutes: 10080
Level 1 incremental backup repeat interval in minutes: 1440
Level 1 incremental backup type : DIFFERENTIAL
Backup target type: STANDBY
Backup destinations:
catalog::channel device type disk format '/tmp/rman/backups/%d_%u'
dbs1,dbs2:device type disk parallelism 2:channel 1 device type disk format
'/tmp/rman/backups/1/%u';channel 2 device type disk format '/tmp/rman/
backups/2/%u'
catalog::configure channel device type disk format '/tmp/rman/backups/%d_%u'
dbs1,dbs2:configure device type disk parallelism 2:configure channel 1
device type disk format '/tmp/rman/backups/1/%u';configure channel 2 device
type disk format '/tmp/rman/backups/2/%u'
Backup start times:
all:00:00
```

CONFIG BACKUP Output for Multiple Recovery Catalogs

The output of `CONFIG BACKUP` issued without any parameters on a configuration with multiple recovery catalogs prints the recovery catalog users and connect identifiers for different shards and shard catalog. The last used recovery catalog will be printed as well. For example:

```
GDSCTL> config backup
...
Recovery catalog database user:
last::rcadmin_west
shard_east::rcadmin_east
shard_west::rcadmin_west
Recovery catalog databaseconnect identifier:
last::<theconnect identifier of rc_west>
shard_east::<theconnect identifier of rc_east>
shard_west::<theconnect identifier of rc_west>
...
```

> **Note:**
>
> - In the actual output for the above example, three real connector identifiers will be printed.
> - The example shows only the parts related to recovery catalog settings.

Running On-Demand Backups

The `GDSCTL RUN BACKUP` command lets you start backups for the shard catalog database and a list of shards.

All on-demand backups are level 0 incremental backups. On-demand backups have no impact on the automated backup schedules configured for the shard catalog database and the shards.

Internally, on-demand backups are started by DBMS Scheduler jobs on the database servers. The jobs are created on-the-fly when the on-demand backup command `RUN BACKUP` is run.

On-demand backup jobs are temporary jobs, and they are automatically dropped after the backups have finished.

The names of the temporary jobs are prefixed with tag `MANUAL_BACKUP_JOB_`.

To use `RUN BACKUP`, you must have already set up the backup configuration with the `CONFIG BACKUP` command.

The `RUN BACKUP` command requires the shard catalog database and any primary shards to be backed up to be open.

```
GDSCTL> RUN BACKUP -shard dbs1
```

The `-shard` option lets you specify a set of shards, shardspaces or shardgroups on which to run the backup. To take an on-demand backup on shardspace `dbs1`, you can run `RUN BACKUP` as shown in the example above.

See the GDSCTL reference in *Oracle Database Global Data Services Concepts and Administration Guide* for detailed command syntax and options.

Viewing Backup Job Status

Use GDSCTL command `STATUS BACKUP` to view the detailed state on the scheduled backup jobs in the specified shards. Command output includes the job state (enabled or disabled) and the job run details.

By default, the command displays the job run details of all the runs that the automated backup jobs have had from 30 days ago in the specified shards. If the job run details for different periods are needed, options -start_time and -end_time must be used.

Run `STATUS BACKUP` as shown in the following examples.

The following `STATUS BACKUP` command example lists the job state and all job run details from the SDB catalog and the primary shard “rdbmsb_cdb2_pdb1”:

```
GDSCTL> status backup -catpwd -shard catalog,rdbmsb_cdb2_pdb1;
"GSMCATUSER" password:***

Retrieving scheduler backup job status for database "rdbms" ...
Jobs:
  Incremental Level 0 backup job is enabled
    Job schedule start time: 2020-07-27 00:00:00.000 -0400
    Job repeat interval: freq=daily;interval=1
  Incremental Level 1 backup job is enabled
    Job schedule start time: 2020-07-27 00:00:00.000 -0400
    Job repeat interval: freq=minutely;interval=60
  Global restore point create job is enabled
    Job schedule start time: 2020-07-27 23:59:55.960 -0400
    Job repeat interval: freq=hourly

Run Details:
  Incremental Level 1 backup job status: SUCCEEDED
    Job run actual start time: 2020-07-26 14:00:00.177 -0400
    Job run slave process ID: 9023
  Incremental Level 1 backup job status: SUCCEEDED
    Job run actual start time: 2020-07-26 22:00:01.305 -0400
Job run slave process ID: 59526
…
Global restore point create job status: SUCCEEDED
    Job run actual start time: 2020-07-27 15:28:37.603 -0400
    Job run slave process ID: 44227
  …
  Global restore point create job status: SUCCEEDED
    Job run actual start time: 2020-07-27 17:28:38.251 -0400
    Job run slave process ID: 57611

Retrieving scheduler backup job status for database "rdbmsb_cdb2_pdb1" ...
Jobs:
  Incremental Level 0 backup job is enabled
    Job schedule start time: 2020-07-28 00:00:00.000 -0400
    Job repeat interval: freq=daily;interval=1
  Incremental Level 1 backup job is enabled
    Job schedule start time: 2020-07-28 00:00:00.000 -0400
    Job repeat interval: freq=minutely;interval=60

Run Details:
  Incremental Level 1 backup job status: SUCCEEDED
    Job run actual start time: 2020-07-26 14:00:00.485 -0400
    Job run slave process ID: 9056
  …
  Incremental Level 1 backup job status: SUCCEEDED
    Job run actual start time: 2020-07-27 14:33:42.702 -0400
    Job run slave process ID: 9056
  Incremental Level 0 backup job status: SUCCEEDED
    Job run actual start time: 2020-07-27 00:00:01.469 -0400
    Job run slave process ID: 75176
```

The following command lists the scheduler backup job state and the details of the job runs in the time frame from 2020/07/26 12:00:00 to 07/27 00:00 from the SDB catalog and the primary shard “rdbmsb_cdb2_pdb1”:

```
GDSCTL> status backup -start_time "2020-07-26 12:00:00" -end_time
"2020-07-27 00:00:00" -catpwd -shard catalog,rdbmsb_cdb2_pdb1;
"GSMCATUSER" password:***

Retrieving scheduler backup job status for database "rdbms" ...
Jobs:
  Incremental Level 0 backup job is enabled
    Job schedule start time: 2020-07-27 00:00:00.000 -0400
    Job repeat interval: freq=daily;interval=1
  Incremental Level 1 backup job is enabled
    Job schedule start time: 2020-07-27 00:00:00.000 -0400
    Job repeat interval: freq=minutely;interval=60
  Globa1 restore point create job is enabled
    Job schedule start time: 2020-07-27 23:59:55.960 -0400
    Job repeat interval: freq=hourly

Run Details:
  Incremental Level 1 backup job status: SUCCEEDED
    Job run actual start time: 2020-07-26 14:00:00.177 -0400
    Job run slave process ID: 9023
  …
  Incremental Level 1 backup job status: SUCCEEDED
    Job run actual start time: 2020-07-26 23:50:00.293 -0400
    Job run slave process ID: 74171
  Globa1 restore point create job status: SUCCEEDED
    Job run actual start time: 2020-07-26 14:28:38.263 -0400
    Job run slave process ID: 11987
  …
  Globa1 restore point create job status: SUCCEEDED
    Job run actual start time: 2020-07-26 23:28:37.577 -0400
    Job run slave process ID: 69451

Retrieving scheduler backup job status for database
"rdbmsb_cdb2_pdb1" ...
Jobs:
  Incremental Level 0 backup job is enabled
    Job schedule start time: 2020-07-28 00:00:00.000 -0400
    Job repeat interval: freq=daily;interval=1
  Incremental Level 1 backup job is enabled
    Job schedule start time: 2020-07-28 00:00:00.000 -0400
    Job repeat interval: freq=minutely;interval=60

Run Details:
  Incremental Level 1 backup job status: SUCCEEDED
    Job run actual start time: 2020-07-26 14:00:00.485 -0400
    Job run slave process ID: 9056
  Incremental Level 1 backup job status: SUCCEEDED
    Job run actual start time: 2020-07-26 22:11:50.931 -0400
    Job run slave process ID: 9056
```

Listing Backups

Use `GDSCTL LIST BACKUP` to list backups usable to restore a sharded database or a list of shards to a specific global restore point.

The command requires the shard catalog database to be open, but the shards can be in any of the started states: nomount, mount, or open.

You can specify a list of shards to list backups for in the command. You can also list backups usable to restore the control files of the listed databases and list backups for standby shards.

The following example shows the use of the command to list the backups from shard `cdb2_pdb1` recoverable to restore point `BACKUP_BEFORE_DB_MAINTENANCE`.

```
GDSCTL> LIST BACKUP -shard cdb2_pdb1 -restorepoint
BACKUP_BEFORE_DB_MAINTENANCE
```

If option `-controlfile` is used, `LIST BACKUPS` will only list the backups usable to restore the control files of the specified shards. If option `-summary` is used, the backup will be listed in a summary format.

```
GDSCTL> list backup -shard shd1,shd2 -controlfile -summary
```

Validating Backups

Run the `GDSCTL VALIDATE BACKUP` command to validate sharded database backups against a specific global restore point for a list of shards. The validation confirms that the backups to restore the databases to the specified restore point are available and not corrupted.

The shard catalog database must be open, but the shard databases can be either mounted or open. If the backup validation is for database control files, the shards can be started nomount.

The following example validates the backups of the control files from the shard catalog databases recoverable to restore point `BACKUP_BEFORE_DB_MAINTENANCE`.

```
GDSCTL> VALIDATE BACKUP -shard shd1,shd2 -controlfile -restorepoint
BACKUP_BEFORE_DB_MAINTENANCE
```

Backup validation for shards are done one shard a time sequentially.

Deleting Backups

Use the `GDSCTL DELETE BACKUP` command to delete backups from the recovery repository.

The `DELETE BACKUP` command deletes the sharded database backups identified with specific tags from the recovery repository. It deletes the records in the recovery database for the backups identified with the provided tags, and, if the media where the files are located is accessible, the physical files from the backup sets from those backups. This is done for each of the target databases. You will be prompted to confirm before the actual deletion starts.

To run this command, the shard catalog database must be open, but the shard databases can be either mounted or open.

The following is an example of deleting backups with tag `odb_200414205057124_0400` from shard `cdb2_pdb1`.

```
GDSCTL> DELETE BACKUP -shard cdb2_pdb1 -tag ODB_200414205057124_0400
"GSMCATUSER" password:

This will delete identified backups, would you like to continue [No]?y

Deleting backups for database "cdb2_pdb1" ...
```

Creating and Listing Global Restore Points

A restore point for a sharded database that we call a global restore point, actually maps to a set of normal restore points in the individual primary databases in a sharded database.

These restore points are created at a common SCN across all of the primary databases in the sharded database. The restore points created in the primary databases are automatically replicated to the Data Guard standby databases. When the databases are restored to this common SCN, the restored sharded database is guaranteed to be in a consistent state.

The global restore point creation must be mutually exclusive with sharded database chunk movement. When the job runs, it first checks whether any chunk moves are going on and waits for them to finish. Sometimes the chunk moves might take a long time. Also, new chunk moves can start before the previous ones have finished. In that case the global restore point creation job might wait for a very long time before there is an opportunity to generate a common SCN and create a global restore point from it. Therefore, it is not guaranteed that a global restore point will be created every hour.

To create the global restore point, run the `GDSCTL` command `CREATE RESTOREPOINT` as shown here.

```
GDSCTL> CREATE RESTOREPOINT
```

The global restore point creation job is configured on the shard catalog database. The name of the job is `AUTOMATED_SDB_RESTOREPOINT_JOB`. Full logging for this job is enabled.

You can optionally enter a name for the restore point by using the `-name` option as shown here.

```
GDSCTL> CREATE RESTOREPOINT -name CUSTOM_SDB_RESTOREPOINT_JOB
```

The job is initially disabled, so you must use `GDSCTL ENABLE BACKUP` to enable the job. The job runs every hour and the schedule is not configurable.

To list all global restore points, run `LIST RESTOREPOINT`.

```
GDSCTL> LIST RESTOREPOINT
```

This command lists all of the available global restore points in the sharded database that were created during the specified time period with SCNs (using the `-start_scn` and `-end_scn` options) in the specified SCN interval (using the `-start_time` and `-end_time` options).

The following command lists the available restore points in the sharded database with the SCN between 2600000 and 2700000.

```
GDSCTL> LIST RESTOREPOINT -start_scn 2600000 -end_scn 2700000
```

The command below lists the available restore points in the sharded database that were created in the time frame from 2020/07/27 00:00:00 to 2020/07/28 00:00:00.

```
GDSCTL> LIST RESTOREPOINT -start_time "2020-07-27 00:00:00" -end_time
"2020-07-28 00:00:00"
```

Restoring Shards From Backup

The `GDSCTL RESTORE BACKUP` command lets you restore a sharded database to a specific global restore point.

This command is used to restore shard database to a specific global restore point. It can also be used to restore only the shard database control files.

The typical procedure for restoring a sharded database is:

1. List the available restore points.
2. Select a restore point to validate the backups.
3. Restore the databases to the selected restore point.

You should validate the backups for a shard against the selected restore point to verify that all the needed backups are available before you start to restore the shard to the restore point.

Note that you are not prohibited from restoring the shard catalog or a specific shard to an arbitrary point in time. However, doing so may put that target in an inconsistent state with the rest of the sharded database and you may need to take corrective action outside of the restore operation.

The database to be restored must be in NOMOUNT state. This command alters the database to MOUNT state after it has restored the control file.

The `RESTORE BACKUP` command requires the shard catalog database to be open.

For data file restore, the shards must be in MOUNT state, but if the command is to restore the control files, the shard databases must be started in NOMOUNT state. To bring the databases to the proper states will be a manual step.

To restore the shard database control files, the database must be started in nomount mode. The control files will be restored from AUTOBACKUP. To restore the database data files, the database must be mounted. The shard catalog database must be open for this command to work.

The following example restores the control files of shard `cdb2_pdb1` to restore point `BACKUP_BEFORE_DB_MAINTENANCE`.

```
GDSCTL> RESTORE BACKUP -shard cdb2_pdb1 -restorepoint
BACKUP_BEFORE_DB_MAINTENANCE -controlfile
```

The restore operation can be done for the shards in parallel. When the restore for the shards happens in parallel, you should not close GDSCTL until the command has finished running, because interrupting the restore operation can result in database corruption or get the sharded database into an inconsistent state.

Backup validation only logically restores the database while `RESTORE BACKUP` will do both the physical database restore and the database recovery. Therefore, after `RESTORE BACKUP` is done, usually the restored the databases need to be opened with the `resetlogs` option.

After the database restore is completed, you should open the database and verify that the database has been restored as intended and it is in a good state.

Restoring the Shard Catalog from Backup

To configure a shard catalog to be restored from backup:

- The sharded database must have been configured for backup and restore.
- The CDB must be started in NOMOUNT state for control file restore, in MOUNT or OPEN state for the shard catalog restore.

Note that no database connection to the shard catalog is needed for GDSCTL to run the `RESTORE BACKUP` command to restore the sharded database control files and the shard catalog PDB.

Shard catalog restore uses GDSCTL `RESTORE BACKUP` with some different parameters than are used for a shard restore, as shown in the syntax here.

```
GDSCTL> RESTORE BACKUP -shard CATALOG
    -cdb connect-string
    -catalog_name pdb-name
    -catalog_dbid dbid
    [-scn scn]
    [-controlfile]
    [-restore_only | -recover_only]
```

Because the shard catalog database must not be open when it is restored, some information obtainable from the catalog that is needed for the database restore will not be available automatically, so it will need to be provided with the `RESTORE BACKUP` command. This includes:

- A common user with SYSDG privilege in the root container and SYSBACKUP privilege for all containers in the shard catalog CDB and its password
- A connect identifier to the shard catalog CDB root (`-cdb connect-string`)
- The name of the shard catalog PDB (`-catalog_name pdb-name`)
- The shard catalog DBID (`-catalog_dbid dbid`)

- If the shard catalog needs to be restored to a specific global restore point, instead of the name of the global restore point, the associated SCN must be provided for the command (`-scn scn`)

Removing Backup Configuration from a Shard

When you remove a shard from the sharded database configuration some backup artifacts remain on the database. You can remove these artifacts with GDSCTL `CONFIG BACKUP` option `-REMOVE`.

When a shard is removed from a sharded database configuration, the shard will no longer be included in automated backup jobs, but the artifacts created on the database host when the shard was configured for backup (for example, the backup wallets and the backup jobs) are not deleted.

To remove these artifacts from the shard before it is removed from the sharded database, run `CONFIG BACKUP` with the `-REMOVE` option, and provide a list of shards that should be removed, as shown here.

```
GDSCTL> CONFIG BACKUP -remove -shard shard_list
```

The command does the following tasks:

- Deletes the shard PDB container-level backup wallet.
- Deletes the DBMS scheduler backup jobs from the shard database.
- If the CDB is not a shared target database by some other sharded database, which happens when multiple sharded databases are placed inside the same set of CDBs, deletes the CDB root container level backup wallet.

This command should only be used if the shard is removed and is not expected to be added back to the same sharded database.

Running RMAN Commands from GDSCTL

RMAN commands can be submitted from the GDSCTL CLI to run against multiple shards and the shard catalog, either in serial or in parallel.

Prerequisites

To run RMAN commands from the GDSCTL CLI, the sharded database must have been configured for backup and restore using the GDSCTL `CONFIG BACKUP` command.

Some RMAN commands also require the target database in a specific state: OPEN, MOUNT or NOMOUNT, to be able to run. Also, some RMAN commands can be run only when RMAN is connected to the CDB root as the target database. Therefore, before submitting RMAN commands to run against a shard, make sure the target shard and its CDB are in the specific state required by the commands.

Using the GDSCTL RMAN Command

GDSCTL provides the `RMAN` command to allow you to run RMAN commands in a GDSCTL session.

You can pass RMAN commands either by referencing a file or including the RMAN statements in the GDSCTL RMAN command.

- Enter a semi-colon after each RMAN statement:

  ```
  GDSCTL> RMAN -shard cdb1_pdb1 rman-stmt1;rman-stmt2;
  ```

  The RMAN statement must be contained in single or double quotation marks if the provided RMAN commands contain spaces and quotation marks:

  ```
  GDSCLT> RMAN -shard cdb1_pdb1 'rman-stmt1;rman-stmt2;'
  ```

- Specify the file path of a RMAN command file:

  ```
  GDSCTL> RMAN -shard cdb2_pdb1 -cmd_file file-path
  ```

The following options are available:

`-async`	Specifies that all tasks created to run this command will run in the background. By default, these tasks run in the foreground.
`-catpwd password`	Specifies the password for user GSMCATUSER, which is prompted if not specified. It needs to be specified only once for the entire GDSCTL session.
`-check_syntax`	Runs the command as a validation the RMAN command syntax.
`-from_cdb userid/password`	Lets you specify a common user in the shard CDB root and its password. This option is required if the provided RMAN commands must be run from the CDB root, meaning that RMAN will be connected to the CDB root as the target database to run the RMAN commands. The provided user must have SYSBACKUP privileges. By default, the provided RMAN commands are run from the shard PDB.
`-shard shard-list`	Lets you specify a comma separated list of shard identifiers. Each identifier can be a shardspace, a shardgroup, or a shard name. The default is “no shards”

Error Handling for Automated Backup Operations

Automated Backup Error Handling

After the RMAN `BACKUP` command completes, the scheduler job continues running. It checks the RMAN output for errors. If no errors are found, the file is deleted, the job continues and is expected to complete successfully. If errors are found in the RMAN output file, then:

- The RMAN output file is retained.
- A job run error status is recorded in the table `ALL_SCHEDULER_JOB_RUN_DETAILS`.
- The RMAN output file name is stored in the `ADDITIONAL_INFO` column of the `ALL_SCHEDULER_JOB_RUN_DETAILS` table, along with a path to the original RMAN output file.

Background Task Error Handling

When a command is run in the background, the fetched RMAN output is kept in memory. After the command is completed, the task checks the RMAN output for errors. If errors are detected, the last 1024 characters of the RMAN output are displayed in the GDSCTL console.

In this case, the entire the RMAN output are logged in the GDSCTL log file as well, which is specified using command `CONFIGURE -LOG_FILE`.

Propagation of Parameter Settings Across Shards

When you configure system parameter settings at the shard catalog, they are automatically propagated to all shards of the sharded database.

Oracle Globally Distributed Database provides centralized management by allowing you to set parameters on the shard catalog. Then the settings are automatically propagated to all shards of the sharded database.

Propagation of system parameters happens only if done under `ENABLE SHARD DDL` on the shard catalog, then include `SHARD=ALL` in the `ALTER` statement.

```
SQL>alter session enable shard ddl;
SQL>alter system set enable_ddl_logging=true shard=all;
```

> **Note:**
>
> Propagation of the `enable_goldengate_replication` parameter setting is not supported.

Patching and Upgrading Oracle Globally Distributed Database

There are special considerations for patching and upgrading an Oracle Globally Distributed Database deployment.

Patching and Upgrading Oracle Globally Distributed Database

Applying an Oracle patch to a sharded database environment can be done on a single shard or all shards; however, the method you use depends on the replication option used for the environment and the type of patch being applied.

Oracle Globally Distributed Database uses consolidated patching to update a shard director (GSM) ORACLE_HOME, so you must apply the Oracle Database release updates to the ORACLE_HOME to get security and Global Data Services fixes.

Patching a Sharded Database

Most patches can be applied to a single shard at a time; however, some patches should be applied across all shards. Use Oracle's best practices for applying patches to single shards just as you would a non-sharded database, keeping in mind the replication method that is being used with the sharded database. Oracle `opatchauto` can be used to apply patches to multiple shards at a time, and can be done in a rolling manner. Data Guard configurations are applied one after another, and in some cases (depending on the patch) you can use Standby First patching.

If a patch addresses an issue with multi-shard queries, replication, or the sharding infrastructure, it should be applied to all of the shards in the sharded database.

Upgrading a Sharded Database

Upgrading the Oracle Globally Distributed Database environment is not much different from upgrading other Oracle Database and global service manager environments; however, the components must be upgraded in a particular sequence such that the shard catalog is upgraded first, followed by the shard directors, and finally the shards.

> **See Also:**
>
> *Oracle OPatch User's Guide*
>
> *Oracle Database Global Data Services Concepts and Administration Guide* for information about upgrading the shard directors.
>
> *Oracle Data Guard Concepts and Administration* for information about patching and upgrading in an Oracle Data Guard configuration.

Upgrading Oracle Globally Distributed Database Components

The order in which Oracle Globally Distributed Database components are upgraded is important for limiting downtime and avoiding errors as components are brought down and back online.

Before upgrading any Oracle Globally Distributed Database components you must

- Complete any pending `MOVE CHUNK` operations that are in progress.
- Do not start any new `MOVE CHUNK` operations.
- Do not add any new shards during the upgrade process.

1. Upgrade the shards with the following points in mind.
   - For system-managed sharded databases: upgrade each set of shards in a Data Guard Broker configuration in a rolling manner.
   - For user-defined sharded databases: upgrade each set of shards in a shardspace in a rolling manner.
   - For composite sharded databases: in a given shardspace, upgrade each set of shards in a Data Guard Broker configuration in a rolling manner.

- If you are upgrading an Oracle Database 18c sharded database configuration containing pluggable database (PDB) shards, follow the PDB-specific upgrade instructions in Compatibility and Migration from Oracle Database 18c.

2. Upgrade the shard catalog database.

   For best results the catalog should be upgraded using a rolling database upgrade; however, global services will remain available during the upgrade if the catalog is unavailable, although service failover will not occur.

3. Upgrade any shard directors that are used to run GDSCTL clients, and which do not also run a global service manager server, before you update the shard directors running global service managers.

4. For shard directors running global service managers, do the following steps on one global service manager at a time.

   To ensure zero downtime, at least one shard director server should always be running. Shard director servers at an earlier version than the catalog will continue to operate fully until catalog changes are made.

   a. Stop one of the global service managers to be upgraded.

   b. Install the 21c global service manager kit

   c. Copy the `tnsnames.ora,gsm.ora,gsm_observer_1.dat,gsmwallt` directory from the previous version to the new version.

   d. Connect to the new version GDSCTL and start the global service manager which was stopped in the old version

   e. Stop the old global service manager and start it the new global service manager.

> **See Also:**
>
> *Oracle Data Guard Concepts and Administration* for information about using DBMS_ROLLING to perform a rolling upgrade.
>
> *Oracle Data Guard Concepts and Administration* for information about patching and upgrading databases in an Oracle Data Guard configuration.

Post-Upgrade Steps for Oracle Globally Distributed Database 21c

If you have a fully operational Oracle Globally Distributed Database environment in a release earlier than 21c, no wallets exist, and no deployment will be done by Oracle Globally Distributed Database after an upgrade to 21c to create them. You must perform manual steps to create the wallets.

> **Note:**
>
> The steps must be followed in EXACTLY this order.

1. Modify the GSMCATUSER password on the shard catalog database using ALTER USER, which automatically creates the wallet on the primary shard catalog.

   ```
   SQL> alter user gsmcatuser identified by gsmcatuser_password;
   ```

   This statement creates a shard wallet file if it does not exist, but does not change the password if the previous password is re-used.

   You can keep the same password for both GSMCATUSER accounts, but you must run ALTER USER to create the wallet.

2. If the GSMCATUSER password was changed, run GDSCTL modify catalog to update the system with the new GSMCATUSER password.

   This step is not necessary if the GSMCATUSER password was not changed.

3. If the GSMCATUSER password was changed, also run GDSCTL modify gsm for each shard director (GSM) to inform the shard director of the new password.

   Again, this step is not necessary if the GSMCATUSER password was not changed.

4. Modify the GSMUSER password on each primary shard using ALTER USER, which automatically creates the wallet on that shard.

   ```
   SQL> alter user gsmuser identified by gsmuser_password;
   ```

   This statement creates a shard wallet file if it does not exist, but does not change the password if the previous password is re-used.

   You can keep the same password for both GSMCATUSER accounts, but you must run ALTER USER to create the wallet.

5. If the GSMUSER password was changed, run GDSCTL modify shard on each primary shard to update the system with the new GSMUSER password for that shard.

   This step is not necessary if the GSMUSER password was not changed.

6. Run GDSCTL sync database on each shard to populate the shard wallet file with the required information.

7. After completing all of the above steps, locate the primary wallet file using the method described below, and copy this file to the wallet_root location on the standby database.

   All standby databases need an identical copy of the wallet file corresponding to its primary.

   This also applies to standby shard catalogs (if any).

   Before copying the wallet, you should check that wallet_root has been set on the standby database, if it has not already been set, then set it before copying the wallet file.

   Note that wallet_root is a global parameter that can only be set in the container database (CDB).

   The wallets themselves are created on the shard catalog, and any shard catalog replicas, during the GDSCTL create shardcatalog command, and on the shards during the GDSCTL deploy command.

All primary databases and their replicas must have a sharding-specific wallet file present to ensure proper operation. The location of the wallet files is under the directory specified by the `wallet_root` database initialization parameter. If it is not set before running `create shardcatalog` or `deploy`, `wallet_root` is set to `$ORACLE_BASE/admin/db_unique_name` on either the shard catalog or shards, respectively.

For example, assume the following after logging into the shard catalog or into a shard:

```
SQL> select guid from v$pdbs where con_id =
sys_context('userenv','con_id');

GUID
--------------------------------
C23E7C78D5B77D50E0537517C40ACE4A

SQL> select value from v$parameter where name='wallet_root';

VALUE
--------------------------------------------------------------------------------
------
your-path-to-keystore
```

Given these values, the sharding-specific wallet file name is

your-path-to-keystore/C23E7C78D5B77D50E0537517C40ACE4A/shard/cwallet.sso

Compatibility and Migration from Oracle Database 18c

When upgrading from an Oracle Database 18c installation which contains a single PDB shard for a given CDB, you must update the shard catalog metadata for any PDB.

Specifically, in 18c, the name of a PDB shard is the `DB_UNIQUE_NAME` of its CDB; however, in later Oracle Database releases, the shard names are *db_unique_name_of_CDB_pdb_name*.

To update the catalog metadata to reflect this new naming methodology, and to also support the new `GSMROOTUSER` account as described in About the GSMROOTUSER Account, perform the following steps during the upgrade process as described in Upgrading Oracle Globally Distributed Database Components.

1. After upgrading any CDB that contains a PDB shard, ensure that the GSMROOTUSER account exists, is unlocked, has been assigned a password, and has been granted SYSDG, SYSBACKUP, and gsmrootuser_role privileges.

   The following SQL statements in SQL*Plus will successfully set up GSMROOTUSER while connected to the root container (CDB$ROOT) of the CDB.

   ```
   SQL> alter session set "_oracle_script"=true;
   Session altered.

   SQL> create user gsmrootuser;
   User created.

   SQL> alter user gsmrootuser identified by new_GSMROOTUSER_password
     account unlock;
   User altered.
   ```

```
SQL> grant sysdg, sysbackup, gsmrootuser_role to gsmrootuser
container=current;
Grant succeeded.

SQL> alter session set "_oracle_script"=false;
Session altered.
```

2. After upgrading the catalog database to the desired Oracle Database version, run the following PL/SQL procedure to update the catalog metadata to reflect the new name for the PDB shards present in the configuration.

   This procedure must be run for each Oracle Database 18c PDB shard.

   The first parameter to `pdb_fixup` is the value of `db_unique_name` in the CDB that contains the PDB shard. In Oracle Database 18c, this is the same as the shard name as shown by `gdsctl config shard`.

   The second parameter is the PDB name of the shard PDB as shown by `show con_name` in SQL*Plus when connected to the shard PDB.

   The `pdb_fixup` procedure will update the catalog metadata to make it compatible with the new naming method for PDB shards.

   ```
   SQL> connect sys/password as sysdba
   Connected.
   SQL> set serveroutput on
   SQL> execute gsmadmin_internal.dbms_gsm_pooladmin.pdb_fixup('cdb1',
   'pdb1');
   ```

3. After upgrading all of the shard directors to the desired version, run the following `GDSCTL` command once for each CDB in the configuration to inform the shard directors of the password for the `GSMROOTUSER` in each CDB.

   ```
   GDSCTL> modify cdb -cdb CDB_name -pwd new_GSMROOTUSER_password
   ```

Downgrading an Oracle Globally Distributed Database

Oracle Globally Distributed Database does not support downgrading.

Sharded database catalogs and shards cannot be downgraded.

Managing Oracle Globally Distributed Database with Enterprise Manager Cloud Control

Oracle Enterprise Manager Cloud Control lets you discover, monitor, and manage an Oracle Globally Distributed Database and its components.

See the following topics for information about Oracle Globally Distributed Database discovery, monitoring, and management using Enterprise Manager Cloud Control:

- Prerequisite: Enable Sharded Database Metrics
- Prerequisite: Discover the Sharded Database Topology
- Monitoring a Sharded Database with Enterprise Manager Cloud Control

- Overview of Sharded Database Management Using Oracle Enterprise Manager Cloud Control
- Managing Shards with Oracle Enterprise Manager Cloud Control
- Managing Chunks with Oracle Enterprise Manager Cloud Control
- Shard Director Management
- Region Management
- Shardspace Management
- Shardgroup Management
- Services Management

Prerequisite: Enable Oracle Globally Distributed Database Metrics

By default Oracle Globally Distributed Database performance metrics are disabled. They can be enabled from the Enterprise Manager Cloud Console or the monitoring template.

There are two methods of gathering metrics, which require you to follow different setup steps as explained in each section below.

Using Default Enterprise Manager Database Metrics

By default metrics shown in the Enterprise Manager Cloud Console Sharded Database pages are the default database metrics, require that you create a metrics query user, and are only gathered on the shard databases discovered in Enterprise Manager.

The default database metrics do not give you data as frequently as the enhanced sharded database metrics described later.

Because multi-shard queries are used to gather metrics, you must also create a user that can access all shards in the sharded database to run the queries.

To use default metrics:

1. Create a new metrics query account on every shard and the shard catalog manually.

```
create user SHARD_SYS identified by password;
grant connect, create session, gsmadmin_role to SHARD_SYS;
GRANT ALL PRIVILEGES TO SHARD_SYS; /*Needed to get all the schemas stats*/
GRANT SELECT ANY DICTIONARY TO SHARD_SYS; /*Needed to get all the schemas
stats*/
```

2. Use the same metrics query account credentials to discover the shard catalog and all shard databases in Enterprise Manager.

   See Prerequisite: Discovering the Sharded Database Topology

3. To enable the default metrics:

```
$emctl set property
 -sysman_pwd password
 -name oracle.sysman.db.ha.sdb.dd.usesdbmetrics
 -value false
```

Using Enhanced Sharded Database Metrics

With Sharded Database enhanced metrics you can gather information about the shards from the shard catalog, so it is not required that you discover all of the shard databases in Enterprise Manager to get complete metrics for the sharded database topology.

To use enhanced metrics:

1. Discover the shard catalog in Enterprise Manager.

   See Prerequisite: Discovering the Sharded Database Topology

2. Enable the Sharded Database metrics using the Console or using the monitoring template.

```
$emctl set property
 -sysman_pwd password
 -name oracle.sysman.db.ha.sdb.dd.usesdbmetrics
 -value true
```

Prerequisite: Discover the Oracle Globally Distributed Database Topology

In Enterprise Manager Cloud Control, you can discover the shard catalog and optionally the shard databases, then add the shard directors, sharded databases, shardspaces, and shardgroups using guided discovery.

As a prerequisite to managing the sharded database in Cloud Control, you must first discover at minimum the shard director hosts and the shard catalog database. Optionally to manage all of the shards in the sharded database, you must also discover the shard databases.

Because the shard catalog database and each of the shards is a database itself, you can use standard database discovery procedures.

Managing the shards is only possible when the individual shards are discovered using database discovery. Discovering the shards is optional to discovering a sharded database, because you can have a sharded database configuration without the shards.

1. In Enterprise Manager Cloud Control, select **Setup**, choose **Add Target**, then choose **Add Target Manually**.
2. In the Add Targets Manually page, click **Add Using Guided Process** in the Add Non-Host Target Using Guided Process panel.
3. In the Add Using Guided Process dialog, locate and select **Sharded Database**, and click **Add**.
4. In the Add Sharded Database: Catalog Database page, click the browse icon next to **Catalog Database** to locate the shard catalog database.
5. In the Select Targets dialog, click the target name corresponding to the catalog database and click **Select**.

   The Catalog Database and Monitoring Credentials fields are filled in if they exist. The monitoring credential is used to query the shard catalog database to get the configuration information.

The monitoring user (usually DBSNMP) should be granted the `GDS_CATALOG_SELECT` role and has read only privileges on the shard catalog repository tables.

```
SQL> grant GDS_CATALOG_SELECT to dbsnmp;
```

Click **Next** to proceed to the next step.

In the Add Sharded Database: Components page you are shown information about the sharded database that is managed by the catalog database, including the sharded database name, its domain name, the sharding method employed on the sharded database, and a list of discovered shard directors.

6. To set monitoring credentials on a shard director, click the plus sign icon on the right side of the list entry.

   A dialog opens allowing you to set the credentials.

   Click **OK** to close the dialog, and click **Next** to proceed to the next step.

7. In the Add Sharded Database: Review page, verify that all of the shard directors, shardspaces, and shardgroups were discovered.

8. Click **Submit** to finalize the steps.

   An Enterprise Manager Deployment Procedure is submitted and you are returned to the Add Targets Manually page.

   At the top of the page you will see information about the script that was submitted to add all of the discovered components to Cloud Control.

9. Click the link to view the provisioning status of the sharded database components.

   In another browser window you can go to the Cloud Control All Targets page to observe the status of the sharded database.

When the target discovery procedure is finished, sharded database targets are added in Cloud Control. You can open the sharded database in Cloud Control to monitor and manage the components.

Oracle Globally Distributed Database Management with Oracle Enterprise Manager Cloud Control

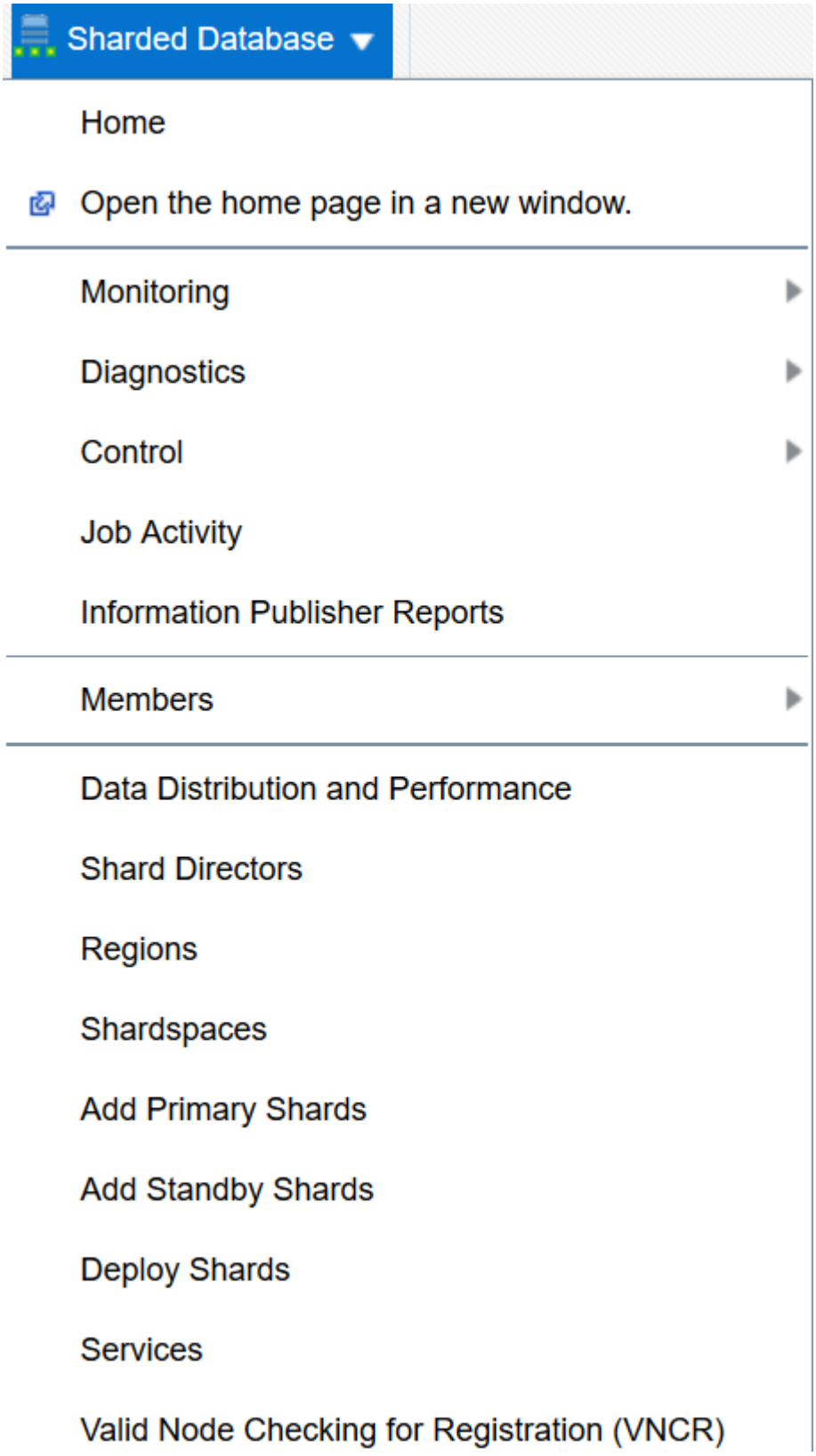
Your sharded database can be configured, deployed, monitored, and managed using Oracle Enterprise Manager Cloud Control

Any discovered sharded database objects can be found in the All Targets page in Enterprise Manager.

Shown below are the Oracle Globally Distributed Database objects Shard Director and Shard Database in the Databases target type category.

Shown below are the sharded database objects Shardgroup and Shardspace in the Groups, Systems and Services target type category.

On the Sharded Database page, you can access most of the management tools from the Sharded Database menu, such as Add Primary Shards, Add Standby Shards, and Deploy Shards, as shown below.



Management tools for other sharded database objects are located in the menus of other Sharded Database object pages, which are described in the procedures requiring that access to those pages.

See also:

- Monitoring a Sharded Database with Enterprise Manager Cloud Control
- Managing Shards with Oracle Enterprise Manager Cloud Control
- Managing Chunks with Oracle Enterprise Manager Cloud Control
- Shard Director Management
- Region Management
- Shardspace Management
- Shardgroup Management
- Services Management

Monitoring an Oracle Globally Distributed Database

Oracle Globally Distributed Database can be monitored using Enterprise Manager Cloud Control or GDSCTL.

Querying System Objects Across Shards

Use the SHARDS() clause to query Oracle-supplied tables to gather performance, diagnostic, and audit data from V$ views and DBA_* views.

The shard catalog database can be used as the entry point for centralized diagnostic operations using the SQL `SHARDS()` clause. The `SHARDS()` clause allows you to query the same Oracle supplied objects, such as V$, DBA/USER/ALL views and dictionary objects and tables, on all of the shards and return the aggregated results.

As shown in the examples below, an object in the `FROM` part of the `SELECT` statement is wrapped in the `SHARDS()` clause to specify that this is not a query to local object, but to objects on all shards in the sharded database configuration. A virtual column called `SHARD_ID` is automatically added to a `SHARDS()`-wrapped object while processing a multi-shard query to indicate the source of every row in the result. The same column can be used in predicate for pruning the query.

A query with the `SHARDS()` clause can only be run on the shard catalog database.

Examples

The following statement queries performance views

```
SQL> SELECT shard_id, callspersec FROM SHARDS(v$servicemetric)
 WHERE service_name LIKE 'oltp%' AND group_id = 10;
```

The following statement gathers statistics.

```
SQL> SELECT table_name, partition_name, blocks, num_rows
 FROM SHARDS(dba_tab_partition) p
 WHERE p.table_owner= :1;
```

The following example statement shows how to find the `SHARD_ID` value for each shard.

```
SQL> select ORA_SHARD_ID, INSTANCE_NAME from SHARDS(sys.v_$instance);

    ORA_SHARD_ID INSTANCE_NAME
    ------------ ----------------
               1 sh1
              11 sh2
              21 sh3
              31 sh4
```

The following example statement shows how to use the `SHARD_ID` to prune a query.

```
SQL> select ORA_SHARD_ID, INSTANCE_NAME
 from SHARDS(sys.v_$instance)
 where ORA_SHARD_ID=21;

    ORA_SHARD_ID INSTANCE_NAME
    ------------ ----------------
              21 sh3
```

> **See Also:**
>
> *Oracle Database SQL Language Reference* for more information about the `SHARDS()` clause.

Monitoring an Oracle Globally Distributed Database with Enterprise Manager Cloud Control

Oracle Globally Distributed Database targets are found in the All Targets page in Enterprise Manager Cloud Control.

To monitor Oracle Globally Distributed Database components you must first enable statistics gathering and then discover the Oracle Globally Distributed Database. See Prerequisite: Enable Sharded Database Metrics and Prerequisite: Discover the Oracle Globally Distributed Database Topology for more information.

Sharded Database Home Page

The target home page for a sharded database shows you a summary of the sharded database components and their statuses.

Summary

The Summary pane, in the top left of the page, shows the following information:

- **Sharded Database Name:** Sharded database name
- **Sharded Database Domain Name:** Sharded database domain name

- **Catalog Database:** Shard catalog database name. You can click the name to view more information about the shard catalog database.
- **Catalog Version:** Oracle Database version of the shard catalog
- **Sharding Type:** Sharding method used to shard the database. This could be System-managed, User-defined, or Composite.
- **Replication Type:** Replication technology used for high availability.
- **Shard Directors:** Number and status of the shard directors
- **Master Shard Director:** Primary shard director name. You can click the shard director name to view more information about the primary shard director, including the shard director (global service manager) version, current status, ports used, and incidents.

Members

The Members pane, in the upper right of the page, shows some relevant information about each of the sharded database components.

The pane is divided into tabs for each component: Shardspaces, Shardgroups, Shard Directors, Shards, Catalog Databases, and Global Services. Click on a tab to view the information about each type of component

- **Shardspaces:**

  Shardspaces are only displayed for databases sharded with the user-defined or composite sharding method.

  The Shardspaces tab displays the shardspace names, status, number of chunks, and protection mode. The shardspace names can be clicked to reveal more details about the selected shardspace.

  You can click the shardspace name to view more details, including information about the shardgroups within the shardspace (for composite sharding) and incidents.

- **Shardgroups:**

  Shardgroups are only displayed for databases sharded with the system-managed or composite sharding method.

  The Shardgroups tab displays the shardgroup names, status, the shardspace to which it belongs, the number of chunks, Data Guard role, and the region to which it belongs.

  You can click the shardgroup name to reveal more details about the selected component, including information about the shards within the shardgroup, and incidents.

  Note that for a database sharded using the system-managed sharding method, `shardspaceora` is the shardspace created by default to contain all of the shardgroups. It is managed by the sharded database and will not appear in the Shardspaces tab.

- **Shard Directors:**

  The Shard Directors tab displays the shard director names, status, region, host, and Oracle home.

You can click the shard director names to reveal more details about the selected shard director, including the shard director (global service manager) version, current status, ports used, and incidents.

You can also click the shard director host to view more details about the host system.

- **Shards:**

  The Shards tab displays the shard names, Data Guard roles, target type, target status, the shardspaces and shardgroups to which they belong, the regions to which they belong, and the state (deployed or .

  In the Names column, you can expand the primary shards to display the information about their corresponding standby shards.

  You can hover the mouse over the Deployed column icon and the deployment status details are displayed. You can click on the shard, shardspace, and shardgroup names to reveal more details about the selected component.

- **Catalog Databases**

  The Catalog Databases tab lists the shard catalog databases and displays the shard catalog database name, type, status, and role for each catalog database.

  You can click on the catalog database name to view more information about the database.

- **Global Services:**

  The Global Services tab displays the name, status, and Data Guard role of the sharded database global services. Above the list is shown the total number of services and an icon showing how many services are in a particular status. You can hover your mouse pointer over the icon to read a description of the status icon.

Incidents

The Incidents pane displays messages and warnings about the various components in the sharded database environment. More information about how to use this pane is in the Cloud Control online help.

Sharded Database Menu

The Sharded Database menu, located in the top left corner, provides you with access to tools to manage the sharded database components.

Target Navigation

The Target Navigation pane gives you easy access to more details about any of the components in the sharded database.

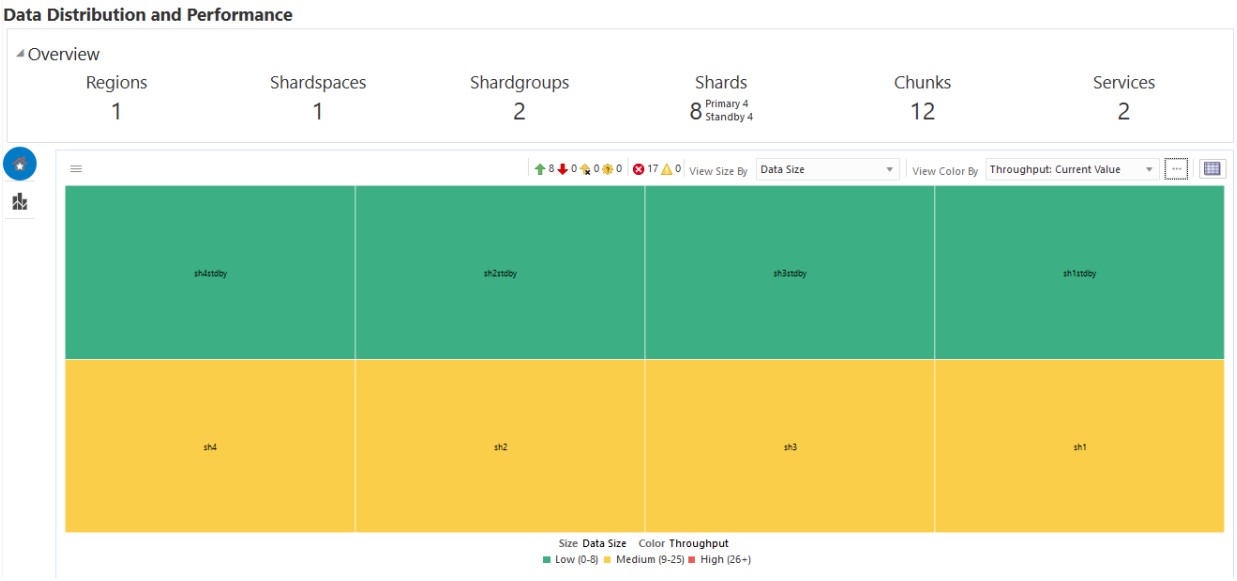
Clicking the navigation tree icon on the upper left corner of the Sharded Database home page opens the Target Navigation pane. This pane shows all of the discovered components in the sharded database in tree form.

Expanding a shardspace reveals the shardgroups in them. Expanding a shardgroup reveals the shards in that shardgroup.

Any of the component names can be clicked to view more details about them.

Data Distribution and Performance Page

In Enterprise Manager Cloud Control, the Sharded Database page, Data Distribution and Performance, gives you an overall view of the data in your sharded database and how the shards are performing.



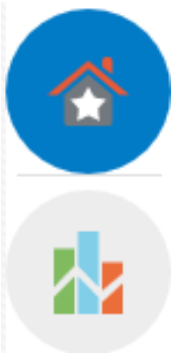
Overview

The Overview section at the top of the page displays number of regions, shardspaces, shardgroups, shards (broken down into primary and standby), chunks, and services in the sharded database configuration that are represented by the data in the chart. If you apply a filter to the chart these numbers change.

Data Distribution and Performance Chart Views

The two icons at the top left corner of the chart toggle the chart between two views:

Figure 7-1 Home and Top Shards Icons

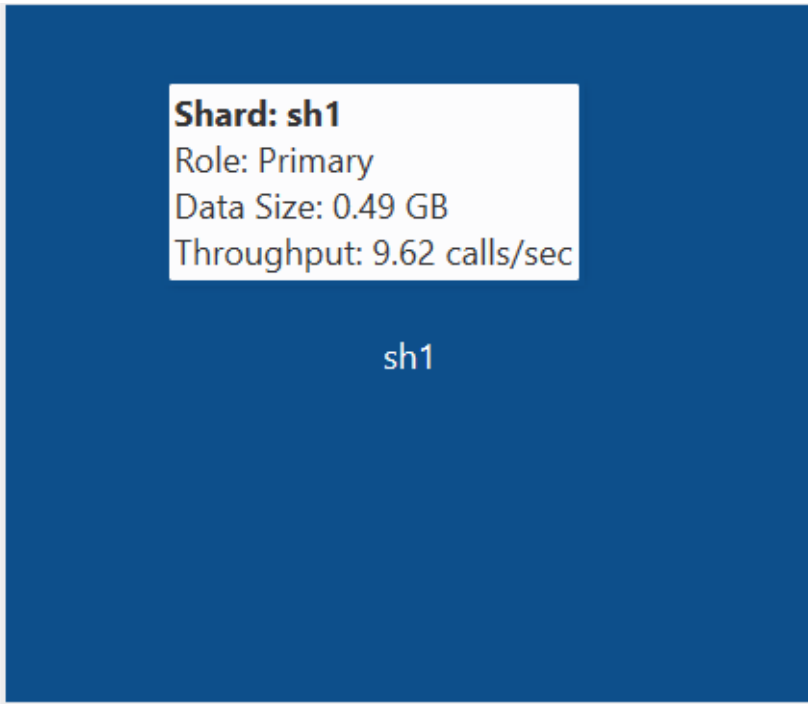


- **Home:** is the default view. Home displays data for all shards in the sharded database by default. You can filter the chart and change the metrics on display as described below.
- **Top Shards:** shows you charts for the top 5, 10, or 20 shards for certain metrics.

Shard Blocks

The color-coded chart displays data by shard. Each shard is indicated by a block.

Figure 7-2 Shard Block with Mouse Over Text



Each block is labeled with the shard name. Moving the mouse over a block displays the Shard name, Data Guard Role, Number of Chunks in the shard, and the Service Time (msec/call).

> **Note:**
>
> If you are using default database metrics then you will not see data from any undiscovered shards in the chart.
> If you are using enhanced metrics, the data for all shards is displayed because the shards are discovered by the shard catalog.

Home View Summary Icons

The row of icons above the chart display the following information:

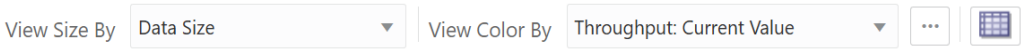
Figure 7-3 Home View Summary Icons

- **Up:** (Green arrow pointing up) Number of shard databases that are up
- **Down:** (Red arrow pointing down) Number of shards that are down
- **Unmonitored:** (Yellow arrow with "X") Number of shards that are unmonitored. This is the number of shards not discovered by Enterprise Manager.
- **Other:** (Yellow gear with question mark "?")Sharded database targets discovered in Enterprise Manager, but that have some issue with target monitoring, such as an unreachable agent, or an availability evaluation error.
- **Critical:** (Red circle with "X") Number of critical incidents
- **Warning:** (Yellow triangle with exclamation point "!") Number of warning incidents

Chart View Controls

Compare metrics on each of the shards by size and color of the blocks in the chart.

Figure 7-4 Chart View Controls



- **View Size By:** changes the size distribution of the blocks by the metric selected
- **View Color By:** changes the comparative color of the blocks by the metric selected

  By default, the colors are light, medium, and dark blue, which indicates that the thresholds for the lightest and darkest color categories are set to arbitrary Enterprise Manager defaults.

  Click **Configure Threshold** (button with three dots) to set custom thresholds for low and high categories in each metric. Charts configured with custom thresholds are shown in a different color spectrum with green=low, yellow=medium, and red=high.
- **Tree Map Table View:** (button with table at the top right corner of the chart) displays a table view of the data shown in the chart

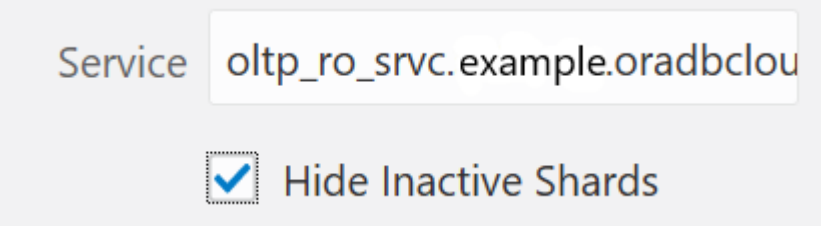
Filters

Click the hamburger icon at the top left corner of the chart to apply filters to the data.

Figure 7-5 Filters Icon

- **Shard Search:** Filter by shard name. You can use an asterisk (*) to select a group of shards with matching name patterns.
- **Key Search:** Lets you enter a shard key value to view the shards that contain data with that key. In the resulting chart you can right-click a block and select **Shard-Level Data Distribution** to drill down into a particular shard.
- **SQL ID Search:** Display which shards are processing a query by the SQL ID for the query, which you can find in the `V$SQL_SHARD` view in the catalog database.
- **Sort By:** Sort the blocks in the chart by size in the default tiled view, in a sequence of bars, or show only the top or bottom 5 blocks.
- **Filter By:** Lets you display only shards in the specified Role, Shardgroup, or Service.

  Hide Inactive Shards: When using the Service filter, you will see all of the shards; however, shards on which the service is not running are shown in grey (inactive), and you can use the checkbox to hide the inactive shards.

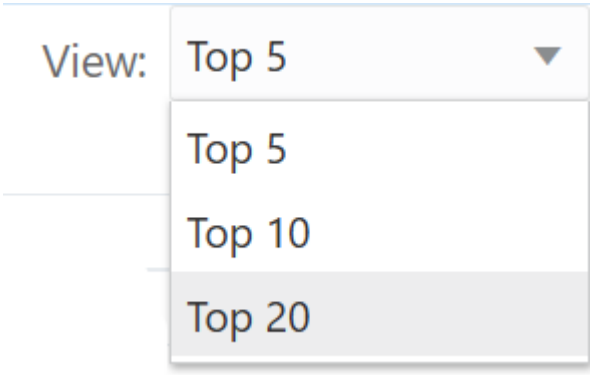


- **Group By:** Toggles that display aggregates for the group, which is indicated by a box line around the group of shards.
  - **Shardgroup** displays a shardgroup box at the top of the grouping, which displays aggregate info about the shardgroup on hover, and you can drill down for shardgroup-based data.
  - **Region** displays a region box at the top of the group, which displays aggregate info about the region on hover.
  - **Data Guard Aggregate Group** groups each shard and its standbys as a single entity, so that you can see the data set being handled by a particular shard and its standbys as a whole.

Top Shards View

Click the Top Shards button on the left side of the chart to view graphs with metrics on the shards with the highest Data Size, Number of Chunks, Throughput, and Service Time.

Use the View list at the top right corner of the view to display the top 5, 10, or 20 shards in each graph.



Monitoring Oracle Globally Distributed Database with GDSCTL

There are numerous `GDSCTL CONFIG` commands that you can use to obtain the health status of individual shards, shardgroups, shardspaces, and shard directors.

Monitoring a shard is just like monitoring a normal database, and standard Oracle best practices should be used to monitor the individual health of a single shard. However, it is also important to monitor the overall health of the entire sharded environment. The GDSCTL commands can also be scripted and through the use of a scheduler and can be done at regular intervals to help ensure that everything is running smoothly.

> **See Also:**
>
> *Oracle Database Global Data Services Concepts and Administration Guide* for information about using the `GDSCTL CONFIG` commands

Shard Management

You can manage shards in your Oracle Globally Distributed Database deployment with Oracle Enterprise Manager Cloud Control and GDSCTL.

About Adding Shards

New shards can be added to an existing sharded database environment to scale out and to improve fault tolerance.

For fault tolerance, it is beneficial to have many smaller shards than a few very large ones. As an application matures and the amount of data increases, you can add an entire shard or multiple shards to the sharded database to increase capacity.

When you add a shard to a sharded database, if the environment is sharded by consistent hash (system-managed), then chunks from existing shards are automatically moved to the new shard to rebalance the sharded environment.

When using user-defined sharding, populating a new shard with data may require manually moving chunks from existing shards to the new shard using the GDSCTL `split chunk` and `move chunk` commands.

Oracle Enterprise Manager Cloud Control can be used to help identify chunks that would be good candidates to move, or split and move to the new shard.

When you add a shard to the environment, verify that the standby server is ready, and after the new shard is in place take backups of any shards that have been involved in a `move chunk` operation.

All of the DDLs that have been processed in the sharded database are applied in the same order to the shard before it becomes accessible to clients.

The `SHOW DDL -COMPRESSED` option displays a set of compressed DDLs generated. It also shows you if any DDL has failed on any of the shards.

Work Flow for Adding Shards

Task 1: Create the shard databases

Before you add new shards to the sharded database configuration, you must install the Oracle Database software on the shard host systems and configure new databases for each primary and standby shard. Following these steps, referring to the linked topics for details:

1. Install the Oracle Database Software
2. Create the Shard Databases

Task 2: Validate the shard databases

Validate the shard database to verify that all of the shard database requirements have been met.

- Validate the Shard Database using SQL*Plus
- Validating a Shard using Enterprise Manager

Task 3: Add the shard databases to the configuration

Add each primary and standby shard to the sharded database configuration.

- Add the Shard CDBs and Add the Shard PDBs using GDSCTL
- Adding Primary Shards and Adding Standby Shards using Enterprise Manager

Task 4: Deploy the updated sharded database configuration

The final task is to deploy the updated sharded database configuration with the new shard and its standbys that you added.

- Deploy the Sharding Configuration using GDSCTL
- Deploying Shards using Enterprise Manager
  Note that using Enterprise Manager sharded database management tools, you can optionally add and deploy a shard in a single step. However, if you choose not to deploy when adding the shard to the configuration, then use this procedure to deploy it.

Removing a Shard From the Pool

It may become necessary to remove a shard from the sharded database environment, either temporarily or permanently, without losing any data that resides on that shard.

For example, removing a shard might become necessary if a sharded environment is scaled down after a busy holiday, or to replace a server or infrastructure within the data center. Prior to decommissioning the shard, you must move all of the chunks from the shard to other shards that will remain online. As you move them, try to maintain a balance of data and activity across all of the shards.

If the shard is only temporarily removed, keep track of the chunks moved to each shard so that they can be easily identified and moved back once the maintenance is complete.

If a shard is part of a sharded database configured for centralized automatic backup, you can remove backup artifacts from the database before removing the shard from the sharded database configuration. See Removing Backup Configuration from a Shard.

You can remove shards using GDSCTL or Oracle Enterprise Manager Cloud Control:

- *Oracle Database Global Data Services Concepts and Administration Guide* for information about using the `GDSCTL REMOVE SHARD` command
- Removing a Shard with Oracle Enterprise Manager Cloud Control

Replacing a Shard

If a shard fails, or if you just want to move a shard to a new host for other reasons, you can replace it using the `ADD SHARD -REPLACE` command in GDSCTL.

When a shard database fails and the database can be recovered on the same host (using RMAN backup/restore or other methods), there is no need to replace the shard using the -replace parameter. If the shard cannot be recovered locally, or for some other reason you want to relocate the shard to another host or CDB, it is possible to create its replica on the new host. The sharding configuration can be updated with the new information by specifying the -replace option in GDSCTL command ADD SHARD.

The following are some cases where replacing a shard using ADD SHARD -REPLACE is useful.

- The server (machine) where the shard database was running suffered irreparable damage and has to be replaced
- You must replace a working server with another (more powerful, for example) server
- A shard in a PDB was relocated from one CDB to another

In all of these cases the number of shards and data distribution across shards does not change after ADD SHARD is run; a shard is replaced with another shard that holds the same data. This is different from ADD SHARD used without the -replace option when the number of shards increases and data gets redistributed.

Upon running ADD SHARD -REPLACE, the old shard parameters, such as connect_string, db_unique_name, and so on, are replaced with their new values. A new database can have different db_unique_name than the failed one. When replacing a standby in a Data Guard configuration, the DBID of the new database must match the old one, as Data Guard requires all of the members of the configuration to have same DBID.

Before Using Replace

Before you use ADD SHARD -REPLACE, verify the following:

- You have restored the database correctly (for example, using RMAN restore or other method). The new database shard must have the same sharding metadata as the failed one. Perform basic validation to ensure that you do not accidently provide a connect string to the wrong shard.
- The shard that failed must have been in a deployed state before failure happened.
- The shard that failed must be down when running the ADD SHARD -REPLACE command.
- Fast-start failover observer must be running, if fast-start failover is enabled (which it is by default).

Replacing a Shard in a Data Guard Environment

The `ADD SHARD -REPLACE` command can only be used to replace a standby shard if the primary is still available. In order to replace a primary shard that failed, wait for one of the remaining standbys to switch over to the primary role before trying to replace the failed shard.

When a switchover is not possible (primary and all the standbys are down), you must run ADD SHARD -REPLACE for each member starting with the primary. This creates a new broker configuration from scratch.

In MAXPROTECTION mode with no standbys available, the primary database shuts down to maintain the protection mode. In this case, the primary database cannot be opened if the standby is not available. To handle the replace operation in this scenario, you must first downgrade Data Guard protection mode using DGMGRL (to MAXAVAILABILITY or MAXPERFORMANCE) by starting up the database in mounted mode. After the protection mode is set, open the primary database and perform the replace operation using GDSCTL. After the replace operation finishes you can revert the protection mode back to the previous level using DGMGRL.

When replacing a standby in a Data Guard configuration, the DBID of the new database must match the old one, as Data Guard requires all of the members of the configuration to have same DBID.

Example 7-1 Example 1: Replacing the primary shard with no standbys in the configuration

The initial configuration has two primary shards deployed and no standbys, as shown in the following example. The Availability for shdc is shown as a dash because it has gone down in a disaster scenario.

```
$ gdsctl config shard

Name    Shard Group    Status    State       Region    Availability
----    -----------    ------    -----       ------    ------------
shdb    dbs1           Ok        Deployed    east      ONLINE
shdc    dbs1           Ok        Deployed    east      -
```

To recover, you create a replica of the primary from the backup, using RMAN for example. For this example, a new shard is created with db_unique_name shdd and connect string inst4. Now, the old shard, shdc, can be replaced with the new shard, shdd, as follows:

```
$ gdsctl add shard -replace shdc -connect inst4 -pwd password

DB Unique Name: SHDD
```

You can verify the configuration as follows:

```
$ gdsctl config shard

Name     Shard Group      Status    State       Region    Availability
----     -----------      ------    -----       ------    ------------
shdb     dbs1             Ok        Deployed    east      ONLINE
shdd     dbs1             Ok        Deployed    east      ONLINE
```

Example 7-2 Example 2: Replacing a standby shard

Note that you cannot replace a primary shard when the configuration contains a standby shard. In such cases, if the primary fails, the replace operation must be performed after one of the standbys becomes the new primary by automatic switchover.

The initial configuration has two shardgroups: one primary and one standby, each containing two shards, when the standby, shdd goes down.

```
$ gdsctl config shard

Name    Shard Group      Status    State       Region    Availability
----    -----------      ------    -----       ------    ------------
shdb    dbs1             Ok        Deployed    east      ONLINE
shdc    dbs1             Ok        Deployed    east      ONLINE
shdd    dbs2             Ok        Deployed    east      -
shde    dbs2             Ok        Deployed    east      READ ONLY
```

Create a new standby. Because the primary is running, this should be done using the `RMAN DUPLICATE` command with the `FOR STANDBY` option. Once the new standby, shdf, is ready, replace the old shard, shdd, as follows:

```
$ gdsctl add shard -replace shdd -connect inst6 -pwd password

DB Unique Name: shdf
```

You can verify the configuration as follows:

```
$ gdsctl config shard

Name    Shard Group      Status    State       Region    Availability
----    -----------      ------    -----       ------    ------------
shdb    dbs1             Ok        Deployed    east      ONLINE
shdc    dbs1             Ok        Deployed    east      ONLINE
shde    dbs2             Ok        Deployed    east      READ ONLY
shdf    dbs2             Ok        Deployed    east      READ ONLY
```

Common Errors

ORA-03770: incorrect shard is given for replace

This error is thrown when the shard given for the replace operation is not the replica of the original shard. Specifically, the sharding metadata does not match the metadata stored in the shard catalog for this shard. Make sure that the database was copied correctly, preferably using RMAN. Note that this is not an exhaustive check. It is assumed that you created the replica correctly.

ORA-03768: The database to be replaced is still up: shardc

The database to be replaced must not be running when running the `add shard -replace` command. Verify this by looking at the output of GDSCTL command `config shard`. If the shard failed but still shows ONLINE in the output, wait for some time (about 2 minutes) and retry.

> **See Also:**
>
> *Oracle Database Global Data Services Concepts and Administration Guide* for information about the ADD SHARD command.

Converting a Physical Standby to a Snapshot Standby

When using Oracle Data Guard as the replication method for a sharded database, Oracle Globally Distributed Database supports only the addition of a primary or physical standby shard; other types of Data Guard standby databases are not supported when adding a new standby to the sharded database.

However, a shard that is already part of the sharded database can be converted from a physical standby to a snapshot standby.

1. Stop all global services on the shard using the GDSCTL command `STOP SERVICE`.

2. Disable all global services on the shard using the GDSCTL command `DISABLE SERVICE`.
3. Convert the shard to a snapshot standby using the procedure described in the Oracle Data Guard documentation Converting a Physical Standby Database into a Snapshot Standby Database.

   At this point, the shard remains part of the sharded database, but will not accept connections which use the sharding key.

If the database is converted back to a physical standby, the global services can be enabled and started again, and the shard becomes an active member of the sharded database.

Migrating a Non-PDB Shard to a PDB

Do the following steps if you want to migrate shards from a traditional single-instance database to Oracle multitenant architecture. Also, you must migrate to a multitenant architecture before upgrading to Oracle Database 21c or later releases.

1. Back up each existing non-PDB shard, and then create a new CDB, and a PDB inside it.
2. Restore each shard to the PDB inside the CDB.
3. Run the `GDSCTL ADD CDB` command to add the new CDB.

   ```
   GDSCTL> add cdb -connect cdb_connect_string -pwd
   gsmrootuser_password
   ```

4. Run the `GDSCTL ADD SHARD -REPLACE` command, specifying the connect string of the PDB, *shard_connect_string*, which tells the sharding infrastructure to replace the old location of the shard with new PDB location.

   For **system-managed** or **composite** sharding, run `ADD SHARD` with the parameters shown here.

   ```
   GDSCTL> add shard -replace db_unique_name_of_non_PDB -connect
   shard_connect_string -pwd gsmuser_password
   -shardgroup shardgroup_name -cdb cdb_name
   ```

   For **user-defined** sharding, the command usage is slightly different.

   ```
   GDSCTL> add shard -replace db_unique_name_of_non_PDB -connect
   shard_connect_string -pwd gsmuser_password
   -shardspace shardspace_name -deploy_as db_mode -cdb cdb_name
   ```

Managing Shards with Oracle Enterprise Manager Cloud Control

You can manage database shards using Oracle Enterprise Manager Cloud Control

To manage shards using Cloud Control, they must first be discovered. Because each database shard is a database itself, you can use standard Cloud Control database discovery procedures.

Shards are managed from within their respective shardgroups. To manage a shard you must first navigate to the shardgroup which contains the shard you wish to manage. This can be done from the All Targets page or the Sharded Database page.

In the Shardgroup page, open the **Shardgroup** menu, located in the top left corner of the shardgroup target page, and choose **Manage Shards**.

Figure 7-6 Shardgroup Menu

If prompted, enter the shard catalog credentials, select the shard director to manage under **Shard Director Credentials**, select the shard director host credentials, and log in.

The following topics describe shard management using Oracle Enterprise Manager Cloud Control:

Validating a Shard

Validate a shard prior to adding it to your Oracle Globally Distributed Database deployment.

You can use Oracle Enterprise Manager Cloud Control to validate shards before adding them to your Oracle Globally Distributed Database deployment. You can also validate a shard after deployment to confirm that the settings are still valid later in the shard life cycle. For example, after a software upgrade you can validate existing shards to confirm correctness of their parameters and configuration.

To validate shards with Cloud Control, they should be existing targets that are being monitored by Cloud Control.

1. In the Shardgroup page, open the **Shardgroup** menu, located in the top left corner of the shardgroup target page, and choose **Manage Shards**.
2. If prompted, enter the shard catalog credentials, select the shard director to manage under **Shard Director Credentials**, select the shard director host credentials, and log in.
3. In the Manage Shards page, select a shard from the list and click **Validate**.
4. Click **OK** to confirm you want to validate the shard.
5. Click the link in the **Information** box at the top of the page to view the provisioning status of the shard.

When the shard validation script runs successfully check for errors reported in the output.

Adding Primary Shards

You can use Oracle Enterprise Manager Cloud Control to add a primary shards to your Oracle Globally Distributed Database deployment.

To add a primary shard using Cloud Control it must be an existing target being monitored by Cloud Control.

> **Note:**
>
> It is highly recommended that you validate a shard before adding it to your configuration. You can either use Cloud Control to validate the shard (see Validating a Shard), or run the `DBMS_GSM_FIX.validateShard` procedure against the shard using SQL*Plus (see Validate the Shard Database).

1. Open the **Sharded Database** menu, located in the top left corner of the Sharded Database target page, and choose **Add Primary Shards**.
2. If prompted, enter the shard catalog credentials, select the shard director to manage under **Shard Director Credentials**, select the shard director host credentials, and log in.

3. To add and deploy the shards in the same operation, select **Deploy All Shards in the sharded database** to deploy all shards added to the sharded database configuration.

   The deployment operation validates the configuration of the shards and performs final configuration steps. Shards can be used only after they are deployed.

4. Click **Add**.
5. In the **Database** field of the Shard Details dialog, select the shard database target and click **Select**.
6. In composite sharding you can select also the shardspace to which to add the shard.
7. Configure any Advanced Settings:

   Connect Descriptor (you can use the default Enterprise Manager connect descriptor or specify another connect descriptor in the **Connect Descriptor** box)

   CPU Utilization Threshold (%)

   Disk Threshold (ms)

8. Click **OK**.
9. Enter the `GSMUSER` credentials if necessary, then click **Next**.
10. Indicate when the `ADD SHARD` operation should occur, then click **Next**.
    - **Immediately**: the shard is provisioned upon confirmation
    - **Later**: schedule the timing of the shard addition using the calendar tool in the adjacent field
11. Review the configuration of the shard to be added and click **Submit**.
12. Click the link in the **Information** box at the top of the page to view the provisioning status of the shard.

If you did not select **Deploy All Shards in the sharded database** in the procedure above, deploy the shard in your Oracle Globally Distributed Database deployment as described in Deploying Shards.

Adding Standby Shards

Use Oracle Enterprise Manager Cloud Control to add a standby shards to your deployment.

To add a standby shard using Cloud Control the database must be an existing target being monitored by Cloud Control.

> **Note:**
>
> It is highly recommended that you validate a shard before adding it to your deployment. You can either use Cloud Control to validate the shard (see Validating a Shard), or run the `DBMS_GSM_FIX.validateShard` procedure against the shard using SQL*Plus (see Validate the Shard Database).

1. Open the **Sharded Database** menu, located in the top left corner of the Sharded Database target page, and choose **Add Standby Shards**.
2. If prompted, enter the shard catalog credentials, select the shard director to manage under **Shard Director Credentials**, select the shard director host credentials, and log in.

3. To add and deploy the shards in the same operation, select **Deploy All Shards in the sharded database** to deploy all shards added to the sharded database configuration.

   The deployment operation validates the configuration of the shards and performs final configuration steps. Shards can be used only after they are deployed.
4. In the **Primary Shards** list, select a primary shard for which the new shard database will act as a standby.
5. At the top of the **Standby Shards** list, click **Add**.
6. In the **Database** field of the Shard Details dialog, select the standby shard.
7. Select the shardgroup to which to add the shard.

   Only shardgroups that do not already contain a standby for the selected primary are shown.
8. Click **OK**.
9. Enter the `GSMUSER` credentials if necessary, then click **Next**.
10. Indicate when the `ADD SHARD` operation should occur, then click **Next**.
    - **Immediately**: the shard is provisioned upon confirmation
    - **Later**: schedule the timing of the shard addition using the calendar tool in the adjacent field
11. Review the configuration of the shard to be added and click **Submit**.
12. Click the link in the **Information** box at the top of the page to view the provisioning status of the shard.

If you did not select **Deploy All Shards in the sharded database** in the procedure above, deploy the shard in your deployment as described in Deploying Shards.

Deploying Shards

Use Oracle Enterprise Manager Cloud Control to deploy shards that have been added to your deployment.

1. Open the **Sharded Database** menu, located in the top left corner of the Sharded Database target page, and choose **Deploy Shards**.
2. If prompted, enter the shard catalog credentials, select the shard director to manage under **Shard Director Credentials**, select the shard director host credentials, and log in.
3. Select the **Perform Rebalance** check box to redistribute data between shards automatically after the shard is deployed.

   If you want to move chunks to the shard manually, uncheck this box.
4. Click **Submit**.
5. Click the link in the **Information** box at the top of the page to view the provisioning status of the shard.

Editing a Shard

You can update a shard's CPU Utilization Threshold (%), Disk Threshold (ms), ONS Port, SCAN Address, Connect Descriptor, and `GSMUSER` Password in the Manage Shards page in Oracle Enterprise Management Cloud Control.

1. In the Shardgroup page, open the **Shardgroup** menu, located in the top left corner of the shardgroup target page, and choose **Manage Shards**.
2. If prompted, enter the shard catalog credentials, select the shard director to manage under **Shard Director Credentials**, select the shard director host credentials, and log in.
3. In the Manage Shards page, select a shard from the list and click **Edit**.
4. Click **OK** to save any changes made in the Edit Shard dialog.

Removing a Shard

You can remove a shard from your sharded database configuration in the Manage Shards page in Oracle Enterprise Management Cloud Control.

1. In the Shardgroup page, open the **Shardgroup** menu, located in the top left corner of the shardgroup target page, and choose **Manage Shards**.
2. If prompted, enter the shard catalog credentials, select the shard director to manage under **Shard Director Credentials**, select the shard director host credentials, and log in.
3. In the Manage Shards page, select a shard from the list and click **Remove**.

   Use the **Force** option to remove the specified shard even if it is inaccessible and/or contains chunks. Using this option might result in a lower number of replicas or total unavailability for a certain range of data.
4. Click **OK** to confirm that you want to remove the shard.

Chunk Management

You can manage chunks in your deployment with Oracle Enterprise Manager Cloud Control and GDSCTL.

Resharding and Hot Spot Elimination

The process of redistributing data between shards, triggered by a change in the number of shards, is called resharding. Automatic resharding is a feature of the system-managed sharding method that provides elastic scalability of a sharded database.

Sometimes data in a sharded database needs to be migrated from one shard to another. Data migration across shards is required in the following cases:

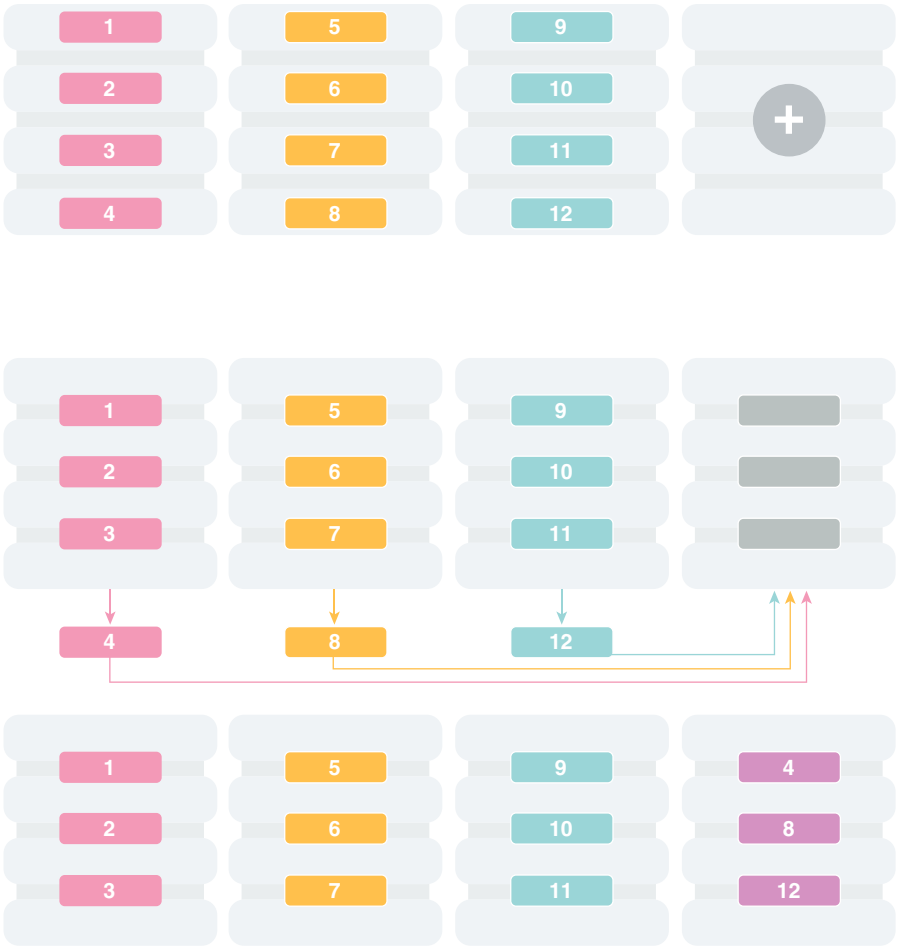
- When one or multiple shards are added to or removed from a sharded database
- When there is skew in the data or workload distribution across shards

The unit of data migration between shards is the chunk. Migrating data in chunks guaranties that related data from different sharded tables are moved together.

When a shard is added to or removed from a sharded database, multiple chunks are migrated to maintain a balanced distribution of chunks and workload across shards.

Depending on the sharding method, resharding happens automatically (system-managed) or is directed by the user (composite). The following figure shows the stages of automatic resharding when a shard is added to a sharded database with three shards.

Figure 7-7 Resharding a Sharded Database



A particular chunk can also be moved from one shard to another, when data or workload skew occurs, without any change in the number of shards. In this case, chunk migration can be initiated by the database administrator to eliminate the hot spot.

RMAN Incremental Backup, Transportable Tablespace, and Oracle Notification Service technologies are used to minimize impact of chunk migration on application availability. A chunk is kept online during chunk migration. There is a short period of time (a few seconds) when data stored in the chunk is available for read-only access only.

FAN-enabled clients receive a notification when a chunk is about to become read-only in the source shard, and again when the chunk is fully available in the destination shard on completion of chunk migration. When clients receive the `chunk read-only` event, they can either repeat connection attempts until the chunk migration is completed, or access the read-only chunk in the source chunk. In the latter case, an attempt to write to the chunk will result in a run-time error.

> **Note:**
>
> Running multi-shard queries while a sharded database is resharding can result in errors, so it is recommended that you do not deploy new shards during multi-shard workloads.

Moving Chunks

Sometimes it becomes necessary to move a chunk from one shard to another. To maintain scalability of the sharded environment, it is important to attempt to maintain an equal distribution of the load and activity across all shards.

As the environment matures in a composite sharded database, some shards may become more active and have more data than other shards. In order to keep a balance within the environment you must move chunks from more active servers to less active servers. There are other reasons for moving chunks:

- When a shard becomes more active than other shards, you can move a chunk to a less active shard to help redistribute the load evenly across the environment.
- When using range, list, or composite sharding, and you are adding a shard to a shardgroup.
- When using range, list, or composite sharding, and you a removing a shard from a shardgroup.
- After splitting a chunk it is often advisable to move one of the resulting chunks to a new shard.

When moving shards to maintain scalability, the ideal targets of the chunks are shards that are less active, or have a smaller portion of data. Oracle Enterprise Manager and AWR reports can help you identify the distribution of activity across the shards, and help identify shards that are good candidates for chunk movement.

> **Note:**
>
> Any time a chunk is moved from one shard to another, you should make a full backup of the databases involved in the operation (both the source of the chunk move, and the target of the chunk move.)

You can manage chunks using GDSCTL or Oracle Enterprise Manager Cloud Control:

- *Oracle Database Global Data Services Concepts and Administration Guide* for information about using the `GDSCTL MOVE CHUNK` command
- Moving Chunks with Oracle Enterprise Manager Cloud Control

Updating an In-Process Chunk Move Operation

While a `MOVE CHUNK` operation is in process, you can use the `GDSCTL ALTER MOVE` command to suspend, resume, or cancel any or all chunks scheduled to be moved (where the move is not yet started) in the operation.

There are three variations on this command: `-SUSPEND` is used to postpone chunk migration operation, `-RESUME` is used to restart the move process, and `-CANCEL` cancels chunk migration.

In addition, the `-CHUNK` and `-SHARD` options are used to filter the list of scheduled chunk moves. You can use the `CONFIG CHUNKS -SHOW_RESHARD` command to get a list of scheduled chunk moves.

Suspending Chunk Moves

`ALTER MOVE -SUSPEND` postpones chunk migration for a specified scope until you wish resume or cancel the operation. The shards on which to suspend operation must be specified, and you can list source and target shards. You can also specify a list of specific chunks to suspend.

If any chunk in the defined scope is already being moved (any state other than "scheduled"), that chunk will not be suspended.

For example, the following command suspends all scheduled chunk moves to or from shard1.

```
GDSCTL> alter move -suspend -shard shard1
```

Restarting Chunk Moves

`ALTER MOVE -RESUME` resets any "move failed" flags on specified shards, and restarts any stalled or suspended chunk moves.

You can optionally provide a list of source and target shards that will have their "move failed" flags reset before the moves restart. If no shards are specified, the suspended moves are restarted once any moves in process are complete.

For example, the following command restarts chunk moves on any suspended or "failed" chunk moves scheduled to or from shard1.

```
GDSCTL> alter move -resume -shard shard1
```

Canceling Chunk Moves

`ALTER MOVE -CANCEL` removes specified chunks from the move chunk schedule.

The `-CHUNK` option specifies that all listed chunks will be removed from the schedule, and `-SHARD` specifies that all chunk moves to/from this database will be removed from the schedule. If no chunks or shards are specified, then all chunk moves not already in process are canceled.

If any chunk in the defined scope is currently being moved (any state other than "scheduled"), that chunk move will not be canceled.

Chunks that are canceled cannot be resumed/restarted. You must issue a new `MOVE CHUNK` command to move these chunks.

For example, the following command removes chunks 1, 2, and 3 from the chunk move schedule, if they are not already being moved.

```
GDSCTL> alter move -cancel -chunk 1,2,3
```

Splitting Chunks

Splitting a chunk in a sharded database is required when chunks become too big, or only part of a chunk must be migrated to another shard.

Oracle Globally Distributed Database supports the online split of a chunk. Theoretically it is possible to have a single chunk for each shard and split it every time data migration is required. However, even though a chunk split does not affect data availability, the split is a time-consuming and CPU-intensive operation because it scans all of the rows of the partition being split, and then inserts them one by one into the new partitions. For composite sharding, it is time consuming and may require downtime to redefine new values for the shard key or super shard key.

Therefore, it is recommended that you pre-create multiple chunks on each shard and split them either when the number of chunks is not big enough for balanced redistribution of data during re-sharding, or a particular chunk has become a hot spot.

Even with system-managed sharding, a single chunk may grow larger than other chunks or may become more active. In this case, splitting that chunk and allowing automatic re-sharding to move one of the resulting chunks to another shard maintains a more equal balanced distribution of data and activity across the environment.

Oracle Enterprise Manager heat maps show which chunks are more active than other chunks. Using this feature will help identify which chunks could be split, and one of the resulting chunks could then be moved to another shard to help rebalance the environment.

You can manage chunks using GDSCTL or Oracle Enterprise Manager Cloud Control:

- *Oracle Database Global Data Services Concepts and Administration Guide* for information about using the `GDSCTL SPLIT CHUNK` command
- Splitting Chunks with Oracle Enterprise Manager Cloud Control

Splitting Chunks into Shardspaces Based on Super Key

In a sharded database using the composite sharding data distribution method, the data can be organized into different shardspaces based on super shard key column values. You can split the existing data chunks per super shard key values into new shardspaces.

Splitting chunks by super sharding key is a unique operation of partition/chunk split which also requires data reorganization and movement. Many internal steps during the course of this operation make use of online DDL support.

Transactionally Consistent data movement with Online DDL across a chunk spanning across partitions of multiple tables allows benefits of online DDL mechanism for the data spread across multiple tables in a table family using sharding key.

Use Case

For example, you have a sharded database with tables Customers, Orders, and Lineitems, which are related to each other through the `customer_id` as reference key and sharding key. The sharded database uses the composite sharding data distribution method with Class for customers as the super shard key.

You want to arrange the location of the data by customer Class so that in the future, the data can be rearranged to provide different levels of service or resources to premium Class customers.

Due to business needs, a requirement arises for a new shardspace where data for only Gold and Silver Class customers reside in the future, though the data is not currently distributed that way. For example, there could be additional reporting or business services or data security services offered only to premium customers for additional costs, and segregating these customers in a different shardspace hosted in a different availability domain or region in the cloud makes it more convenient. Or, there is more efficient, but expensive hardware which can host only few shards and not all of the shards, so you want to allot those to the premium classes.

Splitting the Data into Shardspaces Based on Super Key

A `PARTITIONSET` operator, `SPLIT`, is introduced to support splitting chunks by super sharding key, as shown in this `ALTER TABLE` syntax.

```
ALTER TABLE tablename
SPLIT PARTITIONSET partitionset_name
INTO
(partitionset partitionset_name
values [(list of values)] | [LESS THAN (value)]
 [lob_column1 store as (tablespace set_name1),
 lob_column2 store as (tablespace set_name2) … ]
[[SUBPARTITIONS store in (<tablespace set_name1,
tablespace set_name2, …]|[tablespace_set
tablespaceset_name]],
  partitionset partitionset_name
[lob_column1 store as (tablespace set_name1),
 lob_column2 store as (tablespace set_name2) … ]
[[SUBPARTITIONS store in (<tablespace set_name1,
Tablespace set_name2, …]|[tablespace_set
tablespaceset_name]]) ;
```

For example, to split a customers partitionset into Gold customers and non-Gold customers:

```
ALTER TABLE customers
SPLIT PARTITIONSET all_customers
INTO (PARTITIONSET gold_customers
VALUES ('gold')
 TABLESPACE SET ts2,
PARTITIONSET non_gold_customers)
```

As shown above, the command resembles `ALTER TABLE SPLIT PARTITION`; however, there are certain rules for this syntax in the `SPLIT PARTITIONSET` case as follows.

1. The number of shards in the target shardspace must be the same as the source shardspace.
2. The target shardspace must be clean and not have any chunks.
3. The number of chunks in the target shardspace must be the same as the source shardspace.
4. Only two resulting partitionsets are allowed from one source partitionset. So the command can have at most 3 partitionsets specified.
5. You can specify the `STORAGE` clauseor `TABLESPACE SET` clause with only one resulting partitionset and not both.
6. Whichever partitionset has a `TABLESPACE SET` or `STORAGE` clause is considered the target partitionset; that is, the new partitionset on a different shardspace. All of the tablespace sets specified in the clause need to be on the target, or new shardspace.
7. `PARTITIONSET` without `TABLESPACE SET` clause is considered local; that is, it will be on the same shardspace the as existing source shardspace.
8. The `VALUES` clause must always be specified with the first `PARTITIONSET` in the list. This enables either the upper or lower side of the range to be moved to the target shardspace in case of range partitioning for partitionsets. This clause can't be specified or is implicit for the second resulting partitionset.
9. Resulting partitionsets must have distinct names.

Note that in case of range partitioning for partitionset keys, the command can retain the upper range or lower range in the original partitionset, by indicating the storage clause with the intended target partitionset. For example,

```
ALTER TABLE employees SPLIT PARTITIONSET employees
into (PARTITIONSET junior_employees values less than 10,
PARTITIONSET senior_employees TABLESPACE SET ts4)
```

The above command has 'senior_employees' as the new, or target, partitionset. Rows of partitionset employees having partitionset key column values less than 10 remain in the origin partitionset, and this partitionset is renamed 'junior_employees' at the end of the operation.

Rows of partitionset employees having partitionset key column values equal to or greater than 10 are moved to a new partitionset 'senior_employees' with a shardspace different than that of partitionset 'employees'. Tablespace set clause associated with partitionset 'senior_employees' and tablespace set 'ts4' being in the target shardspace indicates that this partitionset will be in the new target shardspace.

Managing Chunks with Oracle Enterprise Manager Cloud Control

You can manage sharded database chunks using Oracle Enterprise Manager Cloud Control.

The following topics describe chunk management using Oracle Enterprise Manager Cloud Control:

Moving Chunks with Oracle Enterprise Manager Cloud Control

You can move chunks from one shard to another in your deployment using Oracle Enterprise Manager Cloud Control.

1. From a shardspace management page, open the **Shardspace** menu, located in the top left corner of the Sharded Database target page, and choose **Manage Shardgroups**.
2. Select a shardgroup in the list and click **Move Chunks**.
3. In the Move Chunks dialog, select the source and destination shards between which to move the chunks.
4. Select the chunks that you want to move by choosing one of the options.
   - **Enter ID List**: enter a comma separates list of chunk ID numbers
   - **Select IDs From Table**: click the chunk IDs in the table
5. Indicate when the chunk move should occur.
   - **Immediately**: the chunk move is provisioned upon confirmation
   - **Later**: schedule the timing of the chunk move using the calendar tool in the adjacent field
6. Click **OK**.
7. Click the link in the **Information** box at the top of the page to view the provisioning status of the chunk move.

Splitting Chunks with Oracle Enterprise Manager Cloud Control

You can split chunks in your deployment using Oracle Enterprise Manager Cloud Control.

1. Open the **Sharded Database** menu, located in the top left corner of the Sharded Database target page, and choose **Shardspaces**.
2. If prompted, enter the shard catalog credentials, select the shard director to manage under **Shard Director Credentials**, select the shard director host credentials, and log in.
3. Select a shardspace in the list and click **Split Chunks**.
4. Select the chunks that you want to split by choosing one of the options.
   - **Enter ID List**: enter a comma separate list of chunk ID numbers
   - **Select IDs From Table**: click the chunk IDs in the table
5. Indicate when the chunk split should occur.
   - **Immediately**: the chunk split is provisioned upon confirmation
   - **Later**: schedule the timing of the chunk split using the calendar tool in the adjacent field
6. Click **OK**.
7. Click the link in the **Information** box at the top of the page to view the provisioning status of the chunk split.

When the chunk is split successfully the number of chunks is updated in the **Shardspaces** list. You might need to refresh the page to see the updates.

Shard Director Management

You can add, edit, and remove shard directors in your deployment with Oracle Enterprise Manager Cloud Control.

Creating a Shard Director

Use Oracle Enterprise Manager Cloud Control to create and add a shard director to your deployment.

1. Open the **Sharded Database** menu, located in the top left corner of the Sharded Database target page, and choose **Shard Directors**.
2. If prompted, enter the shard catalog credentials, select the shard director to manage under **Shard Director Credentials**, select the shard director host credentials, and log in.
3. Click **Create**, or select a shard director from the list and click **Create Like**.

   Choosing **Create** opens the Add Shard Director dialog with default configuration values in the fields.

   Choosing **Create Like** opens the Add Shard Director dialog with configuration values from the selected shard director in the fields. You must select a shard director from the list to enable the **Create Like** option.
4. Enter the required information in the Add Shard Director dialog, and click **OK**.

   > **Note:**
   >
   > If you do not want the shard director to start running immediately upon creation, you must uncheck the **Start Shard Director After Creation** checkbox.
5. Click **OK** on the confirmation dialog.
6. Click the link in the **Information** box at the top of the page to view the provisioning status of the shard director.

When the shard director is created successfully it appears in the **Shard Directors** list. You might need to refresh the page to see the updates.

Editing a Shard Director Configuration

Use Oracle Enterprise Manager Cloud Control to edit a shard director configuration in your deployment.

You can change the region, ports, local endpoint, and host credentials for a shard director in Cloud Control. You cannot edit the shard director name, host, or Oracle home.

1. Open the **Sharded Database** menu, located in the top left corner of the Sharded Database target page, and choose **Shard Directors**.
2. If prompted, enter the shard catalog credentials, select the shard director to manage under **Shard Director Credentials**, select the shard director host credentials, and log in.
3. Select a shard director from the list and click **Edit**.

   Note that you cannot edit the shard director name, host, or Oracle home.

4. Edit the fields, enter the GSMCATUSER password, and click **OK**.
5. Click the link in the **Information** box at the top of the page to view the provisioning status of the shard director configuration changes.

Removing a Shard Director

Use Oracle Enterprise Manager Cloud Control to remove shard directors from your deployment.

If the shard director you want to remove is the administrative shard director, as indicated by a check mark in that column of the **Shard Directors** list, you must choose another shard director to be the administrative shard director before removing it.

1. Open the **Sharded Database** menu, located in the top left corner of the Sharded Database target page, and choose **Shard Directors**.
2. If prompted, enter the shard catalog credentials, select the shard director to manage under **Shard Director Credentials**, select the shard director host credentials, and log in.
3. Select a shard director from the list and click **Delete**.
4. Click the link in the **Information** box at the top of the page to view the provisioning status of the shard director removal.

When the shard director is removed successfully it no longer appears in the **Shard Directors** list. You might need to refresh the page to see the changes.

Region Management

You can add, edit, and remove regions in your deployment with Oracle Enterprise Manager Cloud Control.

Creating a Region

Create sharded database regions in your deployment using Oracle Enterprise Manager Cloud Control.

1. Open the **Sharded Database** menu, located in the top left corner of the Sharded Database target page, and choose **Regions**.
2. If prompted, enter the shard catalog credentials, select the shard director to manage under **Shard Director Credentials**, select the shard director host credentials, and log in.
3. Click **Create**.
4. Enter a unique name for the region in the Create Region dialog.
5. Optionally, select a buddy region from among the existing regions.
6. Click **OK**.
7. Click the link in the **Information** box at the top of the page to view the provisioning status of the region.

When the region is created successfully it appears in the **Regions** list. You might need to refresh the page to see the updates.

Editing a Region Configuration

Edit sharded database region configurations in your deployment using Oracle Enterprise Manager Cloud Control.

You can change the buddy region for a sharded database region in Cloud Control. You cannot edit the region name.

1. Open the **Sharded Database** menu, located in the top left corner of the Sharded Database target page, and choose **Regions**.
2. If prompted, enter the shard catalog credentials, select the shard director under **Shard Director Credentials**, select the shard director host credentials, and log in.
3. Select a region from the list and click **Edit**.
4. Select or remove a buddy region, and click **OK**.
5. Click the link in the **Information** box at the top of the page to view the provisioning status of the region configuration changes.

When the region configuration is successfully updated the changes appear in the **Regions** list. You might need to refresh the page to see the updates.

Removing a Region

Remove sharded database regions in your deployment using Oracle Enterprise Manager Cloud Control.

1. Open the **Sharded Database** menu, located in the top left corner of the Sharded Database target page, and choose **Regions**.
2. If prompted, enter the shard catalog credentials, select the shard director under **Shard Director Credentials**, select the shard director host credentials, and log in.
3. Select a region from the list and click **Delete**.
4. Click the link in the **Information** box at the top of the page to view the provisioning status of the region removal.

When the region configuration is successfully removed the changes appear in the **Regions** list. You might need to refresh the page to see the updates.

Shardspace Management

You can add, edit, and remove shardspaces in your deployment with Oracle Enterprise Manager Cloud Control.

Creating a Shardspace

Create shardspaces in your composite sharded database deployment using Oracle Enterprise Manager Cloud Control.

Only databases that are sharded using the composite method can have more than one shardspace. A system-managed sharded database can have only one shardspace.

1. Open the **Sharded Database** menu, located in the top left corner of the Sharded Database target page, and choose **Shardspaces**.

2. If prompted, enter the shard catalog credentials, select the shard director to manage under **Shard Director Credentials**, select the shard director host credentials, and log in.
3. Click **Create**.

> **Note:**
>
> This option is disabled in the Shardspaces page for a system-managed sharded database.

4. Enter the values in the fields in the Add Shardspace dialog, and click **OK**.
   - **Name**: enter a unique name for the shardspace (required)
   - **Chunks**: Enter the number of chunks that should be created in the shardspace (default 120)
   - **Protection Mode**: select the Data Guard protection mode (default Maximum Performance)
5. Click the link in the **Information** box at the top of the page to view the provisioning status of the shardspace.

When the shardspace is created successfully it appears in the **Shardspaces** list. You might need to refresh the page to see the updates.

Adding a Shardspace to a Composite Sharded Database

Learn to create a new shardspace, add shards to the shardspace, create a tablespace set in the new shardspace, and add a partitionset to the sharded table for the added shardspace. Then verify that the partitions in the tables are created in the newly added shards in the corresponding tablespaces.

To add a new shardspace to an existing sharded database, make sure that the composite sharded database is deployed and all DDLs are propagated to the shards.

1. Create a new shardspace, add shards to the shardspace, and deploy the environment.
   a. Connect to the shard catalog database.

```
GDSCTL> connect mysdbadmin/mysdbadmin_password
```

   b. Add a shardspace and add a shardgroup to the shardspace.

```
GDSCTL> add shardspace -chunks 8 -shardspace cust_asia
GDSCTL> add shardgroup -shardspace cust_asia -shardgroup
asia_shgrp1 -deploy_as primary -region region3
```

   c. Add shards

```
GDSCTL> add shard -shardgroup asia_shgrp1 -connect
shard_host:TNS_listener_port/shard_database_name -pwd
GSMUSER_password
GDSCTL> add shard asia_shgrp1 -connect
```

```
shard_host:TNS_listener_port/shard_database_name -pwd GSMUSER_password
```

d. Deploy the environment.

```
GDSCTL> deploy
```

Running `DEPLOY` ensures that all of the previous DDLs are replayed on the new shards and all of the tables are created. The partition is created in the default SYS_SHARD_TS tablespace.

2. On the shard catalog create the tablespace set for the shardspace and add partitionsets to the sharded root table.

   a. Create the tablespace set.

```
SQL> CREATE TABLESPACE SET
  TSP_SET_3 in shardspace cust_asia using template
  (datafile size 100m autoextend on next 10M maxsize
   unlimited extent management
   local segment space management auto );
```

   b. Add the partitionset.

```
SQL> ALTER table customers add PARTITIONSET asia VALUES ('ASIA”')
TABLESPACE SET TSP_SET_3 ;
```

   c. When lobs are present, create the tablespace set for lobs and mention the lob storage information in the add partitionset command.

```
SQL> alter table customers add partitionset asia VALUES ('ASIA')
tablespace set TSP_SET_3 lob(docn) store as (tablespace set
LOBTSP_SET_4)) ;
```

   d. When the root table contains subpartitions, use the store as clause to specify the tablespace set for the subpartitions.

```
SQL> alter table customers add partitionset asia VALUES ('ASIA')
tablespace set TSP_SET_3 subpartitions store in(SUB_TSP_SET_1,
SUB_TSP_SET_2);
```

The `ADD PARTITIONSET` command ensures that the child tables are moved to the appropriate tablespaces.

3. Verify that the partitions in the new shardspace are moved to the new tablespaces.

   Connect to the new shards and verify that the partitions are created in the new tablespace set.

```
SQL> select table_name, partition_name, tablespace_name, read_only from
dba_tab_partitions;
```

Shardgroup Management

You can add, edit, and remove shardgroups in your deployment with Oracle Enterprise Manager Cloud Control.

Creating a Shardgroup

Create shardgroups in your deployment using Oracle Enterprise Manager Cloud Control.

1. Select a shardspace to which to add the shardgroup.
2. Open the **Shardspace** menu, located in the top left corner of the shardspace target page, and choose **Manage Shardgroups**.
3. Click **Create**.
4. Enter values in the Create Shardgroup dialog, and click **OK**.
5. Click the link in the **Information** box at the top of the page to view the provisioning status of the shardgroup.

   For example, with the values entered in the screenshots above, the following command is run:

```
GDSCTL Command: ADD SHARDGROUP -SHARDGROUP 'north' -SHARDSPACE
'shardspaceora'
 -REGION 'north' -DEPLOY_AS 'STANDBY'
```

When the shardgroup is created successfully it appears in the **Manage Shardgroups** list. You might need to refresh the page to see the updates.

Services Management

You can manage services in your deployment with Oracle Enterprise Manager Cloud Control.

To manage Oracle Globally Distributed Database services, open the **Sharded Database** menu, located in the top left corner of the Sharded Database target page, and choose **Services**. On the Services page, using the controls at the top of the list of services, you can start, stop, enable, disable, create, edit, and delete services.

Selecting a service opens a service details list which displays the hosts and shards on which the service is running, and the status, state, and Data Guard role of each of those instances. Selecting a shard in this list allows you to enable, disable, start, and stop the service on the individual shards.

Creating a Service

Create services in your deployment using Oracle Enterprise Manager Cloud Control.

1. Open the **Sharded Database** menu, located in the top left corner of the Sharded Database target page, and choose **Services**.
2. If prompted, enter the shard catalog credentials, select the shard director to manage under **Shard Director Credentials**, select the shard director host credentials, and log in.
3. Click **Create**, or select a service from the list and click **Create Like**.

   Choosing **Create** opens the Create Service dialog with default configuration values in the fields.

Choosing **Create Like** opens the Create Like Service dialog with configuration values from the selected service in the fields. You must select a service from the list to enable the **Create Like** option.

4. Enter the required information in the dialog, and click **OK**.

> **Note:**
>
> If you do not want the service to start running immediately upon creation, you must uncheck the **Start service on all shards after creation** checkbox.

5. Click the link in the **Information** box at the top of the page to view the provisioning status of the service.

When the service is created successfully it appears in the **Services** list. You might need to refresh the page to see the updates.

8

Query and DML Processing

On a sharded database, queries and DML can be routed to the shards for processing with or without a sharding key. If a key is provided by the application a database request is routed directly to the shards, but if no key is provided the request is processed by the shard catalog, and then directed to the necessary shards for processing.

- How Database Requests are Routed to the Shards
- Connecting to the Query Coordinator
- Query Coordinator Operation
- Query Processing for Single-Shard Queries
- Query Processing for Multi-Shard Queries
- Gathering Optimizer Statistics on Sharded Tables
- Supported Query Constructs and Example Query Shapes
- Supported DMLs and Examples

How Database Requests are Routed to the Shards

In Oracle Globally Distributed Database, database query and DML requests are routed to the shards in two main ways, depending on whether a sharding key is supplied with the request.

These two routing methods are called *direct routing* and *proxy routing*.

Direct Routing

You can connect directly to the shards to process queries and DML by providing a sharding key with the database request. Direct routing is the preferred way of accessing shards to achieve better performance, among other benefits.

Proxy Routing

Queries that need data from multiple shards, and queries that do not specify a sharding key, cannot be routed directly by the application. Those queries require a proxy to route requests between the application and the shards. Proxy routing is handled by the shard catalog *query coordinator*.

Routing Queries and DMLs Directly to Shards

Applications can have their requests routed directly to the shards if they provide a sharding key. With the direct routing mechanism, requests can only query and manipulate the data that belongs to the shard they were routed to.

Direct access to the data on the shards has several advantages.

- Offers better performance: Overall, applications experience better performance compared to routing requests to the shards indirectly through the shard catalog (by

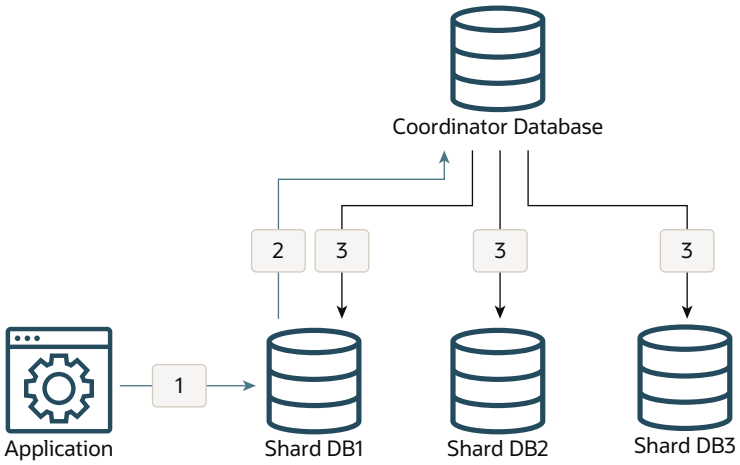
proxy). With direct routing there is no need for the requests and the results to pass through a coordinator database.

- Accommodates geographic distribution of shards: Applications can access the data in shards localized in their region.
- Eases load balancing: Load balancing application requests across the shards can be easily achieved by moving the data across shards using chunk moves.
- Supports all type of queries:
  - `SELECT`, `INSERT`, and `UPDATE` on sharded tables: The scope of these requests is the data that belong to the shards accessed.
  - `SELECT`, `INSERT`, and `UPDATE` on duplicated tables: The scope of theses requests is all of the data in the duplicated tables. Because the primary copies of a duplicated tables reside in the coordinator database, the DMLs on the duplicated tables are re-routed to the coordinator database.

The following figure illustrates DML on duplicated tables using direct routing to a shard.

1. The Application sends the DML request directly to one of the shards, Shard DB1.
2. The DML is forwarded from Shard DB1 to the Coordinator Database, where it is run on the primary duplicated tables.
3. The Coordinator Database refresh mechanism runs periodically to update the instances of the duplicated tables on all of the shards.

Figure 8-1 DML on a Duplicated Table with Direct Routing



For more information about direct routing, see Client Application Request Routing.

For information about developing applications for direct routing, see Developing Applications for Oracle Globally Distributed Database

Routing Queries and DMLs by Proxy

Using the shard catalog query coordinator as a proxy, Oracle Globally Distributed Database can handle request routing for queries and DMLs that do not specify a sharding key.

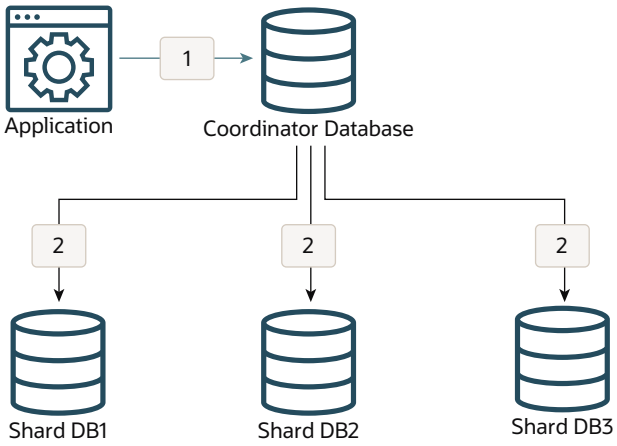
By using the coordinator as a proxy, Oracle Globally Distributed Database provides you with the flexibility to allow any database application to run SQL statements without the need to specify the shards where the query should be processed.

For more information about the coordinator, see Query Processing and the Query Coordinator.

The following figure illustrates DML on duplicated tables using proxy routing.

1. The Application sends the DML request to the Coordinator Database where it is run on the primary duplicated tables.
2. The Coordinator Database refresh mechanism runs periodically to update the instances of the duplicated tables on all of the shards.

Figure 8-2 DML on a Duplicated Table with Proxy Routing



The remaining topics in this chapter discuss routing and processing database requests by proxy.

Connecting to the Query Coordinator

The query coordinator, a component of the shard catalog, contains the metadata of the sharded topology and provides query processing support for sharded databases.

To perform multi-shard queries, connect to the multi-shard coordinator using the `GDS$CATALOG` service on the shard catalog database.

```
sqlplus app_schema/app_schema@shardcatvm:1521/GDS\$CATALOG.oradbcloud
```

For more information about the coordinator, see Query Processing and the Query Coordinator

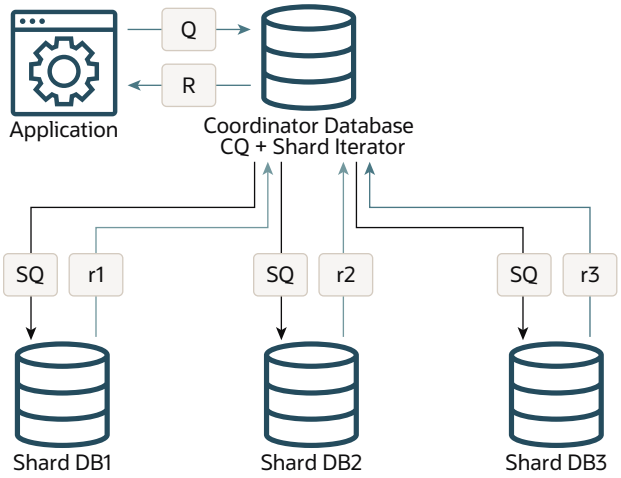
Query Coordinator Operation

The SQL compiler in the shard catalog identifies the relevant shards automatically, and coordinates the query processing across all of the participating shards. Database links are used for the communication between the coordinator and the shards.

As shown in the following figure, at a high level, the coordinator rewrites each incoming query, Q, into two queries, Coordinator Query (CQ) and Shard Query (SQ) where SQ, where SQ (Shard Query) is the part of Q that runs on each participating shard, and CQ (Coordinator Query) is the part of Q that runs on the coordinator shard.

A query, Q, is rewritten into `CQ ( Shard_Iterator( SQ ) )`, where the **Shard_Iterator** is the operator that connects to the shards and runs SQ. It can be run in parallel or serially.

Figure 8-3 Query Coordinator Operation



The following is an example of an aggregate query, Q1, rewritten into Q1'.

```
Q1 : SELECT COUNT(*) FROM customers

Q1': SELECT SUM(sc) FROM (Shard_Iterator(SELECT COUNT(*) sc FROM s1
(i) ))
```

There are two main elements in this process.

1. The relevant shards are identified.
2. The query is rewritten into a distributive form and iterated across the relevant shards.

During the query compilation on the coordinator database, the query compiler analyzes the predicates on the sharding key, and extracts the predicates that can be used to identify the participating shards, that is, the shards that will contribute rows for the sharded tables referenced in the query. The rest of the shards are referred to as **pruned** shards.

In the case where only one participating shard was identified, the full query is routed to that shard for processing. This is called a **single-shard query**.

If there is more than one participating shard, the query is called a **multi-shard query** and it is rewritten. The rewriting process takes into account the expressions computed by the query as well as the query shape.

Query Processing for Single-Shard Queries

A single-shard query is a query which needs to scan data from only one shard and does not need to lookup data from any other shards.

The single-shard query is similar to a client connecting to a specific shard and issuing a query on that shard. In this scenario, the entire query will be processed on the single participating shard, and the coordinator just passes processed rows back to the client. The plan on the coordinator is similar to the remote mapped cursor.

For example, the following query is fully mapped to a single shard because the data for customer 123 is located only on that shard.

```
SELECT count(*) FROM customers c, orders o WHERE c.custno = o.custno and
c.custno = 123;
```

The query contains a condition on the shard key that maps to one and only one shard which is known at query compilation time (literals) or query start time (bind). The query is fully processed on the qualifying shard.

Single-shard queries are supported for:

- Equality and In-list, such as `Area = 'West'`
- Conditions containing literal, bind, or expression of literals and binds, such as

  ```
  Area = :bind
  ```

  ```
  Area = CASE :bind <10 THEN 'West' ELSE 'East' END
  ```

- `SELECT`, `UPDATE`, `DELETE`, `INSERT`, `FOR UPDATE`, and `MERGE`. `UPSERT` is not supported.

Query Processing for Multi-Shard Queries

A multi-shard query is a query that must scan data from more than one shard, and the processing on each shard is independent of any other shard.

A **multi-shard query** maps to more than one shard and the coordinator might need to do some processing before sending the result to the client. For example, the following query gets the number of orders placed by each customer.

```
SELECT count(*), c.custno FROM customers c, orders o WHERE c.custno =
o.custno
 GROUP BY c.custno;
```

The query is transformed to the following by the coordinator.

```
SELECT sum(count_col), custno FROM (SELECT count(*) count_col, c.custno
 FROM customers c, orders o
 WHERE c.custno = o.custno GROUP BY c.custno) GROUP BY custno;
```

The inline query block is mapped to every shard just as a remote mapped query block. The coordinator performs further aggregation and `GROUP BY` on top of the result set from all shards. The unit of processing on every shard is the inline query block.

Multi-Shard Queries and Global Read Consistency

A multi-shard query must maintain global read consistency (CR) by issuing the query at the highest common SCN across all the shards. See Specifying Consistency Levels in a Multi-Shard Query for information about how to set consistency levels.

Passing Hints in Multi-Shard Queries

Any hint specified in the original query on the coordinator is propagated to the shards.

Tracing and Troubleshooting Slow Running Multi-Shard Queries

Set the trace event `shard_sql` on the coordinator to trace the query rewrite and shard pruning. One of the common performance issues observed is when the GROUP BY is not pushed to the shards because of certain limitations of the sharding. Check if all of the possible operations are pushed to the shards and the coordinator has minimal work to consolidate the results from shards.

Specifying Consistency Levels in a Multi-Shard Query

You can use the initialization parameter MULTISHARD_QUERY_DATA_CONSISTENCY to set different consistency levels when running multi-shard queries across shards.

You can specify different consistency levels for multi-shard queries. For example, you might want some queries to avoid the cost of SCN synchronization across shards, and these shards could be globally distributed. Another use case is when you use standbys for replication and slightly stale data is acceptable for multi-shard queries, as the results could be fetched from the primary and its standbys.

The default mode is strong, which performs SCN synchronization across all shards. Other modes skip SCN synchronization. The `delayed_standby_allowed` level allows fetching data from the standbys as well, depending on load balancing and other factors, and could contain stale data.

This parameter can be set either at the system level or at the session level.

> **See Also:**
>
> *Oracle Database Reference* for more information about MULTISHARD_QUERY_DATA_CONSISTENCY usage.

Multi-Shard Query Coordinator Availability and Scalability

The multi-shard query coordinator, a component of the shard catalog, can be kept highly available and scaled to meet its workload with these recommendations.

The availability of the multi-shard coordinator impacts proxy-routing based workloads, so it is highly recommended that the coordinator be protected with Data Guard in

Maximum Availability protection mode (zero data loss failover) with fast-start failover enabled. The coordinator may optionally be Oracle RAC-enabled for additional availability and scalability.

To improve the scalability and availability of multi-shard query workloads, Oracle Active Data Guard standby shard catalog databases in read-only mode can act as multi-shard query coordinators. For each active replica of the catalog database, a special coordinator service, `GDS$COORDINATOR.`*cloud_name* (where *cloud_name* is the value specified for the `configname` parameter in the `GDSCTL CREATE SHARDCATALOG` command, and is `oradbcloud` by default) is running and registered on all shard directors.

Clients can connect to this service on any of the replicas and perform multi-shard queries, allowing shard directors to distribute the multi-shard query workload with respect to runtime load balancing and decrease the load on in the primary shard catalog, which is the central component of Oracle Globally Distributed Database.

Additionally, if the database's region is set, and the client specifies the region in the connection string, a shard director routes a connection with respect to regional affinity.

Availability of the multi-shard query coordinator has zero impact on workloads using direct routing.

Pushing PL/SQL Function Queries to the Shards

The `SHARD_ENABLE` keyword allows the PL/SQL `CREATE` function statement to indicate that a query that references this function can be pushed down into the shards.

Note that the `parallel_enable_clause` uses the `PARALLEL_ENABLE` keyword, which is used to parallelize the execution of a query with the PL/SQL function within one shard.

However, the `SHARD_ENABLE` keyword, is used to parallelize the query across **all shards**. Therefore, these two keywords are different and can be used simultaneously to achieve parallel execution within and across shards.

Existing PL/SQL functions called from a cross-shard query (CSQ) will be executed on the shard catalog and won't be pushed to the shards. To benefit from the PL/SQL functions support, you need to re-run `CREATE` or replace the PL/SQL functions with the keyword `SHARD_ENABLE`.

See SHARD_ENABLE Clause in *Oracle Database PL/SQL Language Reference* for more information about syntax and usage.

Gathering Optimizer Statistics on Sharded Tables

You can gather statistics on sharded tables from the coordinator database.

The statistic preference parameter `COORDINATOR_TRIGGER_SHARD`, when set to `TRUE` on all of the shards, allows the coordinator database to import the statistics gathered on the shards.

The PL/SQL procedures `DBMS_STATS.GATHER_SCHEMA_STATS()` and `DBMS_STATS.GATHER_TABLE_STATS()` gather statistics on sharded tables and duplicated tables in the shards and in the coordinator database. See also, REPORT_GATHER_TABLE_STATS Function.

Manual Statistics Gathering

1. Set `COORDINATOR_TRIGGER_SHARD` to `TRUE` on all of the shards.

This step is performed only one time and only on the shards. If, for example, you have a schema named `sharduser`:

```
connect / as sysdba
EXECUTE
DBMS_STATS.SET_SCHEMA_PREFS('SHARDUSER','COORDINATOR_TRIGGER_SHARD',
'TRUE');
```

2. Gather statistics across the shards.

   The user should be an all-shards user and needs to have privileges to access dictionary tables.

   a. On the shards run the following.

   ```
   connect sharduser/password
   EXEC DBMS_STATS.GATHER_SCHEMA_STATS(ownname => 'SHARDUSER',
   options => 'GATHER');
   ```

   b. When all shards are completed, to pull aggregated statistics run the following on the coordinator.

   ```
   connect sharduser/password
   EXEC DBMS_STATS.GATHER_SCHEMA_STATS(ownname => 'SHARDUSER',
   options => 'GATHER');
   ```

   c. Check the statistics on all of the shards.

   ```
   connect sharduser/password

   ALTER SESSION SET nls_date_format='DD-MON-YYYY HH24:MI:SS';
     col TABLE_NAME form a40
     set pagesize 200 linesize 200

   SELECT TABLE_NAME, NUM_ROWS, sharded, duplicated, last_analyzed
     FROM user_tables
     WHERE table_name not like 'MLOG%' and table_name not like
   'RUPD%'
     and table_name not like 'USLOG%';
   ```

Automatic Statistics Gathering

1. Set `COORDINATOR_TRIGGER_SHARD` to `TRUE` on all of the shards.

   This step is performed only one time and only on the shards. If, for example, you have a schema named `sharduser`:

   ```
   connect / as sysdba
   EXECUTE
   DBMS_STATS.SET_SCHEMA_PREFS('SHARDUSER','COORDINATOR_TRIGGER_SHARD',
   'TRUE');
   ```

2. Schedule a job to pull aggregated statistics on the shards and on the coordinator database.

The user should be an all-shards user and must have privileges to access dictionary tables.

Start the following job on the shards:

```
connect sharduser/password
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
   job_name => 'Gather_Stats_2',
   job_type => 'PLSQL_BLOCK',
   job_action => 'BEGIN DBMS_STATS.GATHER_SCHEMA_STATS(ownname =>
''DEMO'', options => ''GATHER''); END;',
   start_date => SYSDATE,
   repeat_interval =>
'freq=daily;byday=MON,TUE,WED,THU,FRI,SAT,SUN;byhour=14;byminute=10;byseco
nd=00',
   end_date => NULL,
   enabled => TRUE,
   comments => 'Gather table statistics');
END;
/
```

After the job on all of the shards is finished, start the following job on the coordinator.

```
connect sharduser/password
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
   job_name             => 'Gather_Stats_2',
   job_type             => 'PLSQL_BLOCK',
   job_action           => 'BEGIN DBMS_STATS.GATHER_SCHEMA_STATS(ownname
=> ''DEMO'', options => ''GATHER''); END;',
   start_date           =>  SYSDATE,
   repeat_interval      =>
'freq=daily;byday=MON,TUE,WED,THU,FRI,SAT,SUN;byhour=15;byminute=10;byseco
nd=00',
   end_date             =>  NULL,
   enabled              =>  TRUE,
   comments             => 'Gather table statistics');
END;
/
```

Supported Query Constructs and Example Query Shapes

Oracle Globally Distributed Database supports single-shard and multi-shard query shapes with some restrictions.

The following are restrictions on query constructs in Oracle Globally Distributed Database.

- **CONNECT BY Queries** `CONNECT BY` queries are not supported.
- **MODEL Clause** The `MODEL` clause is not supported.
- **User-Defined PL/SQL in the WHERE Clause** User-defined PL/SQL is allowed in multi-shard queries only in the `SELECT` clause. If it is specified in the `WHERE` clause then an error is thrown.

- **XLATE and XML Query type** XLATE and XML Query type columns are not supported.
- **Object types** You can include object types in `SELECT` lists, `WHERE` clauses, and so on, but custom constructors and member functions of type object type are not permitted in `WHERE` clauses.

  Furthermore, for duplicated tables, non-final types, that is, object types that are created with the `NOT FINAL` keyword, cannot be used as a column data type. For sharded tables, non-final types can be used as a column data type but the column must be created with keywords `NOT SUBSTITUTABLE AT ALL LEVELS`.

> **Note:**
>
> Queries involving only duplicated tables are run on the coordinator.

The following topics show several examples of query shapes supported in Oracle Globally Distributed Database.

Queries on Sharded Tables Only

For a single-table query, the query can have an equality filter on the sharding key that qualifies a shard. For join queries, all of the tables should be joined using equality on the sharding key.

The following examples show queries where only sharded tables participate.

Example 8-1 Inner Join

```
SELECT … FROM s1 INNER JOIN s2 ON s1.sk=s2.sk
WHERE any_filter(s1) AND any_filter(s2)
```

Example 8-2 Left Outer Join

```
SELECT … FROM s1 LEFT OUTER JOIN s2 ON s1.sk=s2.sk
```

Example 8-3 Right Outer Join

```
SELECT … FROM s1 RIGHT OUTER JOIN s2 ON s1.sk=s2.sk
```

Example 8-4 Full Outer Join

```
SELECT … FROM s1 FULL OUTER JOIN s2 ON s1.sk=s2.sk
WHERE any_filter(s1) AND any_filter(s2)
```

Queries Involving Both Sharded and Duplicated Tables

A query involving both sharded and duplicated tables can be either a single-shard or multi-shard query, based on the predicates on the sharding key. The only difference is that the query contains a non-sharded table.

> **Note:**
>
> Joins between a sharded table and a duplicated table can be on any column, using any comparison operator, = < > <= >=, or arbitrary join expressions.

Example 8-5 Inner Join

```
SELECT … FROM s1 INNER JOIN r1 ON any_join_condition(s1,r1)
WHERE any_filter(s1) AND any_filter(r1)
```

Example 8-6 Left or Right Outer Join

In this case, the sharded table is the first table in `LEFT OUTER JOIN`.

```
SELECT … FROM s1 LEFT OUTER JOIN r1 ON any_join_condition(s1,r1)
WHERE any_filter(s1) AND any_filter(r1)

SELECT … FROM r1 LEFT OUTER JOIN s1 ON any_join_condition(s1,s2)
AND any_filter(r1) AND filter_one_shard(s1)
```

In this case, the sharded table is the second table in `RIGHT OUTER JOIN`.

```
SELECT … FROM r1 RIGHT OUTER JOIN s1 ON any_join_condition(s1,r1)
WHERE any_filter(s1) AND any_filter(r1)

SELECT … FROM s1 RIGHT OUTER JOIN r1 ON any_join_condition(s1,s2)
AND filter_one_shard(s1) AND any_filter(r1)
```

In some cases, the duplicated table is the first table in `LEFT OUTER JOIN`, or the sharded table is first and it maps to a single shard, based on filter predicate on the sharding key.

```
SELECT … FROM r1 LEFT OUTER JOIN s1 ON any_join_condition(s1,s2)
AND any_filter(r1) AND any_filter(s1)
```

In some cases, the duplicated table is the second table in `RIGHT OUTER JOIN`, or the sharded table is second and it maps to a single shard based on filter predicate on sharding key.

```
SELECT … FROM s1 RIGHT OUTER JOIN r1 ON any_join_condition(s1,s2)
AND any_filter (s1) AND any_filter(r1)
```

Example 8-7 Full Outer Join

```
SELECT … FROM s1 FULL OUTER JOIN r1 ON s1.sk=s2.sk
WHERE any_filter(s1) AND any_filter(s2)
```

In this case, the sharded table requires access to multiple shards:

```
SELECT … FROM s1 FULL OUTER JOIN r1 ON s1.non_sk=s2.non_sk
WHERE any_filter(s1) AND any_filter(s2)
```

Example 8-8 Semi-Join (EXISTS)

```
SELECT … FROM s1 EXISTS
(SELECT 1 FROM r1 WHERE r1.anykey=s1.anykey)

SELECT … FROM r1 EXISTS
(SELECT 1 FROM s1 WHERE r1.anykey=s1.anykey and filter_one_shard(s1))
```

In this case, the sharded table is in a subquery that requires the participation of multiple shards.

```
SELECT … FROM r1 EXISTS
(SELECT 1 FROM s1 WHERE r1.anykey=s1.anykey)
```

Example 8-9 Anti-Join (NOT EXISTS)

```
SELECT … FROM s1 NOT EXISTS
(SELECT 1 FROM r1 WHERE r1.anykey=s1.anykey)
```

In this case, the sharded table is in the sub-query.

```
SELECT … FROM r1 NOT EXISTS
(SELECT 1 FROM s1 WHERE r1.anykey=s1.anykey
```

Supported Aggregate Functions

The following aggregations are supported by proxy routing in Oracle Globally Distributed Database.

- COUNT
- SUM
- MIN
- MAX
- AVG

Queries with User-Defined Types

User-defined SQL object types and user-defined SQL collection types are referred to as user-defined types. Oracle Globally Distributed Database supports queries with user-defined types.

Example 8-10 Create Table with User-Defined Types

The following example creates an all-shard type and type body, then creates a sharded table referencing the type.

```
ALTER SESSION ENABLE SHARD DDL;

CREATE OR REPLACE TYPE person_typ AS OBJECT (
    first_name   VARCHAR2(20),
```

```
    last_name    VARCHAR2(25),
    email        VARCHAR2(25),
    phone        VARCHAR2(20),
    MEMBER FUNCTION details (
    self IN person_typ
    ) RETURN VARCHAR2
);
/

CREATE OR REPLACE TYPE BODY person_typ AS
    MEMBER FUNCTION details (
    self IN person_typ
    ) RETURN VARCHAR2 IS
        result VARCHAR2(100);
    BEGIN
        result := first_name || ' ' || last_name || ' ' || email || ' ' ||
phone;
        RETURN result;
    END;
END;
/

CREATE SHARDED TABLE Employees
( Employee_id      NUMBER NOT NULL
, person      person_typ
, signup_date DATE NOT NULL
, CONSTRAINT RootPK PRIMARY KEY(CustNo)
)
PARTITION BY CONSISTENT HASH (CustNo)
PARTITIONS AUTO
TABLESPACE SET ts1
;
```

Example 8-11 Insert Data Using Type Constructor

```
INSERT INTO Employees values ( 1, person_typ('John', 'Doe',
'jdoe@example.com', '123-456-7890'), to_date('24 Jun 2020', 'dd Mon YYYY'));
```

Example 8-12 Multi-Shard Query of a User-Defined Type Column

```
SELECT e.person FROM Employees e;


SELECT e.person.first_name, e.person.last_name FROM Employees e;


SELECT e.person.details() FROM Employee e where e.person.first_name = 'John’;


SELECT signup_date from Employees e where e.person = person_typ('John',
'Doe', 'jdoe@example.com', '123-456-7890’);
```

Execution Plans for Proxy Routing

In a multi-shard query, each shard produces an independent execution plan which is optimized for the data size and compute resources available on the shard.

You do not need to connect to individual shards to see the explain plan for SQL fragments. Interfaces provided in `dbms_xplan.display_cursor()` display on the coordinator the plans for the SQL segments run on the shards, and `[V/X]$SHARD_SQL` uniquely maps a shard SQL fragment of a multi-shard query to the target shard database.

SQL Segment Interfaces for `dbms_xplan.display_cursor()`

Two interfaces can display the plan for a SQL segment run on shards. The interfaces take shard IDs as the argument to display the plans from the specified shards. The `ALL_SHARDS` format displays the plans from all of the shards.

To print all of the plans from all shards use the `format` value `ALL_SHARDS` as shown here.

```
select * from table(dbms_xplan.display_cursor(sql_id=>:sqlid,

cursor_child_no=>:childno,
                                                format=>'BASIC
+ALL_SHARDS',
                                                shard_ids=>shard_ids))
```

To print selective plans from the shards, pass shard IDs in the `display_cursor()` function. For plans from multiple shards, pass an array of numbers containing shard IDs in the `shard_ids` parameter as shown here.

```
select * from table(dbms_xplan.display_cursor(sql_id=>:sqlid,

cursor_child_no=>:childno,
                                                 format=>'BASIC',
                                                 shard_ids=>ids))
```

To return a plan from one shard pass the shard ID directly to the `shard_id` parameter, as shown here.

```
select * from table(dbms_xplan.display_cursor(sql_id=>:sqlid,

cursor_child_no=>:childno,
                                                format=>'BASIC',
                                                shard_id=>1))
```

V$SQL_SHARD

`V$SQL_SHARD` uniquely maps a shard SQL fragment of a multi-shard query to the target shard database. This view is relevant only for the shard coordinator database to store a list of shards accessed for each shard SQL fragment for a given multi-shard query. Every time a multi-shard query runs, it can run a shard SQL fragment on different set

of shards, so the shard IDs update each time it is runs. This view maintains the SQL ID of a shard SQL fragment for each REMOTE node and the SHARD IDs on which the shard SQL fragment was run.

```
Name                                      Null?    Type
----------------------------------------- --------
----------------------------
 SQL_ID                                            VARCHAR2(13)
 CHILD_NUMBER                                      NUMBER
 NODE_ID                                           NUMBER
 SHARD_SQL_ID                                      VARCHAR2(13)
 SHARD_ID                                          NUMBER
 SHARD_CHILD_NUMBER                                NUMBER
```

- **SQL_ID** – SQL ID of a multi-shard query on coordinator
- **CHILD_NUMBER** – cursor child number of a multi-shard query on coordinator
- **NODE_ID** – ID of REMOTE node for a shard SQL fragment of a multi-shard query
- **SHARD_SQL_ID** – SQL ID of the shard SQL fragment for given remote NODE ID
- **SHARD_ID** – IDs of shards where the shard SQL fragment was run
- **SHARD _CHILD_NUMBER**– cursor child number of a shard SQL fragment on a shard (default 0)

The following is an example of a multi-shard query on the sharded database and the execution plan.

```
SQL> select count(*) from departments a where exists (select distinct
department_id
 from departments b where b.department_id=60);
------------------------------------------------
| Id  | Operation          | Name              |
------------------------------------------------
|   0 | SELECT STATEMENT   |                   |
|   1 |  SORT AGGREGATE    |                   |
|   2 |   FILTER           |                   |
|   3 |    VIEW            | VW_SHARD_377C5901 |
|   4 |     SHARD ITERATOR |                   |
|   5 |      REMOTE        |                   |
|   6 |    VIEW            | VW_SHARD_EEC581E4 |
|   7 |     SHARD ITERATOR |                   |
|   8 |      REMOTE        |                   |
------------------------------------------------
```

A query of SQL_ID on the `V$SQL_SHARD` view.

```
SQL> Select * from v$sql_shard where SQL_ID = '1m024z033271u';
SQL_ID        NODE_ID   SHARD_SQL_ID  SHARD_ID
------------- -------  -------------- --------
1m024z033271u       5   5z386yz9suujt        1
1m024z033271u       5   5z386yz9suujt       11
1m024z033271u       5   5z386yz9suujt       21
1m024z033271u       8   8f50ctj1a2tbs         11
```

> **See Also:**
>
> *Oracle Database PL/SQL Packages and Types Reference*
>
> *Oracle Database Reference*

Supported DMLs and Examples

DMLs in Oracle Globally Distributed Database can target either duplicated tables or sharded tables. There are no limitations on DMLs when the target is a duplicated table.

DMLs (mainly Insert, Update and Delete) targeting sharded tables can be

- Simple DMLs where only the target table is referenced
- DMLs referencing other tables
- Merge statements

Simple DMLs Where Only the Target Table is Referenced

The following are several examples of supported DMLs.

Example 8-13 Update all of the rows

```
UPDATE employees SET salary = salary *1.1;
```

Example 8-14 Insert one row

```
INSERT INTO employees VALUES (102494, 'Jane Doe, ...
    );
```

Example 8-15 Delete one row

```
DELETE employees WHERE employee_id = 103678;
```

DMLs Referencing Other Tables

DMLs on sharded tables can reference other sharded tables, duplicated tables, or local tables.

Example 8-16 DML referencing duplicated table

In this example, `employees` is a sharded table and `ref_jobs` is a duplicated table.

```
DELETE employees
            WHERE job_id IN (SELECT job_id FROM ref_jobs
                             WHERE job_id = 'SA_REP');
```

Example 8-17 DML referencing another sharded table

```
UPDATE departments SET department_name = 'ABC'
            WHERE department_id IN (SELECT department_id
                                    FROM employees
                                    WHERE salary < 10000);
```

Example 8-18 Insert as select from a local table

```
INSERT INTO employees SELECT * FROM local_employees;
```

Example 8-19 DML affecting one shard

A DML statement might affect only one shard, or it can involve multiple shards. For example, the `DELETE` statement shown here affects only one shard because there is a predicate on the sharding key (`employee_id`) in the `WHERE` clause..

```
DELETE employees WHERE employee_id = 103678;
```

Example 8-20 DML affecting multiple shards

The following statement affects all of the rows in the `EMPLOYEES` table because it does not have a `WHERE` clause.

```
UPDATE employees SET salary = salary *1.1;
```

To run this `UPDATE` statement on all shards, the shard coordinator iterates over all of the primary shard databases and invokes the `UPDATE` statement remotely. The coordinator starts a distributed transaction and performs two phase commit to guarantee the consistency of the distributed transaction. If there is an in-doubt transaction, you must recover it manually.

Example Merge Statements

The `MERGE` statement can target a sharded table or a duplicated table. The merge is allowed as long as the `MERGE` operation itself can be pushed to the shards.

Example 8-21 Merge statement with sharded table employees as the target table

In this example, the `employee_id` column is the sharding key, and the join predicate on the source query is on the sharding key, so the `MERGE` statement will get pushed to all of the shards to be processed.

```
MERGE INTO employees D
   USING (SELECT employee_id, salary, department_id FROM employees
   WHERE department_id = 80) S
   ON (D.employee_id = S.employee_id)
   WHEN MATCHED THEN UPDATE SET D.salary = D.salary + S.salary*.01
     DELETE WHERE (S.salary > 8000)
   WHEN NOT MATCHED THEN INSERT (D.employee_id, D.salary)
     VALUES (S.employee_id, S.salary*0.1)
     WHERE (S.salary <= 8000);
```

Example 8-22 Merge statement with duplicated table as the target table

In this example, the target table is the duplicated table `ref_employees`. The source query references the sharded table `employees` and the join predicate is on the sharding key `employee_id`.

```
MERGE INTO ref_employees D
   USING (SELECT employee_id, salary, department_id FROM employees
   WHERE department_id = 80) S
   ON (D.employee_id = S.employee_id)
   WHEN MATCHED THEN UPDATE SET D.salary = D.salary + S.salary*.01
     DELETE WHERE (S.salary > 8000)
   WHEN NOT MATCHED THEN INSERT (D.employee_id, D.salary)
     VALUES (S.employee_id, S.salary*0.1)
      WHERE (S.salary <= 8000);
```

Limitations in Multi-Shard DML Support

The following DML features are not supported by multi-shard DML in Oracle Globally Distributed Database.

- **Error Logging** The `ERROR LOG` clause with DML is not supported by multi-shard DML. A user error is raised in this case.
- **Array DML** Array DML is not supported by multi-shard DML. ORA-2681 is raised in this cases.
- **RETURNING Clause** The `RETURNING INTO` clause is not supported by regular distributed DMLs; therefore, it is not supported. ORA-22816 is raised if you try to use the `RETURNING INTO` clause in multi-shard DMLs.
- **MERGE and UPSERT** The `MERGE` statement is partially supported, that is, a `MERGE` statement affecting only single shard is supported. ORA error is raised if a `MERGE` statement requires the modification of multiple shards.
- **Multi-Table INSERT** Multi-table inserts are not supported by database links; therefore, multi-table inserts are not supported.
- **Updatable Join View** ORA-1779 is thrown when the updatable join view has a join on a sharded table on sharding keys. The reason for this error is that the primary key defined on a sharded table is combination of internal column `SYS_HASHVAL` + sharding key and you cannot specify `SYS_HASHVAL` in the updatable join view. Because of this restriction you cannot establish the key-preserved table resulting in raising ORA-1779.
- **Triggers**

9

Developing Applications for Oracle Globally Distributed Database

You can develop your application to direct requests to a shard within a sharded database.

- Direct Routing to a Shard
- APIs Supporting Direct Routing
- JDBC Sharding Data Source

Direct Routing to a Shard

Oracle clients and connections pools are able to recognize sharding keys specified in the connection string for high performance data dependent routing. A shard routing cache in the connection layer is used to route database requests directly to the shard where the data resides.

In direct, key-based, routing to a shard, a connection is established to a single, relevant shard which contains the data pertinent to the required transaction using a sharding key.

A sharding key is used to route database connection requests at a user session level during connection checkout. The composite sharding method requires both a sharding key and a super sharding key. Direct, key-based, routing requires the sharding key (or super sharding key) be passed as part of the connection. Based on this information, a connection is established to the relevant shard which contains the data pertinent to the given sharding key or super sharding key.

Once the session is established with a shard, all SQL queries and DMLs are run in the scope of the given shard. This routing is fast and is used for all OLTP workloads that perform intra-shard transactions. It is recommended that direct routing be employed for all OLTP workloads that require the highest performance and availability.

In support of Oracle Globally Distributed Database, key enhancements have been made to Oracle connection pools and drivers. JDBC, Universal Connection Pool (UCP), OCI Session Pool (OCI), and Oracle Data Provider for .NET (ODP.NET) provide APIs to pass sharding keys during the connection creation. Apache Tomcat, IBM Websphere, Oracle WebLogic Server, and JBOSS can leverage JDBC/UCP support and use sharding. PHP, Python, Perl, and Node.js can leverage OCI support.

A shard topology cache is a mapping of the sharding key ranges to the shards. Oracle Integrated Connection Pools maintain this shard topology cache in their memory. Upon the first connection to a given shard (during pool initialization or when the pool connects to newer shards), the sharding key range mapping is collected from the shards to dynamically build the shard topology cache.

Caching the shard topology creates a fast path to the shards and expedites the process of creating a connection to a shard. When a connection request is made with a sharding key, the connection pool looks up the corresponding shard on which this particular sharding key exists (from its topology cache). If a matching connection is available in the pool then the pool returns a connection to the shard by applying its internal connection selection algorithm.

A database connection request for a given sharding key that is in any of the cached topology map, goes directly to the shard (that is, bypassing the shard director). Connection Pool also subscribes to RLB notifications from the SDB and dispenses the best connection based on runtime load balancing advisory. Once the connection is established, the client runs transactions directly on the shard. After all transactions for the given sharding key are complete, the application must return the connection to the pool and obtain a connection for another key.

If a matching connection is not available in the pool, then a new connection is created by forwarding the connection request with the sharding key to the shard director.

Once the pools are initialized and the shard topology cache is built based on all shards, a shard director outage has no impact on direct routing.

APIs Supporting Direct Routing

Oracle connection pools and drivers support Oracle Globally Distributed Database.

JDBC, UCP, OCI, and Oracle Data Provider for .NET (ODP.NET) recognize sharding keys as part of the connection check. Apache Tomcat, Websphere, and WebLogic leverage UCP support for sharding and PHP, Python, Perl, and Node.js leverage OCI support.

Oracle JDBC APIs

Oracle Java Database Connectivity (JDBC) provides APIs for connecting to database shards in an Oracle Globally Distributed Database configuration.

The JDBC driver recognizes the specified sharding key and super sharding key and connects to the relevant shard that contains the data. Once the connection is established to a shard, then any database operations, such as DMLs, SQL queries and so on, are supported and run in the usual way.

A shard-aware application gets a connection to a given shard by specifying the sharding key using the database sharding APIs.

- The `OracleShardingKey` interface indicates that the current object represents a sharding key that is to be used with a sharded database.
- The `OracleShardingKeyBuilder` interface builds the compound sharding key with subkeys of various supported data types. This interface uses the new JDK 8 builder pattern for building a sharding key.
- The `OracleConnectionBuilder` interface builds connection objects with additional parameters other than user name and password.
- The `OracleDataSource` class provides database sharding support with the `createConnectionBuilder` and `createShardingKeyBulider` methods.
- The `OracleXADataSource` class provides database sharding support with the `createConnectionBuilder` method
- The `OracleConnection` class provides database sharding support with the `setShardingKeyIfValid` and `setShardingKey` methods.
- The `OracleXAConnection` class provides database sharding support with the `setShardingKeyIfValid` and `setShardingKey` methods.

See the *Oracle Database JDBC Developer's Guide* for more information and examples.

Example 9-1 Sample Shard-Aware Application Code Using JDBC

The following code snippet shows how to use JDBC sharding APIs

```
OracleDataSource ods = new OracleDataSource();
   ods.setURL("jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(HOST=myhost)
(PORT=1521)(PROTOCOL=tcp))
(CONNECT_DATA=(SERVICE_NAME=myorcldbservicename)))");
   ods.setUser("hr");
   ods.setPassword("hr");

  // Employee name is the sharding Key in this example.
  // Build the Sharding Key using employee name as shown below.

   OracleShardingKey employeeNameShardKey = ods.createShardingKeyBuilder()
                                                .subkey("Mary",
JDBCType.VARCHAR)// First Name
                                                .subkey("Claire",
JDBCType.VARCHAR)// Last Name
                                                .build();

   OracleShardingKey locationSuperShardKey =
ods.createShardingKeyBuilder() // Building a super sharding key using
location as the key
                                                 .subkey("US",
JDBCType.VARCHAR)
                                                 .build();

   OracleConnection connection = ods.createConnectionBuilder()
                                    .shardingKey(employeeNameShardKey)
                                    .superShardingKey(locationSuperShardKey)
                                    .build();
```

Related Topics

- JDBC Support for Database Sharding in *Oracle Database JDBC Developer's Guide*

Oracle Call Interface

Oracle Call Interface (OCI) provides an interface for connecting to database shards in an Oracle Globally Distributed Database configuration.

To make requests that read from or write to a chunk, your application must be routed to the appropriate database (shard) that stores that chunk during the connection initiation step. This routing is accomplished by using a data key. The data key enables routing to the specific chunk by specifying its sharding key or to a group of chunks by specifying its super sharding key.

In order to get a connection to the correct shard containing the chunk you wish to operate on, you must specify a key in your application before getting a connection to a sharded Oracle database for either stand-alone connections or connections obtained from an OCI Session pool. For an OCI Session pool, you must specify a data key before you check out connections from the pool.

At a high-level, the following steps have to be followed to form sharding keys and shard group keys and get a session with an underlying connection:

1. Allocate the sharding key descriptor by calling `OCIDescriptorAlloc()` and specifying the descriptor type parameter as `OCI_DTYPE_SHARDING_KEY` to form the sharding key.
2. Allocate the shard group key descriptor by calling `OCIDescriptorAlloc()` and specifying the descriptor type parameter as `OCI_DTYPE_SHARDING_KEY` to form the shard group key.
3. Call `OCISessionGet()` using the initialized authentication handle from the previous step containing the sharding key and shard group key information to get the database connection to the shard and chunk specified by the sharding key and group of chunks as specified by the shard group key.

See *Oracle Call Interface Developer's Guide* for information about creating connections to OCI Session pools, stand-alone connections, and custom pool connections.

Related Topics

- OCI Interface for Using Shards in *Oracle Call Interface Developer's Guide*

Oracle Universal Connection Pool APIs

Oracle Universal Connection Pool (UCP) provides APIs for connecting to database shards in an Oracle Globally Distributed Database configuration.

A shard-aware application gets a connection to a given shard by specifying the sharding key using the enhanced sharding API calls `createShardingKeyBuilder` and `createConnectionBuilder`.

At a high-level, the following steps have to be followed in making an application work with a sharded database:

1. Update the URL to reflect the shard directors and global service.
2. Set the following pool parameters at the pool level and the shard level.
   - `setInitialPoolSize` sets the initial number of connections to be created when UCP is started
   - `setMinPoolSize` sets the minimum number of connections maintained by pool at runtime
   - `setMaxPoolSize` sets maximum number of connections allowed on connection pool
   - `setMaxConnectionsPerShard` sets max connections per shard
3. Build a sharding key object with `createShardingKeyBuilder`.
4. Establish a connection using `createConnectionBuilder`.
5. Run transactions within the scope of the given shard.

Example 9-2 Establishing a Connection Using UCP Sharding API

The following is a code fragment which illustrates how the sharding keys are built and connections established using UCP Sharding API calls.

```
...

PoolDataSource pds =
     PoolDataSourceFactory.getPoolDataSource();

  // Set Connection Pool properties
pds.setURL(DB_URL);
pds.setUser("hr");
pds.setPassword("****");
pds.setInitialPoolSize(10);
pds.setMinPoolSize(20);
pds.setMaxPoolSize(30);

// build the sharding key object

OracleShardingKey shardingKey =
    pds.createShardingKeyBuilder()
      .subkey("mary.smith@example.com", OracleType.VARCHAR2)
      .build();

  // Get an UCP connection for a shard
Connection conn =
    pds.createConnectionBuilder()
     .shardingKey(shardingKey)
     .build();
...
```

Example 9-3 Sample Shard-Aware Application Code Using UCP Connection Pool

In this example the pool settings are defined at the pool level and at the shard level.

```
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import oracle.jdbc.OracleShardingKey;
import oracle.jdbc.OracleType;
import oracle.jdbc.pool.OracleDataSource;
import oracle.ucp.jdbc.PoolDataSource;
import oracle.ucp.jdbc.PoolDataSourceFactory;

public class MaxConnPerShard
{
  public static void main(String[] args) throws SQLException
  {
    String url = "jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(HOST=shard-dir1)
(PORT=3216)
 (PROTOCOL=tcp))(CONNECT_DATA=(SERVICE_NAME=shsvc.shpool.oradbcloud)
(REGION=east)))";
```

```
    String user="testuser1", pwd = "testuser1";

    int maxPerShard = 100, initPoolSize = 20;

    PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();

pds.setConnectionFactoryClassName(OracleDataSource.class.getName());
    pds.setURL(url);
    pds.setUser(user);
    pds.setPassword(pwd);
    pds.setConnectionPoolName("testpool");
    pds.setInitialPoolSize(initPoolSize);

    // set max connection per shard
    pds.setMaxConnectionsPerShard(maxPerShard);
    System.out.println("Max-connections per shard is:
"+pds.getMaxConnectionsPerShard());

    // build the sharding key object
    int shardingKeyVal = 123;
    OracleShardingKey sdkey = pds.createShardingKeyBuilder()
        .subkey(shardingKeyVal, OracleType.NUMBER)
        .build();

    // try to build maxPerShard connections with the sharding key
    Connection[] conns = new Connection[maxPerShard];
    for (int i=0; i<maxPerShard; i++)
    {
      conns[i] = pds.createConnectionBuilder()
          .shardingKey(sdkey)
          .build();

Statement stmt = conns[i].createStatement();
      ResultSet rs = stmt.executeQuery("select sys_context('userenv',
'instance_name'),
       sys_context('userenv', 'chunk_id') from dual");
      while (rs.next()) {
        System.out.println((i+1)+" - inst:"+rs.getString(1)+",
chunk:"+rs.getString(2));
      }
      rs.close();
      stmt.close();
    }

    System.out.println("Try to build "+(maxPerShard+1)+"
connection ...");
    try {
      Connection conn = pds.createConnectionBuilder()
          .shardingKey(sdkey)
          .build();

      Statement stmt = conn.createStatement();
      ResultSet rs = stmt.executeQuery("select sys_context('userenv',
'instance_name'),
       sys_context('userenv', 'chunk_id') from dual");
```

```
      while (rs.next()) {
        System.out.println((maxPerShard+1)+" - inst:"+rs.getString(1)+",
         chunk:"+rs.getString(2));
      }
      rs.close();
      stmt.close();

      System.out.println("Problem!!! could not build connection as max-
connections per
        shard exceeded");
      conn.close();
    } catch (SQLException e) {
      System.out.println("Max-connections per shard met, could not build
connection
        any more, expected exception: "+e.getMessage());
    }
    for (int i=0; i<conns.length; i++)
    {
      conns[i].close();
    }
  }
}
```

Related Topics

- UCP APIs for Database Sharding Support in *Oracle Universal Connection Pool Developer's Guide*

Oracle Data Provider for .NET APIs

Oracle Data Provider for .NET (ODP.NET) provides APIs for connecting to database shards in an Oracle Globally Distributed Database configuration.

Using ODP.NET APIs, a shard-aware application gets a connection to a given shard by specifying the sharding key and super sharding key with APIs such as the `SetShardingKey(OracleShardingKey shardingKey, OracleShardingKey superShardingKey)` instance method in the `OracleConnection` class.

At a high level, the following steps are necessary for a .NET application to work with a sharded database:

1. Use ODP.NET, Unmanaged Driver.

   Sharding is supported with or without ODP.NET connection pooling. Each pool can maintain connections to different shards of the sharded database.

   > **Note:**
   >
   > Oracle Data Provider for .NET (ODP.NET), Unmanaged Driver is deprecated in Oracle Database 23c, and can be desupported in a future release. Oracle recommends existing unmanaged ODP.NET applications migrate to ODP.NET, Managed Driver.

2. Use an `OracleShardingKey` class to set the sharding key and another instance for the super sharding key.

3. Invoke the `OracleConnection.SetShardingKey()` method prior to calling `OracleConnection.Open()` so that ODP.NET can return a connection with the specified sharding key and super sharding key.

   These keys must be set while the `OracleConnection` is in a Closed state, otherwise an exception is thrown.

Example 9-4 Sample Shard-Aware Application Code Using ODP.NET

```
using System;
using Oracle.DataAccess.Client;

class Sharding
{
  static void Main()
  {
    OracleConnection con = new OracleConnection
      ("user id=hr;password=hr;Data Source=orcl;");
    //Setting a shard key
    OracleShardingKey shardingKey = new
OracleShardingKey(OracleDbType.Int32, 123);
    //Setting a second shard key value for a composite key
    shardingKey.SetShardingKey(OracleDbType.Varchar2, "gold");
    //Creating and setting the super shard key
    OracleShardingKey superShardingKey = new OracleShardingKey();
    superShardingKey.SetShardingKey(OracleDbType.Int32, 1000);

    //Setting super sharding key and sharding key on the connection
    con.SetShardingKey(shardingKey, superShardingKey);
    con.Open();

    //perform SQL query
  }
}
```

Related Topics

- Database Sharding in *Oracle Data Provider for .NET Developer's Guide.*

JDBC Sharding Data Source

Oracle Java Database Connectivity (JDBC) sharding data source enables Java connectivity to a sharded database without requiring the application to provide a sharding key.

Using the JDBC sharding data source, you do not need to identify and build the sharding key and the super sharding key to establish a connection. The sharding data source scales out to sharded databases transparently because it does not involve any change to the application code.

To use the JDBC sharding data source, set the connection property `oracle.jdbc.useShardingDriverConnection` to `true` as shown here.

```
Properties prop = new Properties();
prop.setProperty("oracle.jdbc.useShardingDriverConnection", "true");
```

The default value of `oracle.jdbc.useShardingDriverConnection` is `false`.

See the *Oracle Database JDBC Developer's Guide* for more information and examples.

Related Topics

- Overview of the Sharding Data Source in *Oracle Database JDBC Developer's Guide*

10

Migrating to an Oracle Globally Distributed Database

Migration from an existing non-sharded database to a sharded database consists of two phases: schema migration and data migration. Oracle Globally Distributed Database provides guidelines for migrating your existing database schema and data to a sharded database.

The following approaches are recommended for database migration.

- Migration with Oracle Data Pump
- Using External Tables to Load Data into a Sharded Database
- Using Oracle GoldenGate to Replicate Data Between Sharded and Non-Sharded Databases

Migration with Oracle Data Pump

Using the examples and guidelines provided in the following topics, you can extract DDL definitions and data from the source database with the Oracle Data Pump export utility, and then use the Data Pump import utility against the database export files to populate the target sharded database.

If you already created the schema for your sharded database, you can go directly to the data migration topic.

Schema Migration

Transition from a non-sharded database to a sharded database requires some schema changes. At a minimum, the keyword `SHARDED` or `DUPLICATED` should be added to `CREATE TABLE` statements. In some cases, the partitioning of tables should be changed as well, or a column with the shading key added.

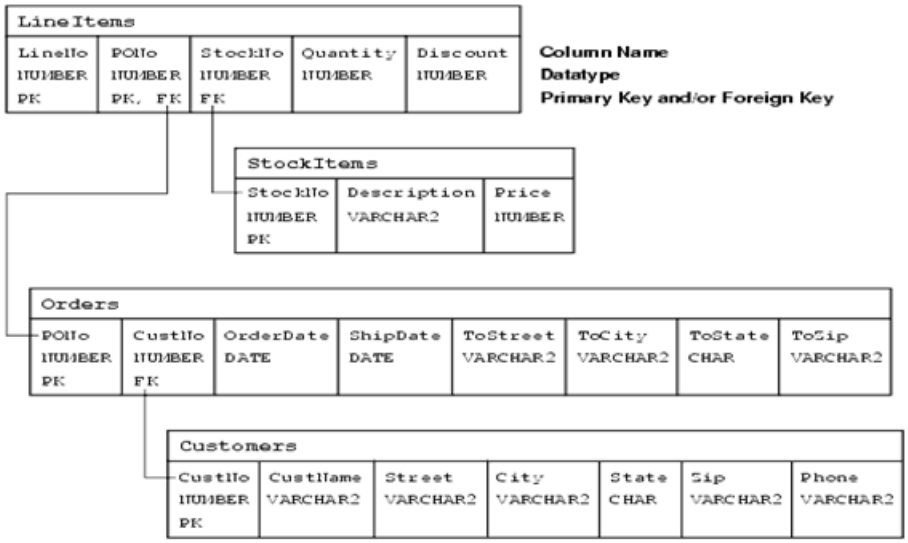
To properly design the sharded database schema, you must analyze the schema and workload of the non-sharded database and make the following decisions.

- Which tables should be sharded and which should be duplicated
- What are the parent-child relationships between the sharded tables in the table family
- Which sharding method is used on the sharded tables
- What to use as the sharding key

If these decisions are not straightforward, you can use the Sharding Advisor to help you to make them. Sharding Advisor is a tool that you run against a non-sharded Oracle Database that you are considering to migrate to an Oracle Globally Distributed Database environment.

To illustrate schema and data migration from a non-sharded to sharded database, we will use a sample data model shown in the following figure.

Figure 10-1 Schema Migration Example Data Model



The data model consists of four tables, Customers, Orders, StockItems, and LineItems, and the data model enforces the following primary key constraints.

- `Customer.(CustNo)`
- `Orders.(PONo)`
- `StockItems.(StockNo)`
- `LineItems.(LineNo, PONo)`

The data model defines the following referential integrity constraints.

- `Customers.CustNo -> Orders.CustNo`
- `Orders.PONo -> LineItems.PONo`
- `StockItems.StockNo -> LineItems.StockNo`

The following DDL statements create the example non-sharded schema definitions.

```
CREATE TABLE Customers (
 CustNo     NUMBER(3) NOT NULL,
 CusName    VARCHAR2(30) NOT NULL,
 Street     VARCHAR2(20) NOT NULL,
 City       VARCHAR2(20) NOT NULL,
 State      CHAR(2) NOT NULL,
 Zip        VARCHAR2(10) NOT NULL,
 Phone      VARCHAR2(12),
 PRIMARY KEY (CustNo)
);

CREATE TABLE Orders (
 PoNo       NUMBER(5),
 CustNo     NUMBER(3) REFERENCES Customers,
 OrderDate  DATE,
 ShipDate   DATE,
```

```
 ToStreet   VARCHAR2(20),
 ToCity     VARCHAR2(20),
 ToState    CHAR(2),
 ToZip      VARCHAR2(10),
 PRIMARY KEY (PoNo)
);

CREATE TABLE LineItems (
 LineNo     NUMBER(2),
 PoNo       NUMBER(5) REFERENCES Orders,
 StockNo    NUMBER(4) REFERENCES StockItems,
 Quantity   NUMBER(2),
 Discount   NUMBER(4,2),
 PRIMARY KEY (LineNo, PoNo)
);

CREATE TABLE StockItems (
 StockNo    NUMBER(4) PRIMARY KEY,
 Description VARCHAR2(20),
 Price      NUMBER(6,2)
);
```

Migrating the Sample Schema

As an example, to migrate the sample schema described above to a sharded database, do the following steps.

1. Get access to the source database export directory.

   The database administrator has to authorize the database user for required access to the database export directory, as shown here.

   ```
   CREATE OR REPLACE DIRECTORY expdir AS '/some/directory';
   GRANT READ, WRITE ON DIRECTORY expdir TO uname;
   GRANT EXP_FULL_DATABASE TO uname;
   ```

   With a full database export, the database administrator must grant you the `EXP_FULL_DATABASE` role, `uname`. No additional role is required for a table level export.

2. Extract the DDL definitions from the source database.

   A convenient way to extract the DDL statements is to create a Data Pump extract file. You can export only metadata, or only a part of the schema containing the set of tables you are interested in migrating, as shown in this example.

   ```
   expdp uname/pwd directory=EXPDIR dumpfile=sample_mdt.dmp
   logfile=sample_mdt.log INCLUDE=TABLE:\"IN \( \'CUSTOMERS\', \'ORDERS\',
   \'STOCKITEMS\', \'LINEITEMS\' \) \" CONTENT=METADATA_ONLY
   FLASHBACK_TIME=SYSTIMESTAMP
   ```

   Then, use the Data Pump import utility against this database export file.

   ```
   impdp uname/pwd@orignode directory=expdir dumpfile=sample_mdt.dmp
   sqlfile=sample_ddl.sql
   ```

In this example, the `impdp` command does not actually perform an import of the contents of the dump file. Rather, the `sqlfile` parameter triggers the creation of a script named `sample_ddl.sql` which contains all of the DDL from within the export dump file.

Trimming down the export in this way more efficiently captures a consistent image of the database metadata without a possibly lengthy database data dump process. You still must get the DDL statements in text format to perform the DDL modifications required by your sharded database schema design.

3. Modify the extracted DDL statements for the sharded database.

   For the sample schema shown above, the corresponding DDL statements for the sharded database may look like the following. This is an example with system-managed sharding.

```
CREATE SHARDED TABLE Customers (
 CustNo     NUMBER(3) NOT NULL,
 CusName    VARCHAR2(30) NOT NULL,
 Street     VARCHAR2(20) NOT NULL,
 City       VARCHAR2(20) NOT NULL,
 State      CHAR(2) NOT NULL,
 Zip        VARCHAR2(10) NOT NULL,
 Phone      VARCHAR2(12),
 CONSTRAINT RootPK PRIMARY KEY (CustNo)
)
PARTITION BY CONSISTENT HASH (CustNo)
PARTITIONS AUTO
TABLESPACE SET ts1
;

CREATE SHARDED TABLE Orders (
 PoNo       NUMBER(5) NOT NULL,
 CustNo     NUMBER(3) NOT NULL,
 OrderDate  DATE,
 ShipDate   DATE,
 ToStreet   VARCHAR2(20),
 ToCity     VARCHAR2(20),
 ToState    CHAR(2),
 ToZip      VARCHAR2(10),
 CONSTRAINT OrderPK PRIMARY KEY (CustNo, PoNo),
 CONSTRAINT CustFK Foreign Key (CustNo) REFERENCES Customers
(CustNo)
)
PARTITION BY REFERENCE (CustFK)
;
CREATE SHARDED TABLE LineItems (
 LineNo      NUMBER(2) NOT NULL,
 PoNo        NUMBER(5) NOT NULL,
 CustNo      NUMBER(3) NOT NULL,
 StockNo     NUMBER(4) NOT NULL,
 Quantity    NUMBER(2),
 Discount    NUMBER(4,2),
 CONSTRAINT LinePK PRIMARY KEY (CustNo, LineNo, PoNo),
 CONSTRAINT LineFK FOREIGN KEY (CustNo, PoNo) REFERENCES Orders
(CustNo, PoNo)
```

```
)
PARTITION BY REFERENCE (LineFK)
;

CREATE DUPLICATED TABLE StockItems (
 StockNo     NUMBER(4) PRIMARY KEY,
 Description VARCHAR2(20),
 Price       NUMBER(6,2)
);
```

Here are some observations about the schema of the sharded database.

- Customers-Orders-LineItems form a table family of `SHARDED` tables, with Customers as the root table and child tables are partitioned by reference. StockItems is a `DUPLICATED` table.
- `CustNo` is chosen as the sharding key. Hence, this column must be included in all the tables of the table family. Note that in the non-sharded database, the LineItems table did not have a `CustNo` column, but it was included in the sharded version on the table. The sharding key column also needs to be present in all primary and foreign key constraints in sharded tables.
- StockItems is now a duplicated table. The primary copy of a duplicated table resides on the shard catalog database. Thus, the foreign key constraint in the LineItems table referencing StockItems table cannot be enforced and is removed.

4. Run the modified DDLs against the target database.

   Connect to the shard catalog database and run

   ```
   ALTER SESSION ENABLE SHARD DDL;
   ```

   Then run the DDLs listed above to create the sharded and duplicated tables.

   It is recommended that you validate the sharding configuration using the `GDSCTL VALIDATE` command, before loading the data.

   ```
   gdsctl> validate
   ```

   If you see inconsistencies or errors, you must correct the problem using the `GDSCTL` commands `SHOW DDL` and `RECOVER`. After successful validation, the sharded database is ready for data loading.

Migrating Data to a Sharded Database

Transitioning from a non-sharded database to a sharded database involves moving the data from non-sharded tables in the source database to sharded and duplicated tables in the target database.

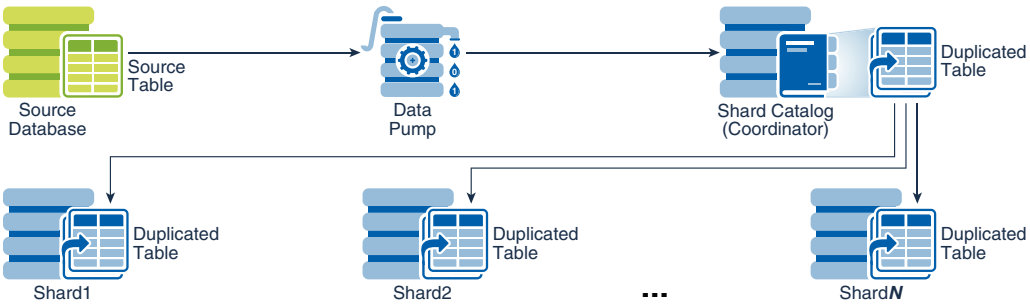
Moving data from non-sharded tables to duplicated tables is straightforward, but moving data from non-sharded tables to sharded tables requires special attention.

Loading Data into Duplicated Tables

You can load data into a duplicated table using any existing database tools, such as Data Pump, SQL Loader, or plain SQL. The data must be loaded to the shard catalog database. Then it gets automatically replicated to all shards.

Because the contents of the duplicated table is fully replicated to the database shards using materialized views, loading a duplicated table may take longer than loading the same data into a regular table.

Figure 10-2 Loading Duplicated Tables



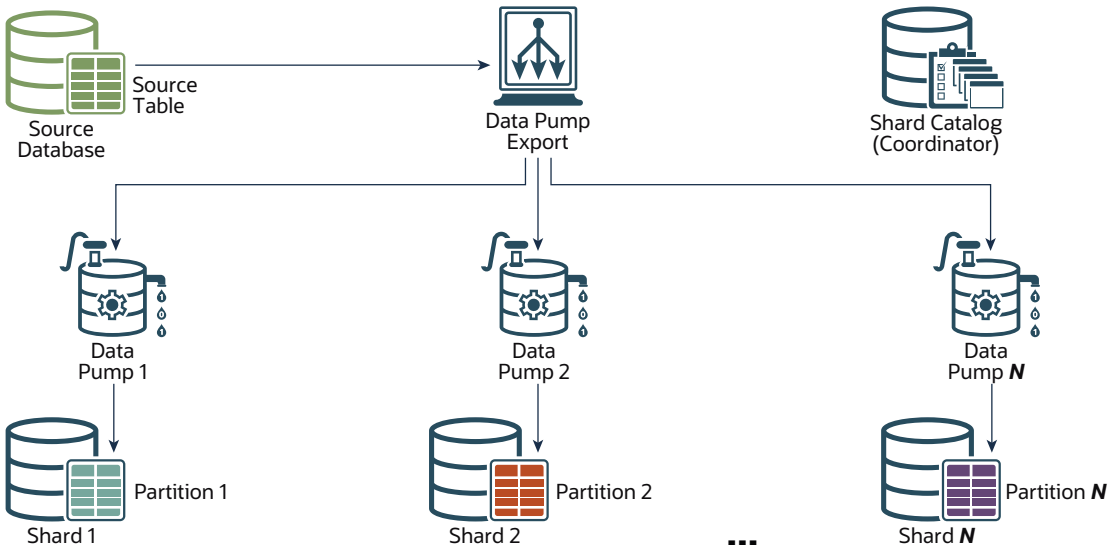
Loading Data into Sharded Tables

When loading a sharded table, each database shard accommodates a distinct subset of the data set, so the data in each table must be split (partitioned) across shards during the load.

You can use the Oracle Data Pump utility to load the data across database shards in subsets. Data from the source database can be exported into a Data Pump dump file. Then Data Pump import can be run on each shard concurrently by using the same dump file.

The dump file can be either placed on shared storage accessible to all shards, or copied to the local storage of each shard. When importing to individual shards, Data Pump import ignores the rows that do not belong to the current shard.

Figure 10-3 Loading Sharded Tables Directly to the Database Shards



Loading the data directly into the shards is much faster, because all shards are loaded in parallel. It also provides linear scalability; the more shards there are in the sharded database, the higher data ingestion rate is achieved.

Oracle Data Pump Support for Sharding Metadata

Oracle Data Pump supports migration to sharded databases with support for sharded DDL. You can migrate sharded database objects to a target database based on source database shard objects.

Oracle Data Pump provides support for sharding DDL in the API `dbms_metadata.get_ddl()`. A transform parameter, `INCLUDE_SHARDING_CLAUSES`, facilitates this support. If this parameter is set to `true`, and the underlying object contains it, then the `get_ddl()` API returns sharding DDL for `create table`, `sequence`, `tablespace` and `tablespace set`. To prevent sharding attributes from being set on import, the default value for `INCLUDE_SHARDING_CLAUSES` is set to `false`.

See *Oracle Database Utilities* topics TRANSFORM and Placing Conditions on Transforms, and *Oracle Database PL/SQL Packages and Types Reference* topic SET_TRANSFORM_PARAM and SET_REMAP_PARAM Procedures for details, examples, and reference.

Loading the Sample Schema Data

As an example, the following steps illustrate how to move the sample schema data from a non-sharded to sharded database. The syntax examples are based on the sample Customers-Orders-LineItems-StockItems schema introduced in the previous topics.

1. Export the data from your database tables.

```
expdp uname/pwd@non_sharded_db directory=expdir
dumpfile=original_tables.dmp logfile=original_tables.log SCHEMAS=UNAME
```

```
INCLUDE=TABLE:\"IN \( \'CUSTOMERS\', \'ORDERS\', \'STOCKITEMS\' )
\" FLASHBACK_TIME=SYSTIMESTAMP CONTENT=DATA_ONLY
```

If the source table (in the non-sharded database) is partitioned, then export to dump files in non-partitioned format (`data_options=group_partition_table_data`).

Example, if the Orders table is a partitioned table on the source database, export it as follows.

```
$ cat ordexp.par
directory=expdir
logfile=ordexp.log
dumpfile=ord_%U.dmp
tables=ORDERS
parallel=8
COMPRESSION=ALL
content=data_only
DATA_OPTIONS=GROUP_PARTITION_TABLE_DATA

$ expdp user/password parfile=ordexp.par
```

Because the `SHARDED` and `DUPLICATED` tables were already created in the target database, you only export the table content (`DATA_ONLY`).

Data Pump export utility files are consistent on a per table basis. If you want all of the tables in the export to be consistent at the same point in time, you must use the `FLASHBACK_SCN` or `FLASHBACK_TIME` parameters as shown in the example above. Having a consistent “as of” point in time database export files is recommended.

2. Make the export file (`original_tables.dmp`) accessible by the target database nodes before you start importing the data to the sharded database.

   You can either move this file (or multiple files in the case of parallel export) to the target database system or share the file over the network.

3. Prepare all the target databases (shard catalog and shards) for import.

   The database administrator has to authorize the database user for required access to the database import directory, as shown here.

   ```
   CREATE OR REPLACE DIRECTORY expdir AS '/some/directory';
   GRANT READ, WRITE ON DIRECTORY expdir TO uname;
   GRANT IMP_FULL_DATABASE TO uname;
   ```

4. Load the `DUPLICATED` table (StockItems) using the shard catalog.

   The following is an example of the import command.

   ```
   impdp uname/pwd@catnode:1521/ctlg directory=expdir
   dumpfile=original_tables.dmp logfile=imp_dup.log tables=StockItems
   content=DATA_ONLY
   ```

5. Load the `SHARDED` tables on the shards directly.

The best way to load the exported `SHARDED` tables (Customers, Orders) is to run the Data Pump on each shard (shrd1,2,…, N) directly. The following is an example of the import command on the first shard.

```
impdp uname/pwd@shrdnode:1521/shrd1 directory=expdir
DUMPFILE=original_tables.dmp LOGFILE=imp_shd1.log TABLES=”Customers,
Orders, LineItems” CONTENT=DATA_ONLY
```

Repeat this step on all of the other shards. Note that the same dump file (`original_tables.dmp`) is used to load data for all of the shards. Data Pump import will ignore rows that do not belong to the current shard. This operation can be run in parallel on all shards.

To benefit from fast loading into very large partitioned tables with parallelism, the data pump parameter `DATA_OPTIONS` should include the value `_FORCE_PARALLEL_DML` (requires patch 31891464).

```
$ cat ordimp.par
directory=expdir
logfile=ordimp.log
dumpfile=ord_%U.dmp
tables=ORDERS
parallel=8
content=data_only
DATA_OPTIONS=_force_parallel_dml
$ impdp user/password parfile=ordimp.par
```

Without patch 31891464, you can alternatively migrate data using an external table of type `DATA PUMP`, as shown in the following example.

a. Export on the source database.

```
CREATE TABLE ORDERS_EXT
 ORGANIZATION EXTERNAL
    ( TYPE ORACLE_DATAPUMP
      DEFAULT DIRECTORY "expdir"
      ACCESS PARAMETERS ( DEBUG = (3 , 33489664))
      LOCATION ('ord1.dat',
                'ord2.dat',
                'ord3.dat',
                'ord4.dat')
    )
PARALLEL 8
REJECT LIMIT UNLIMITED
AS SELECT * FROM user.ORDERS;
```

b. Import into each target shard.

```
CREATE TABLE ORDERS_EXT
 ORGANIZATION EXTERNAL
    ( TYPE ORACLE_DATAPUMP
      DEFAULT DIRECTORY "expdir"
      ACCESS PARAMETERS ( DEBUG = (3 , 33489664))
      LOCATION ('ord1.dat',
```

```
                'ord2.dat',
                'ord3.dat',
                'ord4.dat')
    )
PARALLEL 8
REJECT LIMIT UNLIMITED
;
INSERT /*+ APPEND ENABLE_PARALLEL_DML PARALLEL(a,12)
pq_distribute(a, random) */ INTO "user"."ORDERS" a
SELECT /*+ full(b) parallel(b,12) pq_distribute(b, random)*/
*
FROM "ORDERS_EXT"
WHERE <predicate*>;
Commit;
```

(*) The predicate in the `WHERE` clause depends on the sharding method. For user-defined sharding by range, for example, it will be based on the range of CustNo on a particular shard. For system-managed (consistent hash-based) sharding, see the use case in Using External Tables to Load Data into a Sharded Database.

Note:

You can make Data Pump run faster by using the `PARALLEL` parameter in the `expdp` and `impdp` commands. For export, this parameter should be used in conjunction with the %U wild card in the `DUMPFILE` parameter to allow multiple dump files be created, as shown in this example.

```
expdp uname/pwd@orignode SCHEMAS=uname directory=expdir
dumpfile=samp_%U.dmp logfile=samp.log
FLASHBACK_TIME=SYSTIMESTAMP PARALLEL=4
```

The above command uses four parallel workers and creates four dump files with suffixes _01, _02, _03, and _04. The same wild card can be used during the import to allow you to reference multiple input files.

Migrating Data Without a Sharding Key

As an example, the following steps illustrate how to migrate data to a sharded table from a source table that does not contain the sharding key.

The examples of the Data Pump export and import commands in the previous topic do not include the LineItems table. The reason is that this table in the non-sharded database does not contain the sharding key column (CustNo). However, this column is required in the sharded version of the table.

Because of the schema mismatch between the non-sharded and sharded versions of the table, data migration for LineItems must be handled differently, as shown in the following steps.

1. On the source, non-sharded, database, create a temporary view with the missing column and SQL expression to generate value for this column.

```
CREATE OR REPLACE VIEW Lineitems_View AS
  SELECT l.*,
         (SELECT o.CustNo From Orders o WHERE l.PoNo=o.PoNo) CustNo
FROM LineItems l;
```

   This creates a view `LineItems_View` with the column `CustNo` populated based on the foreign key relationship with the Orders table.

2. Export the new view with VIEWS_AS_TABLES option of the data pump export utility.

```
expdp uname/pwd@non_sharded_db directory=expdir
DUMPFILE=original_tables_vat.dmp LOGFILE=original_tables_vat.log
FLASHBACK_TIME=SYSTIMESTAMP CONTENT=DATA_ONLY
TABLES=Uname.Customers,Uname.Orders,Uname.StockItems
VIEWS_AS_TABLES=Uname.LineItems_
```

3. Import the data to sharded tables by directly running the data pump import on individual shards (shrd1, shrd2,.., shrdN).

   The following is an example of running the import on the first shard.

```
impdp uname/pwd@shrdnode:1521/shrd1 directory=expdir
DUMPFILE=original_tables_vat.dmp LOGFILE=imp_shd_vat1.log
CONTENT=DATA_ONLY
TABLES=Uname.Customers,Uname.Orders,Uname.LineItems_View
VIEWS_AS_TABLES=Uname.LineItems_View REMAP_TABLE=Lineitems_View:Lineitems
```

   The examples uses the `impdp` tool `VIEWS_AS_TABLES` option to import the view `LineItems_View` exported as a table during export operation. And the parameter `REMAP_TABLE` is used to indicate that this data should actually be inserted in the original table LineItems.

Using External Tables to Load Data into a Sharded Database

Using the examples and guidelines in the following topics, you can load data into a sharded database by creating external tables and then loading the data from the external tables into sharded or duplicated tables.

This data loading method is useful when the data to be loaded resides in external files, for example in CSV files.

External tables can be defined using the `ORGANIZATION EXTERNAL` keyword in the `CREATE TABLE` statement. This table must be local to each shard and not sharded or duplicated. Loading the data into the sharded or duplicated table involves a simple `INSERT … SELECT` statement from an external table, with a condition to filter only a subset of data for sharded tables.

You may choose to keep the files on different hosts based on the access time and size of the files. For example, copy the files for duplicated tables on the shard catalog host and keep files for sharded tables on a network share that is accessible to all of the shards. It is also possible to keep a copy of the sharded table files on each shard for faster loading.

For more information about external tables, see External Tables in *Oracle Database Utilities*.

Loading Data into Duplicated Tables

Data for the duplicated tables resides on the shard catalog, so loading the data into the duplicated tables is also done on the shard catalog. The data is then automatically replicated to shards after loading is complete.

Consider the following table defined as a duplicated table.

```
CREATE DUPLICATED TABLE StockItems (
 StockNo     NUMBER(4) PRIMARY KEY,
 Description VARCHAR2(20),
 Price       NUMBER(6,2)
);
```

Loading data into the table StockItems involves the following steps.

1. Create a directory object pointing to the directory containing the data file and grant access to the shard user on this directory.

   ```
   CREATE OR REPLACE DIRECTORY shard_dir AS '/path/to/datafile';
   GRANT ALL on DIRECTORY shard_dir TO uname;
   ```

2. Create an external table that is local to the shard catalog, with the same columns as the duplicated table.

   On the shard catalog, run:

   ```
   ALTER SESSION DISABLE SHARD DDL;
   CREATE TABLE StockItems_Ext (
    StockNo     NUMBER(4) NOT NULL,
    Description VARCHAR2(20),
    Price       NUMBER(6,2)
   )
   ORGANIZATION EXTERNAL
   (TYPE ORACLE_LOADER DEFAULT DIRECTORY shard_dir
       ACCESS PARAMETERS
           (FIELDS TERMINATED BY '|' (
             StockNo,
             Description,
             Price)
       )LOCATION ('StockItems.dat')
    );
   ```

   In this example, the data file for the duplicated table is named `StockItems.dat` and column values are separated by the character '|'.

3. Insert data from the external table into the duplicated table.

   ```
   INSERT INTO StockItems  (SELECT * FROM StockItems_Ext);
   ```

You can use also optimizer hints such as APPEND and PARALLEL (with degree of parallelism) for faster loading depending on your system resources. For example:

```
ALTER SESSION ENABLE PARALLEL DML;
INSERT /*+ APPEND PARALLEL */ INTO StockItems
  (SELECT * FROM StockItems_Ext);
```

or

```
ALTER SESSION ENABLE PARALLEL DML;
INSERT /*+ APPEND PARALLEL(24) */ INTO StockItems
  (SELECT * FROM StockItems_Ext);
```

4. Commit the insert operation.

```
COMMIT;
```

5. Drop the external table.

```
DROP TABLE StockItems_Ext;
```

Repeat these steps for each duplicated table.

Loading Data into Sharded Tables

Loading data into a sharded table needs to be performed on individual shards because data for a sharded table is partitioned across shards. The load can be done concurrently on all the shards, even if the source data file is shared.

The process of loading is similar to the loading of duplicated tables, with an additional filter in the `INSERT … SELECT` statement to filter out the rows that do not belong to the current shard.

As an example, consider the sharded table created as follows.

```
CREATE SHARDED TABLE Customers (
 CustNo     NUMBER(3) NOT NULL,
 CusName    VARCHAR2(30) NOT NULL,
 Street     VARCHAR2(20) NOT NULL,
 City       VARCHAR2(20) NOT NULL,
 State      CHAR(2) NOT NULL,
 Zip        VARCHAR2(10) NOT NULL,
 Phone      VARCHAR2(12),
 CONSTRAINT RootPK PRIMARY KEY (CustNo)
)
PARTITION BY CONSISTENT HASH (CustNo)
PARTITIONS AUTO
TABLESPACE SET ts1
;
```

Loading data into this table involves doing the following steps **on each shard**.

1. Create the directory object in the same way as done for the duplicated tables.

2. Create an external table for Customers table.

```
ALTER SESSION DISABLE SHARD DDL;
CREATE TABLE Customers_Ext (
 CustNo     NUMBER(3) NOT NULL,
 CusName    VARCHAR2(30) NOT NULL,
 Street     VARCHAR2(20) NOT NULL,
 City       VARCHAR2(20) NOT NULL,
 State      CHAR(2) NOT NULL,
 Zip        VARCHAR2(10) NOT NULL,
 Phone      VARCHAR2(12)
)
ORGANIZATION EXTERNAL
(TYPE ORACLE_LOADER DEFAULT DIRECTORY shard_dir
    ACCESS PARAMETERS
    (FIELDS TERMINATED BY '|' (
      CustNo, CusName, Street, City, State, Zip, Phone)
    )LOCATION ('Customers.dat')
 );
```

3. Insert data from external table into sharded table.

```
ALTER SESSION ENABLE PARALLEL DML;

INSERT /*+ APPEND PARALLEL(24) */ INTO Customers
 (SELECT * FROM Customers_Ext WHERE
        SHARD_CHUNK_ID('UNAME.CUSTOMERS', CUSTNO) IS NOT NULL
  );
```

The operator `SHARD_CHUNK_ID` is used to filter the rows that belong to the current shard. This operator returns a valid chunk number for the given sharding key value. The parameters for this operator are the root table name (in this case `UNAME.CUSTOMERS`) and values of the sharding key columns. When a value does not belong to the current shard, this operator returns `NULL`.

Note that this operator is introduced in the current release (Oracle Database 21c). If this operator is not available in your version, you must modify the insert statement as follows for the case of system-managed sharding.

```
INSERT /*+ APPEND PARALLEL(24) */ INTO Customers c
 (SELECT * FROM Customers_Ext WHERE
        EXISTS (SELECT chunk_number FROM gsmadmin_internal.chunks
            WHERE ora_hash(c.CustNo)>= low_key
              AND ora_hash c.CustNo)< high_key)
  );
```

This query user internal sharding metadata to decide the eligibility for the row to be inserted.

4. Commit the insert operation.

```
COMMIT;
```

5. Drop external tables.

```
DROP TABLE Customers_Ext;
```

Repeat the above steps for each sharded table, starting with the root table and descending down the table family hierarchy to maintain any foreign key constraints.

Using Oracle GoldenGate to Replicate Data Between Sharded and Non-Sharded Databases

You can migrate data from a non-sharded database to a sharded database using Oracle GoldenGate.

Migrating data from a non-sharded database to sharded database using Oracle GoldenGate is done in two phases.

Extraction on source database

- All of the tables from the source database are extracted using single extract on the source database.

Replication on target database

- The replication into sharded tables is performed on the shards and the replication into duplicated tables is performed on the shard catalog.

Oracle GoldenGate Replication Prerequisites

Make sure your source and target databases, and Oracle GoldenGate environments meet these prerequisites before attempting sharded database Oracle GoldenGate replication.

Assumptions

1. It is assumed that the tables to be migrated from the non-sharded to sharded database have already been classified into sharded and duplicated tables.
2. The sharding keys for all of the tables to be migrated to sharded tables have already been identified.
3. Sharded and duplicated tables have been pre-created in the target sharded database.
4. Oracle GoldenGate software is already installed on the source and target systems.

Source and Target Databases

- Oracle Database version: 19c (19.15.0.0.0) or later in a Multitenant architecture
- Target database sharding type: System-managed

Oracle GoldenGate Configuration

Oracle GoldenGate Version: 19c Classic Architecture in a hub configuration

Replicating Data from a Non-Sharded Database to a Sharded Database

Example environment

The examples in the steps below use the following topology

System/Object	Source Environment	Target Environment
CDB Name	srccdb	sdbcdb
PDB Name	srcpdb	Shards: sdbpdb1,sdbpdb2,sdbpdb3 Shard catalog: scpdb
Application Schema	app_schema	app_schema
Sharded Tables	Customers, Orders, LineItems	Customers, Orders, LineItems
Duplicated Tables	Products	Products

High Level Steps

1) Create an extract on the source database to capture transactions from the source tables and start it.

2) Capture data from source database for initial load using `expdp` and `flashback_scn`.

3) Perform initial load into the sharded tables on target shards using `impdp`.

4) Perform initial load into the duplicated tables on the target shard catalog using `impdp`.

5) Create the same number of replicats as number of target shards to replicate sharded tables.

6) Create one replicat for the shard catalog to replicate duplicated tables.

7) Start replicats on the target shards using `at csn`

8) Start replicat on the shard catalog using `at csn`

9) Validate the data replication from the source to target tables.

1. Configure the source (non-sharded) database
   a. Create an extract on the source database to capture transactions from source tables and start it.

```
$ ggsci

GGSCI > dblogin useridalias ggadmin
GGSCI > add extract extnshd, integrated tranlog, begin now
GGSCI > register extract extnshd, database container (SRCPDB)
GGSCI > add exttrail ./dirdat/tr, extract extnshd

Add the following parameters in extract parameter file

GGSCI > edit params extnshd

extract extnshd
useridalias  ggadmin
TranlogOptions IntegratedParams (max_sga_size 256)
extTrail ./dirdat/tr
DiscardFile ./dirrpt/extnshd.dsc, Append Megabytes 50
REPORTCOUNT EVERY 2 HOURS, RATE
```

```
Table SRCPDB.app_schema.customers;
Table SRCPDB.app_schema.orders;
Table SRCPDB.app_schema.lineitems;
Table SRCPDB.app_schema.products;

GGSCI> start extract extnshd
```

b. Capture data from the source database for initial load using `expdp`.

```
SQL> select current_scn from v$database;

$ expdp app_schema/xxxxx@SRCPDB
flashback_scn=current_scn_from_previous_step
 directory=DATA_PUMP_DIR dumpfile=app_schema_exp.dmp
 logfile=app_schema_exp.log
```

2. Configure the target (sharded) database.

a. Perform the initial load on the target shard databases and shard catalog using `impdp`.

```
Import into shards
$ impdp app_schema/xxxxx@SDBPDB1 directory=DATA_PUMP_DIR
dumpfile=app_schema_exp.dmp
 logfile=app_schema_imp.log tables=CUSTOMERS,ORDERS,LINEITEMS,
CONTENT=DATA_ONLY
$ impdp app_schema/xxxxx@SDBPDB2 directory=DATA_PUMP_DIR
dumpfile=app_schema_exp.dmp
 logfile=app_schema_imp.log tables=CUSTOMERS,ORDERS,LINEITEMS,
CONTENT=DATA_ONLY
$ impdp app_schema/xxxxx@SDBPDB3 directory=DATA_PUMP_DIR
dumpfile=app_schema_exp.dmp
 logfile=app_schema_imp.log tables=CUSTOMERS,ORDERS,LINEITEMS,
CONTENT=DATA_ONLY

Import into shard catalog
$ impdp app_schema/xxxxx@SCPDB directory=DATA_PUMP_DIR
dumpfile=app_schema_exp.dmp
 logfile=app_schema_imp.log tables=PRODUCTS CONTENT=DATA_ONLY
```

b. Create replicats (same as the number of shards) on the target database.

```
Replicat for sharded tables on Shard 1
======================================
GGSCI > dblogin useridalias ggadmin_shd1
GGSCI > add replicat repshd1, INTEGRATED, exttrail ./dirdat/tr
 CHECKPOINTTABLE ggadmin.GGCHKPT

Add the following parameters in replicat for shard1

GGSCI > edit params repshd1

replicat repshd1
useridalias ggadmin_shd1
HANDLECOLLISIONS
SOURCECATALOG SDBPDB1
```

```
MAP NSHDPDB.APP_SCHEMA.CUSTOMERS , target APP_SCHEMA.CUSTOMERS, &
SQLEXEC (ID chunklookup1, QUERY 'select count(*) count FROM
gsmadmin_internal.chunks
 WHERE ora_hash(:CODE_IN_PARAM) >= low_key and
ora_hash(:CODE_IN_PARAM) < high_key',&
PARAMS (CODE_IN_PARAM = CUSTID),
BEFOREFILTER), &
FILTER (chunklookup1.COUNT = 1);

MAP NSHDPDB.APP_SCHEMA.ORDERS, target APP_SCHEMA.ORDERS, &
SQLEXEC (ID chunklookup2, QUERY 'select count(*) count FROM
gsmadmin_internal.chunks
 WHERE ora_hash(:CODE_IN_PARAM) >= low_key and
ora_hash(:CODE_IN_PARAM) < high_key',&
PARAMS (CODE_IN_PARAM = CUSTID),
BEFOREFILTER), &
FILTER (chunklookup2.COUNT = 1);

MAP NSHDPDB.APP_SCHEMA.LINEITEMS, target APP_SCHEMA.LINEITEMS, &
SQLEXEC (ID chunklookup3, QUERY 'select count(*) count FROM
gsmadmin_internal.chunks
 WHERE ora_hash(:CODE_IN_PARAM) >= low_key and
ora_hash(:CODE_IN_PARAM) < high_key',&
PARAMS (CODE_IN_PARAM = CUSTID),
BEFOREFILTER), &
FILTER (chunklookup3.COUNT = 1);

Replicat for sharded tables on Shard 2
======================================

GGSCI > dblogin useridalias ggadmin_shd2
GGSCI > add replicat repshd2, INTEGRATED, exttrail ./dirdat/tr
 CHECKPOINTTABLE ggadmin.GGCHKPT

Add the following parameters in replicat for shard2

GGSCI > edit params repshd2

replicat repshd2
useridalias ggadmin_shd2
HANDLECOLLISIONS
SOURCECATALOG SDBPDB2
MAP NSHDPDB.APP_SCHEMA.CUSTOMERS , target APP_SCHEMA.CUSTOMERS, &
SQLEXEC (ID chunklookup1, QUERY 'select count(*) count FROM
gsmadmin_internal.chunks
 WHERE ora_hash(:CODE_IN_PARAM) >= low_key and
ora_hash(:CODE_IN_PARAM) < high_key',&
PARAMS (CODE_IN_PARAM = CUSTID),
BEFOREFILTER), &
FILTER (chunklookup1.COUNT = 1);

MAP NSHDPDB.APP_SCHEMA.ORDERS, target APP_SCHEMA.ORDERS, &
SQLEXEC (ID chunklookup2, QUERY 'select count(*) count FROM
gsmadmin_internal.chunks
 WHERE ora_hash(:CODE_IN_PARAM) >= low_key and
```

```
ora_hash(:CODE_IN_PARAM) < high_key',&
PARAMS (CODE_IN_PARAM = CUSTID),
BEFOREFILTER), &
FILTER (chunklookup2.COUNT = 1);


MAP NSHDPDB.APP_SCHEMA.LINEITEMS, target APP_SCHEMA.LINEITEMS, &
SQLEXEC (ID chunklookup3, QUERY 'select count(*) count FROM
gsmadmin_internal.chunks
 WHERE ora_hash(:CODE_IN_PARAM) >= low_key and
ora_hash(:CODE_IN_PARAM) < high_key',&
PARAMS (CODE_IN_PARAM = CUSTID),
BEFOREFILTER), &
FILTER (chunklookup3.COUNT = 1);

Replicat for sharded tables on Shard 3
======================================

GGSCI > dblogin useridalias ggadmin_shd3
GGSCI > add replicat repshd3, INTEGRATED, exttrail ./dirdat/tr
 CHECKPOINTTABLE ggadmin.GGCHKPT

Add the following parameters in replicat for shard3

GGSCI > edit params repshd3

replicat repshd3
useridalias ggadmin_shd3
HANDLECOLLISIONS
SOURCECATALOG SDBPDB3
MAP NSHDPDB.APP_SCHEMA.CUSTOMERS , target APP_SCHEMA.CUSTOMERS, &
SQLEXEC (ID chunklookup1, QUERY 'select count(*) count FROM
gsmadmin_internal.chunks
 WHERE ora_hash(:CODE_IN_PARAM) >= low_key and
ora_hash(:CODE_IN_PARAM) < high_key',&
PARAMS (CODE_IN_PARAM = CUSTID),
BEFOREFILTER), &
FILTER (chunklookup1.COUNT = 1);

MAP NSHDPDB.APP_SCHEMA.ORDERS, target APP_SCHEMA.ORDERS, &
SQLEXEC (ID chunklookup2, QUERY 'select count(*) count FROM
gsmadmin_internal.chunks
 WHERE ora_hash(:CODE_IN_PARAM) >= low_key and
ora_hash(:CODE_IN_PARAM) < high_key',&
PARAMS (CODE_IN_PARAM = CUSTID),
BEFOREFILTER), &
FILTER (chunklookup2.COUNT = 1);


MAP NSHDPDB.APP_SCHEMA.LINEITEMS, target APP_SCHEMA.LINEITEMS, &
SQLEXEC (ID chunklookup3, QUERY 'select count(*) count FROM
gsmadmin_internal.chunks
 WHERE ora_hash(:CODE_IN_PARAM) >= low_key and
ora_hash(:CODE_IN_PARAM) < high_key',&
PARAMS (CODE_IN_PARAM = CUSTID),
```

```
BEFOREFILTER), &
FILTER (chunklookup3.COUNT = 1);

#### NOTE ####
1. Remove Handlecollisions parameter and restart replicats after
deltas
   have been applied on target shards.
2. If sharding key column is of  number datatype, please use
below sqlexec
   filter which has to_number in ora_hash function.

SQLEXEC (ID chunklookup, QUERY 'select count(*) count FROM
gsmadmin_internal.chunks
 WHERE ora_hash(to_number(:CODE_IN_PARAM)) >= low_key
   and ora_hash(to_number(:CODE_IN_PARAM)) < high_key',&


Replicat for duplicate tables on Catalog
========================================

GGSCI > dblogin useridalias ggadmin_cat
GGSCI > add replicat repcat, INTEGRATED, exttrail ./dirdat/tr
 CHECKPOINTTABLE ggadmin.GGCHKPT

Add the following parameters in replicat for shard1
GGSCI > edit params repcat

replicat repcat
useridalias ggadmin_cat
HANDLECOLLISIONS
SOURCECATALOG SCPDB
map NSHDPDB.APP_SCHEMA.PRODUCTS, target APP_SCHEMA.PRODUCTS;
```

c. Start replicats on target shards using `atcsn`.

```
GGSCI> start replicat repshd1, atcsn <SCN captured on source>
GGSCI> start replicat repshd2, atcsn <SCN captured on source>
GGSCI> start replicat repshd3, atcsn <SCN captured on source>
GGSCI> start replicat repcat, atcsn  <SCN captured on source>
GGSCI > info all

Program     Status      Group       Lag at Chkpt  Time Since
Chkpt

MANAGER     RUNNING
EXTRACT     RUNNING     EXTNSHD     00:00:00      00:00:05
REPLICAT    RUNNING     REPCAT      00:00:00      00:00:00
REPLICAT    RUNNING     REPSHD1     00:00:00      00:00:03
REPLICAT    RUNNING     REPSHD2     00:00:00      00:00:06
REPLICAT    RUNNING     REPSHD3     00:09:01      00:00:09
```

3. Validate the data replication from source to target tables.

To validate that rows are replicated from the non-sharded table to the shards, for example, if you have 9000 rows in the source table, and three target shards, about 3000 rows should be distributed to each shard.

11

Raft Replication Configuration and Management

Oracle Globally Distributed Database provides commands and options in the GDSCTL CLI to enable and manage Raft replication in a system-managed sharded database.

For a thorough discussion of how Raft replication is implemented in Oracle Globally Distributed Database with diagrams, see Using Raft Replication in Oracle Globally Distributed Database in Oracle Globally Distributed Database Architecture and Concepts.

> **Note:**
>
> Raft replication is not supported for user-defined and composite data distribution methods in OrOracle Globally Distributed Databaseacle Sharding.

Enabling Raft Replication

You enable Raft replication when you configure the shard catalog.

To enable Raft replication, specify the **native** replication option when you create the shard catalog.

For example,

```
gdsctl> create shardcatalog ... -repl native
```

After the shard catalog is created, you can add shards to the configuration and run the `DEPLOY` command.

> **Note:**
>
> You must have at least 3 shards in your sharded database to use Raft replication.

Specifying Replication Unit Attributes

By default, Oracle Globally Distributed Database determines the number of replication units (RUs) in a shardspace and the number of chunks in an RU.

You can specify the number of primary RUs using the `-repunit` option when you create the shard catalog. Specify a value greater than zero (0).

```
gdsctl> create shardcatalog ... -repunit number
```

The RU value cannot be modified after the first `DEPLOY` command is run on the sharded database configuration. Before you run `DEPLOY`, you can modify the number of RUs using the `MODIFY SHARDSPACE` command.

```
gdsctl> modify shardspace -shardspace shardspaceora -repunit number
```

Note that in system-managed sharding there is one shardspace named `shardspaceora`.

If the `-repunit` parameter is not specified, the default number of RUs is determined at the time of execution of the first `DEPLOY` command.

Ensuring Replicas Are Not Placed in the Same Rack

To ensure high availability, Raft replication group members should not be placed in the same rack.

If specified using the `ADD SHARD` command `-rack rack_id` option, the shard catalog will enforce that shards that contain replicated data are not placed in the same rack. If this is not possible an error is raised.

```
gdsctl> add shard -connect connect_identifier … -rack rack_id
```

Getting Runtime Information for Replication Units

Use `GDSCTL STATUS REPLICATION` to get replication unit runtime statistics, such as the leader and its followers.

`GDSCTL STATUS REPLICATION` can also be entered as `STATUS RU`, or just `RU`).

When option `-ru` is specified, you can get specific information for a particular replication unit.

For example, to get information about replication unit ru1:

```
GDSCTL> status ru -ru 1

Replication units
-------------------------
Database RU# Role Term Log Index Status
-------- --- ---- ---- --------- ------
cdbsh1_sh1 1 Leader 2 21977067 Ok
cdbsh2_sh2 1 Follower 2 21977067 Ok
cdbsh3_sh3 1 Follower 2 21977067 Ok
cdbsh1_sh1 2 Follower 1 32937130 Ok
cdbsh2_sh2 2 Leader 1 32937130 Ok
cdbsh3_sh3 2 Follower 1 32937130 Ok
cdbsh1_sh1 3 Follower 2 16506205 Ok
cdbsh2_sh2 3 Follower 2 16506205 Ok
cdbsh3_sh3 3 Leader 2 16506205 Ok
```

For details about the command syntax, options, and examples, see status ru (RU, status replication) in *Global Data Services Concepts and Administration Guide*.

Scaling with Raft Replication

You can add or remove shards from your Raft replication sharded database with the following instructions.

Adding Shards

To scale up your sharded database, run `GDSCTL ADD SHARD`.

When the shard is deployed, Raft replication automatically splits the replication units (RUs), such that for each shard added, 2 new RUs are created and their leaders placed on the new shard. Chunks from other RUs are moved to the new RUs to rebalance data distribution.

After adding shards you can run `CONFIG TASK` to view the ongoing rebalancing tasks.

If you don't want automatic rebalancing to occur, you can deploy the new shards with `GDSCTL DEPLOY -no_rebalance` and then manually move RUs and chunks to suit your needs.

Removing Shards

You remove a shard using `GDSCTL REMOVE SHARD`; however, you must first move any RUs off of the shard.

You can then manually rebalance the RUs on the remaining shards using `MOVE RU` and `RELOCATE CHUNK`.

See Moving Replication Unit Replicas and Moving A Chunk to Another Replication Unit for information.

Moving Replication Unit Replicas

Use `MOVE RU` to move a follower replica of a replication unit from one shard database to another.

For example,

```
gdsctl> move ru -ru 1 -source dba -target dbb
```

Notes:

- Source database shouldn't contain the replica leader
- Target database should not already contain another replica of the replication unit

> **Note:**
>
> Because running this command requires a tablespace set for the destination chunk, create a minimum of 1 tablespace set before running this command.

See move ru (replication_unit) in *Global Data Services Concepts and Administration Guide* for syntax and option details.

Changing the Replication Unit Leader

Using `SWITCHOVER RU`, you can change which replica is the leader for the specified replication unit.

The `-db` option makes the specified database the new leader of the given RU.

```
gdsctl> switchover ru -ru 1 -database dba
```

To then automatically rebalance the leaders, use `SWITCHOVER RU -rebalance`.

For full syntax and option details, see switchover ru (replication_unit) in *Global Data Services Concepts and Administration Guide*.

Copying Replication Units

You can copy a replication unit from one shard database to another using `COPY RU`. This allows you to instantiate or repair a replica of a replication unit on the target shard database.

For example, to copy replication unit 1 from dba to dbb:

```
gdsctl> copy ru -ru 1 -source dba -target dbb
```

Notes:

- Neither source database nor target database should be the replica leader for the given replication unit
- If the target database already contains this replication unit it will be replaced by full replica of the replication unit on the source database
- If `-replace` is specified, the replication unit is removed from the target database
- If the target database doesn't contain the specified replication unit, then the total number of members for the given replication unit should be less than replication factor (3), unless `-replace` is specified.

  ```
  gdsctl> copy ru -ru 1 -source dba -target dbc -replace dbb
  ```

> **Note:**
>
> Because running this command requires a tablespace set for the destination chunk, create a minimum of 1 tablespace set before running this command.

For syntax and option details, see copy ru (replication_unit) in *Global Data Services Concepts and Administration Guide*.

Moving A Chunk to Another Replication Unit

To move a chunk from one Raft replication unit to another replication unit, use the `GDSCTL RELOCATE CHUNK` command.

To use `RELOCATE CHUNK`, the source and target replication unit leaders must be located on the same shard, and their followers must also be on the same shards. If they are not on the same shard, use `SWITCHOVER RU` to move the leader and `MOVE RU` to move the followers to co-located shards.

When moving chunks, specify the chunk ID numbers, the source RU ID from which to move them, and the target RU ID to move them to, as shown here.

```
GDSCTL> relocate chunk -chunk 3, 4 -sourceru 1, -targetru 2
```

The specified chunks must be in the same source replication unit. If `-targetru` is not specified, an new empty target replication unit is created.

`GDSCTL MOVE CHUNK` is not supported for moving chunks in a sharded database with Raft replication enabled.

> **Note:**
>
> Because running this command requires a tablespace set for the destination chunk, create a minimum of 1 tablespace set before running this command.

See also relocate chunk in *Global Data Services Concepts and Administration Guide*.

Splitting Chunks in Raft Replication

You can manually split chunks with `GDSCTL SPLIT CHUNK` in a Raft replication-enabled sharded database.

If you want to move some data within a chunk to a new RU, you can use `GDSCTL SPLIT CHUNK` to manually split the chunk.

You can then use `RELOCATE CHUNK` to move the new chunk to another RU if you wish. See Moving A Chunk to Another Replication Unit.

> **Note:**
>
> Because running this command requires a tablespace set for the destination chunk, create a minimum of 1 tablespace set before running this command.

Getting the Replication Type

To find out if your sharded database is using Raft replication, run `CONFIG SDB` to see the replication type in the command output.

In the command output, the Replication Type is listed as **Native** when Raft replication is enabled.

For example,

```
GDSCTL> config sdb

GDS Pool administrators
-----------------------

Replication Type
-----------------------
Native

Shard type
-----------------------
System-managed

Shard spaces
-----------------------
shardspaceora

Services
-----------------------
oltp_ro_srvc
oltp_rw_srvc
```

Enable or Disable Reads from Follower Replication Units

Use the database initialization parameter `SHARD_ENABLE_RAFT_FOLLOWER_READ` to enable or disable reads from follower replication units in a shard.

Set this parameter to `TRUE` to enable reads, or set to `FALSE` to disable reads.

This parameter can have different values on different shards.

See also: SHARD_ENABLE_RAFT_FOLLOWER_READ in *Oracle Database Reference*.

Tuning Flow Control to Mitigate Follower Lag

Flow control in Raft replication coordinates Raft group followers to optimize performance, efficiently utilize memory, and smooth out replication pipeline hiccups, such as variable network latency.

Followers may not consistently maintain the same speed. Occasionally, one might be slightly faster, while at other times, slightly slower.

To tune flow control set the RAFT_FLOW_CONTROL parameter on the shard where a follower is lagging.

For example,

```
exec gsmadmin_internal.dbms_gsm_dbadmin.setParameter(ru_id,
RAFT_FLOW_CONTROL, param_value);
```

Where:

ru_id is the RU ID number. Use NULL to specify all RUs.

param_value is one of the following flow control settings.

- AT_DISTANCE: As long as the slow follower is within a threshold distance (see "Configuring Threshold Distance" below), in terms of LCRs, from the fast follower, the fast follower is throttled.

  This is the RAFT_FLOW_CONTROL default setting.

  However, if the slow follower falls behind by more than the threshold distance, then it is disconnected, at which point it may or may not be able to catch up, depending on why there is a lag between the two followers. For example, if the slow follower is lagging because the network connection to it is bad for a very long time, it will be disconnected. This is also the case if the slow follower is actually a down follower.

- TILL_TIMEOUT: As long as the slow follower has received an LCR within a threshold time (see "Configuring Threshold Timeout" below), from the fast follower, the fast follower is throttled.

  However, if the slow follower falls behind by more than the threshold time, then it is disconnected, at which point it may or may not be able to catch up, depending on why there is a lag between the two followers. For example, if the slow follower is lagging because the network connection to it is bad for a very long time, it will be disconnected. This is also the case if the slow follower is actually a down follower.

- AT_LOGLIMIT: Flow control does not kick in during normal operation at all, but only starts if the log file is about to be overwritten by the leader but the slow follower still needs LCRs from the file being overwritten. When this situation occurs, the leader waits for the slow follower to consume the LCRs from the file to be overwritten.

  If the slow follower is actually a down follower, then with this option the leader waits for the slow follower to come online again when the RU's raft log limit is reached.

Configuring Threshold Distance

Threshold distance, expressed as a percentage of the in-memory queue size for LCRs, is used by the AT_DISTANCE option for flow control.

The default value is 10.

To set the threshold distance to another value, run:

```
exec gsmadmin_internal.dbms_gsm_dbadmin.setParameter(ru_id,
RAFT_FLOW_CONTROL_NS_DISTANCE_PCT, number);
```

Configuring Threshold Timeout

Threshold timeout, in milliseconds, is used by the TILL_TIMEOUT option for flow control.

The default value is 0 (zero).

To set the threshold timeout, run:

```
exec gsmadmin_internal.dbms_gsm_dbadmin.setParameter(ru_id,
RAFT_FLOW_CONTROL_TIMEOUT_MS, milliseconds);
```

Setting Transaction Consensus Timeout

You can change the timeout value for a transaction to get consensus in Raft replication.

To configure the transaction consensus timeout, set the `TXN_ACK_TIMEOUT_SEC` parameter, which specified the maximum time a user transaction waits for the consensus of its commit before raising ORA-05086.

```
exec gsmadmin_internal.dbms_gsm_dbadmin.setParameter(ru_id,
TXN_ACK_TIMEOUT_SEC, seconds);
```

Where

`ru_id` is the RU ID number. Use `NULL` to specify all RUs.

By default, Raft replication waits 90 seconds for a transaction to get consensus. However, if the leader is disconnected from the other replicas, it may not get consensus for its commits; if there is low memory in the replication pipeline, the replication of LCRs slows down, resulting in the delayed delivery of acknowledgments. In many cases such as these, 90 seconds may be too long to wait, so you may want to error out a transaction much earlier, depending on your application requirements.

The minimum valid value is 1 second.

Viewing Parameter Settings

Use the `SHARD_RAFT_PARAMETERS` static data dictionary view to see parameters set at an RU level on each shard.

The values for these parameters, if set, can be seen in this view. The columns in the view are:

`ORA_SHARD_ID`: shard ID

`RU_ID`: replication unit ID

`NAME`: parameter name

`VALUE`: parameter value

For details about this view, see SHARD_RAFT_PARAMETERS in *Oracle Database Reference*.

Dynamic Performance Views for Raft Replication

There are several dynamic performance (V$) views available for Raft replication.

- V$SHARD_ACK_SENDER
- V$SHARD_ACK_RECEIVER
- V$SHARD_APPLY_COORDINATOR
- V$SHARD_APPLY_LCR_READER
- V$SHARD_APPLY_READER
- V$SHARD_APPLY_SERVER
- V$SHARD_LCR_LOGS
- V$SHARD_LCR_PERSISTER
- V$SHARD_LCR_PRODUCER
- V$SHARD_NETWORK_SENDER
- V$SHARD_MESSAGE_TRACKING
- V$SHARD_REPLICATION_UNIT
- V$SHARD_TRANSACTION

For descriptions and column details for these views, see Dynamic Performance Views in *Oracle Database Reference*.

Raft Replication Restrictions

The following restrictions apply to Raft replication in Oracle Globally Distributed Database.

GDSCTL MOVE CHUNK is not supported for Raft replication. To move chunks from one replication unit to another, use RELOCATE CHUNK. See Moving A Chunk to Another Replication Unit.

12

Deploying and Managing a Directory-Based Oracle Globally Distributed Database

Directory-based sharding allows you to explicitly associate key value with shards dynamically at run time, which gives you fine-grained control over mapping of key values to shards

Directory-Based Sharding Roadmap

Set up a directory-based sharded database, including configuring the sharded database, creating schema objects, and doing lifecycle management operations.

1. Deploy a Directory-Based Sharded Database

A directory-based configuration follows the same steps as you would for a user-defined sharded database, with a few differences.

Most of the information you need is found in Oracle Globally Distributed Database Deployment for planning, installing and creating the databases for the sharded database topology.

To configure the topology for Directory-based sharding, do the following tasks:

1. Create a shard catalog for user-defined sharding. See Creating a Shard Catalog for Directory-Based Sharding.
2. Add and start shard directors. See Add and Start Shard Directors.
3. Create shardspaces, and shards in those shardspaces. See Add Shardspaces If Needed, Add the Shard CDBs, and Add the Shard PDBs.
4. Create tablespaces in the shardspaces. See User-Defined Sharding for examples.

   Note that each tablespace has to be created individually, and explicitly associated with a shardspace.
5. Verify the topology, add shards and host metadata, deploy the configuration, and start global database services. See Oracle Globally Distributed Database Deployment.

2. Create Schema Objects

- To create tables sharded by directory, see Creating Tables Sharded by Directory.
- To add (and remove) keys to the directory, see Managing Keys in Directory-Based Sharding.

3. Run DML and Queries

See DML Support on Tables Sharded by Directory.

4. Perform Lifecycle Operations

Over the lifetime of your directory-based sharded database, you'll need to do tasks such as:

- Add and remove keys. See Managing Keys in Directory-Based Sharding
- Add partitions. See Adding a New Tablespace and Chunks (Partition) in a Shardspace

- Move chunks. See Chunk Management in Directory Based Sharding.
- Split chunks. See Splitting Partitions (Chunks).
- Query the directory view for metadata. See Sharding Key Directory Public View.

Creating a Shard Catalog for Directory-Based Sharding

Directory-based sharding is an enhancement of the User-defined sharding method, so the shard catalog is configured with the user-defined sharding option.

```
GDSCTL> create shardcatalog -database catalog_connect_string
 -user mysdbadmin/mysdbadmin_password -sharding user
 -protectmode maxperformance
```

More details about creating a shard catalog, including specifying the shard catalog Connect String and connecting to the shard catalog can be found in Create the Shard Catalog.

Creating Tables Sharded by Directory

Create directory-based sharded tables using `PARTITION BY DIRECTORY` in the `CREATE SHARDED TABLE` statement.

For example:

```
CREATE SHARDED TABLE customers
( id             NUMBER NOT NULL
, name           VARCHAR2(30)
, address        VARCHAR2(30)
, status         VARCHAR2(1)
,
CONSTRAINT cust_pk PRIMARY KEY(id)
)
PARTITION BY DIRECTORY (id)
( PARTITION p1 TABLESPACE tbs1,
  PARTITION p2 TABLESPACE tbs2,
  PARTITION p3 TABLESPACE tbs3…);
```

Note:

- Unlike in user-defined sharding, key values are not specified for the partitions in the CREATE TABLE statement.
- The directory table is automatically created during root table creation. The definition of the directory table is:

  <shard user schema>.*<root_table>*$SDIR
- If a child table is created with parent clause in a different schema from the root table, an additional privilege is required for the child table's schema owner. (This is only for directory-based sharding and is not required for regular user-defined sharding.)

  This is because there is a foreign key constraint on the child table to the directory table's sharding key columns, to ensure that no rows can be inserted into the child table without the sharding key value being present in the directory mapping. As a consequence, the child table's schema needs a reference privilege on the directory table's sharding key columns.

  See "Granting References" below.

Granting References

This case is illustrated in this example:

- Root table `dealerships` is under schema `user1`, and has `account_id` as the sharding key.
- Child table `salespeople` is under schema `user2`, and is defined via "`parent user1.dealerships`".

Before this `salespeople` child table can be created, you need:

- `grant all privileges on user1.dealerships to user2;`

  This is the same as needed for user-defined sharding.
- `grant references (account_id) on user1.dealerships$sdir to user2;`

  This is new for directory-based sharding.

Note that `dealerships$sdir` is the internally generated directory table name; it has the format of *`root_table_name`*`$sdir`.

Without the 2nd grant, the child table creation DDL will succeed on the shard catalog but will fail on the shards (as the foreign key is only added on the shards).

Managing Keys in Directory-Based Sharding

The directory table contains the metadata for mapping keys to partitions. You can use the `DBMS_SHARDING_DIRECTORY` PL/SQL API to add and remove keys.

> **Note:**
>
> When adding and removing keys there are APIs that include commit and those that do not. Unless the commit versions of the APIs are used, the directory content is not propagated to the shards until commit is issued explicitly.

Adding Keys

You can add a key to the directory with the specified partition name using `addKeyToPartition` or `addKeyToPartitionCommit`.

The `addKeyToPartitionCommit` procedure is exactly the same as the `addKeyToPartition` procedure with the same parameters, except that it performs a commit automatically at the end.

```
PROCEDURE addKeyToPartition[Commit]
     (schema_name    IN varchar2,    -- root table schema name
      root_table     IN varchar2,    -- root table name
      partition_name IN  varchar2,   -- name of the partition
      key …)                        -- shard key column value
```

Note that the key column value needs to be in the same order as specified in the `CREATE TABLE` statement with the correct types. The procedure can only succeed if the provided key does not yet exist in the directory.

Removing Keys

You can remove a key from the directory using `removeKey` or `removeKeyCommit`.

The `removeKeyCommit` procedure is exactly the same as the `removeKey` procedure with the same parameters, except that it performs a commit automatically at the end.

```
PROCEDURE removeKey
     (schema_name    IN varchar2,    -- root table schema name
      root_table     IN varchar2,    -- root table name
      key … )                       -- shard key column values
```

Note that the key column values need to be in the same order as specified in the `CREATE TABLE` statement with the correct types. The procedure can only succeed if the provided key exists in the directory, and there are no tables (either root table or child tables) with rows still referencing the key.

Enable Automatic Key-to-Partition Assignment

You can indicate an automatic key-to-partition assignment rule for subsequent new key inserts into the root table.

```
PROCEDURE setAssignmentRule
     (schema_name    IN varchar2,    -- root table schema name
      root_table     IN varchar2,    -- root table name
      rule_id        IN number);   -- rule ID as defined in public constants
```

Once set, the key-to-partition assignment rule stays in effect across different sessions, regardless of system restart, until another call to the procedure is made with a different rule value, or with `NONE` meaning automatic assignment should be turned off.

The following constants are defined for key-to-partition assignment rules.

- `NONE constant number :=0;` -- turn off rule-based assignment
- `LAST_PARTITION constant number := 1;` -- rule for assigning key only to the last added partition
- `ROUNT_ROBIN constant number :=2;` -- rule for assigning key to partition by round robin
- `RANDOM constant number :=3;` -- rule for assigning key to partition randomly
- `CUSTOM constant number :=4;` -- TBD

DML Support on Tables Sharded by Directory

Directory-based sharding offers the same support as other sharding methods for regular DMLs and queries run on the shard with partition pruning support.

Adding a New Tablespace and Chunks (Partition) in a Shardspace

You may need to add a new tablespace and partition to a table sharded by directory when you want to add new groupings of keys on an existing shardspace or a newly added shardspace.

The steps involved are:

1. Create new tablespaces in the desired shardspace.
2. Run `ALTER TABLE ADD PARTITION` *`partition_name`* `TABLESPACE` *`tablespace_name`* on the sharded table, for example:

   ```
   ALTER TABLE customers ADD PARTITION p4 TABLESPACE tb4;
   ```

This results in the creation of an empty partition and chunk in the specified shardspace. Subsequent inserts of new key values can then specify this new partition as the target.

If you specify the assignment rule to be `last partition`, all new key inserts will be automatically assigned to the new partition.

Chunk Management in Directory Based Sharding

As with user-defined sharding, tablespaces created for directory-based sharding are assigned to chunks.

The total number of chunks is defined by the number of partitions specified in the sharded table. The number of chunks for a given shardspace is the number of partitions assigned to it. The `ALTER TABLE ADD`, `DROP`, and `SPLIT PARTITION` commands on the sharded table increases or decrease the number of chunks.

The GDSCTL `SPLIT CHUNK` command, which is used to split a chunk in the middle of the hash range for system-managed sharding, is not supported for directory-based sharding. You must use the `ALTER TABLE SPLIT PARTITION` statement to split a chunk.

Also, just like user-defined sharding, no chunk migration is automatically started when a shard is added to the sharded database. You must run the `GDSCTL MOVE CHUNK` command for each chunk that needs to be moved to another shard.

Splitting Partitions (Chunks)

1. Invoke the `DBMS_SHARDING_DIRECTORY` PL/SQL API `flagKeyForSplit` to mark keys for splitting.

   ```
   PROCEDURE flagKeyForSplit
        (schema_name    IN varchar2,    -- root table schema name
         root_table     IN varchar2,    -- root table name
         key … )                       -- shard key column values
   ```

   Note that the key column values need to be in the same order as specified in the `CREATE TABLE` statement with the correct types. The procedure can only succeed if the provided key exists in the directory.

2. Issue the partition split DDL.

   ```
   ALTER TABLE customers SPLIT PARTITION p1 INTO
   ( PARTITION p1 TABLESPACE tb1,
     PARTITION p3 TABLESPACE tb3 )
     UPDATE INDEXES;
   ```

   Note that, in directory-based sharding, a partition can be split into only **two** partitions at a time.

   This operation will go through all of the keys that have been marked for split in the directory and split the corresponding data out into the new partition.

Sharding Key Directory Public View

The view `root_table_name$shard_dir_view` provides you with the key to partition/chunk/shard mappings for the specified root table.

Table 12-1 root_table_name$SHARD_DIR_VIEW

Name	Type	NULL	Description
KEY columns…	varies	No	Unique sharding key column values
KEY_ID$	RAW(32)	No	Unique SHA-256 ID assigned to the key
CHUNK_ID$	NUMBER	No	The chunk ID to which the key is assigned
PARTITION_NAME	VARCHAR2(128)	No	Name of the root table partition the key is assigned to
SHARDSPACE_NAME	VARCHAR2(128)	No	The shardspace name where the chunk belongs to
SPLIT_FLAG$	NUMBER	Yes	0: not flagged (default) 1: flagged

13

Using the Sharding Advisor

Sharding Advisor simplifies the migration of your existing, non-sharded Oracle database to a Oracle Globally Distributed Database, by analyzing your workload and database schema, and recommending the most effective sharded database configurations.

About Sharding Advisor

The Sharding Advisor is a client-side, command-line tool that you run against any non-sharded, production, 10g or later release, Oracle Database that you are considering migrating to an Oracle Globally Distributed Database.

The Sharding Advisor analysis provides you with the information you need to design a schema that maximizes performance while reducing duplicated data in the new sharded database environment.

The following are benefits of using Sharding Advisor to aid you with schema design.

- Maximize query workload performance
- Minimize multi-shard operations requiring cross-shard joins
- Maximize parallelism for complex queries (spread query processing across all shards)
- Minimize the amount of duplicated data on each shard

The Sharding Advisor utility, `GWSADV`, is installed with Oracle Database as a standalone tool, and connects to your database using authenticated OCI connections.

To get an understanding of your schema and other preferences, Sharding Advisor asks you questions as part of an interactive dialog.

Sharding Advisor then connects to the existing non-sharded database, also called the **source**, analyzes its schema and query workload, and produces a set of alternative designs for the sharded database, including recommendations for an effective sharding key, which tables to shard, and which tables to duplicate on all shards.

Sharding configurations are ranked in terms of query performance, with the ranking favoring configurations that maximize single and multi-shard queries that do not require cross-shard joins, while minimizing multi-shard queries that require cross-shard joins.

You choose the design that best fits your requirements. The designs are ranked by the advisor, so if you don't have specific preferences you can choose the highest ranked design by default.

> **Note:**
>
> There are restrictions to Sharding Advisor capabilities:
> The source database must be Oracle Database 10g or later release.
>
> If you cannot run the Sharding Advisor against the live production database, you can run the Sharding Advisor on a different server that has the schema and workload imported from the production database.
>
> Sharding Advisor discovers the table families based on primary key-foreign key relationships. If the schema does not have any primary key-foreign key constraints, sharding by `PARENT` clause is recommended.
>
> Currently, Sharding Advisor recommends only single-table family, system-managed sharding (sharding by reference) configurations if the source database has foreign key constraints; otherwise, Sharding Advisor recommends sharding using the `PARENT` clause.

Run Sharding Advisor

Run the Sharding Advisor command-line tool against your existing, non-sharded Oracle Database to obtain recommended sharded database configurations.

The user running Sharding Advisor requires the following privileges.

```
SQL> ALTER SYSTEM SET statistics_level=all;
SQL> grant create session to sharding_advisor_user;
SQL> grant alter session to sharding_advisor_user;
SQL> grant select on v_$sql_plan to sharding_advisor_user;
SQL> grant select on v_$sql_plan_statistics_all to
sharding_advisor_user;
SQL> grant select on gv_$sql_plan to sharding_advisor_user;
SQL> grant select on gv_$sql_plan_statistics_all to
sharding_advisor_user;
SQL> grant select on DBA_HIST_SQLSTAT to sharding_advisor_user;
SQL> grant select on dba_hist_sql_plan to sharding_advisor_user;
SQL> grant select on dba_hist_snapshot to sharding_advisor_user;
```

The Sharding Advisor command-line utility, `GWSADV`, runs from `$ORACLE_HOME/bin`.

Run the Sharding Advisor from the command line, as shown here.

```
$ gwsadv -u username -p password -c -w sch=\(schema1,schema2\)
```

> **Note:**
>
> The parenthesis in this command is escaped on Linux systems.

Where `-u` and `-p` are the user name and password of the user that runs the Sharding Advisor.

Use the capture workload parameter, `-c`, the first time you run Sharding Advisor against an existing query workload, to capture the predicate information from the source's `GV$SQL_PLAN_STATISTICS_ALL` view. You don't need to use `-c` in subsequent queries on the same workload.

The required `-w` flag indicates that Sharding Advisor uses the query workload for sharding configuration generation and ranking.

In this case, the `sch` parameter specifies a list of schemas to run Sharding Advisor against. There are several other options you can use with Sharding Advisor, detailed in Sharding Advisor Usage and Options.

Run Sharding Advisor on a Non-Production System

To minimize the impact on a live production system, you can run the Sharding Advisor on a copy of the database schema and workload, located on a different server than the production system.

To get the same results as if it were the live production system, the production database schema and workload can be exported using the Oracle Data Pump utilities and copied to a different server. Then you can run Sharding Advisor on the imported schema.

You only export the database schema and system tables. There is no need to export the actual data.

The following procedure uses the HR schema as an example.

Do the following steps on the source (production) database server.

1. Export the schema using Data Pump Export.

```
> expdp system/password SCHEMAS=HR DIRECTORY=HR_DIR CONTENT=METADATA_ONLY
 DUMPFILE=hr_metadata.dmp LOGFILE=hr_exp.lst
```

2. Export the Automatic Work Repository (AWR) snapshot.

```
SQL> @$ORACLE_HOME/rdbms/admin/awrextr.sql
```

Do the following steps on the target database server.

3. Copy the dump files from the source to the target.

   For example, copy the dump files to `/scratch/dump`.

4. Create a user that can run Sharding Advisor on the schema.

```
SQL> CREATE USER hr IDENTIFIED BY password;
```

5. Create (or replace) the dump file directory variable that Data Pump Import can reference.

```
SQL> CREATE DIRECTORY HR_DIR AS '/scratch/dump'

SQL> CREATE OR REPLACE DIRECTORY  HR_DIR AS '/scratch/dump'
```

6. Import the schema.

```
> impdp system/password DIRECTORY=HR_DIR DUMPFILE=hr.dmp
LOGFILE=imp.lst SCHEMAS=HR
```

7. Load the AWR data.

```
SQL> @$ORACLE_HOME/rdbms/admin/awrload.sql
```

8. Now you can run Sharding Advisor on the target, non-production, copy of the database with the user you created.

```
> gwsadv -u hr -p password -c -awr_snap_begin begin_timestamp -
awr_snap_end end_timestamp -w
```

Review Sharding Advisor Output

Sharding Advisor discovers the table families for each potential sharding column that it extracts from the query workload, and ranks the table families based on query classification rules and a ranking algorithm.

To review the sharding configurations and related information that is owned by the user running Sharding Advisor, you can query the following output database tables, which are stored in the same schema as your source database.

- `SHARDINGADVISOR_CONFIGURATIONS` has one row for each table in a ranked sharded configuration, and provides details for each table, such as whether to shard or duplicate it, and if sharded, its level in a table family hierarchy, its parent table, root table sharding key, foreign key reference constraints, and the estimated size per shard.
- `SHARDINGADVISOR_CONFIGDETAILS` has one row for each ranked sharding configuration, and provides details for each ranked sharding configuration, such as the number and collective size, per shard, of the sharded tables, and the number and collective size of the duplicated tables. It also provides the number of single shard and multi-shard queries to expect in production, as well as the number of multi-shard queries requiring cross-shard joins, based on your source database's current workload, and an estimated cost.
- `SHARDINGADVISOR_QUERYTYPES`, for each query in the workload, lists the query type for each sharding configuration. Note that the same query can be of a different query type depending on the sharding configuration.

Because the Sharding Advisor output is contained in regular database tables, you can run many kinds of SQL queries against them to look at the output from different perspectives.

For example, to display the sharding configurations in ranking order, run

```
SELECT rank, tableName as tname, tabletype as type,
           tablelevel as tlevel, parent, shardby as shardBy,
           shardingorreferencecols as cols, unenforceableconstraints,
           sizeoftable
FROM SHARDINGADVISOR_CONFIGURATIONS
ORDER BY rank, tlevel, tname, parent;
```

For details about the Sharding Advisor output tables and more example queries see Sharding Advisor Output Tables and Sharding Advisor Output Review SQL Examples

Choose a Sharding Advisor Recommended Configuration

There are some aspects of database sharding to take into consideration when deciding which configuration to choose for your sharded database.

Increasing the number of shards will result in higher availability and scalability of the sharded database.

Minimizing duplicated data can conflict with your desire to minimize multi-shard queries that require joins across multiple shards. Because joins in a sharded database are usually performed on related data, storing related data in the same shard can dramatically speed up processing of such joins.

The overall cost, in terms of query workload, of the recommended sharding configurations is based on the number of each query type (single shard, multi-shard, and multi-shard with cross-shard joins) in the workload, where multi-shard queries with cross-shard joins have the highest cost, and single shard queries have the lowest cost. The cost information is in the `COST` column of the Sharding Advisor `SHARDINGADVISOR_CONFIGDETAILS` output table.

Sharding Advisor Usage and Options

Sharding Advisor is a client command-line tool that connects to an existing non-sharded database and provides sharding configuration recommendations.

Syntax

```
gwsadv
 [-n nodeName[:portnum]]
 [-s serviceName]
  -u username
  -p password
 [-c]
 [-awr_snap_begin timestamp]
 [-awr_snap_end timestamp]
  -w
 [sch=(schema1, schema2, …)]
 [-tab importantTabsFile]
 [-pr numpreds:n]
 [-t trace_file]
```

Options

Note that each option must be prefixed with a minus sign (-) except for the `sch` argument.

Option	Description	Required (Y/N)
`-awr_snap_begin timestamp`	Specify the beginning timestamp, in the format 'YYYY-MM-DD HH24:MI:SS', to specify the AWR snapshots to capture the workload from.	N

Option	Description	Required (Y/N)
`-awr_snap_end` *timestamp*	Specify the end timestamp, in the format 'YYYY-MM-DD HH24:MI:SS', to specify the AWR snapshots to capture the workload from.	N
`-c`	Capture a new or changed workload. Use `-pr` to limitthe number of predicates to be captured Required on first run of Sharding Advisor on a new or changed workload. Not required on subsequent runs on the same workload. By default, the workload is captured from the `V$SQL_PLAN_STATISTICS_ALL` table. Alternatively, the workload can be captured from Automatic Workload Repository (AWR) snapshots by using the `-awr_snap_begin` and `-awr_snap_end` options with the `-c` option to specify the beginning and ending time stamps of the AWR snapshots.	N
`-n` *nodeName*`[:`*portnum*`]`	Node name and port number, if connecting to a database on another host	N
`-p` *password*	Oracle password	Y
`-pr` *numpreds*`:`*n*	Limits the number of predicates to be captured when using `-c` to capture a new or changed workload.	N
`-s` *serviceName*	Service name, if connecting to a database on another host	N
`sch`	The `sch` option specifies the list of schemas to run Sharding Advisor against, if you want to run as a different user.	N
`-t` *trace_file*	Enables tracing of all activities performed by sharding advisor. Specify an output file name.	N
`-tab` *importantTabsFile*	Name of file that consists of table names, one per line, in the format *schemaname.tablename*, to restrict the number of tables that the Sharding Advisor needs to analyze.	N
`-u` *username*	Oracle user name	Y

Option	Description	Required (Y/N)
`-w`	Directs Sharding Advisor to use the query workload for sharding configuration generation and ranking.	Y

Usage Notes

The normal usage of the sharding advisor is to not specify the `-pr` option. The query workload capture should be faster now even without the `-pr` option. If however, the you want to speed it up further, the `-pr` option can be used. If it is used, it has to be used in conjunction with the `-c` option. If unspecified, the number of predicates to be captured is not limited.

For procedures describing how to run the Sharding Advisor with example commands see Run Sharding Advisor and Run Sharding Advisor on a Non-Production System.

Sharding Advisor Output Tables

To review the sharding configurations and related information, you can query the following output database tables, which are stored in the same schema as your source database.

SHARDINGADVISOR_CONFIGURATIONS Table

Each row of the SHARDINGADVISOR_CONFIGURATIONS table represents a table in a ranked sharded configuration, and provides information about whether to shard or duplicate it, and if sharded, its level in a table family hierarchy, its parent table, root table sharding key, foreign key reference constraints, and table size per shard.

SHARDINGADVISOR_CONFIGURATIONS Schema

Column	Description
`RANK`	The rank of the sharding configuration based on the ranking algorithm
`TABLENAME`	Name of the table in the sharding configuration
`TABLETYPE`	‘S’ (Sharded), ‘D’ (Duplicated), or ‘L’ (Local)
`TABLELEVEL`	Level of the table in the table family hierarchy, NULL for duplicated tables
`PARENT`	Parent of the table in the table family hierarchy, NULL for duplicated tables
`SHARDBY`	Sharding method. `REFERENCE` for sharding by reference, or `PARENT` for sharding by `PARENT` clause, for child tables.
`SHARDINGORREFERENCECOLS`	Sharding key for the root table, partition by `REFERENCE` or `PARENT` for the child tables in a table family, and NULL for duplicated tables
`UNENFORCEABLECONSTRAINTS`	Foreign key constraints other than the reference columns, which cannot be enforced
`SIZEOFTABLE`	Size of the table per shard

SHARDINGADVISOR_CONFIGDETAILS Table

Each row of the SHARDINGADVISOR_CONFIGDETAILS table represents a ranked sharding configuration, and provides the number and collective size, per shard, of each type of table, the number of each type of query, and based on your source database's current workload, an estimated cost.

SHARDINGADVISOR_CONFIGDETAILS Schema

Column	Description
RANK	The rank of the sharding configuration based on the ranking algorithm
CHOSENBYUSER	'Y' if the sharding configuration is chosen by the user, NULL for other sharding configurations
NUMSHARDEDTABLES	Number of sharded tables in this sharding configuration
SIZEOFSHARDEDTABLES	Cumulative size of sharded tables (per shard) in this sharding configuration
NUMDUPLICATEDTABLES	Number of duplicated tables in this sharding configuration
SIZEOFDUPLICATEDTABLES	Cumulative size of duplicated tables (per shard) in this sharding configuration
NUMSINGLESHARDQUERIES	Number of single shard queries in the query workload for this sharding configuration
NUMMULTISHARDQUERIES	Number of multi-shard queries in the query workload for this sharding configuration
NUMCROSSSHARDQUERIES	Number of multi-shard queries that require an external join in the query workload for this sharding configuration
COST	Cost of the sharding configuration based on the costing algorithm

SHARDINGADVISOR_QUERYTYPES Table

Each row of the SHARDINGADVISOR_QUERYTYPES table represents a query in the workload, and lists the query type and SQL ID. Note that the same query can be of a different query type depending on the sharding configuration.

SHARDINGADVISOR_QUERYTYPES Schema

Column	Description
RANK	The rank of the sharding configuration based on the ranking algorithm
SQLID	The query SQL ID
QUERYTYPE	The type of the query in this sharding configuration: `SINGLE SHARD QUERY`, `MULTI SHARD QUERY`, or `CROSS SHARD QUERY`

Sharding Advisor Output Review SQL Examples

Because the Sharding Advisor output is contained in regular database tables, you can run many kinds of SQL queries against them to look at the output from different perspectives.

Example 13-1 Display the sharding configurations in ranking order

```
SELECT rank, tableName as tname, tabletype as type,
           tablelevel as tlevel, parent, shardby as shardBy,
           shardingorreferencecols as cols, unenforceableconstraints,
           sizeoftable
FROM SHARDINGADVISOR_CONFIGURATIONS
ORDER BY rank, tlevel, tname, parent;
```

Example 13-2 Display the table family of the top ranked sharding configuration

```
SELECT rank, tableName as tname, tabletype as type,
        tablelevel as tlevel, parent, shardby as shardBy,
        shardingorreferencecols as cols, unenforceableconstraints,
        sizeoftable
FROM SHARDINGADVISOR_CONFIGURATIONS
WHERE rank = 1 AND tabletype = 'S'
ORDER BY tlevel, tname, parent;
```

Example 13-3 Display the table families in ranking order

```
SELECT rank, tableName as tname, tabletype as type,
        tablelevel as tlevel, parent, shardby as shardBy,
        shardingorreferencecols as cols, unenforceableconstraints,
        sizeoftable
FROM SHARDINGADVISOR_CONFIGURATIONS
WHERE tabletype = 'S'
ORDER BY rank, tlevel, tname, parent;
```

Example 13-4 Display the duplicated tables of the top ranked sharding configuration

```
SELECT rank, tableName as tname, tabletype as type,
           tablelevel as tlevel, parent, shardby as shardBy,
           shardingorreferencecols as cols, unenforceableconstraints,
           sizeoftable
FROM SHARDINGADVISOR_CONFIGURATIONS
WHERE rank = 1 AND tabletype = 'D'
ORDER BY tlevel, tname, parent;
```

Example 13-5 Display the number of sharding configurations with *table_name* as the root table

```
SELECT COUNT(*)
FROM SHARDINGADVISOR_CONFIGURATIONS
WHERE tablename = 'TABLE_NAME' AND tablelevel = 0;
```

Example 13-6 Display the table families of the sharding configurations with root table *table_name*

```
SELECT rank, tableName as tname, tabletype as type,
        tablelevel as tlevel, parent, shardby as shardBy,
        shardingorreferencecols as cols
FROM SHARDINGADVISOR_CONFIGURATIONS
WHERE tabletype = 'S'
    AND rank IN
        (SELECT rank
        FROM SHARDINGADVISOR_CONFIGURATIONS
        WHERE tablename = 'TABLE_NAME' and tablelevel = 0)
ORDER BY rank, tlevel, tname, parent;
```

Example 13-7 Display the details of the sharding configurations in ranking order

```
SELECT rank, chosenbyuser,
        numshardedtables as stabs, sizeofshardedtables as sizestabs,
        numduplicatedtables as dtabs,
        sizeofduplicatedtables as sizedtabs,
        numsingleshardqueries as numssq,
        nummultishardqueries as nummsq,
        numcrossshardqueries as numcsq, cost
FROM SHARDINGADVISOR_CONFIGDETAILS
ORDER BY rank;
```

Example 13-8 Display the details of your chosen sharding configuration

```
SELECT rank,
        numshardedtables as stabs, sizeofshardedtables as sizestabs,
        numduplicatedtables as dtabs,
        sizeofduplicatedtables as sizedtabs,
        numsingleshardqueries as numssq,
        nummultishardqueries as nummsq,
        numcrossshardqueries as numcsq, cost
FROM SHARDINGADVISOR_CONFIGDETAILS
WHERE CHOSENBYUSER = 'Y'
ORDER BY RANK;
```

Sharding Advisor Security

Sharding Advisor is a client-side utility that connects to the non-sharded database using authenticated OCI connections.

- The Sharding Advisor requires the appropriate credentials (user name and password) to connect to the non-sharded source database. Sharding Advisor can be run as a different user than the user that owns the source database schema that the Sharding Advisor analyzes. This user must have `SELECT` privileges on the tables in the non-sharded schema.
- The user needs `SELECT` privileges on the `GV$SQL_PLAN` and `GV$SQL_PLAN_STATISTICS_ALL` views, and on the `DBA_HIST_SQL_PLAN`,

`DBA_HIST_SQLSTAT`, and `DBA_HIST_SNAPHSOT` tables. The user does not need any other special privileges.

- Sharding Advisor is not vulnerable to privilege escalation and denial of service.
- Sharding Advisor does not store or expose any sensitive data such as passwords, database service names, or user names.
- Sharding Advisor does not expose sensitive details about the inner workings of the product.
- Sharding Advisor does not include any interfaces or APIs which are not externally documented.
- Sharding Advisor does not require any insecure protocols to be enabled.
- Sharding Advisor does not use any insecure modes of operation.
- Sharding Advisor does not store any data or other information in any files.
- All connections to the database are through authenticated OCI connections.
- There are no SETUID executables created.
- No new grants to PUBLIC are done.
- No new default schemas are created, but Sharding Advisor internal tables are created under the user that is used to run Sharding Advisor.

14

Sharding JSON Document Collections

Learn how to shard tables of JSON documents using Oracle Globally Distributed Database with SODA.

Overview of Sharding JSON Documents

Oracle Globally Distributed Database allows JSON documents to scale to massive data and transactions volume, provide fault isolation, and support data sovereignty. Oracle Database has support for native JSON objects. Applications can interact with the sharded database using the SODA (Simple Oracle Document Access) API, which allows you to access data using JSON document attributes.

In Oracle Database, JSON documents can be stored in a database table. The database tables act as JSON collections, and each row is a JSON document. JSON documents are stored in the database as type JSON, which is backed by a highly optimized binary JSON format called OSON.

Although Oracle provides support for JSON operators to create, work with, and retrieve JSON documents, the SODA interface is also supported. SODA provides a more intuitive interface for working with JSON documents.

SODA is an API for NoSQL-style JSON (and not only JSON) document collections in Oracle Database. Using SODA APIs, application can perform CRUD operations on documents in collections. Collections are backed by regular Oracle tables (or views).

Typically, to create a collection, one would use SODA API. That creates the underlying table backing the collection. In order to create a sharded collection, however, a shared table has to be created first. Then, a collection can be created on top of a sharded table, by using a mapped collection feature of SODA.

Working with JSON documents in a sharded database introduces the notion of a sharding key. JSON documents are distributed to the individual database table shards according to the sharding key. The sharding key can either be a field from within the JSON document or an external column such as the ID assigned by the SODA API.

For further reading about JSON and SODA, see JSON in Oracle Database and Overview of SODA.

The topics that follow provide details about how to shard JSON objects in Oracle Database. The high level steps are:

- Deploy a sharded database
- Identify a sharding key that the application can use to fetch data
- Define a data store for JSON in Oracle Database by creating sharded tables
- Map the sharded table with SODA

Then life cycle management tasks detailed are:

- Add documents to the sharded JSON collection in the application

- Fetch document data from the sharded JSON collection in the application

Preparing the Environment

Before you begin configuring an Oracle Globally Distributed Database with SODA, deploy a sharding configuration and start the global services.

A sharded database configuration, including shard directors, shard catalog, and shard databases, and any replicas must be deployed. After deploying the sharded database, you must create and start global database services on the shards to service incoming connection requests from your application.

See Sharded Database Deployment for information about creating and deploying a sharded database configuration.

Creating an All-Shards User with SODA Privileges

Create a user on the shard catalog that has the privileges to create schema objects in the sharded database, and also has the necessary `execute` privileges on the `DBMS_SODA` PL/SQL package.

For the purposes of this document, the user is referred to as the *Sharding/SODA user*, and the user name is `app_schema` in the examples.

To create the Sharding/SODA user:

1. Connect to the shard catalog database (for example, as SYSDBA).
2. Enable `SHARD DDL`.
3. Run `CREATE USER` command, granting the permissions shown in the example below.
   Note that the Sharding/SODA user is created on the PDB, not the CDB.

The following is an example Sharding/SODA user creation script.

```
-- Set the container and create the sharded user
alter session set container=SDBPDB;
alter session enable shard ddl;
create user app_schema identified by password;

-- Grant basic privileges
grant connect, resource, alter session to app_schema;
grant execute on dbms_crypto to app_schema;

-- All privileges below are required. User can also be granted all
privileges
grant create table, create procedure, create tablespace, create
materialized view to app_schema;
grant unlimited tablespace to app_schema;
grant select_catalog_role to app_schema;

-- Grant soda_app for this user
grant soda_app to app_schema;

-- Specific grants on shard plsql
```

```
grant execute on exec_shard_plsql to app_schema;
grant gsmadmin_role to app_schema;
grant gsm_pooladmin_role to app_schema;
```

Note the standard database schema privileges and the standard SODA privileges granted to the user. The `exec_shard_plsql` grant, which gives the user the ability to run PL/SQL procedures on a sharded database, is a sharding-specific privilege required for the Sharding/SODA user.

For more information about Oracle Globally Distributed Database schema design, including sharding user creation and running PL/SQL, see Sharded Database Schema Design.

Choosing a Sharding Key

SODA collections are backed by regular Oracle tables. One of the columns in these tables is the ID column, which contains unique keys for the documents in the collection. This column can be used as the sharding key. Alternatively, you can choose a JSON field in the document content to be the sharding key.

The choice of sharding key is application dependent.

The advantages and disadvantages of each sharding key choice are listed in the sections below.

Using the SODA ID as the Sharding Key

The SODA API automatically manages a unique ID for each SODA document. This ID is used by the SODA API to create and retrieve documents within a collection.

The SODA ID must be provided manually by the application when it is used as a sharding key. This is because when creating a new document on a specific shard, the sharding key is required beforehand in order to connect to the appropriate shard. The SODA API allows for this manual (also known as CLIENT key) assignment of a SODA ID on document creation. Examples are provided in the code samples in Using SODA ID as the Sharding Key.

It is up to the application to decide if this SODA ID represents something meaningful (for example, a Customer ID) or is merely a unique Document ID. In any case, the ID must be unique. This is not a requirement imposed by Oracle Globally Distributed Database but by the SODA API.

A summary of using the SODA ID as the sharding key:

- The sharding key must be unique.
- The sharding key is a document ID, which can be independent of the contents of the JSON fields.
- Whenever a new document is inserted, this ID must be provided by the application.

Using a JSON field as the Sharding Key

A JSON field can be used as the sharding key. This key does not need to be unique.

In this case, each document in a collection has a separate SODA ID (as required by SODA), but it is managed automatically by the SODA API as a separate document ID.

A summary of using a JSON field as the sharding key:

- The sharding key does not need to be unique.

- The sharding key is a field within the JSON of each document.
- The SODA ID does not need to be specified when inserting a new document.

Considerations in choosing a Sharding Key method

Note that in both cases, a sharding key is a field which rarely or never changes. This might be a uniquely assigned Customer or Document ID. It can also be a non-unique ID such as a customer birth date, with day, month and year, or a postal code.

For system-managed sharding, either sharding key method is appropriate for distributing documents across shards.

For user-defined sharding, SODA ID as shard key only makes sense if the ID has a meaningful value and it makes sense to partition this by range, for example.

Given no other constraints, using a JSON field as the sharding key offers greater flexibility and allows the sharding key to be stored naturally as part of the JSON.

System-managed vs. User-defined Sharding
Although similar in many ways, user-defined sharding gives you greater control over where data resides. This can be useful when data needs to be separated geographically, or other reasons arise so that data also requires a physical mapping.

Much of the procedures and examples in later topics apply to both sharding methods. There are two exceptions:

1. On creation of the sharded table which underlies the SODA collection, the physical mapping for user-defined sharding must be specified. You can find an example in which a range of ZIP codes must reside on specific shards in Using a JSON Field as a Sharding Key.
2. SODA queries (QBEs) can rely on this data grouping to be able to perform queries on one shard which includes a range of sharding keys.

How to Implement a Solution

After choosing which type of sharding key to use, refer to the following use cases to see examples of how to create a sharded table for the JSON collection, and how to interact with the sharded table from an application.

- Using SODA ID as the Sharding Key
- Using a Sharding Key Other Than the SODA ID

Using SODA ID as the Sharding Key

You can designate the SODA ID as the sharding key when creating the sharded database schema.

The following examples show you how to create a sharded table for the JSON collection, create the SODA mapping, and access the sharded table from an application with Java and Python code samples.

Creating a Sharded Table for the JSON Collection

To create a sharded table that uses the SODA ID as the sharding key:

1. Connect to the shard catalog as the Sharding/SODA user.

2. Enable `SHARD DDL`.
3. Create a tablespace set.
4. Run `CREATE SHARDED TABLE`, as shown in the example below.

The following example creates a sharded table (Customers) for a JSON collection of customer profile documents (CUSTPROFILE).

A column for the SODA ID (ID) identifies the JSON entries, and is also used as the primary key and sharding key. When creating a JSON entry in the table with SODA, the application populates the ID column with a unique value.

The other columns are the default column names given when SODA creates a table to hold an underlying collection. You can see this for yourself when creating a SODA collection and then examining the created table.

Creating a Sharded Table: System-Managed

```
/* Enable shard DDL */
ALTER SESSION ENABLE SHARD DDL;

/* Create a tablespace set */
CREATE TABLESPACE SET TSP_SET_1 USING TEMPLATE
 (datafile size 100m autoextend on next 10M maxsize unlimited
 extent management local segment space management auto);

/* Create the sharded table */
CREATE SHARDED TABLE CUSTOMERS
(
"ID" VARCHAR2(255) NOT NULL,
"CREATED_ON" timestamp default sys_extract_utc(SYSTIMESTAMP) NOT NULL,
"LAST_MODIFIED" timestamp default sys_extract_utc(SYSTIMESTAMP) NOT NULL,
"VERSION" varchar2(255) NOT NULL,
"CUSTPROFILE" JSON,
PRIMARY KEY (ID),
)
TABLESPACE SET TSP_SET_1
PARTITION BY CONSISTENT HASH (ID) PARTITIONS AUTO;
```

Creating a Sharded Table: User-Defined

If the SODA ID has a meaningful value, then the database can be sharded with the user-defined method, and you can create a sharded table using the example below.

Before creating the sharded table in a user-defined sharded database, ensure that the necessary tablespaces and shardspaces have been created. See User-Defined Sharding and Configure the Oracle Globally Distributed Database Topology for details about creating sharded database objects.

```
/* Enable shard DDL */
ALTER SESSION ENABLE SHARD DDL;

/* Create the sharded table */
CREATE SHARDED TABLE CUSTOMERS
```

```
(
"ID" VARCHAR2(255) NOT NULL,
"CREATED_ON" timestamp default sys_extract_utc(SYSTIMESTAMP) NOT NULL,
"LAST_MODIFIED" timestamp default sys_extract_utc(SYSTIMESTAMP) NOT
NULL,
"VERSION" varchar2(255) NOT NULL,
"CUSTPROFILE" JSON,
PRIMARY KEY (ID),
)
PARTITION BY RANGE (ID)
(PARTITION p1 VALUES LESS THAN ('5000') TABLESPACE ts1,
PARTITION p2 VALUES LESS THAN ('10000') TABLESPACE ts2)
```

Creating a Mapped SODA Collection on the Sharded Table

Create a mapped SODA collection to let SODA know which columns to use when working with the sharded table.

In this task, you first run a procedure to create the mapped collection, which creates the metadata necessary for SODA to recognize the previously created table as a SODA collection.

Afterwards you run an additional procedure, `sys.exec_shard_plsql()`, which ensures that the map collection is created on all shards and all future shards.

Creating a SODA Mapped Collection Across All Shards

As the Sharding/SODA user and with `SHARD DDL` enabled, run the following commands on the shard catalog. The shard catalog propagates the procedure to all of the shards to be processed automatically.

```
GRANT SODA_APP TO PROCEDURE APP_SCHEMA.COLLECTION_PROC_CUSTOMERS;

create or replace procedure COLLECTION_PROC_CUSTOMERS AS
METADATA varchar2(8000);
 COL SODA_COLLECTION_T;
 begin METADATA := '{"tableName":"CUSTOMERS",
 "keyColumn":{"name":"ID","assignmentMethod" : "CLIENT"},
 "contentColumn":{"name":"CUSTPROFILE","sqlType":"JSON"},
 "versionColumn":{"name":"VERSION","method":"UUID"},
 "lastModifiedColumn":{"name":"LAST_MODIFIED"},
 "creationTimeColumn":{"name":"CREATED_ON"},
 "readOnly":false}';
-- Create a collection using "map" mode, based on
-- the table you've created above and specified in
-- the custom metadata under "tableName" field.
COL :=
dbms_soda.create_collection('CUSTOMERS',METADATA,DBMS_SODA.CREATE_MODE_
MAP);
end ;
/

exec
sys.exec_shard_plsql('app_schema.collection_proc_customers()',4+1);
```

Note that the keyColumn is mapped as ID, which holds the unique ID of each document. It is designated as CLIENT here because the application will supply a unique key for each document on insert.

At this point, a new collection has been created.

You can run PL/SQL to list the collections. On the shard catalog, run the following commands, and verify that the output lists the CUSTOMERS collection as shown here.

```
SET SERVEROUTPUT ON
DECLARE
l_coll_list SODA_COLLNAME_LIST_T;
BEGIN
l_coll_list := DBMS_SODA.list_collection_names;

IF l_coll_list.COUNT > 0 THEN
FOR i IN 1 .. l_coll_list.COUNT LOOP
DBMS_OUTPUT.put_line(i || ' : ' || l_coll_list(i));
END LOOP;
END IF;
END;
/
1 : CUSTOMERS

PL/SQL procedure successfully completed.
```

Code Samples

The following code samples in Java and Python show you how to connect to a shard using the sharding key and insert a new document.

Note that when using SODA in a sharded database environment, new documents should be created by connecting to specific shards, and not using the shard catalog.

Java Code Sample

These Java code samples are created for the "Using SODA ID as the Sharding Key" use case.

The Java sample below shows you how to connect to a shard and insert a JSON document into the collection.

```
import java.sql.Connection;
import java.util.Properties;
import java.util.List;

// SODA specific imports
import oracle.soda.rdbms.OracleRDBMSClient;
import oracle.soda.OracleDatabase;
import oracle.soda.OracleCursor;
import oracle.soda.OracleCollection;
import oracle.soda.OracleDocument;
import oracle.soda.OracleException;

// Sharding and UCP imports
```

```
import oracle.jdbc.OracleShardingKey;
import oracle.jdbc.OracleType;
import oracle.jdbc.pool.OracleDataSource;
import oracle.ucp.jdbc.PoolDataSourceFactory;
import oracle.ucp.jdbc.PoolDataSource;


/*
* The sample demonstrates connecting to a Sharded Database using
* Oracle JDBC driver and UCP as a client side connection pool.
*/
public class QuickInsertShard {

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

// TNS_ADMIN - Should be the path where the tnsnames.ora file resides
// dbshard_rw - It is the TNS alias present in tnsnames.ora.
// Note that the connection is to the Shard Director (GSM) and the
service name is the shard RW service
final String DB_URL="jdbc:oracle:thin:@dbshard_rw?TNS_ADMIN=/home/opc/
dbhome/";

// Update the Database Username and Password to the Shard User
final String DB_USER = "app_schema";
String DB_PASSWORD = "<user_password>" ;


// Get the PoolDataSource for UCP
PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();

// Set the connection factory first before all other properties
pds.setConnectionFactoryClassName(OracleDataSource.class.getName());
pds.setURL(DB_URL);
pds.setUser(DB_USER);
pds.setPassword(DB_PASSWORD);
pds.setConnectionPoolName("JDBC_UCP_POOL");

// Default is 0. Set the initial number of connections to be created
// when UCP is started.
pds.setInitialPoolSize(10);
// Default is 0. Set the minimum number of connections
// that is maintained by UCP at runtime.
pds.setMinPoolSize(10);
// Instead of Max Pool Size, we can set the number of max connections
per shard
pds.setMaxConnectionsPerShard(20);

// We cannot get the connection until we have the Shard key which is
part of the SQL
//We first set the sharding key or document id explicitly
String shardingKeyVal="10";

// Now we build the connection using this shard key
OracleShardingKey sdkey =
pds.createShardingKeyBuilder().subkey(shardingKeyVal,
```

```
OracleType.VARCHAR2).build();
System.out.println("Initiating UCP and Creating Connection...");
Connection conn = pds.createConnectionBuilder().shardingKey(sdkey).build();

// Enable the SODA Shared Metadata cache
Properties props = new Properties();
props.put("oracle.soda.sharedMetadataCache", "true");
OracleRDBMSClient cl = new OracleRDBMSClient(props);

// Get a DB Connection for use in SODA
OracleDatabase db = cl.getDatabase(conn);

// Print all the Collections in this DB
List<String> names =  db.admin().getCollectionNames();
for (String name : names)
 System.out.println ("Collection name: " + name);

// Open up the CUSTOMERS Collection
OracleCollection col = db.openCollection("CUSTOMERS");

//For a collection configured with client-assigned document keys,
//you must provide the key for the input document. Build a document with
JSON.
OracleDocument cKeyDoc = db.createDocumentFromString(shardingKeyVal,
"{\"name\": \"Matilda\", \"State\": \"CA\", \"ZIP\":\"94065\"}");

// Insert the document above
//If the key  already identifies a document in the collection
//then this will replace the existing doc.
OracleDocument savedDoc = col.saveAndGet(cKeyDoc);

// Get the document back assuming we only know the key
// We are still connected to the same shard
OracleDocument doc = col.find().key(shardingKeyVal).getOne();
String content = doc.getContentAsString();
System.out.println("Retrieved content is: " + content);

// We are done, so close the connection to the shard
conn.close();

// At this point we could open up a new shard connection using a different
sharding key


 }} // End of QuickInsertShard
```

This Java sample shows how you would perform a multi-shard query.

```
import java.sql.Connection;
import java.util.Properties;
import java.util.List;

// SODA specific imports
import oracle.soda.rdbms.OracleRDBMSClient;
```

```
import oracle.soda.OracleDatabase;
import oracle.soda.OracleCursor;
import oracle.soda.OracleCollection;
import oracle.soda.OracleDocument;
import oracle.soda.OracleException;

// Sharding and UCP imports
import oracle.jdbc.OracleShardingKey;
import oracle.jdbc.OracleType;
import oracle.jdbc.pool.OracleDataSource;
import oracle.ucp.jdbc.PoolDataSourceFactory;
import oracle.ucp.jdbc.PoolDataSource;


/*
* The sample demonstrates connecting to a Sharded Database using
* Oracle JDBC driver and UCP as a client side connection pool.
*/
public class QuickQueryCat {

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

// TNS_ADMIN - Should be the path where the tnsnames.ora file resides
// dbshard_rw - It is the TNS alias present in tnsnames.ora.
// This connection is to the shard director using the catalog service
name.
final String DB_URL="jdbc:oracle:thin:@dbcat?TNS_ADMIN=/home/opc/
dbhome/";

// Update the Database Username and Password to the Shard User
final String DB_USER = "app_schema";
String DB_PASSWORD = "<user_password>" ;


// Get the PoolDataSource for UCP
PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();

// Set the connection factory first before all other properties
pds.setConnectionFactoryClassName(OracleDataSource.class.getName());
pds.setURL(DB_URL);
pds.setUser(DB_USER);
pds.setPassword(DB_PASSWORD);
pds.setConnectionPoolName("JDBC_UCP_POOL");

// Now we get a direct connection to the shard catalog
System.out.println("Initiating UCP and Creating Connection...");
Connection conn = pds.getConnection();

// Enable the SODA Shared Metadata cache
Properties props = new Properties();
props.put("oracle.soda.sharedMetadataCache", "true");
OracleRDBMSClient cl = new OracleRDBMSClient(props);

// Get a DB Connection
OracleDatabase db = cl.getDatabase(conn);
```

```
// Print all the Collections in this DB
List<String> names =  db.admin().getCollectionNames();
for (String name : names)
 System.out.println ("Collection name: " + name);

// Open up the CUSTOMERS Collection
OracleCollection col = db.openCollection("CUSTOMERS");

// Do a search across ALL Shards. In this case all users named Matilda
// Setup the specification and open a cursor
OracleDocument filterSpec = db.createDocumentFromString("{ \"name\" :
\"Matilda\"}");

OracleCursor c = col.find().filter(filterSpec).getCursor();

// Print the results of the query
while (c.hasNext()) {
  OracleDocument resultDoc = c.next();

  // Print the document key and document content
  System.out.println ("Document key: " + resultDoc.getKey() + "\n" +
                      " document content: " +
resultDoc.getContentAsString());
}

// Close the cursor
c.close();

// Here, we could initiate another multi-shard query if desired

// We are done, so close the connection
conn.close();


 }} // End of QuickQueryCat
```

Python Code Sample

This Python sample shows how you can actually work with JSON objects using SODA in a sharded database environment.

To use this sample code in your environment, follow the instructions to install the cx_Oracle module for Python: https://cx-oracle.readthedocs.io/en/latest/user_guide/installation.html

This example shows how to connect to a shard using the sharding key and insert a new document.

Note that when using SODA in a sharded database environment, new documents should be created by connecting to specific shards and not using the shard catalog.

```
# import the cx_Oracle module for Python
import cx_Oracle

# Create a connection pool that will be used for connecting to all shards
```

```
# The components of the dsn are hostname (shard director),
# port (usually 1522), global service (created with GDSCTL)
# The pool is then created and SODA metadata caching is enabled.
dsn=cx_Oracle.makedsn("shard_director_host",1522,service_name="service_
name")
pool=cx_Oracle.SessionPool("app_schema","password",dsn,
soda_metadata_cache=True)

# Connect to a specific shard by using the sharding key, which in this
example is
# set explicitly with "sodaid", but this might be passed in or part of
a loop
# You must know beforehand if you are creating or working with a
document for a specific Customer
#
sodaid="2468"
connection=pool.acquire(shardingkey=[sodaid])

# Set autocommit and open the CUSTOMERS collection
connection.autocommit = True
soda = connection.getSodaDatabase()
collection = soda.openCollection("CUSTOMERS")

# Insert a document
# Because you are specifying the shard key, you must pass that in with
the document (key=custid)
# The value can be a UUID for example but it need not have any
relation to the JSON Content.

content = {'name': 'Matilda', 'State': 'CA', 'ZIP':'94065'}
idcontent=soda.createDocument(content, key=sodaid)
doc = collection.insertOneAndGet(idcontent)

# Fetch the document back by key
doc = collection.find().key(sodaid).getOne()
content = doc.getContent()
print('Retrieved SODA document dictionary is:')
print(content)

# After you have finished, release this connection back into the pool
pool.release(connection)

# If you want to add or work with more customers, start with another
connection
# For example: connection=pool.acquire(shardingkey=["123"]) and so on.

#When you are completely finished working with customers you can shut
down the pool
pool.close()
```

This code sample shows you how to run a multi-shard query to return all customer names whose names begin with an "M".

```
import cx_Oracle

# Create an unpooled connection to the shard catalog
# In general, pooled connections should be used for all connections. This is
shown here only as an example.
# The connect string connects to the shard director, but uses the catalog
service, e.g. GD$catalog.oradbcloud
connection = cx_Oracle.connect("app_schema","password","db_connect_string")

# Open the CUSTOMERS collection
connection.autocommit = True
soda = connection.getSodaDatabase()
collection = soda.openCollection("CUSTOMERS")

# Now query the collection
# It is important to note that this is a query across ALL shards
# In other words, you will get ALL users whose names start with M
documents = collection.find().filter({'name': {'$like':
'M%'}}).getDocuments()
for d in documents:
    content = d.getContent()
    print(content["name"])

# Close the connection
connection.close()
```

Using a JSON Field as a Sharding Key

You can designate a JSON field to be the sharding key when creating your sharded database schema.

The examples in the topics that follow show you how to create a sharded table for the JSON collection, create the SODA mapping, trigger the sharding key column population, and access the sharded table from an application with Java and Python code samples.

Creating a Sharded Table for the JSON Collection

To create a sharded table that uses a sharding key other than the SODA ID:

1. Connect to the shard catalog as the Sharding/SODA user.
2. Enable `SHARD DDL`.
3. Create a tablespace set.
4. Run `CREATE SHARDED TABLE`, as shown in the example below.

The following examples create a sharded table (Customers) for a JSON collection of customer profile documents (CUSTPROFILE).

A column for the SODA ID (ID) identifies the JSON entries. When creating a JSON entry in the table with SODA, the application populates the ID column with a unique value.

A sharding key column (ZIP) is the ZIP code value extracted from the JSON document.

The other columns are the default column names given when SODA creates a table to hold an underlying collection. You can see this for yourself when creating a SODA collection and then examining the created table.

Note that the ID column by itself cannot be the primary key. The PK must be or must include the sharding key, in this case ZIP. In the application examples, both ID and ZIP are used to work with the sharded data. In the example above the PK consists of the sharding key and the SODA ID (ZIP, ID), because ZIP will not be a unique value by itself.

Note that in Oracle 21c, you can use either (ZIP, ID) or (ID, ZIP) as the combined Primary Key. In general, you should expect access to this table to be for these values individually, not as a combination. SODA access for these examples looking for ID and customer queries might be using the JSON field (ZIP in this case), so you will create individual indexes in any case. .

Choosing a good sharding key depends on the usage and application requirements. You can use a unique sharding key, for example a Customer ID, but in that case you could also use the SODA ID to store the sharding key.

Creating a Sharded Table: System-Managed

```
/* Enable shard DDL */
ALTER SESSION ENABLE SHARD DDL;

/* Create a tablespace set */
CREATE TABLESPACE SET TSP_SET_1 USING TEMPLATE
 (datafile size 100m autoextend on next 10M maxsize unlimited
 extent management local segment space management auto);

/* Create the sharded table */
CREATE SHARDED TABLE CUSTOMERS
(
"ID" VARCHAR2(255) NOT NULL,
"CREATED_ON" timestamp default sys_extract_utc(SYSTIMESTAMP) NOT NULL,
"LAST_MODIFIED" timestamp default sys_extract_utc(SYSTIMESTAMP) NOT
NULL,
"VERSION" varchar2(255) NOT NULL,
"ZIP" VARCHAR2(60) NOT NULL,
"CUSTPROFILE" JSON,
PRIMARY KEY (ID,ZIP))
TABLESPACE SET TSP_SET_1
PARTITION BY CONSISTENT HASH (ZIP) PARTITIONS AUTO;
```

Creating a Sharded Table: User-Defined

Ensure that all of the necessary tablespaces and shardspaces have been created.

```
/* Enable shard DDL */
ALTER SESSION ENABLE SHARD DDL;
```

```
/* Create the sharded table */
CREATE SHARDED TABLE CUSTOMERS
(
"ID" VARCHAR2(255) NOT NULL,
"CREATED_ON" timestamp default sys_extract_utc(SYSTIMESTAMP) NOT NULL,
"LAST_MODIFIED" timestamp default sys_extract_utc(SYSTIMESTAMP) NOT NULL,
"VERSION" varchar2(255) NOT NULL,
"ZIP" VARCHAR2(60) NOT NULL,
"CUSTPROFILE" JSON,
PRIMARY KEY (ID,ZIP))
PARTITION BY RANGE (ZIP)
(PARTITION p1 VALUES LESS THAN ('50000') TABLESPACE ts1,
PARTITION p2 VALUES LESS THAN ('99999') TABLESPACE ts2)
```

Creating a Mapped SODA Collection on the Sharded Table

Create a map to let SODA know which columns to use when working with the sharded table, and add the sharded table to the list of collections.

You can run a procedure to create the map, but this procedure also must be run on ALL of the shards in the sharded database. The procedure also needs to be run on any shards added in the future. You can accomplish both of these requirements using a sharding-specific PL/SQL procedure, `sys.exec_shard_plsql()`.

To create a SODA map across all shards:

As the Sharding/SODA user and with `SHARD DDL` enabled, run the following commands on the shard catalog. The shard catalog propagates the procedure to all of the shards to be processed automatically.

```
create or replace procedure COLLECTION_PROC_CUSTOMERS AS
METADATA varchar2(8000);
COL SODA_COLLECTION_T;
begin
METADATA := '{"tableName":"CUSTOMERS",
"keyColumn":{"name":"ID"},
"contentColumn":{"name":"CUSTPROFILE","sqlType":"JSON"},
"versionColumn":{"name":"VERSION","method":"UUID"},
"lastModifiedColumn":{"name":"LAST_MODIFIED"},
"creationTimeColumn":{"name":"CREATED_ON"},
"readOnly":false}';
 -- Create a collection using "map" mode, based on
 -- the table you've created above and specified in
 -- the custom metadata under "tableName" field.
COL :=
dbms_soda.create_collection('CUSTOMERS',METADATA,DBMS_SODA.CREATE_MODE_MAP);
end ;
/

exec sys.exec_shard_plsql('app_schema.collection_proc_customers()',4+1);
```

Note that the keyColumn is ID, the key used by SODA to insert and retrieve collections. There is no reference to the ZIP column because it is not used by SODA in the mapping.

At this point, a new collection has been created just as if you had run a `CREATE COLLECTION` command.

You can run some PL/SQL to list out the collections. On the shard catalog, run the following command, and verify that the output lists the Customers table.

```
SET SERVEROUTPUT ON
DECLARE
l_coll_list SODA_COLLNAME_LIST_T;
BEGIN
l_coll_list := DBMS_SODA.list_collection_names;

IF l_coll_list.COUNT > 0 THEN
FOR i IN 1 .. l_coll_list.COUNT LOOP
DBMS_OUTPUT.put_line(i || ' : ' || l_coll_list(i));
END LOOP;
END IF;
END;
/
1 : CUSTOMERS

PL/SQL procedure successfully completed.

SQL>
```

Creating a Trigger to Populate the Sharding Key

When SODA inserts or updates the document, it automatically populates the underlying table columns described in the collection metadata (that is ID, CUSTPROFILE, LAST_MODIFIED, CREATED_ON, and VERSION). However, you also need to populate the ZIP column, and the value must come from within the JSON document. This is accomplished using a trigger.

Note that this is a BEFORE trigger, which allows you to populate a column even when that column is the primary key.

Run the following statements on the shard catalog as the application schema user. The procedure `sys.exec_shard_plsql` ensures that it is also run on all shards and all future shards.

```
alter session enable shard ddl

create or replace procedure COLLECTION_BF_ZIP_CUSTOMERS AS
begin
EXECUTE IMMEDIATE 'alter session enable shard operations';
EXECUTE IMMEDIATE q'%
Create or Replace TRIGGER CUST_BF_TRIG
BEFORE INSERT or UPDATE on CUSTOMERS
FOR EACH ROW
begin
:new.ZIP := JSON_VALUE(:NEW.CUSTPROFILE, '$.ZIP' error on error error
on empty);
end;
%';
```

```
end;
/

exec sys.exec_shard_plsql('app_schema.collection_bf_zip_customers()',4+1+2);
```

In the example above, ZIP is assumed to be a top-level field in the JSON document. If the value is in a nested field, for example under an ADDRESS field, you must include the field hierarchy, for example `'$.ADDRESS.ZIP'`.

Code Samples

The Java and Python code samples for "Using a JSON Field as the Sharding Key" demonstrate how you can actually work with JSON objects using SODA in a sharded database environment.

In these examples, you connect to a shard using the sharding key and insert a new document.

Note that when using SODA in a sharded database environment, new documents should be created by connecting to specific shards and not using the shard catalog.

Java Code Sample

The Java code sample below shows you how to insert JSON documents in a collection where the data is sharded by a JSON field, ZIP code in this example.

```
import java.sql.Connection;
import java.util.Properties;
import java.util.List;

// SODA specific imports
import oracle.soda.rdbms.OracleRDBMSClient;
import oracle.soda.OracleDatabase;
import oracle.soda.OracleCursor;
import oracle.soda.OracleCollection;
import oracle.soda.OracleDocument;
import oracle.soda.OracleException;

// Sharding and UCP imports
import oracle.jdbc.OracleShardingKey;
import oracle.jdbc.OracleType;
import oracle.jdbc.pool.OracleDataSource;
import oracle.ucp.jdbc.PoolDataSourceFactory;
import oracle.ucp.jdbc.PoolDataSource;


/*
* The sample demonstrates connecting to a Sharded Database using
* Oracle JDBC driver and UCP as a client side connection pool.
*/
public class QuickInsertShardJSONField {

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

```
// TNS_ADMIN - Should be the path where the tnsnames.ora file resides
// dbshard_rw - It is the TNS alias present in tnsnames.ora.
// Note that the connection is to the Shard Director (GSM) and the
service name is the shard RW service
final String DB_URL="jdbc:oracle:thin:@dbshard_rw?TNS_ADMIN=/home/opc/
dbhome/";

// Update the Database Username and Password to the Shard User
final String DB_USER = "app_schema";
String DB_PASSWORD = "<user_password>" ;


// Get the PoolDataSource for UCP
PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();

// Set the connection factory first before all other properties
pds.setConnectionFactoryClassName(OracleDataSource.class.getName());
pds.setURL(DB_URL);
pds.setUser(DB_USER);
pds.setPassword(DB_PASSWORD);
pds.setConnectionPoolName("JDBC_UCP_POOL");

// Default is 0. Set the initial number of connections to be created
// when UCP is started.
pds.setInitialPoolSize(10);
// Default is 0. Set the minimum number of connections
// that is maintained by UCP at runtime.
pds.setMinPoolSize(10);
// Instead of Max Pool Size, we can set the number of max connections
per shard
pds.setMaxConnectionsPerShard(20);

// We cannot get the connection until we have the Shard key which is
part of the SQL
//We first set the sharding key which in our case is the value of the
ZIP code field
String shardingKeyVal="94065";

// Now we build the connection using this shard key
OracleShardingKey sdkey =
pds.createShardingKeyBuilder().subkey(shardingKeyVal,
OracleType.VARCHAR2).build();
System.out.println("Initiating UCP and Creating Connection...");
Connection conn =
pds.createConnectionBuilder().shardingKey(sdkey).build();

// Enable the SODA Shared Metadata cache
Properties props = new Properties();
props.put("oracle.soda.sharedMetadataCache", "true");
OracleRDBMSClient cl = new OracleRDBMSClient(props);

// Get a DB Connection
OracleDatabase db = cl.getDatabase(conn);

// Print all the Collections in this DB
```

```
List<String> names =  db.admin().getCollectionNames();
for (String name : names)
 System.out.println ("Collection name: " + name);

// Open up the CUSTOMERS Collection
OracleCollection col = db.openCollection("CUSTOMERS");

//We do not provide an SODA ID column.
//This is provided by SODA when the document is created
// Note that the ZIP field MUST match what we have specified as the key
OracleDocument cDoc = db.createDocumentFromString("{\"name\": \"Matilda\",
\"State\": \"CA\", \"ZIP\":\"94065\"}");

// Insert the document above
OracleDocument insertedDoc = col.insertAndGet(cDoc);

// Get the document key
String dockey = insertedDoc.getKey();

// Get the document back by key
// We are still connected to the same shard
OracleDocument doc = col.find().key(dockey).getOne();
String content = doc.getContentAsString();
System.out.println("Retrieved content is: " + content);

// We are done, so close the connection to the shard
conn.close();

// At this point we could open up a new shard connection using a different
sharding key


 }} // End of QuickInsertShardJSONField
```

Python Code Sample

This code sample in Python shows how you can actually work with JSON objects using SODA in a sharded database environment.

To use this sample code in your environment, follow the instructions to install the cx_Oracle module for Python: https://cx-oracle.readthedocs.io/en/latest/user_guide/installation.html

In this example, you connect to a shard using the sharding key and insert a new document.

Note that when using SODA in a sharded database environment, new documents should be created by connecting to specific shards and not using the shard catalog.

```
# import the cx_Oracle module for Python
import cx_Oracle

# Create a connection pool that will be used for connecting to all shards
# The components of the dsn are hostname (shard director),
# port (usually 1522), global service (created using GDSCTL)
# We also enable SODA metadata caching
dsn=cx_Oracle.makedsn("shard_director_host",1522,service_name="service_name")
```

```
pool=cx_Oracle.SessionPool("app_schema","password",dsn,soda_metadata_ca
che=True)

# Connect to a specific shard by using the shard key, a ZIP code.
which in this
# example is set explicitly as '94065', but this might be passed in or
part of a loop
# You must know beforehand whether you are creating or working with a
document
# with a specific ZIP code value.
connection=pool.acquire(shardingkey=["94065"])

# set autocommit and open the CUSTOMERS collection
connection.autocommit = True
soda = connection.getSodaDatabase()
collection = soda.openCollection("CUSTOMERS")

# Insert a document
# A system generated SODA key is created by default.
content = {'name': 'Matilda', 'STATE': 'CA', 'ZIP': '94065'}
doc = collection.insertOneAndGet(content)

# The SODA key can now be used to work with this document directly
# We can retrieve it immediately
key = doc.key
print('The key of the new SODA document is: ', key)

# Fetch the document back by this same SODA key.
# This only works because we are still connected to the same shard
doc = collection.find().key(key).getOne()
content = doc.getContent()
print('Retrieved SODA document dictionary is:')
print(content)


# Next, add another customer. We are in the shard containing 94065,
# so we can add a customer with the same ZIP code '94065'
content = {'name': 'Mildred', 'STATE': 'CA', 'ZIP: '94065'}
doc = collection.insertOneAndGet(content)

# Now do a query.
# It is important to note that this query is ONLY executed within this
one shard,
# the shard which contains the part of the sharded table with 94065
ZIP codes.
# In other words, the actual query has the additional bound of
customers whose
# names start with 'M' in 94065
# and any other ZIPs stored on this shard. This is unlikely to be a
useful query
# for system-managed sharding.
documents = collection.find().filter({'name': {'$like':
'M%'}}).getDocuments()
for d in documents:
    content = d.getContent()
```

```
    print(content["name"])

# After you have finished, release this connection back into the pool
pool.release(connection)

# If you want to add or work with more customers with a different
# shard key start with another connection
# For example: connection=pool.acquire(shardingkey=["10012"]) and so on.

# When you are completely finished working with customers, shut down the
pool.
pool.close()
```

This code sample shows you how to run a multi-shard query to return all customer names in all shards whose names begin with an "M".

```
import cx_Oracle

# Create an unpooled connection to the shard catalog
# The connect string connects to the shard director, but uses the catalog
service,
# e.g. GD$catalog.oradbcloud
connection = cx_Oracle.connect("app_schema","password","db_connect_string")

# Open the CUSTOMERS collection
connection.autocommit = True
soda = connection.getSodaDatabase()
collection = soda.openCollection("CUSTOMERS")

# Now query the collection
# It is important to note that this is a query across ALL shards
# In other words, you will get ALL users whose name starts with M across ALL
Zip codes
documents = collection.find().filter({'name': {'$like':
'M%'}}).getDocuments()
for d in documents:
    content = d.getContent()
    print(content["name"])

#Close the connection
connection.close()
```

Additional Information About Sharding with SODA

Performance Tuning

Metadata and Statement Caching

For all implementations, statement caching should be turned on the connection pool. This avoids unnecessary round trips to the database.

To turn on SODA metadata caching:

- In Java:

```
Properties props = new Properties();
props.put("oracle.soda.sharedMetadataCache", "true");
OracleRDBMSClient cl = new OracleRDBMSClient(props);
```

  More information is available at SODA Collection Metadata Caching.

- In Python:

```
# Create the session pool
pool = cx_Oracle.SessionPool(user="hr", password=userpwd,
                dsn="dbhost.example.com/
orclpdb1",soda_metadata_cache=True)
```

  More information is available at Using the SODA Metadata Cache

Threading

For optimal use of resources, an instantiation of OracleClient is only required once as it is shared among threads.

The objects obtained from it, such as OracleDatabase and consequently OracleCollection are not thread-safe and do need to be instantiated when creating new requests.

Index Creation and Management

The shard key must be part of the Primary Key. There are no restrictions on creating additional indexes.

All of the guidelines provided by the SODA documentation on creating and managing indexes continue to apply.

Scaling Out Shards

When adding a new shard to the database configuration, all of the DDL, including the SODA metadata and triggers, are automatically available on the new shard.

No extra configuration is required for SODA/JSON Sharding.

15

Achieving Data Sovereignty with Oracle Globally Distributed Database

The proliferation of cloud computing has brought heightened concerns about industry-standard regulations especially around protecting data and its privacy. Today, most organizations want to know where their data is stored, and who has access to it. This creates a key concern about managing data residency—the requirement that data be stored in a specific geographic location.

There are more than 120 countries already engaged in some form of international privacy laws for data protection to ensure that citizens' data are offered more rigorous protections and controls, be it on-premises or on cloud.

Overview of Data Sovereignty

Data sovereignty generally refers to how data is governed by regulations specific to the region in which it originated. These types of regulations can specify where data is stored, how it is accessed, how it is processed, and the life-cycle of the data.

With the exponential growth of data crossing borders and public cloud regions, more than 100 countries now have passed regulations concerning where data is stored and how it is transferred. Personally identifiable information (PII) in particular increasingly is subject to the laws and governance structures of the nation in which it is collected. Data transfers to other countries often are restricted or allowed based on whether that country offers similar levels of data protection, and whether that nation collaborates in forensic investigations.

Data sovereignty requirements are driven by local regulations which could result in different application architectures. A few of them are:

- Data must be physically stored in a certain geographic location. For example, within the boundaries of a specific country or a region comprising of several countries. It is fine to access and process the data remotely so far as the data is not stored in remote locations. From a technical standpoint, this implies that data stores like databases, object stores, and messaging stores that physically store the persistent data must be in a certain geographic location. However, the application run time which has business logic for processing of data could be outside the geographic location. Examples of such applications parts include application servers, mobile applications, API Gateways, Workflows, and so on.
- Data must be physically stored and processed in a certain geographic location: In this case, storing of data and processing of data must take place within the defined geographic location.

Benefits of Implementing Data Sovereignty with Oracle Globally Distributed Database

Oracle Globally Distributed Database meets data sovereignty requirements and supports applications that require low latency and high availability.

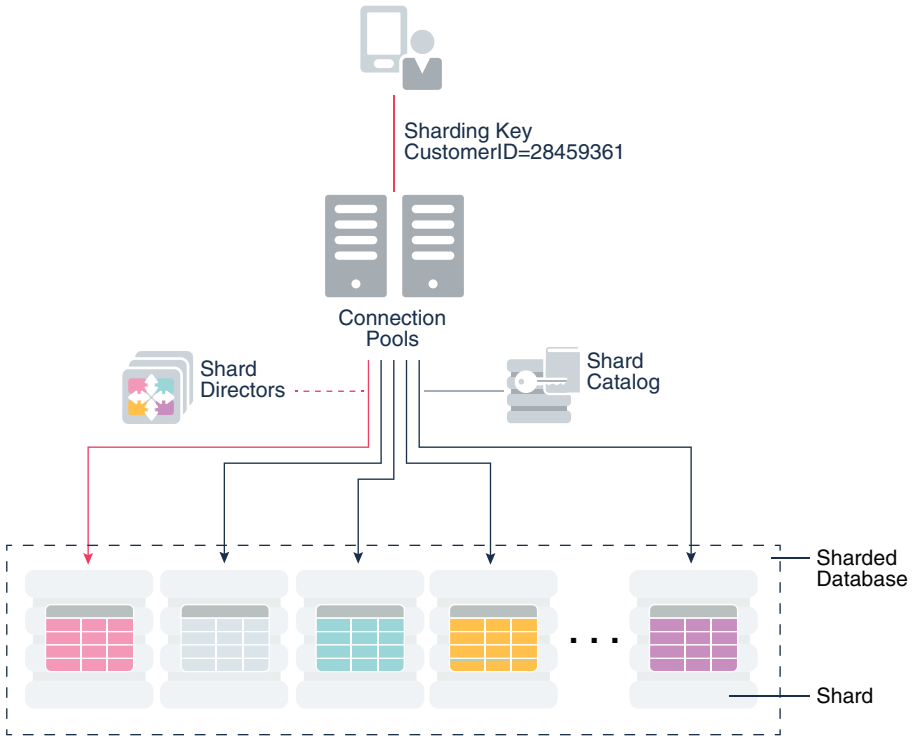
- Sharding makes it possible to locate different parts of the data in different countries or regions – thus satisfying regulatory requirements where data has to be located in a certain jurisdiction.
- It also supports storing particular data closer to its consumers. Oracle Globally Distributed Database automates the entire lifecycle of a sharded database – deployment, schema creation, data-dependent routing with superior run-time performance, elastic scaling, and life-cycle management.
- It also provides the advantages of an enterprise RDBMS, including relational schema, SQL, and other programmatic interfaces, support for complex data types, online schema changes, multi-core scalability, advanced security, compression, high-availability, ACID properties, consistent reads, developer agility with JSON, and much more.

Implementing Data Sovereignty with Oracle Globally Distributed Database

Oracle Globally Distributed Database distributes segments of a data set across many databases (shards) on different computers, on-premises, or in the cloud. These shards can be deployed in multiple regions across the globe. This enables Oracle Globally Distributed Database to create globally distributed databases honoring data residency.

All of the shards in a given database are presented to the application as a single logical database. Applications are seamlessly connected to the right shard based on the queries they run. For example, if an application instance deployed in the US needs data that resides in Europe, the application request is seamlessly routed to an EU data center, without the application having to do anything special.

Figure 15-1 Oracle Globally Distributed Database Architecture



Additionally, Oracle Database security features such as Real Application Security (RAS), Virtual Private Database (VPD), and Oracle Database Vault can be used to limit data access further, even within a region. For example, an administrator in the EU region can further be restricted to see data only from a subset of countries and not all EU countries. Within a Data Sovereignty region, data can be replicated across multiple data centers using Oracle Data Guard.

Oracle Globally Distributed Database management interfaces give you control of the global metadata and provide a view of the physical databases (replicas), data they contain, replication topology, and more. Oracle Globally Distributed Database handles data redistribution when nodes are added or dropped.

You can access worldwide reporting without actually copying the data from the various regions. Sharding can run multi-shard reports without copying any data from any region. Oracle Globally Distributed Database pushes queries to the nodes where the data resides.

Oracle Globally Distributed Database provides comprehensive data sovereignty solutions that focus on the following aspects:

- Data Residency: Data can be distributed across multiple shards, which can be deployed in different geographical locations.
- Data Processing: Application requests are automatically routed to the correct shard irrespective of where the application is running.
- Data Access: Data access within a region can be restricted further using the Virtual Private Database capability of Oracle Database.
- Derivative Data: Ensuring that the data is stored in an Oracle Database, and using Oracle Database features to contain the proliferation of derivative data.
- Data Replication: Oracle Globally Distributed Database can be used with Oracle Data Guard to replicate data within the same Data Sovereignty region.

Data Sovereignty Use Case

A large but imaginary financial institute, Shard Bank, wants to offer credit services to users in multiple counties. Each country where credit service will be provided has its own data privacy regulations and the Personally Identifiable Information (PII) data have to be stored in this country.

The access to the data has to be limited and data administrators in one country cannot see data in others. The solution for this use case is user-defined Sharding with shards configured in different countries and Real Application Security (RAS) or Virtual Private Database (VPD) for data access control.

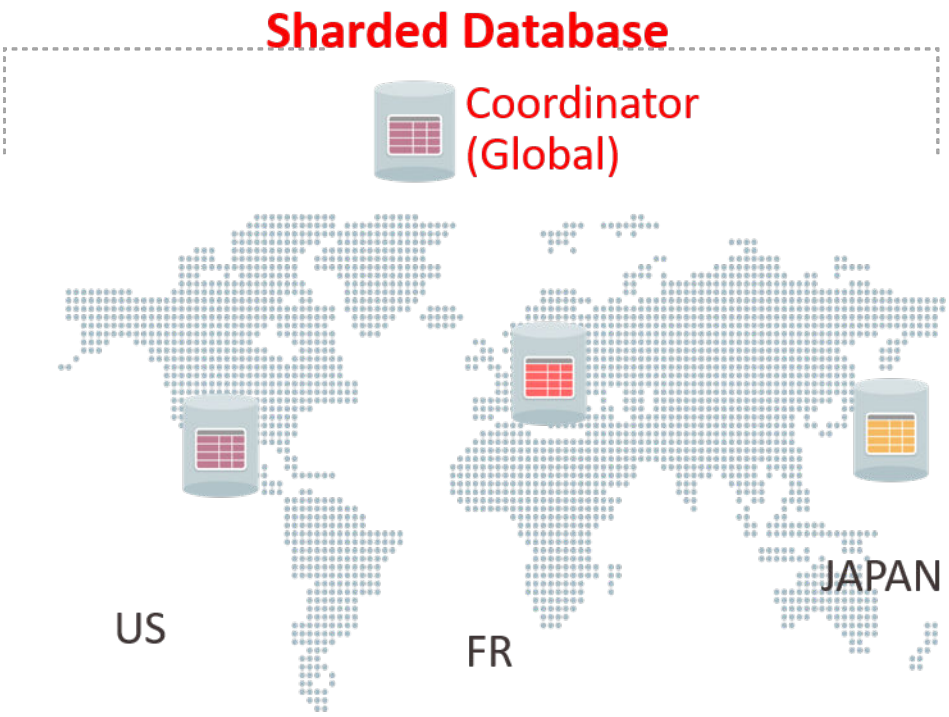
Overview of the Data Sovereignty Solution

This data sovereignty solution provides you with in-country data storage, and still supports a global view of all the data.

The example below demonstrates a hybrid Oracle Globally Distributed Database user-defined deployment between OCI data centers and on-premises across multiple regions. In this configuration, you can store and process all data locally. Each database (in each sovereign region) is made into a shard and the shards belong to a single sharded database. Oracle Globally Distributed Database allows you to query data in one shard (within one

country), and Oracle Globally Distributed Database supports multi-shard queries (that can query data from all the countries).

Figure 15-2 Sharded Database



The global sharded database is sharded by a key indicating the country in which it must reside. In-country applications connect to the local database as usual, and all data is stored and processed locally.

Any multi-shard queries are directed to the shard coordinator. The coordinator rewrites the query and sends it to each shard (country) that has the required data. The coordinator processes and aggregates the results from all of the countries and returns result.

Oracle Globally Distributed Database makes this use case possible with the following capabilities:

- Direct-to-shard routing for in-country queries.
- The user-defined sharding method allows you to use a range or list of countries to partition data among the shards.
- Automatic configuration of replication using Oracle Active Data Guard, and constrain the replicas to be in-country.
- Data federation support (starting with Oracle Database 21c) for converting and adding existing databases into a sharded database. For more information, see Combine Existing Non-Sharded Databases into a Federated Sharded Database.

- Automatic derivation of sharding key (starting with Oracle Database 21c).

The benefits of this approach are:

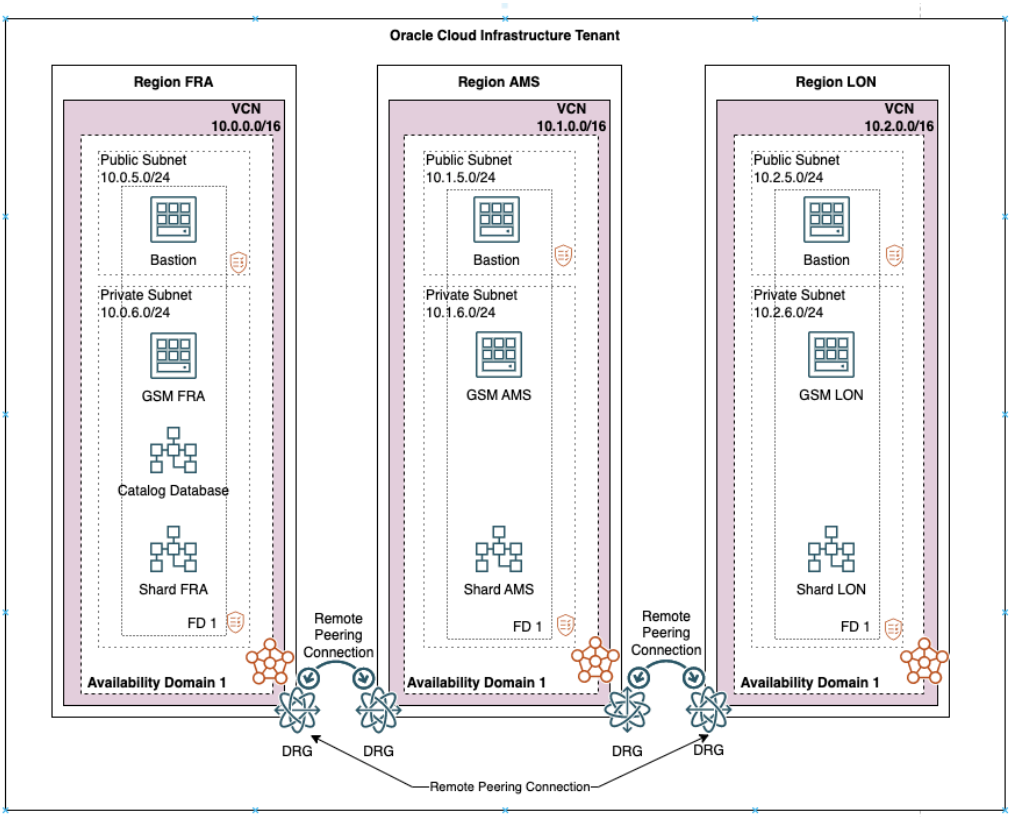
- Each shard can be in a cloud or on-premises within the country.
- Shards can use different cloud providers (multi-cloud strategy) and replicas of a shard can be in a different cloud or on-premises.
- Online resharding allows you to move data between clouds, or to and from the cloud and on-premises.
- Strict enforcement of data sovereignty providing protection from inadvertent cross region data leak.
- Single Multimodel Big Data store with reduced volume of data duplication.
- Better fault isolation as planned/unplanned down time within one region/LOB does not impact other regions/LOBs.
- Ability to split busy partitions and shards as needed.
- Support for full ACID properties is critical for transactional applications.

Deployment Topology for Data Sovereignty

In this example use case, we create a sharded database on Oracle Cloud Infrastructure that spans three regions, Frankfurt (Region1 FRA), Amsterdam (Region 2 AMS), and London (Region 3 LON).

Each region hosts a shard director (Virtual Machine global service manager (GSM)) and one shard (System Database Shard 1, 2, and 3 respectively), and Region 1 (FRA) hosts the shard catalog (System Database GSM Catalog Database).

Figure 15-3 Deployment Topology of Data Sovereignty



Configuring the Data Sovereignty Use Case

Configure the Oracle Globally Distributed Database Data Sovereignty use case by performing the steps given in the following topics.

Configuring VCN Networks in All Three OCI Regions

In Oracle Cloud Infrastructure (OCI), a virtual cloud network is a virtual version of a traditional network on which your instances run. Deploy and configure a virtual cloud network (VCN) in each of our regions (FRA, AMS, and LON).

In each region, create a VCN with two subnets: public and private.

1. Create new route table for private subnet and associate it with private subnet. The default route table should only be used for the public subnet and the private subnet should have a dedicated private route table.
2. Create an internet gateway and associate it with default route table.
3. Create a Network Address Translation (NAT) gateway, Service Gateway, and associate it with route table for private subnet.

- VCN Name/CIDER: Sharding VCN FRA 10.0.0.0/16
- Public Subnet name/CIDER: public_fra 10.0.5.0/24
- Private Subnet name/CIDER: private_fra 10.0.6.0/24

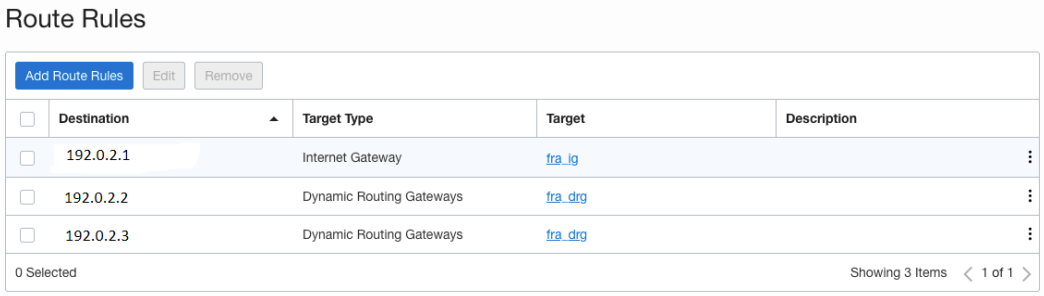
> **Note:**
>
> Repeat the steps in all regions used in the sharding deployment. The subnet CIDER must be different in each region and you must provide region prefix in the VCN/ subnet name.

Configuring Remote VCN Peering Between All Three Regions

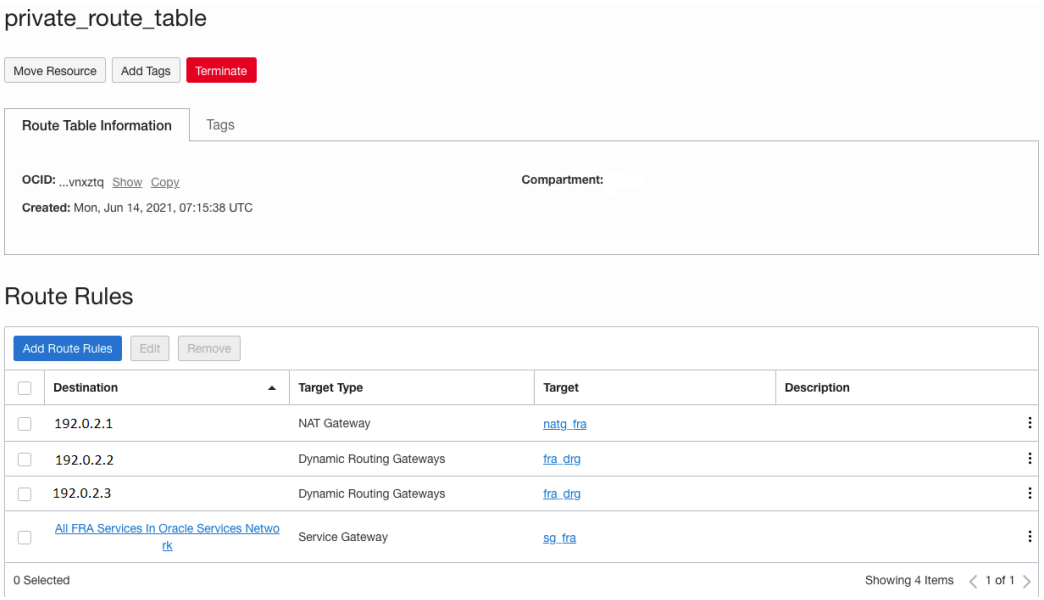
Remote VCN peering is the process of connecting two VCNs in different regions, which allows the VCNs' resources to communicate using private IP addresses without routing the traffic over the internet.

Configure two remote peering connections (RPCs) in each region to connect with the other two regions in the topology.

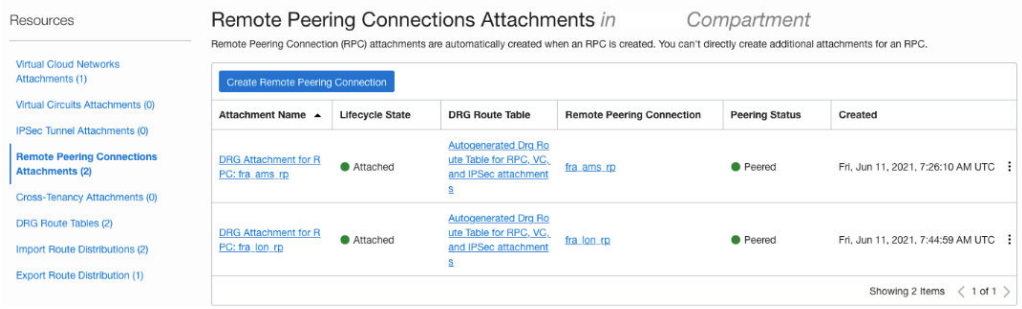
1. See Remote VCN Peering using an RPC for the steps to configure an RPC.
2. Configure routing rules for the public subnet/VCN.



3. Configure routing rules for the private subnet/VCN.



4. Configure security rules.



Configuring Private DNS for Naming Resolution Between the Regions

You create private views for the public and private subnet for each domain in each region, resulting in a total of 6 private zones within 1 zone. Then all entries are added to each private zone configuration.

1. See Private DNS to create and manage private DNS zones.
2. Verify that all names are resolved correctly before you proceed with the next task.

> **Note:**
>
> These steps must be done in each region on all VCNs/VMs so that names can be correctly resolved.

Installing a Global Service Manager in Each Region

Oracle Global Data Services global service manager (GSM) is used by Oracle Globally Distributed Database to route queries from the application to the correct shard in a sharded database.

Download the software and perform the following tasks:

- Download the global service manager (Oracle Database 19c) software into the bastion VM.
- Apply the latest version of OPatch.
- Apply the latest available Oracle Database Bundle Patch on the newly installed global service manager (Oracle Database 19c).

To install a GSM in each region:

1. Create a 200 GB block storage using iSCSI. Configure iSCSI on the OCI Compute for GSM. Mount block storage under `/u01` .

   See Connecting to Volumes With Consistent Device Paths for the mounting block storage process.

2. As the `root` user, install all the required packages.

   ```
   # yum install -y oracle-database-preinstall-19c
   ```

3. As the `root` user, ensure that `/u01` is owned by `oracle:oinstall`.

```
# chown oracle:oinstall /u01
```

4. Download the GSM software to the designated shard director VM and install it in silent mode.

   See Performing a Silent Install of Global Service Manager.

5. Add `gsm home` to `/etc/oratab`.

   ```
   gsm:/u01/app/oracle/product/19.0.0.0/dbhome_1:N
   ```

6. Apply the latest OPatch version.
7. Apply the latest available bundle patch version for Oracle Database 19c.
8. Open GSM port on Firewall:

   ```
   $ systemctl start firewalld.service
   $ systemctl enable firewalld.service
   $ firewall-cmd --permanent --zone=public --add-port=1522/tcp # firewall-
   cmd --reload
   $ firewall-cmd --permanent --zone=public --list-ports
   1522/tcp 22/tcp
   ```

9. Ensure that the required port is open on security lists assigned to GSM VMs to allow applications to connect to GSM.

Collecting TNS entries for Shard Catalog and Sharded Databases

The collection of TNS entries is required to prepare GSM server for configuration of shard catalog and shard databases. The shard catalog requires access only to PDB that stores the shard catalog objects. However for the shard database, prepare the entries for each shared CDB and PDB that stores the application schemas.

1. Prepare the `tnsnames` entries to access the shard catalog database and all shards (Shard Catalog and Shards).
2. Add these entries to `$ORACLE_HOME/network/admin/tnsnames.ora` on the GSM VMs.

> **Note:**
>
> Use FQDN for hostnames in connection strings.

```
db_unique_name =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = host_name_fqdn)(PORT = 1521))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = cdb_service_name)
    )
  )


pdb_name =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = host_name_fqdn)(PORT = 1521))
```

```
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = pdb_service_name)
    )
  )
```

Configuring the Shard Catalog

The shard catalog manages the metadata for Oracle Globally Distributed Database. Configure a database on Region 1 (FRA) which will be the shard catalog database.

1. Connect to all DBCS instances and update `sqlnet` encryption algorithms configured in `sqlnet.ora` file and add the `RC4_256` encryption method as a supported algorithm for client and server.

   > **Note:**
   >
   > The patch is required to enable the AES encryption as the AES encryption is not supported by default by GSM: `Enh 29496977 - GDS ONLY USES RC4_256 TYPE ENCRYPTION`. To enable the AES encryption, apply the patch in Oracle Database 19c. However, this patch is not required in Oracle Database 21c.

   > **Note:**
   >
   > The `RC4_256` algorithm is required only for Oracle Database 19c.

2. Configure the shard catalog database with requirements for Oracle Globally Distributed Database.

   ```
   SQL> alter system set open_links=16 scope=spfile;
   SQL> alter system set open_links_per_instance=16 scope=spfile;
   SQL> shu immediate
   SQL> startup
   ```

3. Configure users on the shard catalog database.

   ```
   SQL> alter user gsmcatuser account unlock.
   SQL> alter user gsmcatuser identified by password;
   # Switch to PDB dedicated for catalog database
   SQL> alter session set container=catalog_db_pdb;
   SQL> create user mysdbadmin identified by password;
   SQL> grant connect, create session, gsmadmin_role to mysdbadmin;
   SQL> grant inherit privileges on user SYS to GSMADMIN_INTERNAL;
   ```

Configuring the Shard Databases

Configure a database in each region which will be a shard in the sharded database configuration.

1. Connect to all DBCS instances and update `sqlnet` encryption algorithms configured in `sqlnet.ora` file and add the `RC4_256` encryption method as a supported algorithm for client and server.

> **Note:**
>
> The patch is required to enable the AES encryption as the AES encryption is not supported by default by GSM: `Enh 29496977 - GDS ONLY USES RC4_256 TYPE ENCRYPTION`. To enable the AES encryption, apply the patch in Oracle Database 19c. However, this patch is not required in Oracle Database 21c.

> **Note:**
>
> The `RC4_256` algorithm is required only for Oracle Database 19c.

2. Run the following commands:

```
SQL> alter database flashback on;
SQL> alter system set dg_broker_start=true;
SQL> alter user GSMROOTUSER account unlock;
SQL> alter user GSMUSER account unlock;
SQL> alter user GSMADMIN_INTERNAL account unlock;
SQL> alter user GSMROOTUSER identified by password;
SQL> alter user GSMUSER identified by password;
SQL> alter user GSMADMIN_INTERNAL identified by password;
SQL> grant sysdg to gsmuser;
SQL> grant SYSBACKUP to gsmuser;
SQL> grant sysdg to GSMROOTUSER;
SQL> grant SYSBACKUP to GSMROOTUSER;
SQL> alter system set global_names=false;
SQL> shu immediate
SQL> startup
# Switch to PDB used as shared database
SQL> alter session set container= pdb_name;
SQL> grant read,write on directory DATA_PUMP_DIR to GSMADMIN_INTERNAL;
SQL> grant sysdg to gsmuser;
SQL> grant SYSBACKUP to gsmuser;
```

Creating the Oracle Globally Distributed Database

Configure the global service manager listener, create a shard catalog database, and add all of the shards to the configuration. The deployment step configures all shards as a single global database.

1. Configure the shard catalog.

> **Note:**
>
> By default system-managed sharding is configured. If you require any other sharding method, specify it during shard catalog creation.

```
GDSCTL> create shardcatalog -database catalog_pdb_tns_entry -
sharding user -user
      mysdbadmin/password -region region1
```

2. Add the GSM listener and start it. Run the listener from GDSCTL.

```
GDSCTL> add gsm -gsm sharddirector1 -listener 1522 -pwd password -
catalog pdb_tns_entry
      -region region1
```

3. Use the following template to add shards to the configuration. Repeat for each shard database.

   Add shard in FRA:

```
GDSCTL> add invitednode shard_hostname
GDSCTL> add cdb -connect cdb_conn_tns_entry -pwd gsmrootuser_pwd
GDSCTL> add shardspace -shardspace primary_shardspace_fra
GDSCTL> add shard -cdb cdb_conn_string -connect pdb_conn_string
 -shardspace primary_shardspace_fra -pwd gsmuser_pwd -deploy_as
PRIMARY
```

   Add shard in AMS:

```
GDSCTL> add invitednode shard_hostname
GDSCTL> add cdb -connect cdb_conn_tns_entry -pwd gsmrootuser_pwd
GDSCTL> add shardspace -shardspace primary_shardspace_ams
GDSCTL> add shard -cdb cdb_conn_string -connect pdb_conn_string
 -shardspace primary_shardspace_ams -pwd gsmuser_pwd -deploy_as
PRIMARY
```

   Add shard in LON:

```
GDSCTL> add invitednode shard_hostname
GDSCTL> add cdb -connect cdb_conn_tns_entry -pwd gsmrootuser_pwd
GDSCTL> add shardspace -shardspace primary_shardspace_lon
GDSCTL> add shard -cdb cdb_conn_string -connect pdb_conn_string
 -shardspace primary_shardspace_lon -pwd gsmuser_pwd -deploy_as
PRIMARY
```

4. Deploy the sharded database configuration.

   Run the `GDSCTL DEPLOY` command, to get the following output:

```
GDSCTL> deploy
deploy: examining configuration...
deploy: requesting Data Guard configuration on shards via GSM
```

```
deploy: shards configured successfully
The operation completed successfully
```

5. Create global database services on the shards to service incoming connection requests from your application. The global service is an extension to the traditional database service. All the properties of traditional services are supported for global services. For sharded databases additional properties are set for global services. See Create and Start Global Database Services.

   For example, database role, replication lag tolerance, region affinity between clients and shards, and so on. For a read-write transactional workload, create a single global service to access data from any primary shard in a sharded database. For highly available shards using Active Data Guard, create a separate read-only global service.

   ```
   GDSCTL> add service -service oltp_rw_srvc -role primary
   ```

Load the data into the shards using the methods described in Migrating to a Sharded Database

Related Topics

- C.35 create shardcatalog
- Create the Shard Catalog Database

Implementing a Session-Based Application Context Policy

Add row-level data access control on the sharded database in conjunction with the Oracle Database virtual private database (VPD) feature for both single shard queries and multi-shard queries. Oracle Global Data Services global service manager (GSM) is used in Oracle Globally Distributed Database to route queries from the application to the correct shard in a sharded database.

1. Create user accounts and sample tables on the shard catalog.

```
connect / as sysdba
alter session enable shard ddl;
create user bt identified by bt;
grant dba, all privileges to bt;

--CREATE USER sysadmin_vpd IDENTIFIED BY password CONTAINER = CURRENT;
CREATE USER sysadmin_vpd IDENTIFIED BY password ; --CONTAINER = CURRENT;

GRANT CREATE SESSION, CREATE ANY CONTEXT, CREATE PROCEDURE, CREATE
TRIGGER, ADMINISTER DATABASE TRIGGER, ALTER SESSION TO sysadmin_vpd;
GRANT EXECUTE ON DBMS_SESSION TO sysadmin_vpd;
GRANT EXECUTE ON DBMS_RLS TO sysadmin_vpd;

CREATE USER CT identified by ct;
CREATE USER DT identified by dt;
GRANT CREATE SESSION TO CT, DT;

GRANT EXECUTE ON sys.exec_shard_plsql to bt, ct, dt, sysadmin_vpd;

connect bt/bt
create tablespace set ts1 in shardspace shd1;
CREATE SHARDED TABLE customers (custid number, name varchar2(20),
```

```
constraint pk1 primary key(custid)) PARTITION BY CONSISTENT
HASH(custid) PARTITIONS AUTO TABLESPACE SET ts1;
-- user-defined:
-- CREATE SHARDED TABLE customers (custid number primary key, name
varchar2(20)) PARTITION BY RANGE (custid) (PARTITION p1 values less
than (100) TABLESPACE  ts1, PARTITION p2 values less than(200)
TABLESPACE  ts2, PARTITION p3 values less than(300) TABLESPACE
ts11, PARTITION p4 values less than(400) TABLESPACE  ts12);

insert into customers(custid, name) values(1,'CT');
insert into customers(custid, name) values(2,'DT');
insert into customers(custid, name) values(4,'ET');
insert into customers(custid, name) values(5,'FT');
commit;

GRANT READ ON customers TO sysadmin_vpd;

create sharded table orders(oid number not null, custid number not
null, constraint ordfk foreign key(custid) references
customers(custid)) partition by reference(ordfk);
-- user-defined:
-- CREATE SHARDED TABLE orders(oid number not null, custid number
not null, constraint orders_fk1 foreign key(custid) references
customers(custid)) partition by reference(orders_fk1);

insert into orders values(9876, 1);
insert into orders values(8888, 2);
insert into orders values(7777, 2);
insert into orders values(7771, 4);
insert into orders values(7772, 4);
insert into orders values(7773, 5);
commit;

GRANT READ ON orders TO CT, DT;
```

2. Create a database session-based application context.

```
CONNECT sysadmin_vpd/password
CREATE OR REPLACE CONTEXT orders_ctx USING orders_ctx_pkg;
```

3. Create a PL/SQL package to set the application context.

```
CONNECT sysadmin_vpd/password
CREATE OR REPLACE PACKAGE orders_ctx_pkg IS
--  PROCEDURE set_custnum SHARD_ENABLE;
  PROCEDURE set_custnum;
 END;
/
CREATE OR REPLACE PACKAGE BODY orders_ctx_pkg IS
  --PROCEDURE set_custnum SHARD_ENABLE
  PROCEDURE set_custnum
  AS
    custnum NUMBER;
    cnt number;
    cname varchar2(256);
```

```
  BEGIN
    -- workaround for bug 33131789: run a CSQ before SET_CONTEXT
    SELECT count(*) INTO cnt FROM BT.CUSTOMERS;
    SELECT SYS_CONTEXT('USERENV', 'SESSION_USER') INTO cname FROM dual;
    SELECT custid INTO custnum FROM BT.CUSTOMERS WHERE name = cname;
    DBMS_SESSION.SET_CONTEXT('orders_ctx', 'cust_no', custnum);
  EXCEPTION
   WHEN NO_DATA_FOUND THEN NULL;
  END set_custnum;
END;
/
```

4. Create a logon trigger to run the application context PL/SQL package.

```
/* create trigger fails to propagate from catalog.
    CREATE TRIGGER set_custno_ctx_...
    DDL Error: ORA-06550: line 1, column 7:
    PLS-00352: Unable to access another database
\'GDS$CATALOG.SYSLOCLINK\'
    ORA-06512: at "GSMADMIN_INTERNAL.EXECUTECOMMAND", line 166
    ORA-06550: line 1, column 7:
    PLS-00201: identifier \'SYS@GDS$CATALOG.SYSLOCLINK\' must be declared
    ORA-06550: line 1, column 7:
    PL/SQL: Statement ignored
    ORA-06512: at "SYS.DBMS_GSM_FIXED", line 3764
    ORA-06512: at "SYS.DBMS_GSM_FIXED", line 3866
    ORA-06512: at "GSMADMIN_INTERNAL.EXECUTECOMMAND", line 118
    ORA-06512: at "GSMADMIN_INTERNAL.EXECUTEDDL", line
  So we create it on shards as well manually. => Use alter session
enable  shard operations before creating the trigger.
*/
/* execute sys.exec_shard_plsql('CREATE OR REPLACE TRIGGER
set_custno_ctx_trig AFTER LOGON ON DATABASE  BEGIN
sysadmin_vpd.orders_ctx_pkg.set_custnum; END;');
ORA-03753: The procedure cannot be propagated.
*/
-- run on catalog and all shards
CONNECT sysadmin_vpd/password
CREATE OR REPLACE TRIGGER set_custno_ctx_trig AFTER LOGON ON DATABASE
BEGIN  sysadmin_vpd.orders_ctx_pkg.set_custnum; END;
/
```

5. Test the logon trigger.

```
connect dt/dt
SELECT SYS_CONTEXT('orders_ctx', 'cust_no') custnum FROM DUAL;
connect ct/ct
SELECT SYS_CONTEXT('orders_ctx', 'cust_no') custnum FROM DUAL;
/* Example output:
SQL> SELECT SYS_CONTEXT('orders_ctx', 'cust_no') custnum FROM DUAL;
CUSTNUM
--------------------------------------------------------------------------------
------
```

```
2
*/
```

6. On the shard catalog and shards, create a PL/SQL policy function to limit user access to their orders only.

```
/* IF you see following error while propagation of DDL to shards,
create the function on catalog and each shards manually.
    PLS-00352: Unable to access another database
\'GDS$CATALOG.SYSLOCLINK\'
*/
connect sysadmin_vpd/password
CREATE OR REPLACE FUNCTION get_user_orders(
  schema_p   IN VARCHAR2,
  table_p    IN VARCHAR2)
 RETURN VARCHAR2
 AS
  orders_pred VARCHAR2 (400);
  cnum NUMBER;
 BEGIN
  SELECT NVL(SYS_CONTEXT('orders_ctx', 'cust_no'), 0) INTO cnum
FROM dual;
  --orders_pred := 'custid = '||cnum;
  orders_pred := 'custid = SYS_CONTEXT(''orders_ctx'',
''cust_no'')';
 RETURN orders_pred;
END;
/
```

7. Create the new security policy.

```
execute sys.exec_shard_plsql(' SYS.DBMS_RLS.ADD_POLICY
(object_schema => ''BT'',   object_name => ''orders'', policy_name
=> ''orders_policy'',  function_schema => ''sysadmin_vpd'',
policy_function  => ''get_user_orders'', statement_types  =>
''select'',  policy_type => DBMS_RLS.CONTEXT_SENSITIVE,  namespace
=> ''orders_ctx'',  attribute => ''cust_no'')');

-- exec sys.exec_shard_plsql('sys.DBMS_RLS.DROP_POLICY(''BT'',
''orders'', ''orders_policy'')');
-- exec sys.exec_shard_plsql('sys.DBMS_RLS.REFRESH_POLICY(''BT'',
''orders'', ''orders_policy'')');
```

8. Test the new policy.

```
connect ct/ct
select * from bt.orders;
connect dt/dt
select * from bt.orders;
/*
connect dt/dt
SQL> select * from bt.orders;
       OID     CUSTID
---------- ----------
      8888        2
```

```
      7777         2

connect ct/ct
SQL> select * from bt.orders;
       OID     CUSTID
---------- ----------
      9876         1
*/
```

16
Troubleshooting

You can enable tracing, locate log and trace files, and troubleshooting common issues.

The following topics describe Oracle Globally Distributed Database troubleshooting in detail:

- Troubleshooting Tips
- Tracing and Debug Information
- Common Error Patterns and Resolutions

Troubleshooting Tips

Use these tips to discover information about the sharded database that you need to help you troubleshoot issues.

Pre-Deployment Network Validation

Several GDSCTL commands have a `-validate_network` option to detect network configuration issues as early as possible during the specification and deployment of sharded databases.

The `-validate_network` can be used in following GDSCTL commands for sharded databases:

- `add {invitednode | invitedsubnet}`
- `add shard`
- `deploy`
- `start gsm`
- `validate` (also includes `-show_errors`)

Checking the Sharding Method

Run `gdsctl config sdb` to check which sharding method, also known as the shard type, is used in the sharded database.

The sharding method can be system-managed, composite, or user-defined.

The sharding method is shown under "Shard type" in the output of `gdsctl config sdb` as shown here.

```
gdsctl> config sdb


GDS Pool administrators
-----------------------
```

```
Replication Type
------------------------
Data Guard

Shard type
------------------------
System-managed

Shard spaces
------------------------
shd1

Services
------------------------
srv1
```

Checking the Replication Type

Run `gdsctl config sdb` to check which method is used for shard replication in the sharded database.

The replication type is shown under "Replication Type" in the output of `gdsctl config sdb` as shown here.

```
gdsctl> config sdb


GDS Pool administrators
------------------------

Replication Type
------------------------
Data Guard

Shard type
------------------------
System-managed

Shard spaces
------------------------
shd1

Services
------------------------
srv1
```

Table 16-1 Replication types in config sdb output

Replication Type	Value Shown in Output
Oracle Data Guard	Data Guard
Raft	

Checking the Oracle Data Guard Protection Mode

You can run `gdsctl config shardspace` on a given shardspace to check the Oracle Data Guard protection mode in your GDSCTL session, rather than switching to DGMGRL.

Data Guard can be configured in three different protection modes: MaxProtection, MaxAvailability, and MaxPerformance.

The Data Guard protection mode is shown under PROTECTION MODE in the `gdsctl config shardspace` command output, as shown here.

```
GDSCTL> config shardspace -shardspace shd1
Shard Group                   Region                        Role
-----------                   ------                        ----
dbs1                          east                          Primary

PROTECTION_MODE               Chunks
---------------               ------
MaxProtection                 6
```

Checking Which Shards Are Mapped to a Key

You can run `gdsctl config chunks -key` to check which shards are mapped to a sharding key.

Example 1: Single Table Family

In the following example, there is only one table family in the sharded database configuration, and the table is partitioned (sharded) on data type number.

In this example, the user is checking which chunk sharding key value "2" is mapped to. In the output it shows sharding key 2 is mapped to chunk "3" and is present in the database "aime1b".

```
GDSCTL> config chunks -key 2
Range Definition
------------------------
Chunks    Range Definition
------    ----------------
3         1431655764-2147483646

Databases
------------------------
aime1b
```

Similarly, this can be done for any data type sharding is done on. Also, a multiple column sharding key can be checked with comma separated values.

The range definition is the range of hash values and can be ignored.

Example 2: Multiple Table Families

In a multiple table family configuration, add the option `-table_family` to specify the table family to which the specified sharding key belongs.

The `config chunks` command lists shards from all shardgroups in the topology. This example also lists a Data Guard standby shardgroup, as shown by the addition of "aime1e" to the Databases (shards) list.

```
GDSCTL> config chunks -key 1 -table_family testuserfam3.customersfam1

Range Definition
------------------------
Chunks    Range Definition
------    ----------------
1         0-357913941

Databases
------------------------
aime1b
aime1e
```

Example 3: Specifying a Multiple Column Sharding Key

When a table is sharded by multiple columns, specify the sharding key value as a comma-separated list as shown here.

```
GDSCTL> config chunks -key 10,mary,2010-04-04

Range Definition
------------------------
Chunks    Range Definition
------    ----------------
4         1288490187-1717986916

Databases
------------------------
aime1b
aime1e
```

Checking Shard Operation Mode (Read-Only or Read-Write)

You can check whether shards are running in read-only or read-write mode by running `gdsctl config chunks -cross_shard`.

The `gdsctl config chunks -cross_shard` command output shows which shards, listed under "Database", are running in read-only and read-write modes, as shown below. The command also lists the chunk ranges on those shards.

```
gdsctl config chunks -cross_shard

Read-Only cross shard targets
------------------------
Database                      From To
--------                      ---- --
tst3b_cdb2_pdb1               1    3
tst3c_cdb3_pdb1               9    10
tst3d_cdb2_pdb1               4    5
tst3e_cdb3_pdb1               6    8
```

```
Chunks not offered for cross-shard
------------------------
Shard space    From To
-----------    ---- --

Read-Write cross-shard targets
------------------------
Database                      From To
--------                      ---- --
tst3b_cdb2_pdb1               1    5
tst3c_cdb3_pdb1               6    10

Chunks not offered for Read-Write cross-shard activity
------------------------
Data N/A
```

Checking DDL Text

Run `gdsctl show ddl -ddl ddl_id` to get the text for the specified DDL.

The DDL numeric identifier is specified with `-ddl ddl_id` to get the text and other details of a particular DDL, as shown here.

```
gdsctl show ddl -ddl 5

DDL Text: CREATE SHARDED TABLE Customers ( CustNo NUMBER NOT NULL, Name
VARCHAR2(50), Address VARCHAR2(250), Location VARCHAR2(20), Class
VARCHAR2(3), CONSTRAINT RootPK PRIMARY KEY(CustNo)) PARTITION BY CONSISTENT
HASH (CustNo) PARTITIONS AUTO TABLESPACE SET ts1
Owner: TESTUSER1
Object name: CUSTOMERS
DDL type: C
Obsolete: 0
Failed shards:
```

> **Note:**
>
> The `show ddl` command output might be truncated. You can run `SELECT ddl_text FROM gsmadmin_internal.ddl_requests` on the shard catalog to see the full text of the statements.

Checking Chunk Migration Status

Run `gdsctl config chunks -show_reshard` to check the status of chunk migration.

A chunk move is a long running operation, whether user-initiated or internal (during incremental deploy), so if you need to check the status, the `gdsctl config chunks -show_reshard` provides the following status indicators as the move progresses.

- empty - indicates no chunk migration in progress

- scheduled - chunk is pending movement, which could be because it is waiting on another chunk move to complete, or the move didn't initiate due to some error
- running - current in progress
- failed - chunk move failed. Check GSM traces and source and target database traces for details.

In the following example, chunk move status is shown in the "Ongoing chunk movement" table in the command output.

```
gdsctl config chunks -show_reshard

Chunks
------------------------
Database                       From      To
--------                       ----      --
tst3b_cdb2_pdb1                1         6
tst3c_cdb3_pdb1                7         10
tst3d_cdb2_pdb1                1         6
tst3e_cdb3_pdb1                7         10

Ongoing chunk movement
------------------------
Chunk     Source                       Target
status
-----     ------                       ------
------
7         tst3c_cdb3_pdb1              tst3b_cdb2_pdb1
Running
8         tst3c_cdb3_pdb1              tst3b_cdb2_pdb1
scheduled
9         tst3c_cdb3_pdb1              tst3b_cdb2_pdb1
scheduled
10        tst3c_cdb3_pdb1              tst3b_cdb2_pdb1
scheduled
```

Checking Table Type (Sharded or Duplicated)

You can check whether tables are sharded or duplicated in dba/all/user_tables using `SELECT TABLE_NAME,SHARDED,DUPLICATED FROM user_tables;`.

In the following example, column "S" indicates whether a table is sharded, and column "D" indicates whether a table is duplicated.

```
SQL> select TABLE_NAME,SHARDED,DUPLICATED from user_tables;

TABLE_NAME      S D
--------------- - -
CUSTOMERS       Y N
DUP1            N Y
LINEITEMS       Y N
MLOG$_DUP1      N N
ORDERS          Y N
```

Checking User Type (Local or ALL_SHARD)

You can find out which users are created as local users and which are sharded database users by selecting the username and ALL_SHARD column in dba/all/user_users.

```
SQL>  select USERNAME,ALL_SHARD from users_users where username='TESTUSER1';

USERNAME        ALL_SHARD
--------------- ---------
TESTUSER1       YES
```

Identifying Tables Created as Sharded Tablespaces

You can find out whether tablespaces are used for a sharded table by selecting the TABLESPACE_NAME and CHUNK_TABLESPACE columns in dba/all/user_tablespaces.

The value in the CHUNK_TABLESPACE column is Y in dba/all/user_tablespaces if it is a tablespace for a sharded table.

```
SQL> select TABLESPACE_NAME,CHUNK_TABLESPACE from user_tablespaces;

TABLESPACE_NAME                C
------------------------------ -
SYSTEM                         N
SYSAUX                         N
TEMP                           N
SYSEXT                         N
TS1                            Y
```

Checking if Shard DDL is Enabled or Disabled

You can check if Shard DDL is enabled or disabled in the current SQL session.

These examples show the result of checking Shard DDL status after enabling and disabling Shard DDL.

```
SQL> alter session enable shard ddl;

Session altered.

SQL> select shard_ddl_status from v$session where AUDSID =
userenv('SESSIONID');

SHARD_DD
--------
ENABLED


SQL> alter session disable shard ddl;

Session altered.
```

```
SQL> select shard_ddl_status from v$session where AUDSID =
userenv('SESSIONID');

SHARD_DD
--------
DISABLED
```

Filtering Data by Sharding Key

You can set the `SHARD_QUERIES_RESTRICTED_BY_KEY` parameter to enable or disable data filtering by a specified sharding key.

The parameter `SHARD_QUERIES_RESTRICTED_BY_KEY` can be set with `ALTER` at the system or session level. If enabled, DMLs will only display select data for specified `SHARDING_KEY` set in the client connection.

In the following example, the client connection is established with a shard with `SHARDING_KEY` specified as "1". However, when the client runs a `SELECT` on the customers table, all of the rows in that table in the shard are displayed.

```
connection established for client with sharding_key=1

SQL> select * from customers order by custno;

    CUSTNO NAME       ADDRESS    LOCATION   CLA
---------- ---------- ---------- ---------- ---
         1 John       Oracle KM  Bangalore  A
        50 Larry      Oracle HQ  SFO        B

2 rows selected.

SQL>
```

Now, as shown below, we enable session level filtering, and the result of the same `SELECT` statement is restricted to only the single row that matches the `SHARD_KEY` specified in the client connection.

```
SQL> alter session set shard_queries_restricted_by_key = true;

Session altered.

SQL> select current_shard_key from dual;

CURRENT_SHARD_KEY
-----------------
                1

1 row selected.

SQL> select * from customers;

    CUSTNO NAME       ADDRESS    LOCATION   CLA
```

```
---------- ---------- ---------- ---------- ---
         1 John       Oracle KM  Bangalore  A
```

Gathering Optimizer Statistics on Sharded Tables

You can gather statistics on sharded tables from the coordinator database.

The statistic preference parameter `COORDINATOR_TRIGGER_SHARD`, when set to `TRUE` on all of the shards, allows the coordinator database to import the statistics gathered on the shards.

The PL/SQL procedures `DBMS_STATS.GATHER_SCHEMA_STATS()` and `DBMS_STATS.GATHER_TABLE_STATS()` gather statistics on sharded tables and duplicated tables in the shards and in the coordinator database. See also, REPORT_GATHER_TABLE_STATS Function.

Manual Statistics Gathering

1. Set `COORDINATOR_TRIGGER_SHARD` to `TRUE` on all of the shards.

   This step is performed only one time and only on the shards. If, for example, you have a schema named `sharduser`:

   ```
   connect / as sysdba
   EXECUTE
   DBMS_STATS.SET_SCHEMA_PREFS('SHARDUSER','COORDINATOR_TRIGGER_SHARD','TRUE'
   );
   ```

2. Gather statistics across the shards.

   The user should be an all-shards user and needs to have privileges to access dictionary tables.

   a. On the shards run the following.

      ```
      connect sharduser/password
      EXEC DBMS_STATS.GATHER_SCHEMA_STATS(ownname => 'SHARDUSER', options
      => 'GATHER');
      ```

   b. When all shards are completed, to pull aggregated statistics run the following on the coordinator.

      ```
      connect sharduser/password
      EXEC DBMS_STATS.GATHER_SCHEMA_STATS(ownname => 'SHARDUSER', options
      => 'GATHER');
      ```

   c. Check the statistics on all of the shards.

      ```
      connect sharduser/password

      ALTER SESSION SET nls_date_format='DD-MON-YYYY HH24:MI:SS';
        col TABLE_NAME form a40
        set pagesize 200 linesize 200

      SELECT TABLE_NAME, NUM_ROWS, sharded, duplicated, last_analyzed
        FROM user_tables
      ```

```
    WHERE table_name not like 'MLOG%' and table_name not like
  'RUPD%'
    and table_name not like 'USLOG%';
```

Automatic Statistics Gathering

1. Set `COORDINATOR_TRIGGER_SHARD` to `TRUE` on all of the shards.

   This step is performed only one time and only on the shards. If, for example, you have a schema named `sharduser`:

   ```
   connect / as sysdba
   EXECUTE
   DBMS_STATS.SET_SCHEMA_PREFS('SHARDUSER','COORDINATOR_TRIGGER_SHARD',
   'TRUE');
   ```

2. Schedule a job to pull aggregated statistics on the shards and on the coordinator database.

   The user should be an all-shards user and must have privileges to access dictionary tables.

   Start the following job on the shards:

   ```
   connect sharduser/password
   BEGIN
   DBMS_SCHEDULER.CREATE_JOB (
      job_name => 'Gather_Stats_2',
      job_type => 'PLSQL_BLOCK',
      job_action => 'BEGIN DBMS_STATS.GATHER_SCHEMA_STATS(ownname =>
   ''DEMO'', options => ''GATHER''); END;',
      start_date => SYSDATE,
      repeat_interval =>
   'freq=daily;byday=MON,TUE,WED,THU,FRI,SAT,SUN;byhour=14;byminute=10;
   bysecond=00',
      end_date => NULL,
      enabled => TRUE,
      comments => 'Gather table statistics');
   END;
   /
   ```

   After the job on all of the shards is finished, start the following job on the coordinator.

   ```
   connect sharduser/password
   BEGIN
   DBMS_SCHEDULER.CREATE_JOB (
      job_name             => 'Gather_Stats_2',
      job_type             => 'PLSQL_BLOCK',
      job_action           => 'BEGIN
   DBMS_STATS.GATHER_SCHEMA_STATS(ownname => ''DEMO'', options =>
   ''GATHER''); END;',
      start_date           =>  SYSDATE,
      repeat_interval      =>
   'freq=daily;byday=MON,TUE,WED,THU,FRI,SAT,SUN;byhour=15;byminute=10;
   bysecond=00',
   ```

```
   end_date             =>  NULL,
   enabled              =>  TRUE,
   comments             => 'Gather table statistics');
END;
/
```

Generate HTML SQL Monitor Output for a Query Running from the Shard Catalog

To generate an HTML SQL monitor output, you can follow these steps:

1. Add hint to query:

   ```
   SELECT /*+ MONITOR */ ...
   ```

   For example, a cross-shard query from the shard catalog:

   ```
   select /*+ MONITOR */ count(*) from CUSTOMER;
   ```

2. Get `SQL_ID` of the query from `v$sql`.

   ```
   SELECT SQL_ID, SQL_FULLTEXT
   FROM V$SQL
   WHERE UPPER(SQL_FULLTEXT) LIKE '%CUSTOMER%'
   AND LAST_ACTIVE_TIME > sysdate -1
   ORDER BY LAST_ACTIVE_TIME DESC;
   ```

3. Generate a report in a file (for example, `report.html` in either the default or a specific folder with the same or different name).

   ```
   SET LONG 1000000
   SET LONGCHUNKSIZE 1000000
   SET LINESIZE 1000
   SET PAGESIZE 0
   SET TRIM ON
   SET TRIMSPOOL ON
   SET ECHO OFFSET FEEDBACK OFF
   spool report.html
   -- replace sql_id values with sql_id of the query
   SELECT DBMS_SQLTUNE.report_sql_monitor( sql_id => 'dfj5upfq6w50j',
     type => 'ACTIVE', report_level => 'ALL') AS report
   FROM dual;
   spool off;
   ```

4. Find the generated SQL Monitor report and view it in a browser or any HTML viewer tool.

Tracing and Debug Information

You can enable tracing for your sharded database and find information in any of several trace files. GDSCTL also has several commands that can display status and error information.

Enabling Tracing

Enable PL/SQL tracing to track down issues in the sharded database.

To get full tracing, set the `GWM_TRACE` level as shown here. The following statement provides immediate tracing, but the trace is disabled after a database restart.

```
ALTER SYSTEM SET EVENTS 'immediate trace name GWM_TRACE level 7';
```

The following statement enables tracing that continues in perpetuity, but only after restarting the database.

```
ALTER SYSTEM SET EVENT='10798 trace name context forever, level 7'
SCOPE=spfile;
```

It is recommended that you set both of the above traces to be thorough.

To trace everything in the sharded database environment, you must enable tracing on the shard catalog and all of the shards. The traces are written to the RDBMS session trace file for either the GDSCTL session on the shard catalog, or the session(s) created by the shard director (a.k.a. GSM) on the individual shards.

Where to Find Alert Logs and Trace Files

There are several places to look for trace and alert logs in the sharded database environment.

Standard RDBMS trace files located in diag/rdbms/.. will contain trace output.

Output from ‘deploy’ will go to job queue trace files *db_unique_name*_jXXX_*PID*.trc.

Output from other GDSCTL commands will go to either a shared server trace file *db_unique_name*_sXXX_*PID*.trc or dedicated trace file *db_unique_name*_ora_*PID*.trc depending on connect strings used.

Shared servers are typically used for many of the connections to the catalog and shards, so the tracing is in a shared server trace file named *SID*_s00*.trc.

GDSCTL has several commands that can display status and error information.

Use `GDSCTL STATUS GSM` to view locations for shard director (GSM) trace and log files.

```
GDSCTL> status
Alias                     SHARDDIRECTOR1
Version                   18.0.0.0.0
Start Date                25-FEB-2016 07:27:39
Trace Level               support
Listener Log File         /u01/app/oracle/diag/gsm/slc05abw/
sharddirector1/alert/log.xml
Listener Trace File       /u01/app/oracle/diag/gsm/slc05abw/
sharddirector1/trace/
ora_10516_139939557888352.trc
Endpoint summary          (ADDRESS=(HOST=shard0)(PORT=1571)
(PROTOCOL=tcp))
```

```
GSMOCI Version           2.2.1
Mastership               N
Connected to GDS catalog Y
Process Id               10535
Number of reconnections  0
Pending tasks.     Total 0
Tasks in  process. Total 0
Regional Mastership      TRUE
Total messages published 71702
Time Zone                +00:00
Orphaned Buddy Regions:  None
GDS region               region1
Network metrics:
   Region: region2 Network factor:0
```

The non-XML version of the alert.log file can be found in the /trace directory as shown here.

```
/u01/app/oracle/diag/gsm/shard-director-node/sharddirector1/trace/alert*.log
```

To decrypt log output in GSM use the following command.

```
GDSCTL> set _event 17 -config_only
```

Primary shard director (GSM) trace/alert files include status and errors on any and all asynchronous commands or background tasks (move chunk, split chunk, deploy, shard registration, Data Guard configuration, shard DDL processing, etc.)

To find pending AQ requests for the shard director, including error status, use GDSCTL CONFIG.

To see ongoing and scheduled chunk movement, use GDSCTL CONFIG CHUNKS -show_reshard

To see shards with failed DDLs, use GDSCTL SHOW DDL -failed_only

To see the DDL error information for a given shard, use GDSCTL CONFIG SHARD -shard *shard_name*

Common Error Patterns and Resolutions

Troubleshoot common errors in Oracle Globally Distributed Database.

Shard Director Fails to Start

If you encounter issues starting the shard director, try the following:

To start Scheduler you must be inside ORACLE_HOME on each shard server.

```
GDSCTL>start gsm -gsm shardDGdirector
GSM-45054: GSM error
GSM-40070: GSM is not able to establish connection to GDS catalog

GSM alert log, /u01/app/oracle/diag/gsm/shard1/sharddgdirector/trace/
```

```
alert_gds.log
GSM-40112: OCI error. Code (-1). See GSMOCI trace for details.
GSM-40122: OCI Catalog Error. Code: 12514. Message: ORA-12514:
TNS:listener does not
currently know of service requested in connect descriptor
GSM-40112: OCI error. Code (-1). See GSMOCI trace for details.
2017-04-20T22:50:22.496362+05:30
Process 1 in GSM instance is down
GSM shutdown is successful
GSM shutdown is in progress
NOTE : if not message displayed in the GSM log then enable GSM trace
level to 16
while adding GSM itself.
```

1. Remove the newly created shard director (GSM) that failed to start.

   ```
   GDSCTL> remove gsm -gsm shardDGdirector
   ```

2. Add the shard director using trace level 16.

   ```
   GDSCTL> add gsm -gsm shardDGdirector -listener port_num -pwd
   gsmcatuser_password
    -catalog hostname:port_num:shard_catalog_name
    -region region1 -trace_level 16
   ```

3. If the shard catalog database is running on a non-default port (other than 1521), set the remote listener.

   ```
   SQL> alter system set
   local_listener='(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)
   (HOST=hostname)(PORT=port_num)))';
   ```

Tablespace Set Creation Fails

A failure in tablespace set creation may be due to `DB_FILES` parameter set too low.

`DB_FILES` parameter default setting is 200. This may be too low for sharded datbases with a large number of shards and chunks. You may also require a larger number of data files in a Raft replication scenario

To calculate the number of database files created for sharding objects on a given shard:

Sharded database files required = (Number of `CREATE TABLESPACE SET` SQL statements executed using `SHARD DDL`) * (Number of chunks present on the shard + 1)

`DB_FILES` must be set to at least the number of files used by sharding (above) **PLUS** non-sharding database files (system, sysaux, and so on) **PLUS** any extra needed by generic RDBMS code (5); therefore:

`DB_FILES` required in each shard = (Number of sharded database files required, as calculated above) + Number of default database files(6) + 5

To check free `DB_FILES` and parameter setting:

```
SQL> select count(*) from v$datafile;

  COUNT(*)
----------
XxxXX

SQL> show parameter db_files

NAME                     TYPE     VALUE
------------------------------------ -----------
------------------------------
db_files                 integer     200
```

Issues Using Deploy Command

```
GDSCTL> deploy
GSM-45029: SQL error
ORA-29273: HTTP request failed
ORA-06512: at "SYS.DBMS_ISCHED", line 3715
ORA-06512: at "SYS.UTL_HTTP", line 1267
ORA-29276: transfer timeout
ORA-06512: at "SYS.UTL_HTTP", line 651
ORA-06512: at "SYS.UTL_HTTP", line 1257
ORA-06512: at "SYS.DBMS_ISCHED", line 3708
ORA-06512: at "SYS.DBMS_SCHEDULER", line 2609
ORA-06512: at "GSMADMIN_INTERNAL.DBMS_GSM_POOLADMIN", line 14284
ORA-06512: at line 1
```

Solution : Check the $ORACLE_HOME/data/pendingjobs for the exact error. ORA-1017 is thrown if any issues on wallet.

1. On problematic Shard host stop the remote scheduler agent.

   ```
   schagent -stop
   ```

2. rename wallet direcotry on Database home

   ```
   mv $ORACLE_HOME/data/wallet $ORACLE_HOME/data/wallet.old
   ```

3. start the remote scheduler agent and it will create new wallet directory

   ```
   schagent -start
   schagent -status
   echo welcome | schagent -registerdatabase 10.10.10.10 8080
   ```

Issues Moving Chunks

If you encounter issues with `MOVE CHUNK`, try the following:

Issue: Initialization parameter `remote_dependencies_mode` has a default value of `timestamp`; therefore, because `prvtgwmut.plb` is run and `DBMS_GSM_UTILITY` recompiled durning upgrade, `GDSCTL MOVE CHUNK` runs into ORA-04062 errors similar to the following.

```
GSM Errors:
server:ORA-03749: Chunk move cannot be performed at this time.
ORA-06512: at "SYS.DBMS_SYS_ERROR", line 79
ORA-06512: at "GSMADMIN_INTERNAL.DBMS_GSM_DBADMIN", line 5497
ORA-04062: timestamp of package "GSMADMIN_INTERNAL.DBMS_GSM_UTILITY"
has been
changed
ORA-06512: at line 1
ORA-06512: at "GSMADMIN_INTERNAL.DBMS_GSM_DBADMIN", line 5366
ORA-06512: at line 1 (ngsmoci_execute)
```

Workaround 1: Restart the source and target shards after upgrade.

Workaround 2: `ALTER SYSTEM SET remote_dependencies_mode=signature` on both source and target.

Issues with Oracle Database Vault

Do not enable Oracle Database Vault on your sharded databases. Oracle Globally Distributed Database does not support Oracle Database Vault.

Issue During Deployment of Role-Separated Environment

The `GSM-45029: SQL ERROR NO MORE DATA TO READ FROM SOCKET` error occurs when you perform administrative operations for Oracle Globally Distributed Database or for Oracle Global Data Services (GDS) and connect through a listener that runs in the Oracle Real Application Clusters (Oracle RAC) or Oracle Restart account in a role-separated environment. The error occurs where the Oracle RAC or Oracle Restart account is different from the Oracle Database account.

Solution:

Start a listener in the Oracle Database account on the sharded catalog database and on each shard, if it is not already running.

The listener can be used to connect and perform administrative operations.

This listener can also be used when you provide an Oracle Database Transparent Network Substrate (TNS) address, when it is required for administrative commands, such as `add shard`.

17

Oracle Globally Distributed Database Solutions

The following solutions show you how to use Oracle Globally Distributed Database to solve a business problem.

- Combine Existing Non-Sharded Databases into a Federated Sharded Database
- Creating Affinity Between Middle-Tier and Shards

Combine Existing Non-Sharded Databases into a Federated Sharded Database

If you have several database installations in different locations that run the same application, and you want to to include the data from all of them, to run data analytics queries for example, you can combine the independent databases into a sharded database to take advantage of Oracle Globally Distributed Database multi-shard queries.

Overview

About Federated Sharding

Learn what a federated sharding configuration is, why you need it, and how it works.

Federated sharding is a sharded database configuration where the shards consist of independent databases with similar schemas.

Creating a sharded database from independent databases reduces the need to import tons of data into a single location for data analytics.

Consider the following benefits to this approach.

- Create a sharding environment using existing, geographically distributed databases; there is no need to provision new systems.
- Run multi-shard queries; access data from many locations in a single query.

Oracle Globally Distributed Database, in a federated sharding configuration, treats each independent database as a shard, and as such can issue multi-shard queries on those shards.

You can create a federated sharding configuration with minor version mismatches between the shards. For example, one region could be on Oracle Database 21.1 and another could be on Oracle Database 21.3. All database shards and the shard catalog must be on Oracle Database 21c or later.

Federated Sharding Schema Requirements

You can convert existing databases running the same application into a federated sharding configuration, without modifying the database schemas or the application.

However, the databases must have the same schema or minor differences. For example, a table can have an extra column in one of the databases.

An application upgrade can trigger changes in the schema, such as when you add a new table, new column, new check constraint, or/and modify a column data type. When part of an overall federated sharding configuration, Oracle Globally Distributed Database handles the schema differences caused by an application upgrade, as long as the overall schema structure stays the same.

Sharded and Duplicated Tables in a Federated Sharding Configuration

Tables that have *different* sets of data on each of the federated databases are equivalent to the sharded tables in a traditional sharded database. Tables with the *same* content on all of the federated databases are equivalent to the duplicated tables in a traditional sharded database.

When you create the federated sharding configuration, the system assumes that all of the tables are sharded, so you must explicitly mark the tables that must be considered duplicated by the multi-shard query coordinator.

Limitations to Federated Sharding

There are some limitations to creating a federated sharding configuration.

- There is no concept of chunk in a federated sharding configuration, so the `GDSCTL MOVE CHUNK` command is not supported.
- Application sharding key-based routing is not supported.
- The existing databases, before being added to a federated sharding configuration, must be upgraded to Oracle Database 21c or later.
- DDLs, cross-shard insert, update, and delete are not supported from the shard catalog in a federated sharding architecture under `ENABLE SHARD DDL`.

Federated Sharding Security

The database users do not need to exist on all of the federated databases, but the schema owners should exist on all of the databases. The privileges and the passwords of these schema owners can be different. Only common privileges are imported for security.

Creating and Deploying a Federated Sharding Configuration

To deploy a federated sharding environment using existing databases, you define the database layout just as you would for user-defined sharding, using `GDSCTL` commands.

The following is a high level description of the process for creating and deploying a federated sharding configuration.

1. Run the GDSCTL CREATE SHARDCATALOG command with the FOR_FEDERATED_DATABASE option to create the federated sharding configuration
2. Add shard directors to the configuration.
3. Add a shardspace to the configuration. A shardspace is defined as an existing database and its replica.
4. Add a shard by adding the existing database to the shardspace, then run DEPLOY.
5. Run GDSCTL SYNC SCHEMA to compare the schemas in the federated sharding configuration and retrieve the common shared schemas. Use SYNC SCHEMA to inspect and apply the DDLs.
6. Use SQL ALTER TABLE on the shard catalog to convert tables containing the same data across the federated shards to duplicated tables.
7. Prepare the shards in the federated sharding configuration for multi-shard queries.

The following topics describe the federated sharding-specific tasks in detail.

Create the Federated Sharding Configuration

The GDSCTL command CREATE SHARDCATALOG is used to create the federated sharding configuration, with the FOR_FEDERATED_DATABASE option used instead of selecting a sharding method in the SHARDING parameter.

The usage for the GDSCTL command CREATE SHARDCATALOG in creating a federated sharding configuration is similar to how it is used to create the shard catalog in user-defined sharding, except that instead of specifying a sharding method in the SHARDING parameter, you use the FOR_FEDERATED_DATABASE option. That is, the FOR_FEDERATED_DATABASE option is mutually exclusive with the SHARDING option.

```
CREATE SHARDCATALOG -DATABASE connect_identifier
  [-USER username[/password]]
  [-REGION region_name_list]
  [-CONFIGNAME config_name]
  [-AUTOVNCR ON/OFF]
  [-FORCE]
  [-SDB sdb_name]
  [-SHARDSPACE shardspace_name_list]
  -FOR_FEDERATED_DATABASE
```

The CREATE SHARDCATALOG syntax statement above shows which parameters are supported. The parameters not shown are not supported when used with the FOR_FEDERATED_DATABASE sharding method, for example, -AGENT_PASSWORD, REPFACTOR, and the Oracle Data Guard protection mode PROTECTMODE.

> **Note:**
>
> Only Oracle Data Guard replication is supported for federated sharding configurations. Oracle Globally Distributed Database doesn't handle the creation and management of the Data Guard configuration, but you can use Data Guard parameters with the `ADD SHARD` command so that you can add the primary and standbys to see the status in `GDSCTL`.

> **See Also:**
>
> The GDSCTL create shardcatalog topic in *Oracle Database Global Data Services Concepts and Administration Guide* for usage notes and command options.

Retrieve, Inspect, and Apply the DDLs

Run the `GDSCTL SYNC SCHEMA` command in phases to create the schema objects common to the existing databases in the shard catalog.

The `GDSCTL SYNC SCHEMA` syntax shown here illustrates the three phases of the opertion.

```
sync[hronize] schema
  [-schema [schemalist | all] [-retrieve_only] [-restart [-force]]
 | -apply [-skip_first]
 | -show [[-ddl ddlnum] [-count n] | [-failed_only]]]
```

`SYNC SCHEMA` should be run in phases, as described here.

1. **Retrieve Phase**

   Run `SYNC SCHEMA` with the `-retrieve_only` option to inspect and verify the DDLs before they are run on the shard catalog.

   ```
   sync schema -schema schemalist -retrieve_only
   ```

   When `SYNC SCHEMA` is run without `-retrieve_only`, the DDL is retrieved and applied at the same time.

2. **Inspection Phase**

   You can examine the DDL statements and their processing status with the `-show` option. The `-ddl ddlnum` option shows the specified DDL, and the `-count n` option specifies the maximum number of entries to show.

   ```
   sync schema -show -ddl ddlnum -count n
   ```

Or you can use the `-failed_only` option to examine only the errored out statements.

```
sync schema -show -failed_only
```

3. **Apply Phase**

   In the final phase, you run the DDLs on the shard catalog to create the schemas and their objects.

   ```
   sync schema -apply
   ```

   If you get an error in the apply phase, there are a couple of ways to work around it:

   - If you can fix the cause of the error, fix and then retry `SYNC SCHEMA -apply`, which retries the failed DDL.
   - If the DDL cannot be fixed or it is not required, you can run `SYNC SCHEMA -apply -skip_first`, which resumes the apply phase from the point of the DDL failure.

   For security reasons, Oracle Globally Distributed Database doesn't offer a way to edit the DDLs.

4. **Import Incremental Changes**

   If there are changes in the schema at a later point, the previous phases can be run again to import incremental changes. For example, when new objects are added, or a new column is added to a table, which will generate an `ALTER TABLE ADD` statement.

> **See Also:**
>
> The sync schema (synchronize schema) topic in *Oracle Database Global Data Services Concepts and Administration Guide* for more `SYNC SCHEMA` usage notes and option details.
>
> SYNC SCHEMA Operations for information about the tasks performed by `SYNC SCHEMA`

Convert Tables to Duplicated Tables

Use `ALTER TABLE` *`table_name`* `externally duplicated` to mark tables as duplicated in a federated sharding configuration.

Any table created by `SYNC SCHEMA` is considered by the multi-shard query layer as an **externally sharded** table. If the table contains the same data on all of the shards, you can alter the table to **externally duplicated**, so that the multi-shard query retrieves the data from one shard only, even if it is a query on a table with no filter predicates on `ORA_SHARDSPACE_NAME`.

```
ALTER TABLE table_name [externally duplicated | externally sharded]
```

Prepare the Shards For Multi-Shard Queries

Create **all shard** users and use the `ORA_SHARDSPACE_NAME` pseudo-column to perform queries on specific shards.

All Shard Users

Before running multi-shard queries from the shard catalog, you must create **all shard** users and grant them access to the sharded and duplicated tables. These users and their privileges should be created in the shard catalog under `shard DDL enabled`.

Create Shardspace-Specific Queries

A **shardspace** in federated sharding is a set consisting of a primary shard and zero or more standby shards. To filter query results for a particular shard[space], a pseudo-column called `ORA_SHARDSPACE_NAME` is added to every `externally sharded` table. The value of this pseudo column in the tables is the name of the shardspace.

Depending on the value of `MULTISHARD_QUERY_DATA_CONSISTENCY`, the rows can be fetched from the primary or from any of the standbys in the shardspace. To run a multi-shard query on a given shard, you can filter the query with the predicate `ORA_SHARDSPACE_NAME = shardspace_name_shard_belongs_to`.

A query like `SELECT CUST_NAME, CUST_ID FROM CUSTOMER`, where the table CUSTOMER is marked as `externally sharded`, runs on all of the shards.

A query like `SELECT CUST_NAME, CUST_ID FROM CUSTOMER WHERE ora_shardspace_name = 'EUROPE'` runs on the shards belonging to the `shardspace_name` Europe. Depending on the `MULTISHARD_QUERY_DATA_CONSISTENCY` parameter value, the query is run on either the primary shard of the shardspace Europe or on its standbys.

You can join sharded tables from different shardspaces. For example, to find the customers from shardspace Europe with orders in shardspace NA, write a query similar to the following.

```
SELECT order_id, customer_name FROM customers c , orders o WHERE
c.cust_id = o.cust_id and
c.ora_shardspace_name = 'Europe' and o.ora_shardspace_name = 'NA'
```

Querying an `externally duplicated` table, with or without the `ORA_SHARDSPACE_NAME` predicate, should go to only one of the shardspaces. The `MULTISHARD_QUERY_DATA_CONSISTENCY` parameter value determines whether to query a primary shard in the shardspace or its replicas.

Federated Sharding Reference

SYNC SCHEMA Operations

DDL Synchronization

DDL synchronization is an operation that `SYNC SCHEMA` runs just after the deployment of the shards in a federated sharding configuration.

The goal of this operation is to import the object definitions from all of the shards, compare the definitions across the shards, and generate DDLs for the objects that exist on all of the shards (common objects). Once the DDLs are run and the objects are created, you can reference these objects in multi-shard queries.

Import Users

A user or schema is a candidate for import by `SYNC SCHEMA` if it exists on all of the shards and owns importable schema objects.

You can narrow the list of users to be imported by passing a list of users in the `-SCHEMA` parameter. For example,

```
gdsctl> sync schema -schema scott


gdsctl> sync schema -schema scott,myschema
```

For case-sensitive schemas use quoted identifiers.

```
gdsctl> sync schema -schema "O'Brien",scott
```

To include all non-Oracle schemas, use the value `ALL` in the `SCHEMA` parameter.

```
gdsctl> sync schema -schema all
```

Before importing the users, `SYNC SCHEMA` verifies that any discovered users exist on all shards, and no user already exists on the shard catalog with the same name. The users are then created on the shard catalog as local users and they are locked. Because these are local users, they only share the same name with shards and are essentially the same as any other user that may have the same name across different databases. Note that these users are not able to login and issue queries because they are not **all shard** users. To issue multi-shard queries, an all shard user must be created.

> **Note:**
>
> Only users local to a PDB are imported. Common CDB users are not imported.

Grant User Roles and Priviledges

For the imported users, `SYNC SCHEMA` compares users' privileges.

`SYNC SCHEMA` grants only the privileges that are granted on all of the shards (common grants). A user **A** who has a DBA role on shard1, but does not have DBA role on shard2, is not granted the DBA role in the shard catalog.

Import Object Definitions

The objects compared and imported by `SYNC SCHEMA` to the shard catalog are the objects that will be referenced in multi-shard queries or used by multi-shard query processing.

These objects are:

- Tables
- Views and Materialized Views (exported as tables)
- Check Constraints
- Object Types
- Synonyms

Running `SYNC SCHEMA` does not import objects related to storage, or objects that have no impact on multi-shard query processing, such as tablespaces, indexes, indextypes, directories, or zone maps.

Schema Object Comparison

The objects, from one shard to another, can have different definitions. `SYNC SCHEMA` compares the different definitions and creates a common definition to enable multi-shard queries against imported objects.

`SYNC SCHEMA` detects the objects' differences at two levels: number of objects, and object definitions.

First, `SYNC SCHEMA` considers the number of objects. It is likely that, during an application upgrade, some objects are added to the schemas. Only objects that are on all of the shards will be imported into the shard catalog.

Second, the object definitions from one shard to another can have different attributes. For the objects that `SYNC SCHEMA` imports, the following differences are noted:

Differences in Tables

When comparing objects in a federated sharding configuration, some differences in tables have an impact on multi-shard queries and some do not.

Column Differences

Only column differences have an impact on multi-shard queries. `SYNC SCHEMA` addresses only this difference.

- The number of columns can be different.
- The data type of a given column can be different.
- The default value of a given column can be different.
- The expression of a virtual column can be different

When a table has a different numbers of columns, `SYNC SCHEMA` will opt for the creation of a table that contains the union of all of the columns. Taking the union of all of the columns, compared to just taking the intersection, will spare you from re-writing multi-shard queries in case of an incremental deploy, when the added shard has fewer columns than indicated in the shard catalog.

When a column has different data types, `SYNC SCHEMA` defines it as the highest (largest) datatype.

When a column has different data types, and one of the columns is a user-defined object type, then that column is not imported into the shard catalog.

When a column has different default values, `SYNC SCHEMA` sets `NULL` as the default value.

Nested table columns are not imported into the shard catalog.

Example: a Customer table is defined on shard1 and shard2 as shown here.

On shard1:

```
Customer( Cust_id number, Name varchar(30),
    Address varchar(50),Zip_code number)
```

On shard2:

```
Customer( Cust_id varchar(20), Name varchar(30),
    Address varchar(50),Zip_code number,
    Country_code number)
```

Note that the column `Cust_id` is a number on shard1 and a varchar(20) on shard2. Also, note that `Country_code` exists on shard2 but does not exist on shard1.

The Customer table created by `SYNC SCHEMA` in the shard catalog has all of the columns, including `Country_code`, and the `Cust_id` type is varchar(20).

```
Customer( Cust_id varchar(20), Name varchar(30),
    Address  varchar(50),Zip_code number,
    Country_code number)
```

`SYNC SCHEMA` keeps track of these differences between schemas in the shard catalog. A query issued on the catalog database that accesses these heterogeneous columns is rewritten to address the differences before it is sent to the shards. On the shard, if there is a data type mismatch, the data is CAST into the "superior" data type as created on the catalog. If the column is missing on the shard, the default value is returned as set on the catalog.

Partition Scheme Differences

Note that this difference has no impact on multi-shard queries, and is ignored.

- Partitioning column can be different.
- Partition type can be different.
- Number of partitions can be different.

Storage Attribute Differences

Note that this difference has no impact on multi-shard queries, and is ignored.

- Tablespaces, on which the table is created, are different.
- The encryption can be different.
- The `INMEMORY` attribute can be different.

Differences in Views

Views on shards are created and handled as tables in the shard catalog. The same restrictions that apply to tables also apply to views.

Differences in Constraints

Only CHECK constraints are created in the shard catalog. The CHECK constraint condition should be same on all of the shards.

Differences in Object Types

Object types and type bodies are only created if they have the same definition on all of the shards.

Troubleshooting Federated Sharding

Solve common federated sharding issues with these troubleshooting tips.

ORA-03851: Operation not supported in federated database

ORA-03701: Invalid parameter combination: federated database and ...

Some of the operations and command options that apply to a traditional sharded database are not applicable to a federated database. This is because:

- There is no concept of a chunk in a federated database. Any chunk-related operation is invalid, for example `SPLIT CHUNK` and `MOVE CHUNK`.
- The Data Guard broker configuration is not set up or managed by the system in federated database, because the existing shards may already have been set up with their own high availability configurations. Operations such as `SET DATAGUARD_PROPERTY` or `MODIFY SHARDSPACE` are not supported.
- The `CREATE SHARD` command is not supported.

ORA-03885: Some primary shards are undeployed or unavailable

The `SYNC SCHEMA` operation requires that all primary shards be available. Check the output of the `CONFIG SHARD` command, and check the status of all primary shards. Fix any issues and retry the operations when the shards become available.

ORA-03871: Some DDL statements are not applied to the catalog

The `SYNC SCHEMA` operation cannot import object definitions from the shards when some statements from the previous issuance are still not applied on the shard catalog. Run `SYNC SCHEMA` with the `-apply` option to run these statements.

Handling Errors During Multi-Shard Queries

If a multi-shard query fails with this error due to a mismatch of the object definition on the shard and the catalog, make sure that the shard catalog has the latest schema changes imported. Any time there are schema changes in the federated database, you must run `SYNC SCHEMA` to import any changes in the schemas on the shards.

Note that subsequent runs of `SYNC SCHEMA` will not drop and recreate the object, but will generate `ALTER` statements to incorporate the definition changes. This ensures that if there are queries already running during the `SYNC SCHEMA` operation, they won't fail with invalid object errors.

Handling Errors During DDL Processing Phase

If DDL fails on the shard catalog, the status of each DDL can be examined with the `SYNC SCHEMA -show` option.

```
gdsctl> sync schema -show
```

Note: The `SYNC SCHEMA -show` command is different from the command `SHOW DDL`. `SHOW DDL` lists DDL statements run by an all-shard user that are first run on the catalog and then propagated to the shards, whereas `SYNC SCHEMA -show` DDL statements are generated from the objects imported from shards.

By default, `SYNC SCHEMA -show` lists a fixed number of the latest DDLs. The `-count` and `-ddl` options can be used to inspect specific range of DDLs. For example,

```
gdsctl> sync schema -show -count 20
gdsctl> sync schema -show -count 20 -ddl 5
```

To check the complete DDL text and error message, if any, use the `-ddl` option.

```
gdsctl> sync schema -show -ddl 5
```

To list only the failed DDL statements, use the `-failed_only` option.

```
gdsctl> sync schema -failed_only
```

Based on the error message of the failed DDL, fix the cause of the error and perform the apply phase.

```
gdsctl> sync schema -apply
```

The `SYNC SCHEMA` command also has a `-restart` option to perform the complete operation from the beginning as if it were run for the first time. This option will `DROP` all existing schemas imported during all previous runs of `SYNC SCHEMA` and any related metadata. Be aware that this will cause any running queries on these objects to fail.

```
gdsctl> sync schema -restart
```

Creating Affinity Between Middle-Tier and Shards

Middle-tier routing allows smart routers to route to the middle tier associated with a sharding key.

You can use the middle-tier routing API to publish the sharded database topology to the router tier so that requests based on specific sharding keys are routed to the appropriate application middle tier, which in turn establishes connections on the given subset of shards.

In a typical sharded database environment, middle-tier connection pools route database requests to specific shards. This can lead to a situation where each middle-tier connection pool establishes connections to each shard. This can create too many connections to the database. The issue can be solved by creating an affinity between the middle tiers and shards. In this scenario it would be ideal to dedicate a middle tier (web server, application

server) for each data center or cloud, and to have client requests routed directly to the middle tier where the shard containing the client data (corresponding to the client shard key) resides. A common term used for this kind of setup is swim lanes, where each swim lane is a dedicated stack, from web server to application server all the way to the database.

Oracle Universal Connection Pool (UCP) solves this problem by providing a middle-tier routing API which can be used to route client requests to the relevant middle tier. The UCP middle tier API is exposed by the `OracleShardRoutingCache` class. An instance of this class represents the UCP internal shard routing cache, which can be created by providing connection properties such as user, password, and URL. The routing cache connects to the sharding catalog to retrieve the key to shard mapping topology and stores it in its cache.

The routing cache is used by UCP middle-tier API `getShardInfoForKey(shardKey,superShardKey)`, which accepts a sharding key as input and returns a set of `ShardInfo` instances mapped to the input sharding key. The `ShardInfo` instance encapsulates a unique shard name and priority of the shard. An application using the middle-tier API can map the returned unique shard name value to a middle tier that has connections to a specific shard. The routing cache is automatically updated when chunks are split or moved to other shards by subscribing to respective ONS events.

The following code example illustrates the usage of Oracle UCP middle-tier routing API.

Example 17-1 Middle-Tier Routing Using UCP API

```
import java.sql.SQLException;
import java.util.Properties;
import java.util.Random;
import java.util.Set;

import oracle.jdbc.OracleShardingKey;
import oracle.jdbc.OracleType;
import oracle.ucp.UniversalConnectionPoolException;
import oracle.ucp.routing.ShardInfo;
import oracle.ucp.routing.oracle.OracleShardRoutingCache;

/**
 * The code example illustrates the usage of UCP's mid-tier routing
feature.
 * The API accepts sharding key as input and returns the set of
ShardInfo
 * instances mapped to the sharding key. The ShardInfo instance
encapsulates
 * unique shard name and priority. The unique shard name then can be
mapped
 * to a mid-tier server which connects to a specific shard.
 *
 */
public class MidtierShardingExample {

  private static String user = "testuser1";
  private static String password = "testuser1";
```

```
  // catalog DB URL
  private static String url = "jdbc:oracle:thin:@//hostName:1521/
catalogServiceName";
  private static String region = "regionName";

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

  static void testMidTierRouting() throws UniversalConnectionPoolException,
      SQLException {

    Properties dbConnectProperties = new Properties();
    dbConnectProperties.setProperty(OracleShardRoutingCache.USER, user);
    dbConnectProperties.setProperty(OracleShardRoutingCache.PASSWORD,
password);
    // Mid-tier routing API accepts catalog DB URL
    dbConnectProperties.setProperty(OracleShardRoutingCache.URL, url);

    // Region name is required to get the ONS config string
    dbConnectProperties.setProperty(OracleShardRoutingCache.REGION, region);

    OracleShardRoutingCache routingCache = new OracleShardRoutingCache(
        dbConnectProperties);

    final int COUNT = 10;
    Random random = new Random();

    for (int i = 0; i < COUNT; i++) {
      int key = random.nextInt();
      OracleShardingKey shardKey = routingCache.getShardingKeyBuilder()
          .subkey(key, OracleType.NUMBER).build();
      OracleShardingKey superShardKey = null;

      Set<ShardInfo> shardInfoSet = routingCache.getShardInfoForKey(shardKey,
          superShardKey);

      for (ShardInfo shardInfo : shardInfoSet) {
        System.out.println("Sharding Key=" + key + " Shard Name="
            + shardInfo.getName() + " Priority=" + shardInfo.getPriority());
      }
    }

  }
}
```

See Also:

Middle-Tier Routing Using UCP in *Oracle Universal Connection Pool Developer's Guide*

18

Oracle Globally Distributed Database Reference

The following topics provide you with reference information to help you plan, configure, deploy, and manage your sharded database configuration.

- Using GDSCTL with Oracle Globally Distributed Database
- SHARDED_TABLE_FAMILIES

Using GDSCTL with Oracle Globally Distributed Database

Several of the Global Data Services GDSCTL commands are used for setting up and deploying an Oracle Globally Distributed Database configuration. Learn how to use the GDSCTL command-line tool and the Oracle Globally Distributed Database-related GDSCTL commands in the following topics.

GDSCTL Operation

Learn how to start GDSCTL, run commands, and get command help text.

Starting GDSCTL

To start GDSCTL, enter `gdsctl` at the operating system prompt.

```
$ gdsctl
```

GDSCTL starts and displays the GDSCTL command prompt.

```
GDSCTL>
```

Running GDSCTL Commands Interactively

You can run GDSCTL commands interactively at either the operating system prompt or the GDSCTL command prompt.

Run a GDSCTL command at the system prompt.

```
$ gdsctl add gsm -gsm gsm1 -catalog 127.0.0.1:1521:db1
```

Run a GDSCTL command at the GDSCTL command prompt.

```
GDSCTL> add gsm -gsm gsm1 -catalog 127.0.0.1:1521:db1
```

Both of these methods achieve the same result. The command syntax examples in this document use the GDSCTL command prompt.

Running GDSCTL Batch Operations

You can gather all the GDSCTL commands in one file and run them as a batch with GDSCTL.

The following command starts GDSCTL and runs the commands contained in the specified script file.

```
$ gdsctl @script_file_name
```

GDSCTL Help Text

You can display help for GDSCTL and GDSCTL commands.

The GDSCTL `HELP` command displays a summary of all GDSCTL commands.

```
GDSCTL> help
```

If you specify a command name after `HELP`, then the help text for that command is shown.

```
GDSCTL> help start gsm
```

You can also use the `-h` option with any GDSCTL command to show the help text for the specified command.

```
GDSCTL> start gsm -h
```

GDSCTL Connections

Some GDSCTL commands require a connection to the shard catalog, and for ceratin operations, GDSCTL must connect to a shard director.

GDSCTL Shard Catalog Connections

If you run GDSCTL commands that require a connection to the shard catalog, then you must run the GDSCTL `CONNECT` command before the first command that requires the connection.

The `CONNECT` command only needs to be run once in a GDSCTL session.

GDSCTL uses Oracle Net Services to connect to the shard catalog database or another database in the sharded database configuration. For these connections you can run GDSCTL from any client or host that has the necessary network configuration.

Unless specified, GDSCTL resolves connect strings with the current name resolution methods (such as TNSNAMES).

The GDSCTL operations that require a connection to a shard catalog are noted in the usage notes for each command.

GDSCTL Shard Director Connections

For certain operations, GDSCTL must connect to a shard director, also known as global service manager.

Unless specified, GDSCTL resolves connect strings with the current name resolution methods (such as TNSNAMES). However, to resolve the shard director name, GDSCTL queries the `gsm.ora` file.

To connect to a shard director, GDSCTL must be running on the same host as the shard director. When connecting to a shard director, GDSCTL looks for the `gsm.ora` file associated with the local shard director.

The following are the GDSCTL operations that require a connection to a shard director.

- `ADD GSM` adds a shard director.
- `START GSM` starts the shard director.
- `STOP GSM` stops the shard director.
- `MODIFY GSM` modifies the configuration parameters of the shard director.
- `STATUS GSM` returns the status of a shard director.
- `SET INBOUND_CONNECT_LEVEL` sets the `INBOUND_CONNECT_LEVEL` listener parameter.
- `SET TRACE_LEVEL` sets the trace level for the listener associated with the specified shard director.
- `SET OUTBOUND_CONNECT_LEVEL` sets the timeout value for the outbound connections for the listener associated with a specific shard director.
- `SET LOG_LEVEL` sets the log level for the listener associated with a specific shard director.

GDSCTL Commands Used with Oracle Globally Distributed Database

A subset of GDSCTL commands is used with Oracle Globally Distributed Database.

- add cdb
- add credential
- add file
- add gsm
- add invitednode (add invitedsubnet)
- add region
- add service
- add shard
- add shardgroup
- add shardspace
- config
- config backup

- config cdb
- config chunks
- config credential
- config file
- config gsm
- config region
- config sdb
- config service
- config shard
- config shardgroup
- config shardspace
- config table family
- config vncr
- configure
- connect
- create restorepoint
- create shardcatalog
- delete backup
- delete catalog
- deploy
- disable backup
- disable service
- enable backup
- enable service
- list backup
- list restorepoint
- modify catalog
- modify cdb
- modify credential
- modify file
- modify gsm
- modify region
- modify service
- modify shard
- modify shardgroup
- modify shardspace
- move chunk

- relocate service
- recover shard
- remove cdb
- remove credential
- remove file
- remove gsm
- remove invitednode (remove invitedsubnet)
- remove region
- remove service
- remove shard
- remove shardgroup
- remove shardspace
- restore backup
- run backup
- services
- set gsm
- set inbound_connect_level
- set log_level
- set outbound_connect_level
- set trace_level
- split chunck
- sql
- start gsm
- start service
- status backup
- status gsm
- status service
- stop gsm
- stop service
- sync schema (synchronize schema)
- validate backup
- validate catalog

SHARDED_TABLE_FAMILIES

The `SHARDED_TABLE_FAMILIES` public view shows all sharded tables and the corresponding root table and schema names.

Column	Data Type	NULL	Description
TABFAM_ID	NUMBER		This unique table family identifier is a numeric value and each table family is assigned a unique number
ROOT_SCHEMA_NAME	VARCHAR2(128)		The schema owning root (parent) table for a table family
ROOT_TABLE_NAME	VARCHAR2(128)		The root (parent) table name for a table family
SCHEMA_NAME	VARCHAR2(128)		The schema name for tables
TABLE_NAME	VARCHAR2(128)	NOT NULL	The table name

Sample Output

The following is sample output from a query on the `SHARDED_TABLE_FAMILIES` view. In this table family `customers1` is root table, and `orders1` and `lineitems1` are the child tables of `customers1`.

```
SQL> select * from SHARDED_TABLE_FAMILIES order by
TABFAM_ID,ROOT_SCHEMA_NAME,ROOT_TABLE_NAME,SCHEMA_NAME,TABLE_NAME;

 TABFAM_ID ROOT_SCHEMA_NAM ROOT_TABLE_NAME SCHEMA_NAM TABLE_NAME
---------- --------------- --------------- ---------- ---------------
         6 TESTUSER1       CUSTOMERS1      TESTUSER1  CUSTOMERS1
         6 TESTUSER1       CUSTOMERS1      TESTUSER1  LINEITEMS1
         6 TESTUSER1       CUSTOMERS1      TESTUSER1  ORDERS1

        10 TESTUSER1       CUSTOMERS2      TESTUSER1  CUSTOMERS2
        10 TESTUSER1       CUSTOMERS2      TESTUSER1  LINEITEMS2
        10 TESTUSER1       CUSTOMERS2      TESTUSER1  ORDERS2

        13 TESTUSER1       CUSTOMERS3      TESTUSER1  CUSTOMERS3
        13 TESTUSER1       CUSTOMERS3      TESTUSER1  LINEITEMS3
        13 TESTUSER1       CUSTOMERS3      TESTUSER1  ORDERS3
```