Oracle® Database

Using Oracle Sharding

19c
E87088-14
May 2025

Oracle Database Using Oracle Sharding, 19c

E87088-14

Copyright © 2018, 2025, Oracle and/or its affiliates.

Primary Authors: Virginia Beecher, Roopam Jain

Contributors: Shailesh Dwivedi, Jean-Francois Verrier, Prakash Jashnani, Pankaj Chandiramani, Mark Dilman, Nourdine Benadjaoud, Rennie Sreekumar Ranjit Kumar , David Colello , Steve Ball, Abhishek Srivastava, Sebastian Binek, Shahab Hamid

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

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

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

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

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

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

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

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

Contents

1 Oracle Sharding Overview

What is Sharding 1-1
About Oracle Sharding 1-1
Oracle Sharding as Distributed Partitioning 1-2
Benefits of Oracle Sharding 1-3
Example Applications using Database Sharding 1-4
Flexible Deployment Models 1-5
High Availability in Oracle Sharding 1-5
Sharding Methods 1-5
Client Request Routing 1-6
Query Execution 1-6
High Speed Data Ingest 1-6
Deployment Automation 1-7
Data Pump Migration 1-7
Lifecycle Management of Shards 1-7

2 Oracle Sharding Architecture and Concepts

Components of the Oracle Sharding Architecture 2-1
- Sharded Database and Shards 2-1
- Shard Catalog 2-2
- Shard Director 2-3
- Global Service 2-3

Partitions, Tablespaces, and Chunks 2-3
Tablespace Sets 2-5
Sharding Methods 2-6
- System-Managed Sharding 2-6
- User-Defined Sharding 2-8
- Composite Sharding 2-10
- Using Subpartitions with Sharding 2-13

Sharded Database Schema Objects 2-15
- Sharded Tables 2-15
- Sharded Table Family 2-17
  - How a Table Family Is Sharded 2-17

Designing Schemas With Multiple Table Families 2-18
Duplicated Tables 2-20
Non-Table Objects Created on All Shards 2-21
Shard-Level High Availability 2-22
About Sharding and Replication 2-22
Using Oracle Data Guard with a Sharded Database 2-23
Using Oracle GoldenGate with a Sharded Database 2-28
Query Processing and the Query Coordinator 2-30
Client Application Request Routing 2-31
Management Interfaces for a Sharded Database 2-32

3 Security in an Oracle Sharding Environment

Using Transparent Data Encryption with Oracle Sharding 3-1
Creating a Single Encryption Key on All Shards 3-2
Configuring TCP/IP with SSL/TLS for Oracle Sharding 3-4
Oracle Database Vault 3-4

4 Sharded Database Deployment

Introduction to Sharded Database Deployment 4-1
Choosing a Shard Creation Method 4-1
Sharded Database Deployment Roadmap 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 4-10
Configure the Sharded Database Topology 4-16
Create the Shard Catalog 4-17
Add and Start Shard Directors 4-19
Add Shardspaces If Needed 4-20
Add Shardgoups If Needed 4-20
Verify the Sharding Topology 4-21
Add the Shard CDBs 4-22
Add the Shards 4-22
Add Shards Using GDSCTL ADD SHARD 4-22
Add Shards Using GDSCTL CREATE SHARD 4-23
Add Host Metadata 4-27
Deploy the Sharding Configuration 4-27
Create and Start Global Database Services 4-30
Verify Shard Status 4-31

Example Sharded Database Deployment 4-32
Example Sharded Database Topology 4-32
Deploy the Example Sharded Database 4-34

5 Using Oracle Database Sharding in Oracle Cloud Infrastructure

Deploy a Sharded Database on Kubernetes 5-1
Deploy a Sharded Database With Terraform 5-1
Deploy a Sharded Database with Docker 5-1

6 Sharded Database Schema Design

Sharded Database Schema Design Considerations 6-1
Choosing Sharding Keys 6-2
Primary Key and Foreign Key Constraints 6-4
Indexes on Sharded Tables 6-5
DDL Execution in a Sharded Database 6-5
Creating Objects Locally and Globally 6-6
DDL Syntax Extensions for Oracle Sharding 6-7
CREATE TABLESPACE SET 6-7
ALTER TABLESPACE SET 6-8
DROP TABLESPACE SET and PURGE TABLESPACE SET 6-8
CREATE TABLE 6-8
ALTER TABLE 6-10
ALTER SESSION 6-11
PL/SQL Procedure Execution in a Sharded Database 6-11
Creating Sharded Database Schema Objects 6-13
Create an All-Shards User 6-13
Creating a Sharded Table Family 6-13
Creating Sharded Tables 6-16
Creating Duplicated Tables 6-19
Updating Duplicated Tables and Synchronizing Their Contents 6-19
Schema Creation Examples 6-20
Create a System-Managed Sharded Database Schema 6-20
Create a User-Defined Sharded Database Schema 6-23
Create a Composite Sharded Database Schema 6-26
Monitor DDL Execution and Verify Object Creation 6-29
DDL Execution Failure and Recovery Examples 6-32
Generating Unique Sequence Numbers Across Shards 6-37

7 Migrating to a Sharded Database

Using Oracle Data Pump to Migrate to a Sharded Database 7-1
- Migrating a Schema to a Sharded Database 7-1
- Migrating the Sample Schema 7-3
- Migrating Data to a Sharded Database 7-5
- Loading the Sample Schema Data 7-7
- Migrating Data Without a Sharding Key 7-9

Using External Tables to Load Data into a Sharded Database 7-10
- Loading Data into Duplicated Tables 7-11
- Loading Data into Sharded Tables 7-12

Using Oracle GoldenGate to Replicate Data Between Sharded and Non-Sharded Databases 7-14
- Oracle GoldenGate Replication Prerequisites 7-14
- Replicating Data from a Non-Sharded Database to a Sharded Database 7-14

8 Query and DML Execution

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 8-4

Query Processing for Multi-Shard Queries 8-5
- Specifying Consistency Levels in a Multi-Shard Query 8-6

Supported Query Constructs and Example Query Shapes 8-6
- Queries on Sharded Tables Only 8-7
- Queries Involving Both Sharded and Duplicated Tables 8-7
- Aggregate Functions Supported by Oracle Sharding 8-9
- Queries with User-Defined Types 8-9
- Execution Plans for Proxy Routing 8-10

Supported DMLs and Examples 8-12
- Simple DMLs Where Only the Target Table is Referenced 8-13
- DMLs Referencing Other Tables 8-13
- Example Merge Statements 8-14
- Limitations in Multi-Shard DML Support 8-14

Gathering Optimizer Statistics on Sharded Tables 8-15

9 Developing Applications for the Sharded Database

Direct Routing to a Shard 9-1

Sharding APIs Supporting Direct Routing 9-2

Oracle JDBC APIs for Oracle Sharding 9-2
Oracle Call Interface for Oracle Sharding 9-3
Oracle Universal Connection Pool APIs for Oracle Sharding 9-4
Oracle Data Provider for .NET APIs for Oracle Sharding 9-7

10 Sharded Database Administration

Managing the Sharding-Enabled Stack 10-1
Starting Up the Sharding-Enabled Stack 10-1
Shutting Down the Sharding-Enabled Stack 10-1
Oracle Globally Distributed Database Users and Roles 10-1
Overview of Users and Roles 10-1
Oracle Sharding Database Roles 10-2
About the GSMUSER Account 10-2
About the GSMROOTUSER Account 10-3
Backing Up and Recovering a Sharded Database 10-3
Modifying a Sharded Database Schema 10-3
Propagation of Parameter Settings Across Shards 10-4
Migrating a Non-PDB Shard to a PDB 10-5
Managing Sharded Database Software Versions 10-5
Patching and Upgrading a Sharded Database 10-5
Performing a Rolling Upgrade 10-6
Upgrading Sharded Database Components 10-7
Downgrading a Sharded Database 10-8
Compatibility and Migration from Oracle Database 18c 10-8
Managing Oracle Sharded Database with Enterprise Manager Cloud Control 10-9
Prerequisite: Enable Sharded Database Metrics 10-10
Discovering Sharded Database Components 10-11
Overview of Sharded Database Management with Oracle Enterprise Manager Cloud Control 10-12
Monitoring a Sharded Database 10-15
Querying System Objects Across Shards 10-15
Monitoring a Sharded Database with GDSCTL 10-16
Monitoring a Sharded Database with Enterprise Manager Cloud Control 10-16
Sharded Database Home Page 10-16
Data Distribution and Performance Page 10-19
Shard Management 10-23
About Adding Shards 10-23
Resharding and Hot Spot Elimination 10-24
Removing a Shard From the Pool 10-26
Adding Standby Shards 10-26
Managing Shards with Oracle Enterprise Manager Cloud Control 10-27

Validating a Shard 10-27
Adding Primary Shards 10-27
Adding Standby Shards 10-28
Deploying Shards 10-29
Managing Shards with GDSCTL 10-29
Validating a Shard 10-29
Adding Shards to a System-Managed SDB 10-31
Replacing a Shard 10-35
Chunk Management 10-38
About Moving Chunks 10-38
Updating an In-Process Chunk Move Operation 10-39
Moving Chunks 10-40
About Splitting Chunks 10-40
Splitting Chunks 10-41
Shard Director Management 10-41
Creating a Shard Director 10-41
Editing a Shard Director Configuration 10-42
Removing a Shard Director 10-42
Region Management 10-43
Creating a Region 10-43
Editing a Region Configuration 10-43
Removing a Region 10-44
Shardspace Management 10-44
Creating a Shardspace 10-44
Adding a Shardspace to a Composite Sharded Database 10-45
Shardgroup Management 10-46
Creating a Shardgroup 10-46
Services Management 10-47
Creating a Service 10-47

11 Achieving Data Sovereignty with Oracle Sharding

Overview of Data Sovereignty 11-1
Benefits of Implementing Data Sovereignty with Oracle Sharding 11-1
Implementing Data Sovereignty with Oracle Sharding 11-2
Use Case of Achieving Data Sovereignty with Oracle Sharding 11-3
Overview of Oracle Sharding Solution 11-3
Deployment Topology of Data Sovereignty with Oracle Sharding 11-5
Configuring Data Sovereignty with Oracle Sharding 11-5
Configuring VCN Networks in All Three OCI Regions 11-6
Configuring Remote VCN Peering Between All Three Regions 11-6
Configuring Private DNS for Naming Resolution Between the Regions 11-7

Installing a Global Service Manager in Each Region 11-8
Collecting TNS entries for Shard Catalog and Sharded Databases 11-8
Configuring the Shard Catalog 11-9
Configuring the Shard Databases 11-10
Creating Oracle Sharding Global Database 11-11

12 Troubleshooting Oracle Sharding

Troubleshooting Tips 12-1
Checking the Sharding Method 12-1
Checking the Replication Type 12-2
Checking the Oracle Data Guard Protection Mode 12-3
Checking Which Shards Are Mapped to a Key 12-3
Checking Shard Operation Mode (Read-Only or Read-Write) 12-4
Checking DDL Text 12-5
Checking Chunk Migration Status 12-5
Checking Table Type (Sharded or Duplicated) 12-6
Checking User Type (Local or ALL_SHARD) 12-7
Identifying Tables Created as Sharded Tablespaces 12-7
Checking if Shard DDL is Enabled or Disabled 12-7
Filtering Data by Sharding Key 12-8
Setting the Duplicated Table Refresh Rate 12-9
Oracle Sharding Tracing and Debug Information 12-9
Enabling Tracing for Oracle Sharding 12-9
Where to Find Oracle Sharding Alert Logs and Trace Files 12-10
Common Error Patterns and Resolutions for Sharded Databases 12-11
Issues Starting Remote Scheduler Agent 12-11
Shard Director Fails to Start 12-11
Errors From Shards Created with CREATE SHARD 12-12
Issues Using Create Shard 12-12
Issues Using Deploy Command 12-14

13 Oracle Sharding Reference

Using GDSCTL with Oracle Sharding 13-1
Starting GDSCTL 13-1
Running GDSCTL Commands Interactively 13-1
Running GDSCTL Batch Operations 13-1
GDSCTL Help Text 13-2
GDSCTL Connections 13-2
GDSCTL Shard Catalog Connections 13-2
GDSCTL Shard Director Connections 13-2

Preface

Review the following topics to:

- Discover how you can use this document to learn about Oracle Sharding
- Get accessibility information for this document
- See a list of related documents that may help you design, develop, deploy, and manage your Oracle Sharding 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 Sharding suitability for their requirements. IT managers can scope out the work needed to implement Oracle Sharding 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 Sharding. Finally, business analysts can use this document as a guide to figure out costing for an Oracle Sharding 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

For more information, see the Oracle database documentation set. These books may be of particular interest:

- *Oracle Database Global Data Services Concepts and Administration Guide*
- *Oracle Database Administrator's Guide*
- *Oracle Data Guard Concepts and Administration*
- *Oracle Data Guard Broker*
- *Using the Oracle GoldenGate Microservices Architecture*
- *Oracle Database JDBC Developer's Guide*
- *Oracle Universal Connection Pool Developer's Guide*
- *Oracle Data Provider for .NET Developer's Guide for Microsoft Windows*
- *Oracle Call Interface Programmer'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, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Changes in This Release for Oracle Sharding

This preface contains:

Changes in Oracle Database 19c

The following are changes in *Using Oracle Sharding* for Oracle Database 19c.

New Features

The following features are new in this release:

Multiple Table Family Support for System-Managed Sharding

The Oracle Sharding feature for Oracle Database 18c supported only one table family (a set of related tables sharing the same sharding key) for each sharded database. In Oracle Database 19c, Oracle Sharding includes support for multiple table families where all data from different table families reside in the same chunks. This feature applies to system-managed sharded databases only. Different applications accessing different table families can now be hosted on one sharded database.

There is one new GDSCTL command, `CONFIG TABLE FAMILY`, and several other commands are extended to support this feature: `ADD SERVICE`, `MODIFY SERVICE`, `CONFIG SERVICE`, `CONFIG CHUNKS`, `STATUS ROUTING`, and `VALIDATE CATALOG`.

There are no new SQL keywords or statements introduced with this feature; however, some restrictions are changed with the use of `CREATE SHARDED TABLE` and `TABLESPACE SET`.

See

- Sharded Table Family
- *Oracle Database Global Data Services Concepts and Administration Guide*
- *Oracle Database SQL Language Reference*

Support for Multiple PDB-Shards in the Same CDB

In Oracle Database 18c, Oracle Sharding introduced the capability for using a single PDB in a CDB as a shard or a shard catalog database. In Oracle Database 19c, Oracle Sharding enables you to use more than one PDB in a CDB for shards or shard catalog databases, with certain restrictions. For example, this feature allows a CDB to contain shard PDBs from different sharded databases (SDBs), each with their own separate catalog databases.

See Compatibility and Migration from Oracle Database 18c for information about how to migrate shard PDBs to 19c.

Generation of Unique Sequence Numbers Across Shards

Before Oracle Database 19c, if you needed a unique number across shards you had to manage it yourself. In Oracle Database 19c, Oracle Sharding allows you to independently generate sequence numbers on each shard which are unique across all shards.

To support this feature, new `SEQUENCE` object clauses, `SHARD` and `NOSHARD`, are included in the `SEQUENCE` object DDL syntax.

See

- Generating Unique Sequence Numbers Across Shards
- *Oracle Database SQL Language Reference*

Support for Multi-Shard Query Coordinators on Shard Catalog Standbys

Before Oracle Database 19c, only the primary shard catalog database could be used as the multi-shard query coordinator. In Oracle Database 19c you can also enable the multi-shard query coordinator on Oracle Active Data Guard standbys of the shard catalog database. This improves the scalability and availability of multi-shard query workload.

See

- Query Processing and the Query Coordinator

Propagation of Parameter Settings Across Shards

Before Oracle Database 19c, database administrators had to configure `ALTER SYSTEM` parameter settings on each shard in a sharded database. This feature provides ease of manageability by allowing administrators to centrally manage and propagate parameter settings from the shard catalog to all of the database shards. Once settings are configured at the shard catalog, they are automatically propagated to all shards of the sharded database.

See Propagation of Parameter Settings Across Shards

Deprecation and Desupport

The following features are deprecated or desupported in this release:

Desupport of Setting Passwords in GDSCTL Command Line

To enhance security, starting with Oracle Database 19c, the ability to specify passwords from the Global Data Services Control Utility (GDSCTL) command-line when called from the operating system prompt is no longer supported.

This desupport applies only to password changes where GDSCTL is called from a user command-line prompt. For example, the following command is desupported:

```
$ gdsctl add database -connect inst1 -pwd gsm_password
```

Specifying the password from the GDSCTL utility itself is still valid. For example, the following command is valid:

```
GDSCTL> add database -connect inst1 -pwd gsm_password
```

This deprecation addresses the security vulnerability when specifying passwords in GDSCTL commands called from the operating system prompt.

1

Oracle Sharding Overview

Learn about what Oracle Sharding can do in this high level conceptual discussion.

Oracle Sharding capabilities and benefits are described in the following topics:

What is Sharding

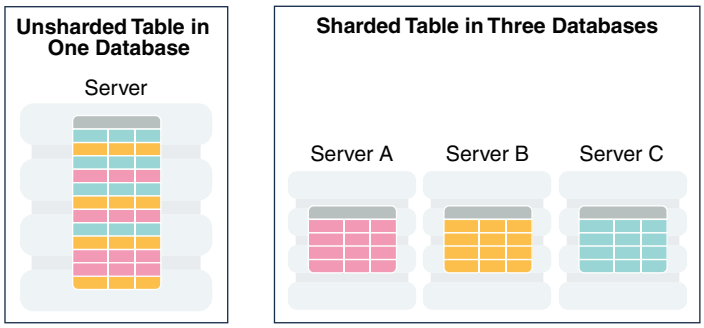
Hyperscale computing is a computing architecture that can scale up or down quickly to meet increased demand on the system. This architecture innovation was originally driven by internet giants that run distributed sites and has been adopted by large-scale cloud providers.

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 Sharding

Oracle Sharding 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 Sharding 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 Active Data Guard.

> **Note:**
>
> Oracle GoldenGate replication support for Oracle Sharding High Availability is deprecated in Oracle Database 21c.

- 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 Sharding 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.

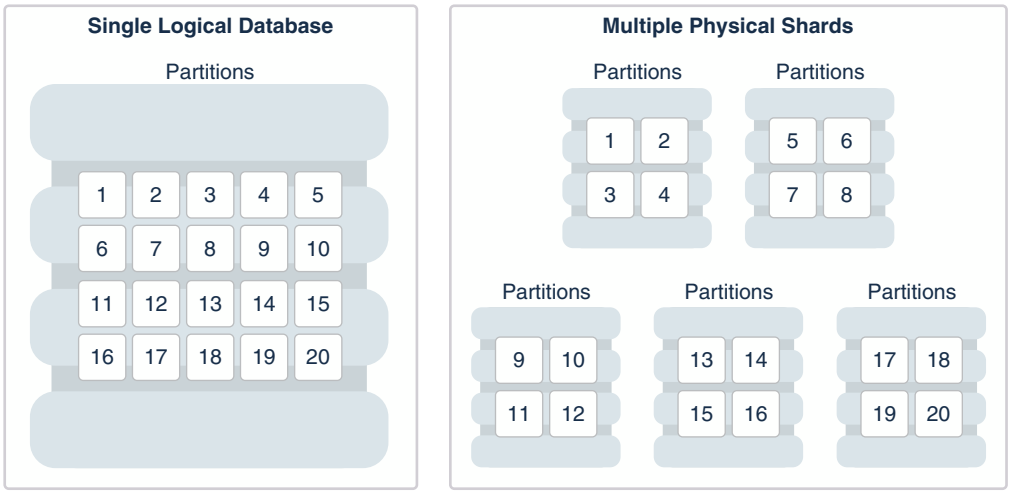
From the perspective of an application, a sharded database in Oracle Sharding looks like a single database; the number of shards, and the distribution of data across those shards, are *completely transparent* to the application.

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 Sharding as Distributed Partitioning



Oracle Sharding automatically distributes the partitions across shards when you execute 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 Sharding

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

Key benefits of Oracle Sharding include:

- **Linear Scalability**

  The Oracle Sharding shared–nothing architecture eliminates performance bottlenecks and provides unlimited scalability. Oracle Sharding 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**

  Sharding enables 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 Database Sharding

Oracle Sharding 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 Sharding 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 Sharding 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 Sharding 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 Sharding 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 Sharding 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 Sharding

Oracle Sharding is tightly integrated with Oracle Data Guard to provide high availability and disaster recovery. Replication is automatically configured and deployed when the sharded database is created.

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. Use this option when application needs strict data consistency and zero data loss.

Oracle GoldenGate is used for fine-grained active-active replication. Though applications must be able to deal with conflicts and data loss upon potential failover.

> **Note:**
>
> Oracle GoldenGate replication support for Oracle Sharding High Availability is deprecated in Oracle Database 21c.

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. 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.

Sharding Methods

Because Oracle Sharding is based on table partitioning, all of the sub-partitioning methods provided by Oracle Database are also supported by Oracle Sharding. A data sharding *method*

controls the placement of the data on the shards. Oracle Sharding supports system-managed, user defined, or composite sharding methods.

- **System-managed sharding** 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 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.
- **Composite sharding** allows you to use two levels of sharding. First the data is sharded by range or list and then it is sharded further by consistent hash.

  In many use cases, especially for data sovereignty and data proximity requirements, the composite sharding method offers the best of both system-managed and user-defined sharding methods, giving you the automation you want and the control over data placement you need.

Client Request Routing

Oracle Sharding 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 Sharding 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 executed. 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 Execution

No changes to query and DML statements are required to support Oracle Sharding. 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 executed on all shards in a configuration, so too can certain Oracle-provided PL/SQL procedures.

Oracle Sharding 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 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. 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

Sharded 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 Pump Migration

Oracle Data Pump is sharding aware and is used to migrate data from a non-sharded Oracle database to a sharded Oracle database.

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 of Shards

The Oracle Sharding 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

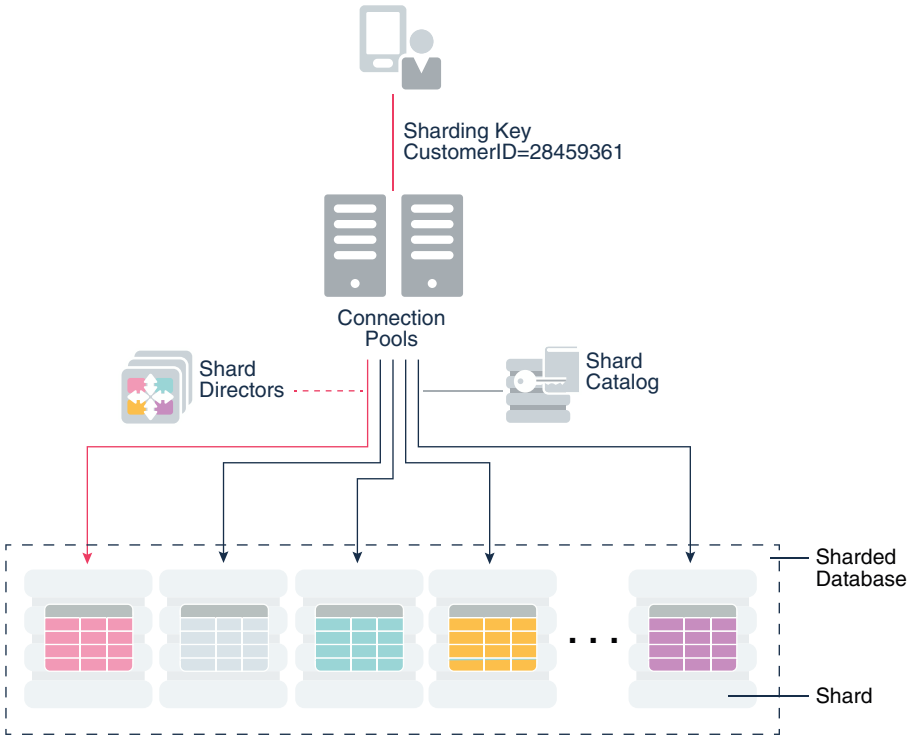
2

Oracle Sharding Architecture and Concepts

Components of the Oracle Sharding Architecture

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

Figure 2-1 Oracle Sharding 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 Sharding 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 Sharding represents a data center or multiple data centers that are in close network proximity.

Shards are replicated for high availability and disaster recovery with Oracle Data Guard. 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.

> **Note:**
>
> Oracle GoldenGate replication support for Oracle Sharding High Availability is deprecated in Oracle Database 21c.

Shard Catalog

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

A shard catalog serves following purposes

- Serves as an administrative server for entire shareded 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 executed by connecting to the shard catalog.

The shard catalog also contains the master 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 executed 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 Sharding, global service managers support routing of connections based on data location. A global service manager, in the context of Oracle Sharding, 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 5 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 a 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 Active Data Guard, a separate read-only global service can be created.

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 an SDB.

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.

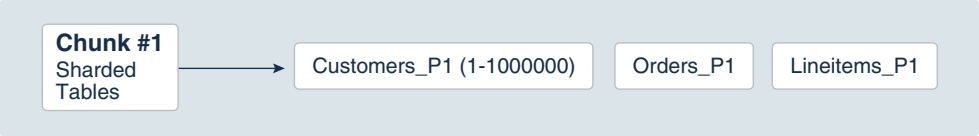
However, it is possible to create different tables from a table family in different tablespace sets, for example the Customers table in tablespace set `ts1` and Orders in tablespace set `ts2`. In

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.

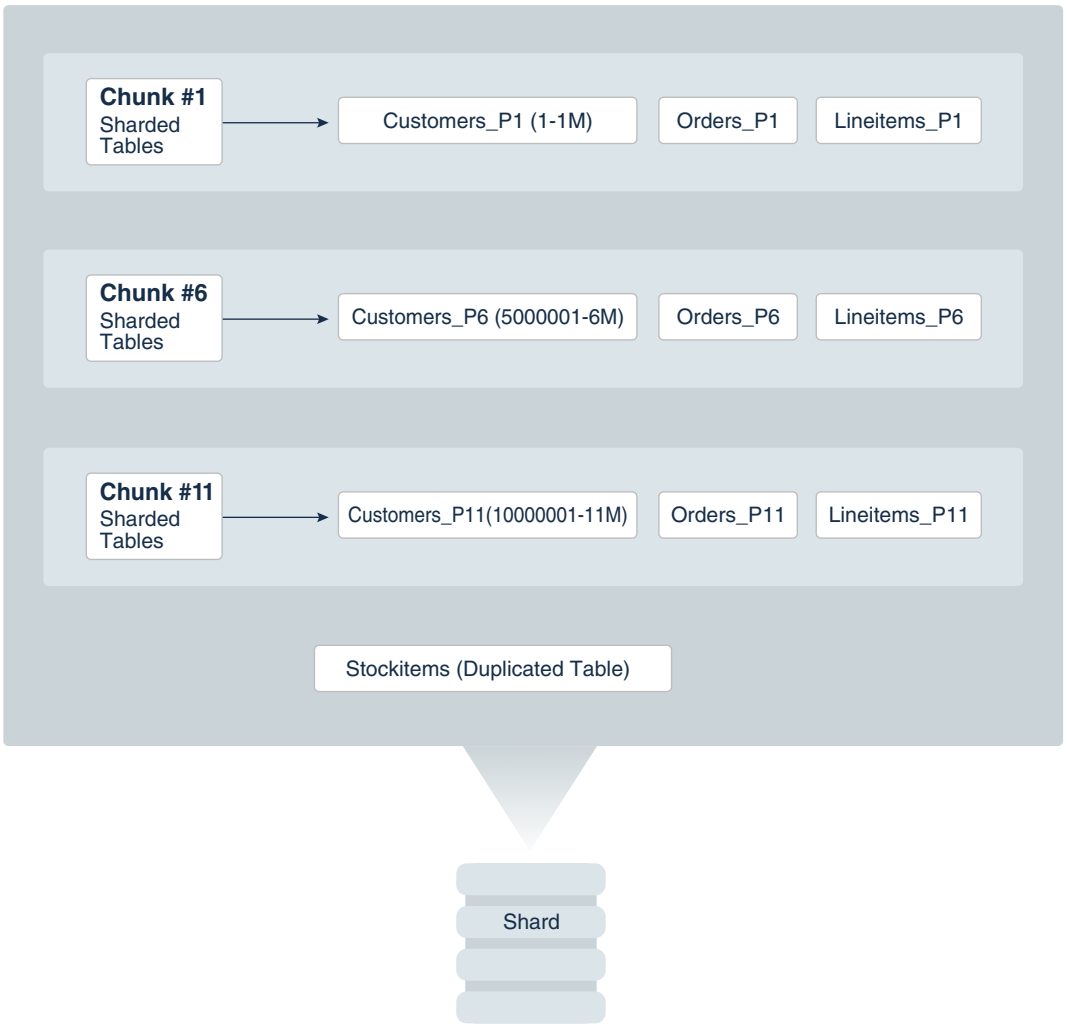
A chunk that contains corresponding partitions from the tables of Cutomers-Orders-LineItems schema is shown in the following figure.

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 Sharding 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.

Sharding Methods

The following topics discuss sharding methods supported by Oracle Sharding, how to choose a method, and how to use subpartitioning.

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.

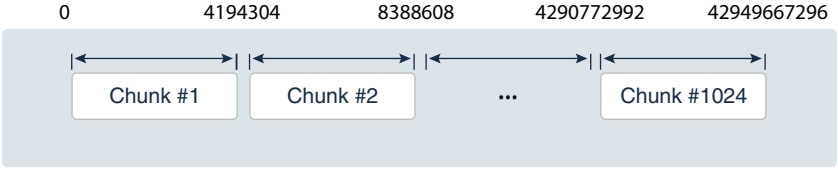
The distribution used in system-managed sharding is intended to eliminate hot spots and provide uniform performance across shards. Oracle Sharding automatically maintains the balanced distribution of chunks when shards are added to or removed from an SDB.

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 Sharding 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 SDB 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 2-4 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 SDB. For example, if 1024 chunks are created in an SDB that contains 16 shards, each shard will contain 64 chunks.

In the event of resharding, when shards are added to or removed from an SDB, 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 an SDB 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 an SDB with system-managed sharding can be specified in the GDSCTL command, `CREATE SHARDCATALOG`. If not specified, the default value, 120 chunks per shard, is used. Once an SDB 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 executing 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
;
```

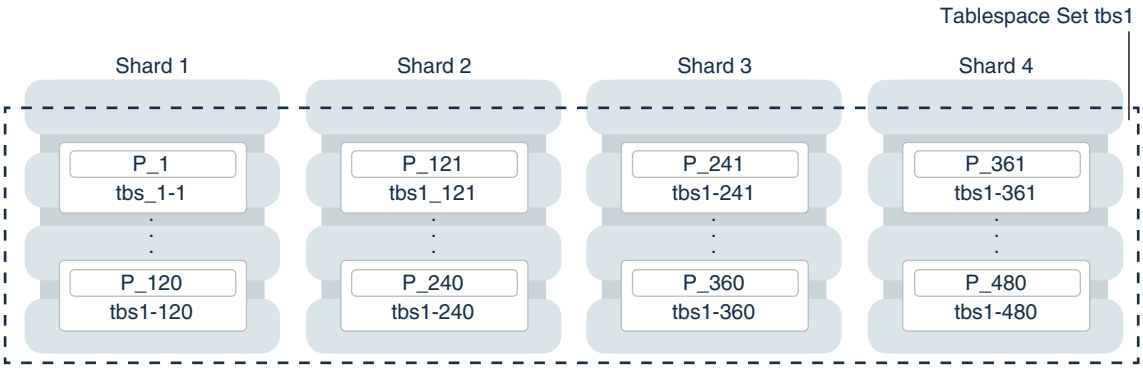
`PARTITIONS AUTO` in this statement means that the number of partitions is automatically set to the number of tablespaces in the tablespace set `ts1` (which is equal to the number of chunks) and each partition will be stored in a separate tablespace.

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 2-5 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.

For a user-defined sharded database, two replication schemes are supported: Oracle Data Guard or Oracle Active Data Guard. User-defined sharding is not supported where Oracle GoldenGate is used as the replication method.

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.

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.

```
 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_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
)
;
```

There is no tablespace set for user-defined sharding. Each tablespace has to be created individually and explicitly associated with a shardspace. 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. See Shard-Level High Availability for details about replication with user-defined sharding. For simplicity, assume that each shardspace consists of a single shard.

The following statements can be used to create the tablespaces for the accounts table in the example above.

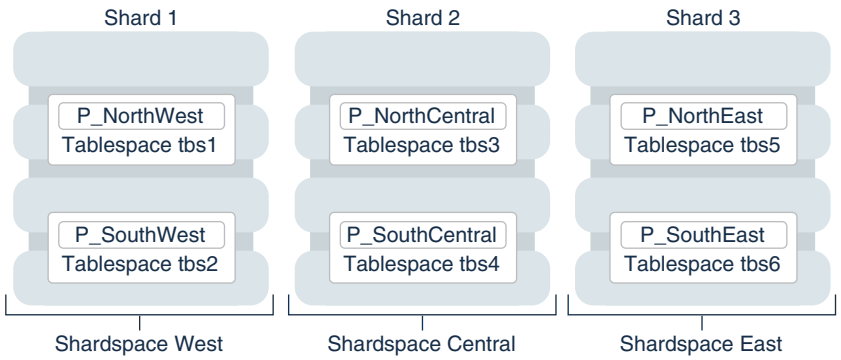
```
CREATE TABLESPACE tbs1 IN SHARDSPACE west;
CREATE TABLESPACE tbs2 IN SHARDSPACE west;

CREATE TABLESPACE tbs3 IN SHARDSPACE central;
CREATE TABLESPACE tbs4 IN SHARDSPACE central;

CREATE TABLESPACE tbs5 IN SHARDSPACE east;
CREATE TABLESPACE tbs6 IN SHARDSPACE east;
```

Before executing the `CREATE TABLESPACE` statements, the shardspaces must be created and populated with shards. For example, you can use the following GDSCTL commands:

```
ADD SHARDSPACE -SHARDSPACE east
ADD SHARDSPACE -SHARDSPACE central
ADD SHARDSPACE -SHARDSPACE west
ADD SHARD -CONNECT shard-1 -SHARDSPACE west;
ADD SHARD -CONNECT shard-2 -SHARDSPACE central;
ADD SHARD -CONNECT shard-3 -SHARDSPACE east;
```

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

Figure 2-6 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 SDB. The user needs to execute the GDSCTL `MOVE CHUNK` command for each chunk that needs to be migrated.

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.

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

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.

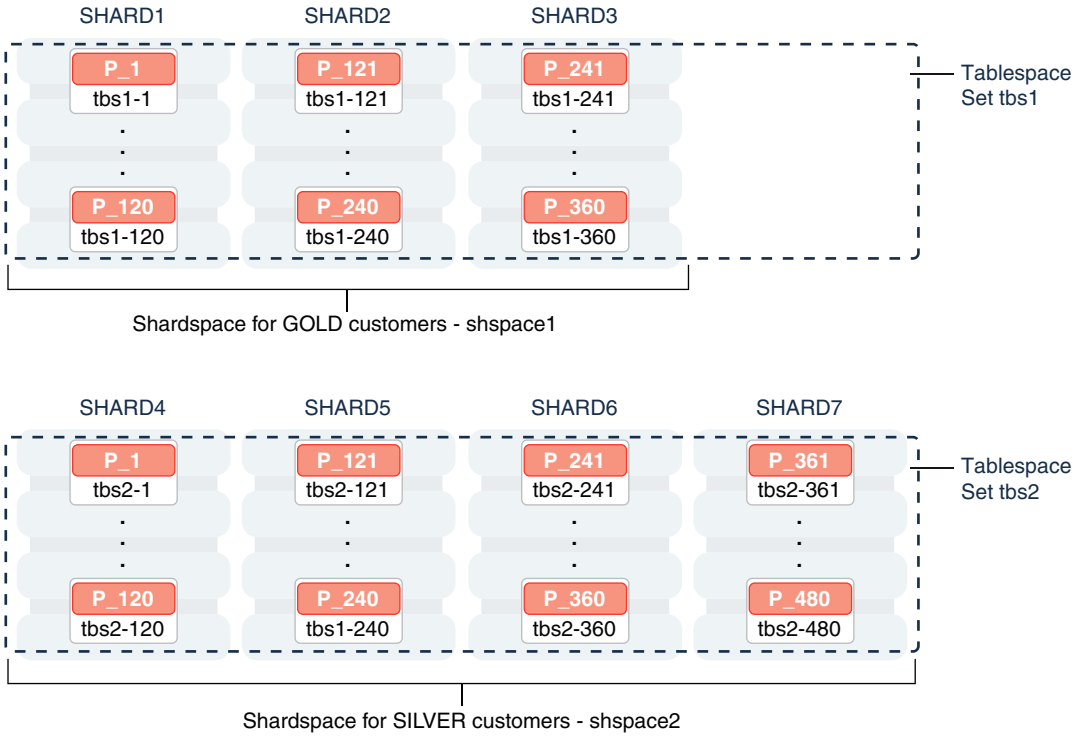
System-managed sharding uses partitioning by consistent hash to randomly distribute data across shards. This provides better load balancing compared to user-defined sharding that uses partitioning by range or list. However, system-managed sharding does not give the user any control on assignment of data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

When sharding by consistent hash on a primary key, there is often a requirement to differentiate subsets of data within an SDB 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 2-7 Composite Sharding



The following commands would be issued to create this configuration. Note that two shardspaces need to be created for this configuration.

```
create SHARDCATALOG -sharding composite -database
     cat_host:1521/cat_pdb.domain -user gsmcatuser/gsmcatuser_pwd
     -region dc1

add gsm -gsm gsm1 -listener 1540 -catalog cat_host:1521/cat_pdb.domain
     -region dc1 -pwd gsmcatuser_pwd
gdsctl start gsm
```

```
add shardspace -shardspace shspace1 -chunks 60
add shardspace -shardspace shspace2 -chunks 120

ADD SHARDGROUP -shardgroup gold -shardspace shspace1 -region dc1 -deploy_as
     primary
ADD SHARDGROUP -shardgroup silver -shardspace shspace2 -region dc1 -deploy_as
     primary

add CDB -connect cdb1_host:1521/cdb1.domain -pwd gsmrootuser_pwd
add CDB -connect cdb2_host:1521/cdb2.domain -pwd gsmrootuser_pwd
add CDB -connect cdb3_host:1521/cdb3.domain -pwd gsmrootuser_pwd
add CDB -connect cdb4_host:1521/cdb4.domain -pwd gsmrootuser_pwd
add CDB -connect cdb5_host:1521/cdb5.domain -pwd gsmrootuser_pwd
add CDB -connect cdb6_host:1521/cdb6.domain -pwd gsmrootuser_pwd
add CDB -connect cdb7_host:1521/cdb7.domain -pwd gsmrootuser_pwd

add shard -cdb cdb1 -shardgroup gold -connect
     cdb1_host:1521/sh1_pdb.domain -pwd gsmuser_pwd
add shard -cdb cdb2 -shardgroup gold -connect
     cdb2_host:1521/sh2_pdb.domain -pwd gsmuser_pwd
add shard -cdb cdb3 -shardgroup gold -connect
     cdb3_host:1521/sh3_pdb.domain -pwd gsmuser_pwd

add shard -cdb cdb4 -shardgroup silver -connect
     cdb4_host:1521/sh4_pdb.domain -pwd gsmuser_pwd
add shard -cdb cdb5 -shardgroup silver -connect
     cdb5_host:1521/sh5_pdb.domain -pwd gsmuser_pwd
add shard -cdb cdb6 -shardgroup silver -connect
     cdb6_host:1521/sh6_pdb.domain -pwd gsmuser_pwd
add shard -cdb cdb7 -shardgroup silver -connect
     cdb7_host:1521/sh7_pdb.domain -pwd gsmuser_pwd

deploy
```

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 Sharding 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:**
>
> In Oracle Database 12*c* Release 2 only a single partition set from a table can be stored in a shardspace.
> The sharding method is specified in the `GDSCTL CREATE SHARDCATALOG` command and cannot be changed later.

Using Subpartitions with Sharding

Because Oracle Sharding 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 execution 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.

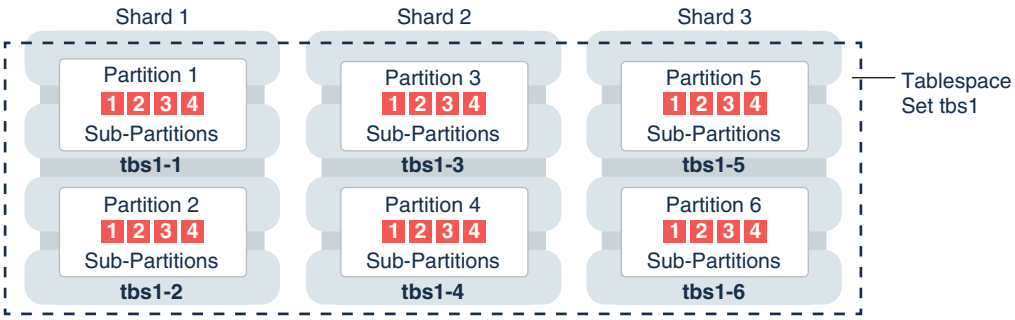
The following example shows system-managed sharding by consistent hash combined with subpartitioning by range.

```
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 2-8 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)
)
;
```

Sharded Database Schema Objects

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 executed on a single shard. The following topics define and illustrate the schema objects that form a sharded database to inform your design.

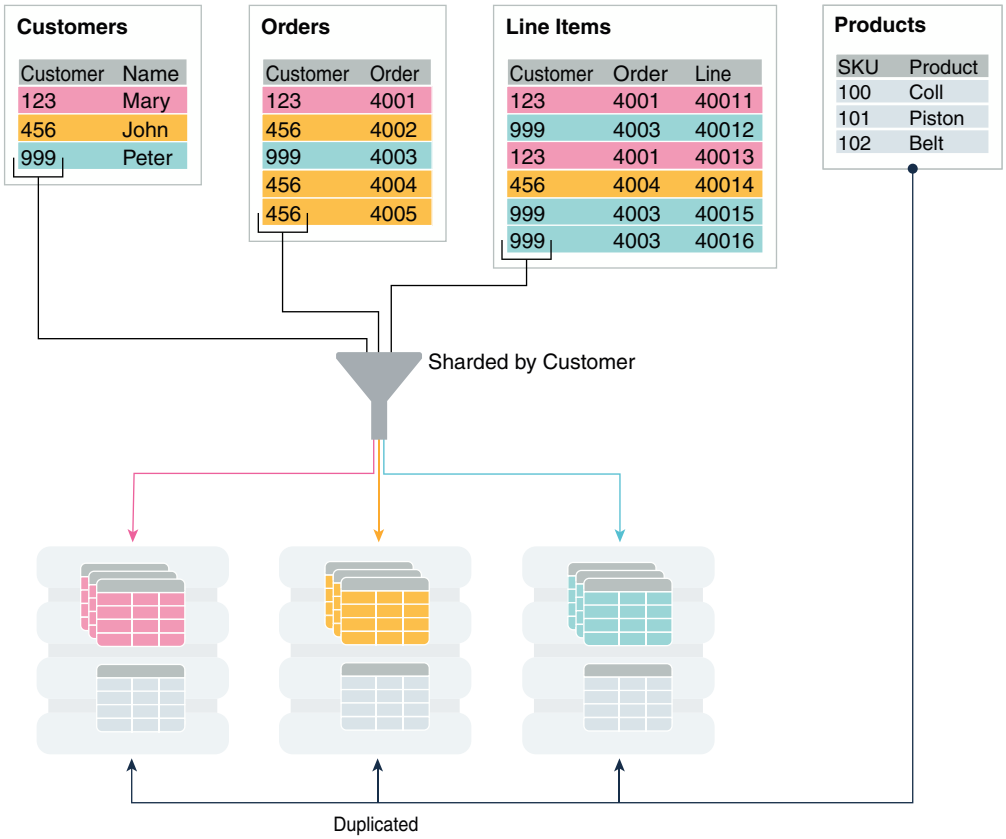
Sharded Tables

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 2-9 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
```

Designing Schemas With Multiple Table Families

A sharded 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
```

Duplicated Tables

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

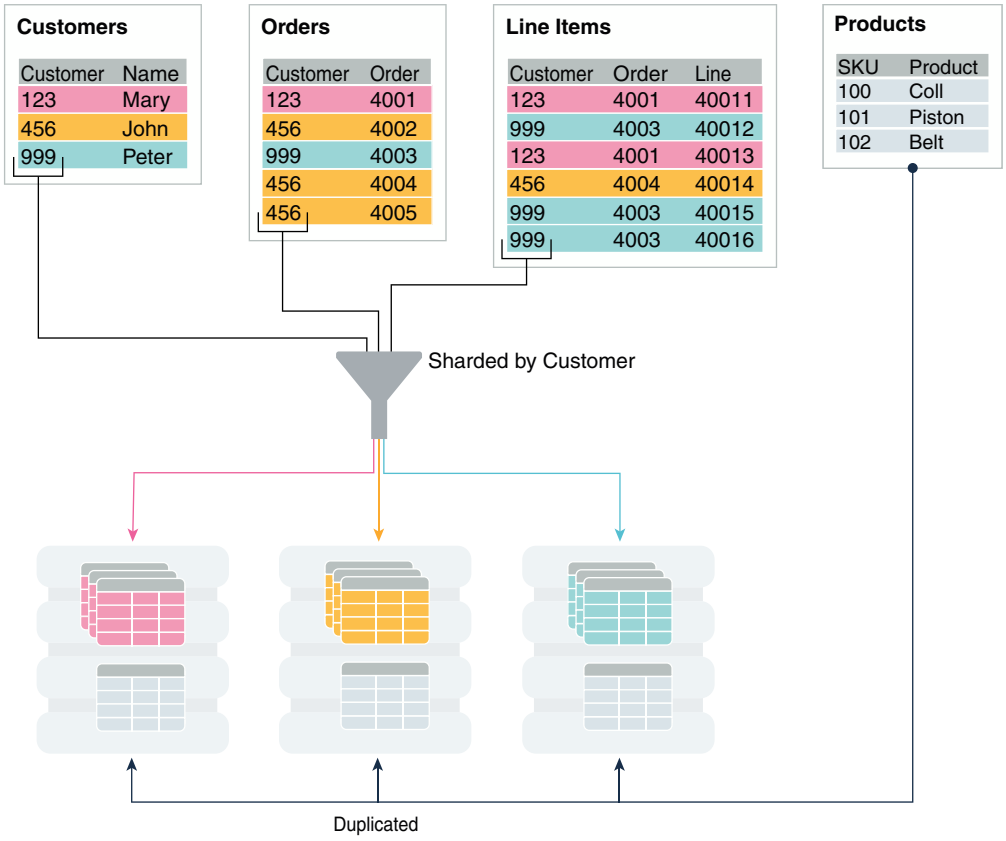
For many applications, the number of database requests handled by a single shard can be maximized by duplicating read-only or read-mostly tables across all shards. This strategy is a good choice for relatively small tables that are not updated frequently, and that are often accessed together with sharded tables. Duplicated tables tend to be updated less frequently than sharded tables and are not expected to be very large.

A sharded database includes both sharded tables that are horizontally partitioned across shards, and duplicated tables that are replicated to all shards. Duplicated tables contain reference information, for example, a Stock Items table that is common to each shard. 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. The 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 must be duplicated on all shards.

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

Figure 2-10 Sharded Tables and a Duplicated Table in a Sharded Database



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.

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.

Shard-Level High Availability

Oracle Sharding is integrated with Oracle Database replication technologies for high availability and disaster recovery at the shard level.

The following topics describe how to use Oracle's replication technologies to make your sharded databases highly available:

About Sharding and Replication

Oracle Sharding is tightly integrated with the Oracle replication and disaster recovery technologies Oracle Data Guard and Oracle GoldenGate.

> **Note:**
>
> Oracle GoldenGate replication support for Oracle Sharding High Availability is deprecated in Oracle Database 21c.

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 one of two technologies—Oracle Data Guard or Oracle GoldenGate—to replicate your data. Oracle Sharding automatically deploys the specified replication topology and enables data replication.

The availability of a sharded database is not affected by an outage or slowdown of one or more shards. Replication is used to provide individual shard-level high availability (Oracle Active

Data Guard or Oracle GoldenGate). 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 Sharding automatically fails over database connections from a shard to its replica in the event of an unplanned outage.

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.

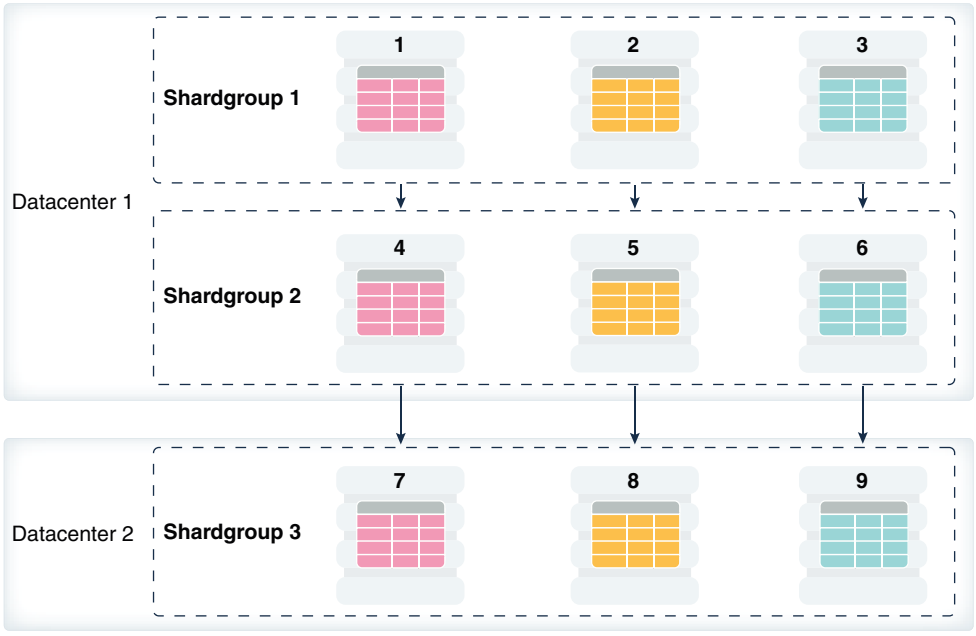
Oracle Data Guard can be used as the replication technology for sharded databases using the system-managed, user-defined, or composite method of sharding.

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 datacenter and configured for synchronous replication. And Shardgroup 3 consists of remote standbys (shards 7-9) located in a different datacenter and configured for asynchronous replication. Oracle Active Data Guard is enabled in this configuration, so each standby is open read-only.

Figure 2-11 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 Sharding 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.

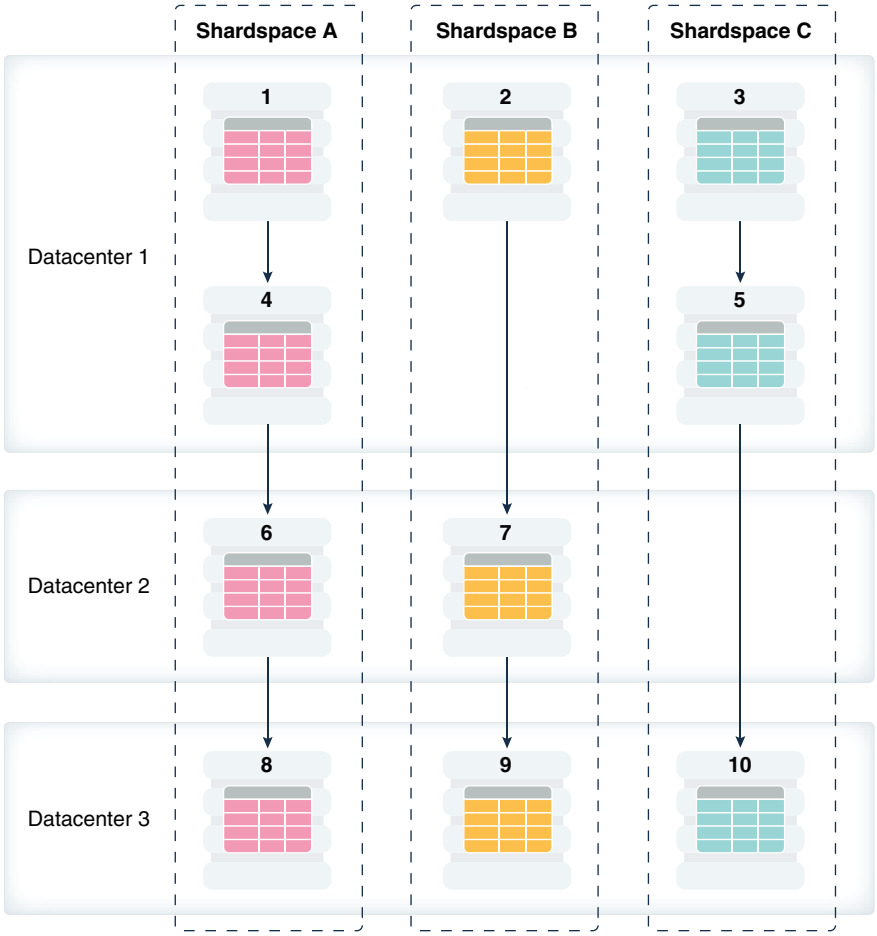
Run the following GDSCTL commands to deploy the example configuration shown in the figure above.

```
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

CREATE SHARD -shardgroup shardgroup1 -destination host01 -credential
oracle_cred
CREATE SHARD -shardgroup shardgroup1 -destination host02 -credential
oracle_cred
CREATE SHARD -shardgroup shardgroup1 -destination host03 -credential
oracle_cred
...
CREATE SHARD -shardgroup shardgroup3  -destination host09 -credential
oracle_cred

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 2-12 User-Defined Sharding with Data Guard Replication



Run the following GDSCTL commands to deploy the example user-defined sharded database with Data Guard replication shown in the figure above.

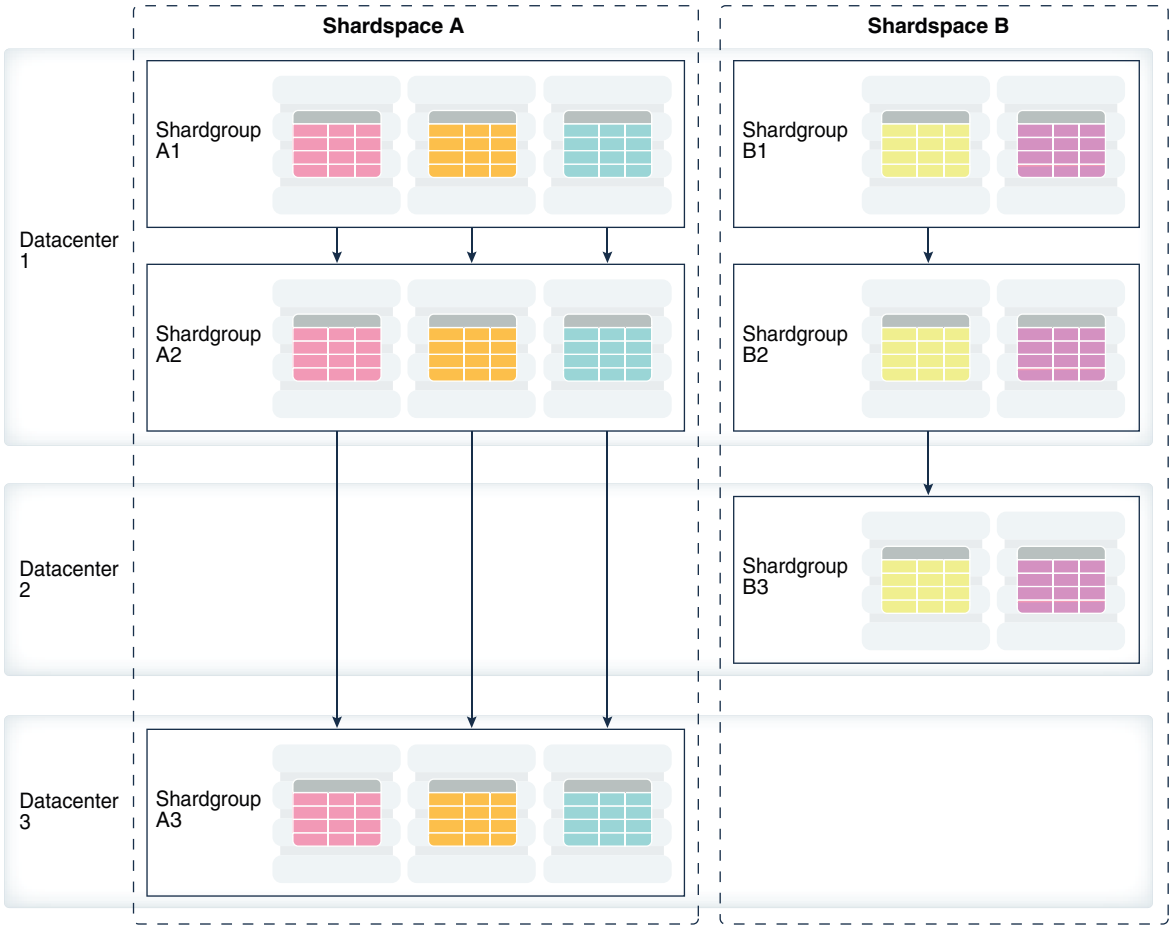
```
CREATE SHARDCATALOG -sharding user -database host00:1521:cat -region
dc1,dc2,dc3

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

CREATE SHARD -shardspace shardspace_a -region dc1 -deploy_as primary  -
destination
host01 -credential oracle_cred -netparamfile /home/oracle/netca_dbhome.rsp

CREATE SHARD -shardspace shardspace_a -region dc1 -deploy_as standby  -
```

```
destination
host04 -credential oracle_cred -netparamfile /home/oracle/netca_dbhome.rsp

CREATE SHARD -shardspace shardspace_a -region dc2 -deploy_as standby  -
destination
host06 -credential oracle_cred -netparamfile /home/oracle/netca_dbhome.rsp

CREATE SHARD -shardspace shardspace_a -region dc3 -deploy_as standby  -
destination
host08 -credential oracle_cred -netparamfile /home/oracle/netca_dbhome.rsp

CREATE SHARD -shardspace shardspace_b -region dc1 -deploy_as primary  -
destination
host08 -credential oracle_cred -netparamfile /home/oracle/netca_dbhome.rs
...

CREATE SHARD -shardspace shardspace_c -region dc3 -deploy_as standby  -
destination
host10 -credential oracle_cred -netparamfile /home/oracle/netca_dbhome.rsp

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 2-13 Composite Sharding with Data Guard Replication



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

```
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

CREATE SHARD -shardgroup shardgroup_a1 -destination host01 -credential
orcl_cred
...

CREATE SHARD -shardgroup shardgroup_b3 -destination host09 -credential
orcl_cred

DEPLOY
```

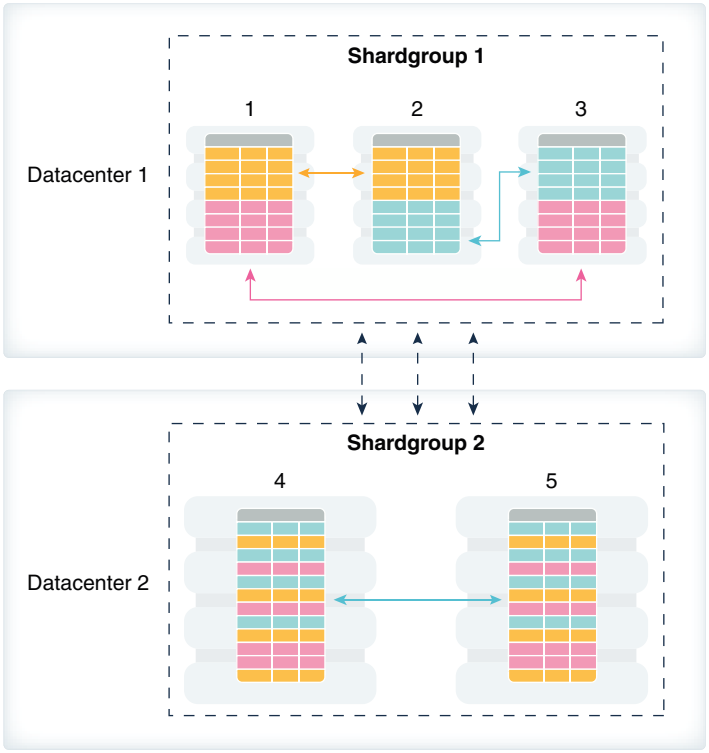
Using Oracle GoldenGate with a Sharded Database

Oracle GoldenGate is used for fine-grained active-active replication where all shards are writable, and each shard can be partially replicated to other shards within a shardgroup.

> **Note:**
>
> Oracle Database 21c supports only multitenant architecture (CDB). Oracle GoldenGate versions 12.3-19.1 only support Oracle Sharding with single-instance Oracle databases (release 11g through 19c.)

In Oracle GoldenGate, replication is handled at the chunk level. For example, in Shardgroup 1 in the following figure, half of the data stored in each shard is replicated to one shard, and the other half to another shard. If any shard becomes unavailable, its workload is split between two other shards in the shardgroup. The multiple failover destinations mitigate the impact of a shard failure because there is no single shard that has to handle all of the workload from the failed shard.

Figure 2-14 System-Managed Sharding with Golden Gate Replication



With Oracle GoldenGate replication, a shardgroup can contain multiple replicas of each row in a sharded table; therefore, high availability is provided within a shardgroup, and there is no need to have a local replica of the shardgroup, as there is in the case of Data Guard replication. The number of times each row is replicated within a shardgroup is called its *replication factor* and is a configurable parameter.

To provide disaster recovery, a shardgroup can be replicated to one or more data centers. Each replica of a shardgroup can have a different number of shards, replication factor, database versions, and hardware platforms. However, all shardgroup replicas must have the same number of chunks, because replication is done at the chunk level.

Shardgroup 2 in the figure above contains the same data as Shardgroup 1, but resides in a different data center. Shards in both data centers are writable. The default replication factor, 2, is used for both shardgroups.

Note that because Shardgroup 2 contains only two shards and the replication factor is 2, the shards are fully replicated, and each of them contains all of the data stored in the sharded database. This means that any query routed to these shards can be executed without going across shards. There is only one failover destination in this shardgroup; if a shard goes down, the load on the other shard doubles.

Oracle Sharding is designed to minimize the number of conflicting updates performed to the same row on different shards. This is achieved designating a *master chunk* for each range of hash values and routing most of requests for the corresponding data to this chunk.

Sometimes it is impossible to avoid update conflicts because of state transitions, such as a chunk move or split, or a shard going up or down. The user may also intentionally allow conflicts in order to minimize transaction latency. For such cases Oracle GoldenGate provides automatic conflict detection and resolution which handles all kinds of conflicts including insert-delete conflicts.

> **Note:**
>
> Oracle GoldenGate does not support the user defined sharding method.
>
> For system-managed sharding with Oracle GoldenGate, a shard must have at least two chunks.
>
> Oracle GoldenGate does not support PDBs as shards.

> **See Also:**
>
> Working with Oracle GoldenGate Sharding in the Fusion Middleware *Using the Oracle GoldenGate Microservices Architecture* guide for more information about using Oracle GoldenGate with Oracle Sharding.

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 execution.

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 execution across all of the participating shards.

In a single-shard query scenario, the entire query is executed 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 executed 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 executes 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 executed 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 SQL execution without a sharding key, you will experience a reduced level of performance compared with direct, key-based, routing.

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 executed 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 executed from the query coordinator which orchestrates parallel execution 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 Sharding.

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.

Management Interfaces for a Sharded Database

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

3
Security in an Oracle Sharding Environment

Using Transparent Data Encryption with Oracle Sharding

Oracle Sharding supports Transparent Data Encryption (TDE), but in order 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 catalog database. For TDE to work properly, especially when data is moved between shards, certain restrictions apply. In order 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 executed 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 key store identified by *password*

Limitations

The following limitations apply to using TDE with Oracle Sharding.

- For `MOVE CHUNK` to work, all 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.

> **See Also:**
>
> *Oracle Database Advanced Security Guide* for more information about TDE

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.

> **Note:**
>
> 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 *keystore_password* should be the same if you prefer to issue wallet open and close commands centrally from the catalog.

   > **Note:**
   >
   > The wallet directory path should match the `ENCRYPTION_WALLET_LOCATION` in the corresponding sqlnet.ora.
   >
   > `ENCRYPTION_WALLET_LOCATION` parameter is being deprecated. You are advised to use the `WALLET_ROOT` static initialization and `TDE_CONFIGURATION` dynamic initialization parameter instead.

   With shard DDL disabled, issue the following statement.

   ```
   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 catalog. In order for data movement to work, you cannot use different encryption keys on each shard.

2. 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));
```

3. 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;
```

(Optional) Enter the result of the step here.

4. 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.
5. 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;
```

6. Restart the shard databases.
7. Activate the key on all of the shards.

   On the catalog with shard DDL enabled

```
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).

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));
```

Configuring TCP/IP with SSL/TLS for Oracle Sharding

Configuring TCP/IP with SSL/TLS for Oracle Sharding has different steps depending on the type of databases you plan to run shards on.

For information about configuring this security feature, see the documents based on the types of database you plan to run shards on.

- **Autonomous Database**

  For Oracle Autonomous Database, TLS is already enabled by default, and you only need to create the remaining security infrastructure, such as vaults, keys, and certificate resources on OCI.

- **Base Database Service**

  For Base Database Service on OCI you will need to enable TLS using the information in Configure TCP/IP with SSL/TLS for Sharding – GSM OCI Mode (Doc ID 2881390.1)

- **On-Premises**

  For on-premises databases, see Configure TCP/IP with SSL/TLS for Sharding – GSM JDBC THIN MODE (Doc ID 2881420.1)

More information is also available in Configuring Oracle Database Native Network Encryption and Data Integrity and Configuring Secure Sockets Layer Authentication

Oracle Database Vault

Do not enable Oracle Database Vault (Data Vault) on your sharded databases. Oracle Sharding does not support Oracle Database Vault.

4

Sharded Database Deployment

Sharded database deployment includes the prerequisites and instructions for installing the required software components, creating the catalog, roles, and the sharded database, configuring replication for high availability, and creating the schema for the sharded database.

The following topics contain the concepts and tasks you need to deploy a sharded database:

Introduction to Sharded Database Deployment

Oracle Sharding 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

Note that there are many different configurations and topologies that can be used for a sharded database. Your particular sharded database may employ a variety of Oracle software components such as Oracle Data Guard, Oracle GoldenGate, and Oracle Real Application Clusters (Oracle RAC) along with different sharding methodologies including system-managed, composite, and user-defined sharding.

Depending on your application's particular architecture and system requirements, you may have several choices from which to choose when designing your system. See Sharding Methods, Shard-Level High Availability for information about the various sharding methodologies and disaster recovery and high-availability options.

Choosing a Shard Creation Method

When deploying a sharded configuration, there are two different `GDSCTL` commands, `ADD SHARD` and `CREATE SHARD`, that can be used to add a shard.

Before you start to configure the sharding topology, decide which shard creation method to use because this decision affects some of the configuration steps.

The differences between the `ADD SHARD` and `CREATE SHARD` methods are explained where necessary in the configuration instructions.

ADD SHARD Method

The `GDSCTL ADD SHARD` command can be used to add a shard to an Oracle Sharding configuration. When using this command, you are responsible for creating the Oracle databases that will become shards during deployment. You can use whatever method you want to create the databases as long as the databases meet the prerequisites for inclusion in an Oracle Sharding configuration.

Some of the benefits of using the `ADD SHARD` method include:

- You have complete control over the process used to create the databases.

- It is straightforward to customize database parameters, naming, and storage locations.
- Both PDB and non-CDB shards are supported.
- There is less Oracle software to configure on the shard hosts.
- There is much less complexity in the deployment process because the shard databases are created before you run any `GDSCTL` commands.

CREATE SHARD Method

The `GDSCTL CREATE SHARD` command can be used to create a shard in an Oracle Sharding configuration. With `CREATE SHARD`, the shard catalog leverages the Oracle Remote Scheduler Agent to run the Database Configuration Assistant (DBCA) remotely on each shard host to create a database for you. This method does not support PDBs, so any shard databases added must be non-CDBs.

Some of the benefits of using the `CREATE SHARD` method include:

- It is easier to create shard databases for non-database administrators.
- It provides a standard way to provision a new database, when no standard is in current practice.
- Any database created with `CREATE SHARD` is automatically configured correctly for Oracle Sharding without the need to run SQL statements against the database or otherwise adjust database parameters.
- You can create standby databases automatically.

Sharded Database Deployment Roadmap

Follow this roadmap to set up hosts, install the required software, and configure and deploy a sharded database.

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).
   - If you are using the `ADD SHARD` method to deploy shards, 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 Sharded Database Topology).
   - `CREATE SHARDCATALOG`
   - `ADD GSM`

- START GSM
- ADD SHARDSPACE
- ADD SHARDGROUP
- ADD CDB
- ADD SHARD
- ADD CREDENTIAL
- ADD FILE
- CREATE SHARD
- ADD INVITEDNODE

4. Run DEPLOY to deploy the sharding topology configuration (see Deploy the Sharding 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 Sharded 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 Sharded Database Deployment walks you through a specific deployment scenario of a representative reference configuration. This section provides examples of every command needed to produce a fully functional sharded database after all of the steps are completed.

Provision and Configure Hosts and Operating Systems

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

- **Oracle Database Enterprise Edition** is required when running an Oracle Sharded 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** requirements are Low Latency GigE.
- **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, this database acts as a 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.
- **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 the shard director is co-located on the same host as the catalog database it 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, replication technology such as Oracle Data Guard or Oracle Golden Gate should be used, and replicas created for all sharded data. Additional hosts will be needed to run these replica or standby databases.

> **Note:**
>
> Oracle GoldenGate replication support for Oracle Sharding High Availability is deprecated in Oracle Database 21c.

Once 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 Sharding 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.

If you will be using the `CREATE SHARD` method to add shards to your configuration, you must also install the Remote Scheduler Agent software on each shard host. The agent does not need to be installed on the shard catalog hosts. See Installing and Configuring the Scheduler Agent on a Remote Host for more information.

The `CREATE SHARD` method also requires the creation of two directories on each shard host, `$ORACLE_BASE/oradata` and `$ORACLE_BASE/fast_recovery_area`. Create these two directories while logged into the shard host as the owner of the Oracle Database software. Permissions should be set the same as for the directories that will hold the data files for your shard database, which is typically full access for the software owner only.

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

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 legacy database or 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> REM run the following command if using a CDB
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> REM run the following command if using a CDB
```

```
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> REM run the following command if using a CDB
  SQL> alter session set container=catalog_pdb_name;

  SQL> show parameter db_files

  NAME                                 TYPE        VALUE
  ------------------------------------ ----------- ------------
  db_files                             integer     1024
  ```

- If you are planning to use `CREATE SHARD` to add shards to the sharding configuration, then the `SHARED_SERVERS` and `DISPATCHERS` database initialization parameters must be set to allow the Remote Scheduler Agent to connect to the catalog over an XDB connection. This is not necessary if `ADD SHARD` will be used.

  Specifically, `SHARED_SERVERS` must be greater than 0 (zero) to allow shared server connections to the shard catalog from the Remote Scheduler Agent processes running on the shard hosts. In addition, the value of `DISPATCHERS` must contain a service for XDB, based on the Oracle SID value.

  ```
  $ sqlplus / as sysdba

  SQL> show parameter shared_servers

  NAME                                 TYPE        VALUE
  ------------------------------------ ----------- ------------
  shared_servers                       integer     5

  SQL> show parameter dispatchers

  NAME                                 TYPE        VALUE
  ------------------------------------ ----------- ----------------------
  Dispatchers                          string      (PROTOCOL=TCP), (PROTO
                                                   COL=TCP)(SERVICE=mysid
                                                   XDB)
  ```

After setting the parameter values appropriately, run the `ALTER SYSTEM REGISTER` command to ensure that the XDB service is available for incoming connection requests.

- 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> REM run the following command if using a CDB
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 legacy database or 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.

  If you are using a PDB as the shard catalog, 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, host only one shard catalog PDB per CDB, and do not unlock the `GSMCATUSER` account in any other PDBs.

  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.
```

If you are using a PDB as the shard catalog, also run the following commands.

```
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 legacy database or 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> REM run the following command if using a CDB
  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 legacy database or PDB.

  The TNS Listener can be created and configured in whatever way you wish. If the shard catalog is a PDB, 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 when using a PDB for the shard catalog, 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

If you are using the `CREATE SHARD` method to add shards to your configuration, then skip this topic as it does not apply to `CREATE SHARD`. Otherwise, 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 cloud infrastructure tools.

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

- If the shard will be a PDB in a CDB, then 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.
```

- If the shard will be a PDB, then create a PDB for use as the shard database. Using the root container (`CDB$ROOT`) of a CDB as a shard is not supported.
- 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> REM run the following command if using a CDB
  SQL> alter session set container=shard_pdb_name;

  SQL> show parameter spfile

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

- 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> REM run the following command if using a CDB
  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
  ```

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

```
$ sqlplus / as sysdba

SQL> REM run the following command if using a CDB
SQL> alter session set container=shard_pdb_name;

SQL> show parameter compatible

NAME                 TYPE        VALUE
-------------------- ----------- -----------------
compatible           string      19.0.0
```

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

```
$ sqlplus / as sysdba

SQL> REM run the following command if using a CDB
SQL> alter session set container=shard_pdb_name;

SQL> select flashback_on from v$database;

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

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

```
$ sqlplus / as sysdba

SQL> REM run the following command if using a CDB
SQL> alter session set container=shard_pdb_name;

SQL> select force_logging from v$database;

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

- 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 datafile. 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 tablespaces in the system.

```
$ sqlplus / as sysdba

SQL> REM run the following command if using a CDB
SQL> alter session set container=shard_pdb_name;

SQL> show parameter db_files
```

```
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------
db_files                             integer     1024
```

- 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> REM run the following command if using a CDB
  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
  ```

- A directory object named `DATA_PUMP_DIR` must be created and accessible in the shard database 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> REM run the following command if using a CDB
  SQL> alter session set container=shard_pdb_name;

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

  Directory created.

  SQL> grant read, write on directory DATA_PUMP_DIR to gsmadmin_internal;

  Grant succeeded.
  ```

- 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> REM run the following command if using a CDB
  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/
```

- 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
  ```

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

  If the shards are PDBs, note that `GSMUSER` is a common user in the CDB. 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> REM run the following commands if using a CDB
  SQL> alter session set container=shard_pdb_name;

  SQL> alter user gsmuser account unlock;
  ```

```
User altered.

SQL> REM all cases run the following command

SQL> grant SYSDG, SYSBACKUP to gsmuser;

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 PDB.

  The TNS Listener can be created and configured in whatever way you wish. If the shard is a PDB, 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 when using a PDB for the shard, 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`).

Validate the Shard Database

To validate that all of the above 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> REM run the following command if using a CDB
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 Sharded Database Topology

The sharded database topology is descibed by the sharding metadata in the shard catalog database. Use `GDSCTL` to configure the sharded database topology.

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 the order presented, 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, composite, user-defined), replication technology (Oracle Data Guard, Oracle GoldenGate), default number of chunks in the database, and others. Make sure that you consult the `GDSCTL` reference documentation for the complete list of possible command options and their defaults.

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

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 of the 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.

If the shard catalog database is a PDB, 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)
    )
  )
```

If you are using the `ADD SHARD` method to create shards, do only the first step. If you are using the `CREATE SHARD` method, do both steps.

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

   Additional Parameters Required for CREATE SHARD Method

   If you will use the `CREATE SHARD` method to add shards to the configuration, then when you run `CREATE SHARDCATALOG` you must set the following additional parameters, which are required for Remote Scheduler Agent registration in the next step.

   - `-agent_password` specifies the password that will be used by the Remote Scheduler Agent to register with the shard catalog.
   - `-agent_port` specifies the port number that the Agent uses to create an XDB connection to the shard catalog. The default for this parameter is 8080.

   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.

   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
   ```

2. **For CREATE SHARD method only:** Register the Remote Scheduler Agents with the shard catalog, and start the agents on each shard host.

   Go to each shard host, log in as the owner of the Oracle software installation, and run the following `schagent` commands in the Oracle Home from which the shard database will run.

   ```
   schagent -registerdatabase catalog_hostname agent_port
   schagent -start
   ```

In the `schagent` command above, replace *catalog_hostname* with the name of the shard catalog host, and replace *agent_port* as with the port number you configured in `CREATE SHARDCATALOG` above.

For example:

```
$ $ORACLE_HOME/bin/schagent -registerdatabase cathost.example.com 8080
$ $ORACLE_HOME/bin/schagent -start
```

After successful agent registration, the shard host can receive remote job requests from the shard catalog during `GDSCTL DEPLOY`. After a successful deployment on a given host, the Remote Scheduler Agent is no longer used during a sharded database's life cycle and can be stopped safely using the following command.

```
$ $ORACLE_HOME/bin/schagent -stop
```

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 Shardgoups 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

If your shards will be PDBs inside CDBs, then add the CDBs containing the shard PDBs to the sharding configuration with the ADD CDB command. If you will be using non-CDBs as your shards, or will be using CREATE SHARD to add shards, skip to the next section.

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 Shards

Depending on whether you use ADD SHARD or CREATE SHARD to add shards to your configuration, follow the appropriate instructions below.

Add Shards Using GDSCTL ADD SHARD

Use the GDSCTL ADD SHARD command to add the shard information to the shard catalog.

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
```

Do not specify the `-cdb` parameter if you are not using PDBs as your shards.

In the examples above, 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, and re-runs `dbms_gsm_fix.validateShard` to check for errors. Then `GDSCTL` constructs the shard name using the following conventions.

- For PDB shards: *db_unique_name_of_CDB_PDB_name*, for example `cdb1_pdb1`
- For legacy database shards: *db_unique_name_of_DB*, which is simply the `db_unique_name`

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 Shards Using GDSCTL CREATE SHARD

Use the `GDSCTL CREATE SHARD` command to create the shard database and add the shard information to the shard catalog.

Run `CREATE SHARD` with the parameters appropriate to your sharding method, as shown in the following examples.

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

```
GDSCTL> create shard -shardgroup shardgroup_name -destination shard_hostname
 -osaccount account_name -ospassword account_password
```

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

```
GDSCTL> create shard -shardspace shardspace_name -deploy_as db_mode
 -destination shard_hostname -osaccount account_name -ospassword
account_password
```

The `-shardgroup` or `-shardspace` parameters specify the location of the shard in your sharding topology, and `-deploy_as` specifies the intended open mode (`primary`, `standby`, `active_standby`) of the shard.

The `-destination` parameter specifies which Remote Scheduler Agent the shard catalog contacts to spawn NETCA and DBCA to create the shard database. This value is typically the host name of the shard host. To see the list of available destinations, select from the `ALL_SCHEDULER_EXTERNAL_DESTS` view on the shard catalog database.

The `-osaccount` and `-ospassword` parameters specify the operating system user name and password to be used when spawning the NETCA and DBCA processes on the shard host. Typically, the user name is the owner of the Oracle Database software.

Password Encryption

To avoid specifying the cleartext password for the account on each `CREATE SHARD` command, you can store the encrypted password in the shard catalog for later use by using the `GDSCTL ADD CREDENTIAL` command. In the `CREATE SHARD` command specify the credential name in the command parameters instead of `-osaccount` and `-ospassword` as shown below.

```
GDSCTL> add credential -credential credential_name
 -osaccount account_name -ospassword account_password

GDSCTL> create shard -shardgroup shardgroup_name -destination shard_hostname
 -credential credential_name
```

What Happens When You Run CREATE SHARD

When you run `CREATE SHARD`, `GDSCTL` validates the input parameters and the shard host setup and then adds shard metadata to the shard catalog, which in turn causes the following operations to be performed during `GDSCTL DEPLOY`:

- Create and start a TNS Listener process on shard hosts at port 1521 with a listener name of “LISTENER” (by default)
- For primary shards, create the shard database using the default DBCA template located on the shard host at `$ORACLE_HOME/assistants/dbca/templates/General_Purpose.dbc` (by default)

  The primary shards will have the following characteristics, by default:

  – Randomly generated passwords for `SYS`, `SYSTEM`, and `GSMUSER`

  - A db_unique_name, db_name, and SID of the form 'shNN' where NN is a sequence-based number to uniquely identify added shards
  - A db_domain value the same as the domain found in the ALL_SCHEDULER_EXTERNAL_DESTS.HOSTNAME column corresponding to the specified destination. If no domain is found, then db_domain is set to the db_domain of the shard catalog database.
  - NLS_CHARACTERSET and NLS_NCHAR_CHARACTERSET values the same as those for the shard catalog database
  - The db_file_name_convert parameter set to '*','$ORACLE_BASE/oradata/'
  - The db_create_file_dest parameter set to $ORACLE_BASE/oradata
  - The remote_login_passwordfile parameter set to EXCLUSIVE
  - The database is in archivelog mode
  - Force logging is enabled
  - Database flashback is on
  - If Oracle Data Guard replication was specified on CREATE SHARDCATALOG, the following parameters are set.
    * dg_broker_start set to TRUE
    * db_recovery_file_dest set to $ORACLE_BASE/fast_recovery_area
    * db_recovery_file_dest_size set to 51200 MB
    * standby_file_management set to AUTO
    * db_flashback_retention_target set to 60

- For standby shards, use DBCA and RMAN to create the standby database based on the existing primary. In general, all primary database parameters are inherited by the standbys.

CREATE SHARD Usage Tips For Database Customization

The GDSCTL CREATE SHARD command has several parameters that let you customize the shard databases.

- The -sys_password and -system_password parameters let you specify the passwords for the SYS and SYSTEM accounts on your new shards.

  Note that the GSMUSER password is always created randomly because this account is not intended to be used for interactive logins. To change the GSMUSER password after deployment, change the password on the database with ALTER USER and then use the GDSCTL MODIFY SHARD command to update the sharding metadata with the new password.

- The -netparam and -netparamfile parameters let you customize the TNS listener name and port number when they are created on the shard host.

  The value specified for these parameters is the file name of a NETCA response file. Examples of response files can be found in $ORACLE_HOME/assistants/netca on the shard hosts.

- Likewise, you can use the -dbtemplate and -dbtemplatefile parameters to specify the DBCA template file to be used when creating the shard database.

  The default template is $ORACLE_HOME/assistants/dbca/templates/General_Purpose.dbc on the shard host.

- You can use the `-dbparam` and `-dbparamfile` parameters to directly pass DBCA command line parameters to the DBCA process on the shard host, just as you would enter them when running DBCA from the command line.

  To see all of the possible parameters for primary database creation, run `dbca -help -createDatabase`. To see parameters for standby database creation, run `dbca -help -createDuplicateDB`. For example, to change the global database name and SID for a primary shard, create a file with a single line as shown below, and specify that file name in `-dbparam` or `-dbparamfile`.

  ```
  -gdbName mydb.example.com -sid mysid
  ```

- When specifying any of these parameters, you can use `-netparamfile`, `-dbtemplatefile`, or `-dbparamfile` with an operating system file name, as seen from `GDSCTL`.

  Alternatively, you can save the contents of the file in the shard catalog database with the `ADD FILE` command and then use `-netparam`, `-dbtemplate`, or `-dbparam` on your `CREATE SHARD` command.

  ```
  GDSCTL> create shard -dbtemplatefile /home/user/mytemplate.dbc -
  netparamfile /home/user/mynetca.rsp ...
  ```

  or

  ```
  GDSCTL> add file -file mytemplate -source /home/user/mytemplate.dbc
  GDSCTL> add file -file mynetca -source /home/user/mynetca.rsp
  GDSCTL> create shard -dbtemplate mytemplate -netparam mynetca ...
  ```

If you wish to ensure that a standby shard on a particular host is in the same Data Guard configuration as a specific primary shard, it is recommended that you create the primary shard first followed by the standby shard using the desired `-destination` value in `CREATE SHARD`. If you create several primary shards sequentially, and then create several standby shards, the primaries and standbys are matched in Data Guard configurations in a non-deterministic way.

Verify the Shard Configuration

Run `GDSCTL CONFIG SHARD` to verify that the shard metadata on the shard catalog is as expected.

```
GDSCTL> config shard
Name      Shard Group          Status    State    Region    Availability
--------- -------------------  ------    -----    ------    ------------
sh1       shardgroup_primary   U         none     region1   -
sh2       shardgroup_primary   U         none     region1   -
sh3       shardgroup_standby   U         none     region2   -
sh4       shardgroup_standby   U         none     region2   -
```

Note that the value for Status is U for “undeployed”, and State and Availability are none and - until the `GDSCTL 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` or `CREATE 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` and `CREATE SHARD` commands add 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
   ```

Deploy the Sharding Configuration

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

When you run the `GDSCTL DEPLOY` command, the output looks like the following if you used the `ADD SHARD` command to configure the shards.

```
GDSCTL> deploy
deploy: examining configuration...
deploy: requesting Data Guard configuration on shards via GSM
```

```
deploy: shards configured successfully
The operation completed successfully
```

If you used `CREATE SHARD` to configure the shards, the `GDSCTL DEPLOY` command output will look similar to the following.

```
GDSCTL> deploy
deploy: examining configuration...
deploy: deploying primary shard 'sh1' ...
deploy: network listener configuration successful at destination 'shard1'
deploy: starting DBCA at destination 'shard1' to create primary shard
'sh1' ...
deploy: deploying primary shard 'sh2' ...
deploy: network listener configuration successful at destination 'shard2'
deploy: starting DBCA at destination 'shard2' to create primary shard
'sh2' ...
deploy: waiting for 2 DBCA primary creation job(s) to complete...
deploy: waiting for 2 DBCA primary creation job(s) to complete...
deploy: DBCA primary creation job succeeded at destination 'shard1' for shard
'sh1'
deploy: DBCA primary creation job succeeded at destination 'shard2' for shard
'sh2'
deploy: deploying standby shard 'sh3' ...
deploy: network listener configuration successful at destination 'shard3'
deploy: starting DBCA at destination 'shard3' to create standby shard
'sh3' ...
deploy: deploying standby shard 'sh4' ...
deploy: network listener configuration successful at destination 'shard4'
deploy: starting DBCA at destination 'shard4' to create primary shard
'sh4' ...
deploy: waiting for 2 DBCA standby creation job(s) to complete...
deploy: waiting for 2 DBCA standby creation job(s) to complete...
deploy: DBCA standby creation job succeeded at destination 'shard3' for shard
'sh3'
deploy: DBCA standby creation job succeeded at destination 'shard4' for shard
'sh4'
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.
- If the `CREATE SHARD` method is used to create shards, PL/SQL code on the shard catalog schedules a Remote Scheduler Agent job on each shard host that runs NETCA, which creates and starts a TNS Listener. Then, a second job is scheduled to run DBCA on the shard host, which creates the shard database. If standbys are to be deployed, then another set of NETCA and DBCA jobs are run, which create the standby databases on their respective hosts after the primary databases are successfully created.

- For shards that are being 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 using PDBs and the `ADD SHARD` method, 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 you used the `CREATE SHARD` method, or used `ADD SHARD` with non-CDBs, then the shard names are the `db_unique_name` value of the shard databases.

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.

If you used the `CREATE SHARD` method to add shards to your configuration, and errors from NETCA or DBCA are returned during `GDSCTL DEPLOY` from Remote Scheduler Agent jobs, then do the following steps to resolve the errors and retry the deployment.

1. Resolve the issues.

   The error message returned by `GDSCTL DEPLOY` should have enough information to view the output from the failed job on the shard host.

   Typically, there will be trace and log files from the NETCA or DBCA execution in `$ORACLE_BASE/cfgtoollogs` on the shard host. Resolve any underlying issues that caused the failure (bad parameters, resource issues on the host, and the like).

2. Re-set the shard host.

   a. Stop any running TNS listeners created during the attempted deployment.

   b. Stop any running shard databases started during the attempted deployment.

   c. Delete `$ORACLE_HOME/network/admin/listener.ora`

   d. Delete all files associated with the failed shard creation from `$ORACLE_BASE/oradata` and `$ORACLE_BASE/fast_recovery_area`

3. Run `GDSCTL DEPLOY` again.

Create and Start Global Database Services

After the shards are successfully deployed, and the correct status is 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.

After the services are started, your sharded database is ready for application schema creation and incoming client connection requests.

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 sharding configuration deployment, verify the detailed status of each 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: 19.0.0.0
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 Sharded Database Deployment

This example explains how to deploy a typical system-managed sharded database with multiple replicas, using Oracle Data Guard for high availability. The shard catalog and the shards in this example are PDBs and the shards are added to the configuration with the `ADD SHARD` command.

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

You are not required to map data to shards in system-managed sharding, 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 sharded Database, see System-Managed Sharding.

Example Sharded Database Topology

Consider the following system-managed sharded database configuration, where shardgroup 1 contains the primary shards, while shardgroups 2 and 3 contain standby replicas.

In addition, let's assume that the replicas in shardgroup 2 are Oracle Active Data Guard standbys (that is, databases open for read-only access), while the replicas in shardgroup 3 are mounted databases that have not been opened.

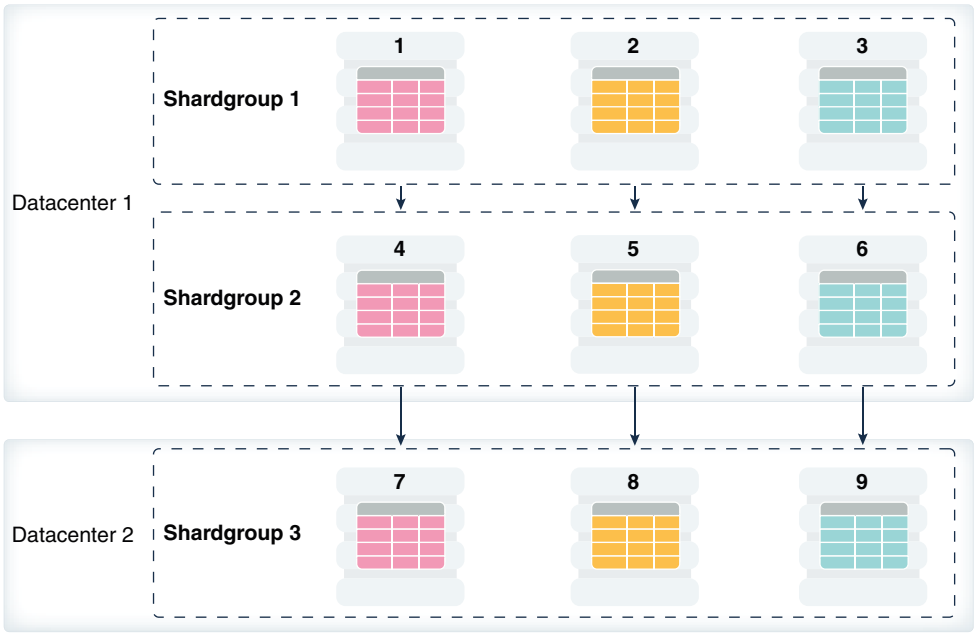


Table 4-1 Example System-Managed Topology Host Names

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 datacenters 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. This example shows how to use two shard directors per region, which is the best practice recommendation. Data center 1 • Shard director host names = gsmhost1 and gsmhost1b • Shard director name = gsm1 and gsm1b Data center 2 • Shard director hast name = gsmhost2 and gsmhost2b • Shard director name = gsm2 and gsm2b
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 Sharded 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, gsmhost1b, gsmhost2 gsmhost2b, and 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, gsmhost1b, gsmhost2, and gsmhost2b.

   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 gsmhost1b, run the following commands in GDSCTL.

```
connect mydbsadmin/mydbsadmin_password@cathost.example.com:1521/
catpdb.example.com
add gsm -gsm gsm1b -region dc1 -catalog cathost.example.com:1521/
catpdb.example.com -pwd gsmcatuser_password
start gsm -gsm gsm1b
```

See Add and Start Shard Directors for details.

c. 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.

d. On host gsmhost2b, run the following commands in GDSCTL.

```
connect mydbsadmin/mydbsadmin_password@cathost.example.com:1521/
catpdb.example.com
add gsm -gsm gsm2b -region dc2 -catalog cathost.example.com:1521/
catpdb.example.com -pwd gsmcatuser_password
start gsm -gsm gsm2b
```

See Add and Start Shard Directors for details.

e. 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 Shardgoups If Needed, Add the Shard CDBs, and Add Shards Using GDSCTL ADD SHARD for details.

f. 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.

g. Run `DEPLOY` from `GDSCTL` to complete the configuration of the sharded database.

   See Deploy the Sharding Configuration for details.

h. 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 Database Sharding in Oracle Cloud Infrastructure

Tooling for Oracle Sharding includes Terraform, Kubernetes, and Docker scripts to automate and further simplify the sharded database deployment operations.

Deploy a Sharded 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 Sharding 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 a Sharded Database With Terraform

Tooling for Oracle Sharding 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 Sharding 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 a Sharded Database with Docker

Oracle Sharding 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 Sharding 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

Sharded Database Schema Design

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 executed on a single shard.

Sharded Database Schema Design Considerations

Design of the database schema has a big impact on the performance and scalability of a sharded database. 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 Sharding 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 executed 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)?

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 Sharding 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 sharding 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.

Indexes on Sharded Tables

Only local indexes can be created on sharded tables. Unique local indexes on sharded tables must contain the sharding key.

Global indexes on sharded tables are not allowed because they can compromise the performance of online chunk movement.

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;
```

DDL Execution in a Sharded Database

To create a schema in a sharded database, you must issue DDL commands on the shard catalog database, which validates the DDLs and executes them locally before they are executed 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 master copy of the sharded database schema. If the shard catalog validation and execution 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 a new shard is added to a sharded database, all of the DDLs that have been executed 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 executing the DDL and returns the status of the execution.

```
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 executing 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 execute shard DDL is not recommended; create a privileged account for this purpose.

For information about DDL syntax extensions for Oracle Sharding, see DDL Syntax Extensions for Oracle Sharding.

Creating Objects Locally and Globally

Objects created using GDSCTL creates global, sharded 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 master 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.

DDL Syntax Extensions for Oracle Sharding

Oracle Sharding 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 Sharding, 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 Oracle Sharding. 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 Oracle Sharding:

- 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
- LONG, abstract (MDSYS data types are supported), REF data types
- Maximum number of columns without primary key is 999
- The `nologging` and `inmemory` options
- XMLType column in a duplicated table cannot be used in non-ASSM tablespace

See CREATE TABLE for more information about using the clauses supporting Oracle Sharding.

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
- Add foreign key constraint
- 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: long, abstract (`MDSYS` datatypes are supported), `REF`
- Column options: vector encode, invisible column, nested tables
- Object types
- 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`.

PL/SQL Procedure Execution in a Sharded Database

In the same way that DDL statements can be executed on all shards in a 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 propogated 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 procudures 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 catalog and it is not propogated 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 executes correctly on the shard catalog database, it will be queued for processing on all the currently deployed shards. Any error in the target procedure execution on the catalog is returned to the SQL*Plus session. Errors during execution on the shards can be tracked in the same way they are for DDLs.

Creating Sharded Database Schema Objects

The following topics show you how to create the schema objects in your sharded database. Refer back to the Sharded Database Schema Objects section in chapter 2 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 sharded 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 executing 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 Sharding.)

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

Creating Sharded Tables

A sharded table is a table that is partitioned into smaller and more manageable pieces among multiple databases, called shards.

Tablespace Set Sizing

In the system-managed and composite sharding methods, when you create a tablespace set on the shard catalog, you must make sure you have enough space for all of 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);
```

Assuming a default of 120 chunks per shard, the command creates 360 tablespaces of an initial tables space 100M each on the shard catalog and on each shard. While that doesn't sound like a lot of storage, when the database administrator allots 100G initially, they are not expecting 3.6TB per shard. If that amount of storage is not planned for, this may lead to a failed DDL, and will require significant effort to recover from.

Creating Sharded Tables in a System-Managed Sharded Database

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.

Creating Sharded Tables in a User-Defined Sharded Database

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 west;

CREATE TABLESPACE tbs3 IN SHARDSPACE central;
CREATE TABLESPACE tbs4 IN SHARDSPACE central;

CREATE TABLESPACE tbs5 IN SHARDSPACE east;
CREATE TABLESPACE tbs6 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_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
)
;
```

Creating Sharded Tables in a Composite Sharded Database

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)
;
```

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. This strategy is a good choice for relatively small tables that are not updated frequently, and that are often accessed together with sharded tables.

There are some limitations on duplicated tables. The following are not supported for duplicated tables.

- `NOLOGGING`
- `ALTER TABLE ADD/DROP CONSTRAINT` for primary key only
- `ALTER TABLE ADD/DROP PRIMARY KEY`
- `ALTER TABLE RENAME COLUMN`
- `PARTITION BY REFERENCE`
- `PARTITION BY SYSTEM`
- `CLUSTERED TABLE`
- Non-final `UDT` or `NESTED TABLE`
- `LONG DATATYPE`
- `COLUMN VECTOR ENCODE`
- `INVISIBLE COLUMN`
- Column encryption
- Information Lifecycle Management (ILM) policy
- CTAS Parallel
- 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.
- Creation of duplicated tables with `inmemory` option is not supported; however, altering a duplicated table to be `inmemory` is supported.

The Products duplicated table can be created using the following statement.

```
CREATE DUPLICATED TABLE Products
( StockNo     NUMBER PRIMARY KEY
, Description VARCHAR2(20)
, Price       NUMBER(6,2))
;
```

Updating Duplicated Tables and Synchronizing Their Contents

Oracle Sharding synchronizes the contents of duplicated tables using Materialized View Replication.

A duplicated table on each shard is represented by a materialized view. The master table for the materialized views is located in the shard catalog. The `CREATE DUPLICATED TABLE`

statement automatically creates the master table, materialized views, and other objects required for materialized view replication.

You can connect to any shard and update a duplicated table directly on the shard. The update is first propagated over a database link from the shard to the master table on the shard catalog. Then the update is asynchronously propagated to all other shards as a result of a materialized view 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.

> **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

Schema Creation Examples

The following examples show the steps you would take to create a schema for a sharded database using the system-managed, user-defined, and composite sharding methods.

Create a System-Managed Sharded Database Schema

Create the tablespace set, sharded tables, and duplicated tables for a sharded 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 execution and verify that the tablespace sets, tables, and chunks were correctly created on all of the shards.

Create a User-Defined Sharded Database Schema

Create the schema user, tablespace set, sharded tables, and duplicated tables for a sharded 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 execution and verify that the tablespace sets, tables, and chunks were correctly created on all of the shards.

Create a Composite Sharded Database Schema

Create the schema user, tablespace set, sharded tables, and duplicated tables for a sharded 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 execution and verify that the tablespace sets, tables, and chunks were correctly created on all of the shards.

Monitor DDL Execution and Verify Object Creation

You can monitor DDL execution using GDSCTL and SQL, to verify that the DDLs are propagated to all of the shards.

Monitor DDL Execution

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 executed using SQL*Plus on the shard catalog, because SQL*Plus does not return the execution 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 Execution Failure and Recovery Examples

The following examples demonstrate the steps to issue a DDL, monitor its execution 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 execution 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 execute on the shard catalog and, as expected, the GDSCTL `show ddl` command shows that no DDL was executed 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 executed 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 execution, the DDL is automatically applied when the shard is back up.

Example 6-2 Recovery from an error on a shard by executing a corrective action on that shard

In this example, the user attempts to create a tablespace set for system-managed sharded tables. But the datafile directory on one of the shards is not writable, so the DDL is successfully executed 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 SDB 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 executing 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 execute 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 execute `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 SDB, and only DDL numbers 1, 2, and 5 are applied.

Generating Unique Sequence Numbers Across Shards

Oracle Sharding allows you to generate globally unique sequence numbers across shards for non-primary key columns, and it is handled by the sharded database.

Customers often 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 a new object, `SHARDED SEQUENCE`. 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, new `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

7

Migrating to a Sharded Database

Migration from an existing non-sharded database to a sharded database consists of two phases: schema migration and data migration. Oracle Sharding provides guidelines for migrating your existing database schema and data to a sharded database.

The following approaches are recommended for database migration.

Using Oracle Data Pump to Migrate to a Sharded Database

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.

Migrating a Schema to a Sharded Database

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.

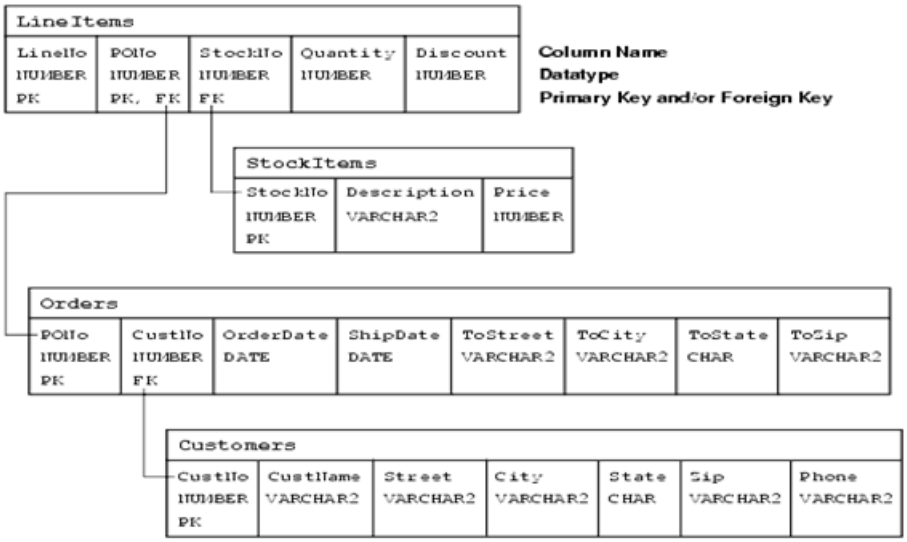
To properly design the sharded database schema, you must analyze the schema and workload of the non-sharded database and make the following decisions.

- 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 Sharding 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 7-1 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 master 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.

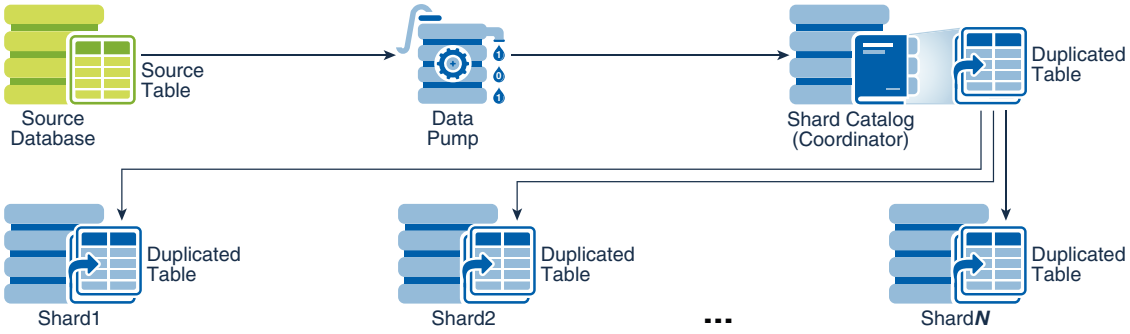
Moving data from non-sharded tables to duplicated tables is straightforward, but moving data from non-sharded tables to sharded tables requires special attention.

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 7-2 Loading Duplicated Tables



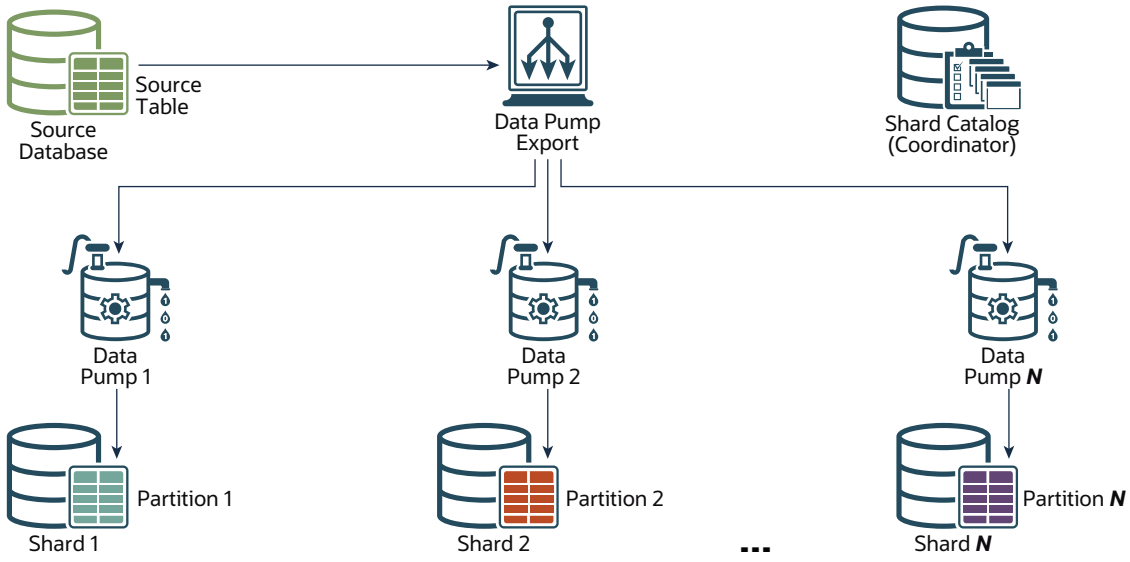
Loading Data into Sharded Tables

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 7-3 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.

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`.

```
$ 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
```

   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.

8

Query and DML Execution

On a sharded database, queries and DML can be routed to the shards for execution 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 execution.

How Database Requests are Routed to the Shards

In Oracle Sharding, 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 execute 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.

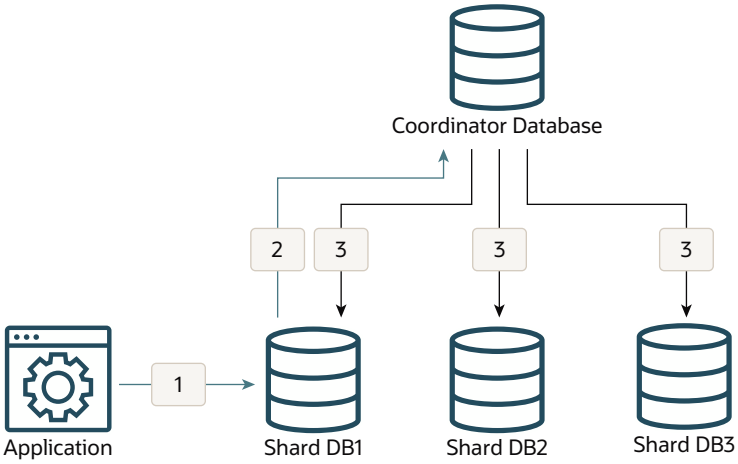
- Offers better performance: Overall, applications experience better performance compared to routing requests to the shards indirectly through the shard catalog (by proxy). With direct routing there is no need for the requests and the results to pass through a coordinator database.
- Accommodates geographic 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 queries is the data that belong to the shards accessed.
  - `SELECT`, `INSERT`, and `UPDATE` on duplicated tables: The scope of theses queries is all of the data in the duplicated tables. Because the master 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 master 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 the Sharded Database

Routing Queries and DMLs by Proxy

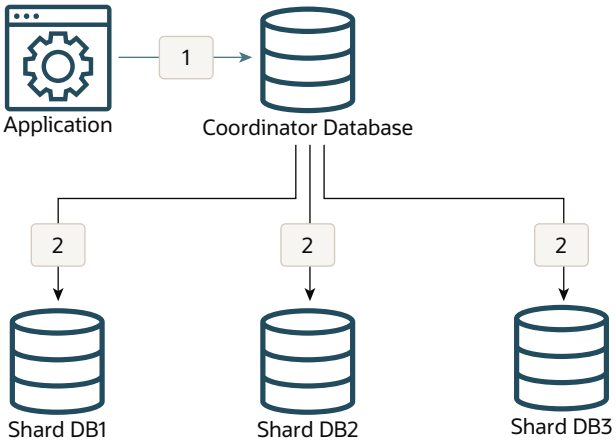
Using the shard catalog query coordinator as a proxy, Oracle Sharding can handle request routing for queries and DMLs that do not specify a sharding key.

By using the coordinator as a proxy, Oracle Sharding 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 executed.

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 master 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



For more information about the coordinator, see Query Processing and the Query Coordinator.

The remaining topics in this chapter discuss routing and processing database requests by proxy.

Connecting to the Query Coordinator

The Oracle Sharding 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 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

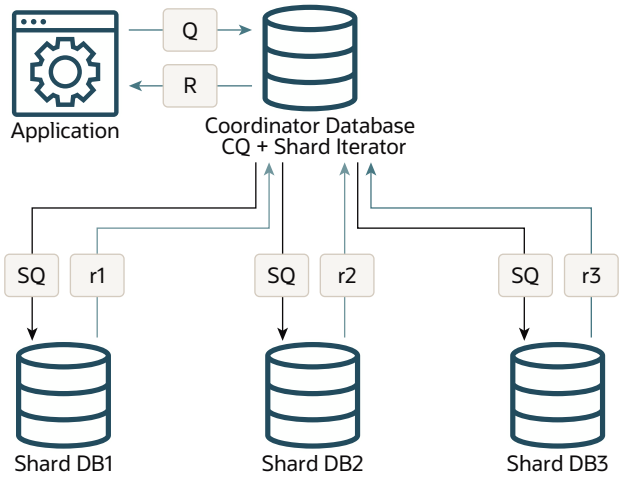
Query Coordinator Operation

The SQL compiler in the shard catalog identifies the relevant shards automatically, and coordinates the query execution 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 execution. 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 executed 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 executed 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 execution 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 executing 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.

Supported Query Constructs and Example Query Shapes

Oracle Sharding supports single-shard and multi-shard query shapes with some restrictions.

The following are restrictions on query constructs in Oracle Sharding.

- **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 Sharding.

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
```

Aggregate Functions Supported by Oracle Sharding

The following aggregations are supported by proxy routing in Oracle Sharding.

- `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 Sharding 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 executed 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 executed 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 execution of a multi-shard query can execute a shard SQL fragment on different set of shards, so every execution updates the shard IDs. 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 executed.

```
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 executed
- **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 sharding 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 remote execution of the `UPDATE` statement. 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 executed.

```
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 Sharding.

- **Parallel DML** Parallel DML is not supported by multi-shard DML. The DML will always run on one shard at a time (serially) in multi-shard DML.
- **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 by Oracle Sharding. 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 by Oracle Sharding, 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 by Oracle Sharding.
- **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**

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;bysecon
   d=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;bysecon
d=00',
   end_date             =>  NULL,
   enabled              =>  TRUE,
   comments             => 'Gather table statistics');
END;
/
```

9

Developing Applications for the Sharded Database

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 supported and executed 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 Sharding, 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 executes transactions directly on the shard. After all transactions for the given sharding key have been

executed, 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.

Sharding APIs Supporting Direct Routing

Oracle connection pools and drivers support Oracle Sharding.

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 for Oracle Sharding

Oracle Java Database Connectivity (JDBC) provides APIs for connecting to database shards in an Oracle Sharding 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 executed 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 an Oracle sharding key that is to be used with Oracle 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 for Oracle Sharding

Oracle Call Interface (OCI) provides an interface for connecting to database shards in an Oracle Sharding 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 Programmer'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 Programmer's Guide*

Oracle Universal Connection Pool APIs for Oracle Sharding

Oracle Universal Connection Pool (UCP) provides APIs for connecting to database shards in an Oracle Sharding 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. Execute 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 for Oracle Sharding

Oracle Data Provider for .NET (ODP.NET) provides APIs for connecting to database shards in an Oracle Sharding 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.

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 for Microsoft Windows*

10

Sharded Database Administration

Oracle Sharding provides tools and some automation for the administration of a sharded database.

The following topics describe sharded database administration in detail:

Managing the Sharding-Enabled Stack

This section describes the startup and shutdown of components in the sharded database configuration. It contains the following topics:

Starting Up the Sharding-Enabled Stack

The following is the recommended startup sequence of the sharding-enabled 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 Sharding-Enabled Stack

The following is the recommended shutdown sequence of the sharding-enabled 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 Sharding some types of users require certain roles and privileges.

For sharded databases there are two 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`.

- 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 Sharding Database Roles

Oracle Sharding provides a set of predefined database roles to help in sharded database administration.

Most of the Oracle Sharding 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 sharded database administrators, so that they can administer a sharded database configuration
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 global service managers to connect to databases in a GDS configuration.

`GSMUSER` exists by default on any Oracle database. In an Oracle Sharding 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.

If you use the `gdsctl create shard` command to create a new shard with the Database Configuration Assistant (DBCA), the `GSMUSER` account is automatically granted the SYSDG and SYSBACKUP privileges and assigned a random password during the deployment process. Because the `GSMUSER` account never needs to be logged into interactively, the value of the password does not need to be known by administrators; however, the password can be changed after deployment if required by using the `alter user` SQL command on the shard, in combination with the `gdsctl modify shard -pwd` command.

> **See Also:**
>
> add shard in *Global Data Services Concepts and Administration Guide*

About the GSMROOTUSER Account

`GSMROOTUSER` is a database account specific to Oracle Sharding 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 a Sharded Database

Because shards are hosted on individual Oracle databases, you can use Oracle Maximum Availability best practices to back up and restore shards individually.

If you are using Data Guard and Oracle Active Data Guard for SDB high availability, be sure to take observers offline and disable Fast Start Failover before taking a primary or standby database offline.

Contact Oracle Support for specific steps to recover a shard in the event of a disaster.

> **See Also:**
>
> Oracle Maximum Availability Architecture for MAA best practices white papers

Modifying a Sharded Database Schema

When making changes to duplicated tables or sharded tables in a sharded database, these changes should be done from the shard catalog database.

Before executing 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 executed 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

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.

Before Oracle Database 19c, you had to configure `ALTER SYSTEM` parameter settings on each shard in a sharded database. In Oracle Database 19c, Oracle Sharding 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.

Migrating a Non-PDB Shard to a PDB

Do the following steps if you want to migrate shards from a legacy single-instance database to Oracle multitenant architecture.

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 Sharded Database Software Versions

This section describes the version management of software components in the sharded database configuration. It contains the following topics:

Patching and Upgrading a Sharded 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 Sharding 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.

When using Oracle GoldenGate be sure to apply patches in parallel across the entire shardspace. 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.

> **Note:**
>
> Because logical standbys are not supported in Oracle Sharding, rolling upgrades may run into a DDL recovery issue because a physical standby database becomes a 'transient logical standby' during a rolling upgrade. To avoid this issue, follow the steps in Performing a Rolling Upgrade.

> **Note:**
>
> Oracle GoldenGate replication support for Oracle Sharding High Availability is deprecated in Oracle Database 21c.

Upgrading a Sharded Database

Upgrading the Oracle Sharding 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.

Performing a Rolling Upgrade

Because logical standbys are not supported in Oracle Sharding, rolling upgrades may run into a DDL recovery issue because a physical standby database becomes a 'transient logical standby' during a rolling upgrade.

To avoid this issue, perform the following steps.

1. Shut down the shard catalog database.

   Shutting down the shard catalog database prevents any shard director (GSM) from becoming the master, and the catalog will not try to apply any DDL in this state, but the shard director will continue in steady-state allowing production applications to connect and run.

2. Perform the rolling upgrade.
3. When the rolling upgrade is complete, start up the shard catalog database.

> **Note:**
>
> During a rolling upgrade, some operations such as automatic failover may not be available while the shard catalog is shut down.

Upgrading Sharded Database Components

The order in which sharded database components are upgraded is important for limiting downtime and avoiding errors as components are brought down and back online.

Before upgrading any sharded 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.

   Shard director upgrades should be done in-place; however, an in-place upgrade causes erroneous error messages unless permissions on the following files for the following platforms are updated to 755:
   - On Linux, Solaris64, and Solaris Sparc64:

     ```
     $ORACLE_HOME/QOpatch/qopiprep.bat
     $ORACLE_HOME/jdk/bin/jcontrol
     $ORACLE_HOME/jdk/jre/bin/jcontrol
     ```

- On AIX:

```
$ORACLE_HOME/QOpatch/qopiprep.bat
$ORACLE_HOME/jdk/jre/bin/classic/libjvm.a
$ORACLE_HOME/jdk/bin/policytool
```

- On HPI:

```
$ORACLE_HOME/jdk/jre/lib/IA64N/server/Xusage.txt
$ORACLE_HOME/jdk/jre/bin/jcontrol
$ORACLE_HOME/QOpatch/qopiprep.bat
```

- On Windows no error messages are expected.

4. Stop, upgrade, and restart all shard director servers one 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.

> **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.

Downgrading a Sharded Database

Oracle Sharding does not support downgrading.

Sharded database catalogs and shards cannot be downgraded.

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 Oracle Database 19c, 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 Sharded 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 executed 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
   ```

Managing Oracle Sharded Database with Enterprise Manager Cloud Control

Oracle Enterprise Manager Cloud Control lets you discover, monitor, and manage a sharded database and its components.

See the following topics for information about sharded 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
- Shard Management
- Chunk Management
- Shard Director Management
- Region Management
- Shardspace Management
- Shardgroup Management
- Services Management

Prerequisite: Enable Sharded Database Metrics

By default sharded 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. Grant the user permission to run the cross shard queries.

   ```
   alter session enable shard ddl;
   ```

2. 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*/
   ```

3. 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

4. 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
```

Discovering Sharded Database Components

In Enterprise Manager Cloud Control, you can discover the shard catalog and 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.

Overview of Sharded Database Management with Oracle Enterprise Manager Cloud Control

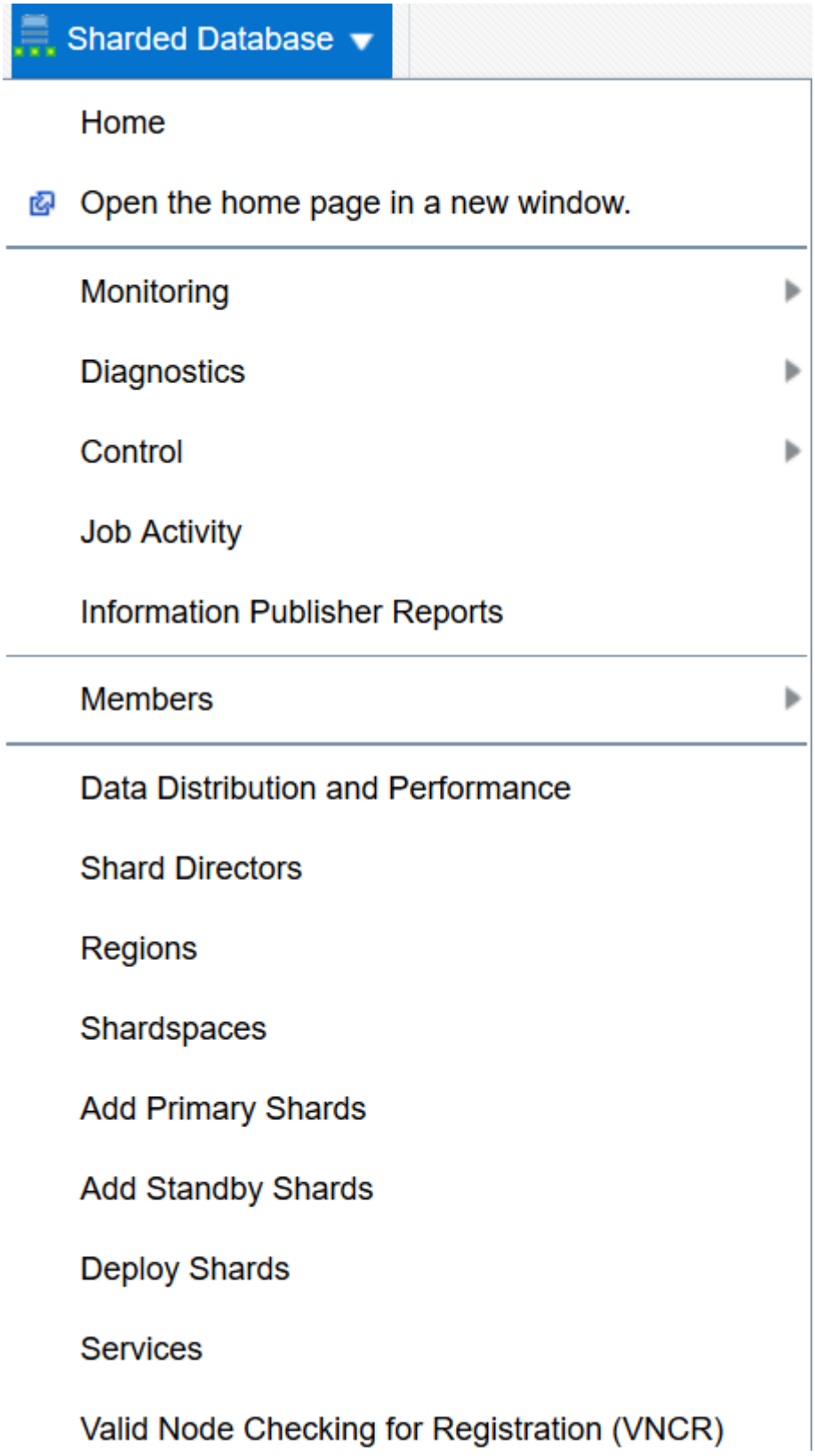
Your sharded database can be configured, deployed, monitored, and managed using Oracle Enterprise Manager Cloud Control

Any discovered sharded database objects can be found in the All Targets page in Enterprise Manager.

Shown below are the Oracle Sharding objects Shard Director and Shard Database in the Databases target type category.

Shown below are the Oracle Sharding 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.

Monitoring a Sharded Database

Sharded databases can be monitored using Enterprise Manager Cloud Control or GDSCTL.

See the following topics to use Enterprise Manager Cloud Control or GDSCTL to monitor sharded databases.

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 during execution of 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 a Sharded 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. When using Oracle GoldenGate for replication it is also important to monitor the lag of each replication stream.

> **See Also:**
>
> *Oracle Database Global Data Services Concepts and Administration Guide* for information about using the `GDSCTL CONFIG` commands

Monitoring a Sharded Database with Enterprise Manager Cloud Control

Sharded database targets are found in the All Targets page in Enterprise Manager Cloud Control.

To monitor sharded database components you must first enable statistics gathering and then discover the sharded database. See Prerequisite: Enable Sharded Database Metrics and Discovering Sharded Database Components 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:** Master shard director name. You can click the shard director name to view more information about the master 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 Oracle Sharding to contain all of the shardgroups. It is managed by Oracle Sharding 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.

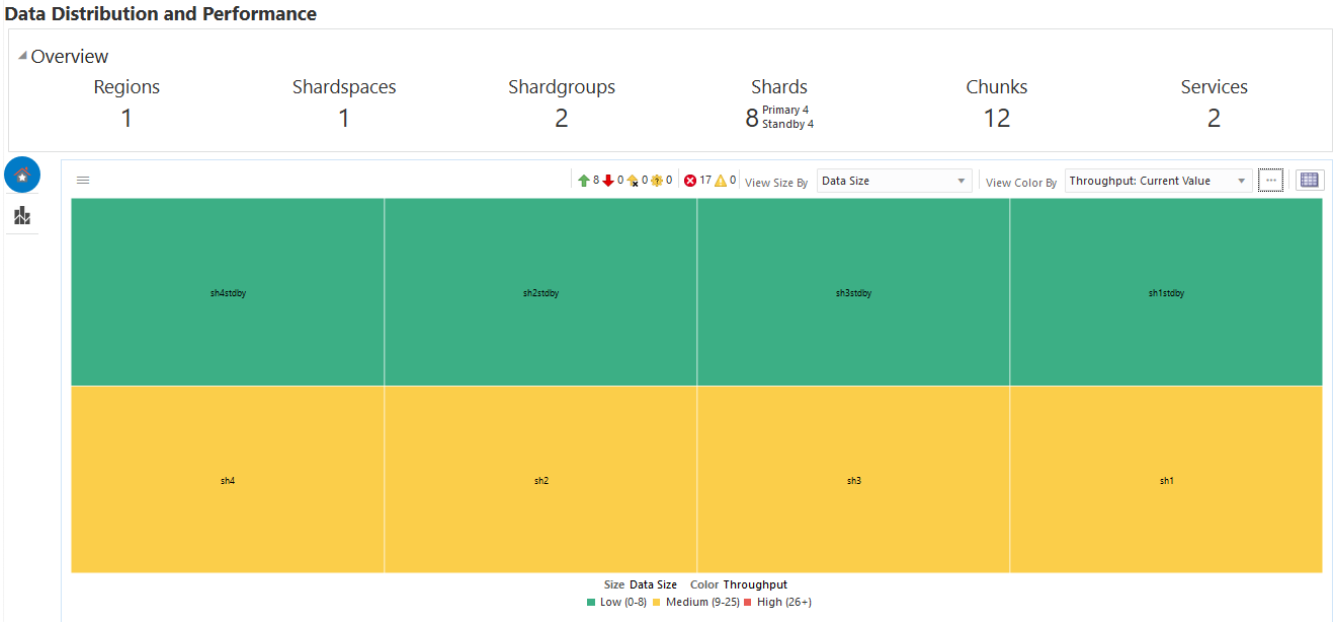
Clicking the navigation tree icon on the upper left corner of the Sharded Database home page opens the Target Navigation pane. This pane shows all of the discovered components in the sharded database in tree form.

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.



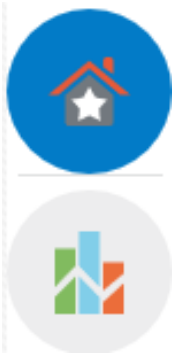
Overview

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 10-1 Home and Top Shards Icons

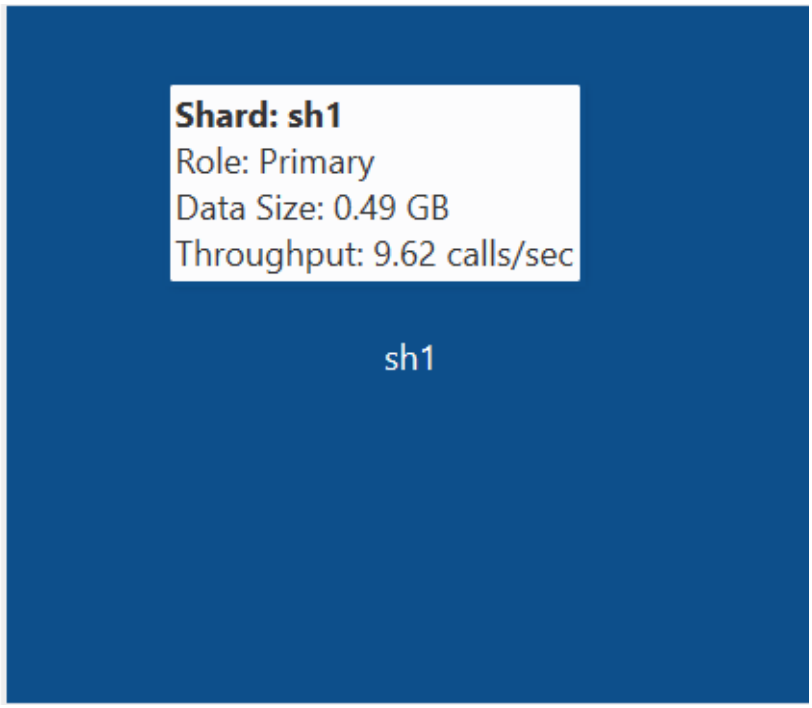


- **Home:** is the default view. Home displays data for all shards in the sharded database by default. You can filter the chart and change the metrics on display as described below.
- **Top Shards:** shows you charts for the top 5, 10, or 20 shards for certain metrics.

Shard Blocks

The color-coded chart displays data by shard. Each shard is indicated by a block.

Figure 10-2 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:

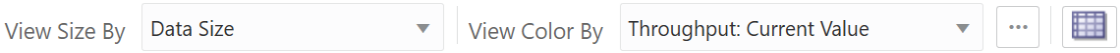
Figure 10-3 Home View Summary Icons

- **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 10-4 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

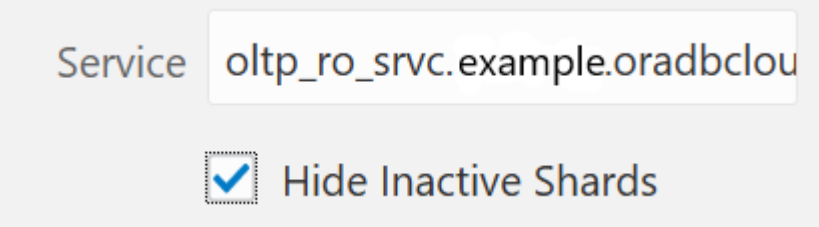
Filters

Click the hamburger icon at the top left corner of the chart to apply filters to the data.

Figure 10-5 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 executing 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.

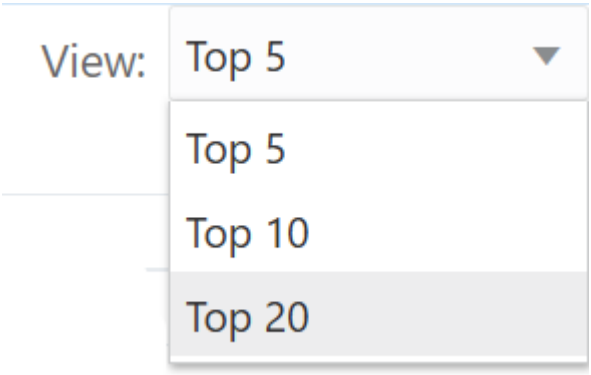


- **Group By:** Toggles that display aggregates for the group, which is indicated by a box line around the group of shards.
  - **Shardgroup** displays a shardgroup box at the top of the grouping, which displays aggregate info about the shardgroup on hover, and you can drill down for shardgroup-based 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.



Shard Management

You can manage shards in your Oracle Sharding deployment with Oracle Enterprise Manager Cloud Control and GDSCTL.

The following topics describe shard management concepts and tasks:

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 SDB to increase capacity.

When you add a shard to a sharded database, if the environment is sharded by consistent hash, 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.

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 an SDB.

Sometimes data in an SDB needs to be migrated from one shard to another. Data migration across shards is required in the following cases:

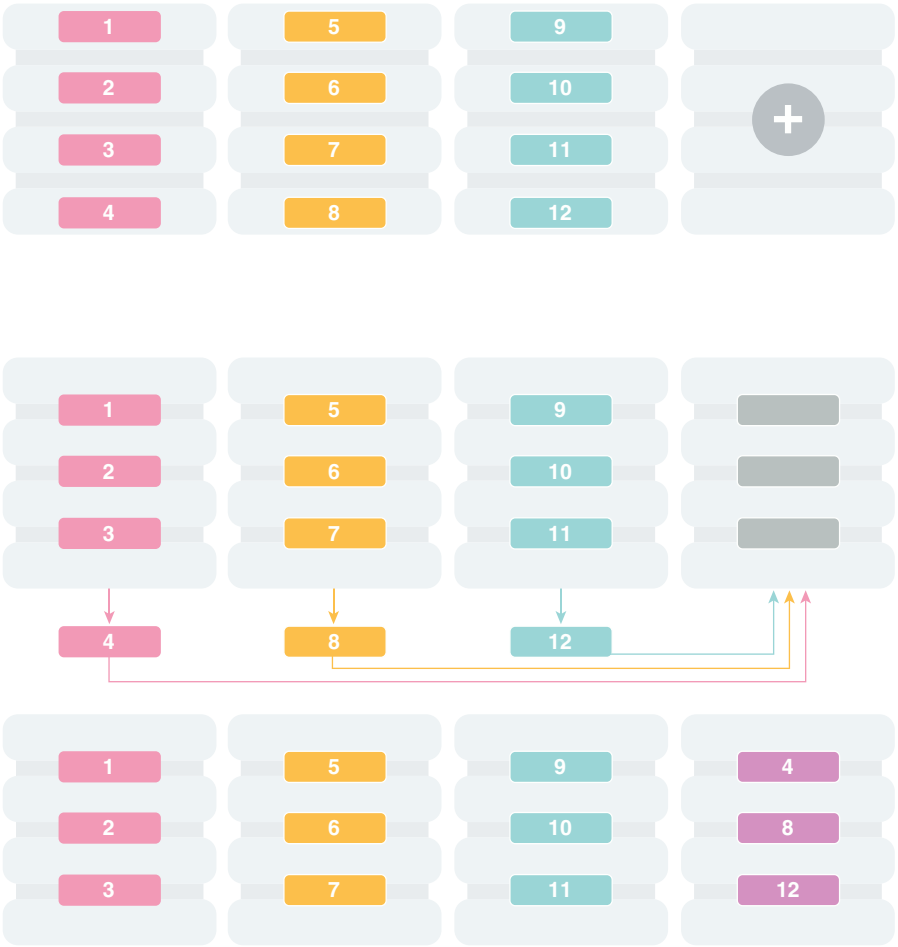
- When one or multiple shards are added to or removed from an SDB
- When there is skew in the data or workload distribution across shards

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 an SDB, 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 an SDB with three shards.

Figure 10-6 Resharding an SDB



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.

> **See Also:**
>
> Adding Shards to a System-Managed SDB
>
> Sharding Methods

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.

> **See Also:**
>
> About Moving Chunks
>
> *Oracle Database Global Data Services Concepts and Administration Guide* for information about using the `GDSCTL REMOVE SHARD` command

Adding Standby Shards

You can add Data Guard standby shards to an Oracle Sharding environment; however there are some limitations.

When using Data Guard as the replication method for a sharded database, Oracle Sharding 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. When converting a physical standby to a snapshot standby, the following steps should be followed:

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 Data Guard documentation.

   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.

> **See Also:**
>
> *Oracle Data Guard Concepts and Administration*

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.

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 Sharding deployment.

You can use Oracle Enterprise Manager Cloud Control to validate shards before adding them to your Oracle Sharding deployment. You can also validate a shard after deployment to confirm that the settings are still valid later in the shard lifecycle. 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. From a shardgroup management 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. 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

Use Oracle Enterprise Manager Cloud Control to add a primary shards to your Oracle Sharding deployment.

Primary shards should be existing targets that are being monitored by Cloud Control.

It is highly recommended that you validate a shard before adding it to your Oracle Sharding environment. 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 Validating a Shard).

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. 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 a shard and click **Select**.
6. In a composite Oracle Sharding environment you can select the shardspace to which to add the shard.
7. Click **OK**.
8. Enter the GSMUSER credentials if necessary, then click **Next**.
9. 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
10. Review the configuration of the shard to be added and click **Submit**.
11. 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 Sharding deployment using the Deploying Shards task.

Adding Standby Shards

Use Oracle Enterprise Manager Cloud Control to add a standby shards to your Oracle Sharding deployment.

Standby shards should be existing targets that are being monitored by Cloud Control.

It is highly recommended that you validate a shard before adding it to your Oracle Sharding environment. 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 Validating a Shard).

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. 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. Choose a primary shard for which the new shard will act as a standby in the **Primary Shards** list.
5. 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 Oracle Sharding deployment using the Deploying Shards task.

Deploying Shards

Use Oracle Enterprise Manager Cloud Control to deploy shards that have been added to your Oracle Sharding environment.

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.

Managing Shards with GDSCTL

You can manage shards in your Oracle Sharding deployment using the GDSCTL command-line utility.

The following topics describe shard management using GDSCTL:

Validating a Shard

Before adding a newly created shard to a sharding configuration, you must validate that the shard has been configured correctly for the sharding environment.

Before you run `ADD SHARD`, run the `validateShard` procedure against the database that will be added as a shard. The `validateShard` procedure verifies that the target database has the initialization parameters and characteristics needed to act successfully as a shard.

The `validateShard` procedure analyzes the target database and reports any issues that need to be addressed prior to running `GDSCTL ADD SHARD` on that database. The `validateShard` procedure does not make any changes to the database or its parameters; it only reports information and possible issues.

The `validateShard` procedure takes one optional parameter that specifies whether the shard will be added to a shard catalog using Data Guard or to a shard catalog using Oracle GoldenGate as its replication technology. If you are using Data Guard, call `validateShard('DG')`. If you are using Oracle GoldenGate, use `validateShard('OGG')`. The default value is Data Guard if no parameter is passed to `validateShard`.

The `validateShard` procedure can also be run after the deployment of a shard to confirm that the settings are still valid later in the shard lifecycle. For example, after a software upgrade or after shard deployment, `validateShard` can be run on existing shards to confirm correctness of their parameters and configuration.

Run `validateShard` as follows:

```
sqlplus / as sysdba
SQL> set serveroutput on
SQL> execute dbms_gsm_fix.validateShard
```

The following is an example of the output.

```
INFO: Data Guard shard validation requested.
INFO: Database role is PRIMARY.
INFO: Database name is DEN27B.
INFO: Database unique name is den27b.
INFO: Database ID is 718463507.
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 not a multitenant container database.
INFO: Database is using a server parameter file (spfile).
INFO: db_create_file_dest set to: '<ORACLE_BASE>/oracle/dbs2'
INFO: db_recovery_file_dest set to: '<ORACLE_BASE>/oracle/dbs2'
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/dt, /dbs/bt, dbs2/DEN27D/, dbs2/
DEN27B/'
INFO: GSMUSER account validated successfully.
INFO: DATA_PUMP_DIR is '<ORACLE_BASE>//oracle/dbs2'.
```

Any lines tagged with `INFO` are informational in nature and confirm correct settings. Lines tagged with `WARNING` may or may not be issues depending on your configuration. For example, issues related to Data Guard parameters are reported, but if your configuration will only include primary databases, then any Data Guard issues can be ignored. Finally, any output with the `ERROR` tag must be corrected for the shard to deploy and operate correctly in a sharding configuration.

Adding Shards to a System-Managed SDB

Adding shards to a system-managed SDB elastically scales the SDB. In a system-managed SDB chunks are automatically rebalanced after the new shards are added.

To prepare a new shard host, do all of the setup procedures as you did for the initial sharded database environment including:

- Install the Oracle Database Software

1. Connect to a shard director host, and verify the environment variables.

```
$ ssh os_user@shard_director_home
$ env |grep ORA
ORACLE_BASE=/u01/app/oracle
ORACLE_HOME=/u01/app/oracle/product/18.0.0/gsmhome_1
```

2. Set the global service manager for the current session, and specify the credentials to administer it.

```
$ gdsctl
GDSCTL> set gsm -gsm sharddirector1
GDSCTL> connect mysdbadmin/mysdbadmin_password
```

3. Verify the current shard configuration.

```
GDSCTL> config shard
Name            Shard Group         Status    State       Region
Availability
----            -----------         ------    -----       ------
------------
sh1             primary_shardgroup  Ok        Deployed    region1
ONLINE
sh2             standby_shardgroup  Ok        Deployed    region2
READ_ONLY
sh3             primary_shardgroup  Ok        Deployed    region1
ONLINE
sh4             standby_shardgroup  Ok        Deployed    region2
READ_ONLY
```

4. Specify the shard group, destination, and the credentials for each new shard.

   In the examples the new shard hosts are called shard5 and shard6, and they are using the default templates for NETCA and DBCA.

```
GDSCTL> add invitednode shard5
GDSCTL> create shard -shardgroup primary_shardgroup -destination shard5
 -credential os_credential -sys_password
GDSCTL> add invitednode shard6
```

```
GDSCTL> create shard -shardgroup standby_shardgroup -destination shard6
 -credential os_credential -sys_password
```

While creating the shards, you can also set the SYS password in the `create shard` using `-sys_password` as shown in the above example. This sets the SYS password after the shards are created during `DEPLOY`.

The above example uses the CREATE SHARD method for creating new shards. To add a preconfigured shard using the ADD SHARD command, do the following after `ADD INVITEDNODE`:

```
GDSCTL> add shard -shardgroup primary_shardgroup
 -connect shard_host:TNS_listener_port/shard_database_name
 -pwd GSMUSER_password
```

If the shard to be added is a PDB, you must use the `-cdb` option in `ADD SHARD` to specify which CDB the PDB shard is in. In addition, `ADD CDB` must be used before the `ADD SHARD` command to add the CDB to the catalog. See *Oracle Database Global Data Services Concepts and Administration Guide* for the syntax for `ADD CDB` and `ADD SHARD`.

> **Note:**
>
> The valid node checking for registration (VNCR) feature provides the ability to configure and dynamically update a set of IP addresses, host names, or subnets from which registration requests are allowed by the shard directors. Database instance registration with a shard director succeeds only when the request originates from a valid node. By default, the shard management tier (based on Oracle Global Data Services framework) automatically adds a VNCR entry for the host on which a remote database is running each time `create shard` or `add shard` is executed. The automation (called auto-VNCR) finds the public IP address of the target host, and automatically adds a VNCR entry for that IP address. If the host has multiple public IP addresses, then the address on which the database registers may not be the same as the address which was added using auto-VNCR and , as a result, registration many be rejected. If the target database host has multiple public IP addresses, it is advisable that you configure VNCR manually for this host using the `add invitednode` or `add invitedsubnet` commands in GDSCTL.
>
> If there are multiple net-cards on the target host (`/sbin/ifconfig` returns more than one public interface), use `add invitednode` to be safe (after finding out which interface will be used to route packets).
>
> If there is any doubt about registration, then use `config vncr` and use `add invitednode` as necessary. There is no harm in doing this, because if the node is added already, auto-VNCR ignores it, and if you try to add it after auto-VNCR already added it, you will get a warning stating that it already exists.

5. Run the `DEPLOY` command to create the shards and the replicas.

```
GDSCTL> deploy
```

6. Verify that the new shards are deployed.

```
GDSCTL> config shard
Name         Shard Group         Status    State       Region
Availability
----         -----------         ------    -----       ------
------------
sh1          primary_shardgroup  Ok        Deployed    region1   ONLINE
sh2          standby_shardgroup  Ok        Deployed    region2   READ_ONLY
sh3          primary_shardgroup  Ok        Deployed    region1   ONLINE
sh4          standby_shardgroup  Ok        Deployed    region2   READ_ONLY
sh5          primary_shardgroup  Ok        Deployed    region1   ONLINE
sh6          standby_shardgroup  Ok        Deployed    region2   READ_ONLY
```

7. Check the chunk configuration every minute or two to see the progress of automatic rebalancing of chunks.

```
$ gdsctl config chunks -show_Reshard

Chunks
------------------------
Database                       From      To
--------                       ----      --
sh1                            1         4
sh2                            1         4
sh3                            7         10
sh4                            7         10
sh5                            5         6
sh5                            11        12
sh6                            5         6
sh6                            11        12

Ongoing chunk movement
------------------------
Chunk     Source          Target             status
-----     ------          ------             ------
```

8. Observe that the shards (databases) are automatically registered.

```
$ gdsctl databases

Database: "sh1" Registered: Y State: Ok ONS: N. Role: PRIMARY Instances: 1
 Region: region1
   Service: "oltp_ro_srvc" Globally started: Y Started: N
            Scan: N Enabled: Y Preferred: Y
   Service: "oltp_rw_srvc" Globally started: Y Started: Y
            Scan: N Enabled: Y Preferred: Y
   Registered instances:
     cust_sdb%1
Database: "sh2" Registered: Y State: Ok ONS: N. Role: PH_STNDBY Instances:
1
 Region: region2
   Service: "oltp_ro_srvc" Globally started: Y Started: Y
            Scan: N Enabled: Y Preferred: Y
   Service: "oltp_rw_srvc" Globally started: Y Started: N
            Scan: N Enabled: Y Preferred: Y
```

```
   Registered instances:
     cust_sdb%11
Database: "sh3" Registered: Y State: Ok ONS: N. Role: PRIMARY Instances: 1
 Region: region1
   Service: "oltp_ro_srvc" Globally started: Y Started: N
            Scan: N Enabled: Y Preferred: Y
   Service: "oltp_rw_srvc" Globally started: Y Started: Y
            Scan: N Enabled: Y Preferred: Y
   Registered instances:
     cust_sdb%21
Database: "sh4" Registered: Y State: Ok ONS: N. Role: PH_STNDBY Instances:
1
 Region: region2
   Service: "oltp_ro_srvc" Globally started: Y Started: Y
            Scan: N Enabled: Y Preferred: Y
   Service: "oltp_rw_srvc" Globally started: Y Started: N
            Scan: N Enabled: Y Preferred: Y
   Registered instances:
     cust_sdb%31
Database: "sh5" Registered: Y State: Ok ONS: N. Role: PRIMARY Instances: 1
 Region: region1
   Service: "oltp_ro_srvc" Globally started: Y Started: N
            Scan: N Enabled: Y Preferred: Y
   Service: "oltp_rw_srvc" Globally started: Y Started: Y
            Scan: N Enabled: Y Preferred: Y
   Registered instances:
     cust_sdb%41
Database: "sh6" Registered: Y State: Ok ONS: N. Role: PH_STNDBY Instances:
1
 Region: region2
   Service: "oltp_ro_srvc" Globally started: Y Started: Y
            Scan: N Enabled: Y Preferred: Y
   Service: "oltp_rw_srvc" Globally started: Y Started: N
            Scan: N Enabled: Y Preferred: Y
   Registered instances:
     cust_sdb%51
```

9. Observe that the services are automatically brought up on the new shards.

```
$ gdsctl services

Service "oltp_ro_srvc.cust_sdb.oradbcloud" has 3 instance(s). Affinity:
ANYWHERE
   Instance "cust_sdb%11", name: "sh2", db: "sh2", region: "region2",
status: ready.
   Instance "cust_sdb%31", name: "sh4", db: "sh4", region: "region2",
status: ready.
   Instance "cust_sdb%51", name: "sh6", db: "sh6", region: "region2",
status: ready.
Service "oltp_rw_srvc.cust_sdb.oradbcloud" has 3 instance(s). Affinity:
ANYWHERE
   Instance "cust_sdb%1", name: "sh1", db: "sh1", region: "region1",
status: ready.
   Instance "cust_sdb%21", name: "sh3", db: "sh3", region: "region1",
status: ready.
```

```
   Instance "cust_sdb%41", name: "sh5", db: "sh5", region: "region1",
status: ready.
```

> **See Also:**
>
> *Oracle Database Global Data Services Concepts and Administration Guide* for information about GDSCTL command usage

Replacing a Shard

If a shard fails and is unrecoverable, 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 executed; 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 executing 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 alive. 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 alive, the primary database shuts down to maintain the protection mode. In this case, the primary database cannot be opened if the standby is not alive. 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 10-1 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 10-2 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
```

Replacing a Shard in an Oracle GoldenGate Environment

The GDSCTL command option ADD SHARD -REPLACE is not supported with Oracle GoldenGate.

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.

Chunk Management

You can manage chunks in your Oracle Sharding deployment with Oracle Enterprise Manager Cloud Control and GDSCTL.

The following topics describe chunk management concepts and tasks:

About 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 SDB, 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.)

> **See Also:**
>
> *Oracle Database Global Data Services Concepts and Administration Guide* for information about using the `GDSCTL MOVE CHUNK` command

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
```

Moving Chunks

You can move chunks from one shard to another in your Oracle Sharding 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.

About 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 Sharding 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 resharding

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.

> **See Also:**
>
> *Oracle Database Global Data Services Concepts and Administration Guide* for information about using the `GDSCTL SPLIT CHUNK` command

Splitting Chunks

You can split chunks in your Oracle Sharding 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 Oracle Sharding deployment with Oracle Enterprise Manager Cloud Control.

The following topics describe shard director management tasks:

Creating a Shard Director

Use Oracle Enterprise Manager Cloud Control to create and add a shard director to your Oracle Sharding 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 Oracle Sharding 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 Oracle Sharding 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 Oracle Sharding deployment with Oracle Enterprise Manager Cloud Control.

The following topics describe region management tasks:

Creating a Region

Create sharded database regions in your Oracle Sharding 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 Oracle Sharding 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 Oracle Sharding 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 Oracle Sharding deployment with Oracle Enterprise Manager Cloud Control.

The following topics describe shardspace management tasks:

Creating a Shardspace

Create shardspaces in your composite Oracle Sharding 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 Oracle Sharding deployment with Oracle Enterprise Manager Cloud Control.

The following topics describe shardgroup management tasks:

Creating a Shardgroup

Create shardgroups in your Oracle Sharding 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 Oracle Sharding deployment with Oracle Enterprise Manager Cloud Control.

To manage Oracle Sharding 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.

The following topics describe services management tasks:

Creating a Service

Create services in your Oracle Sharding 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.

11

Achieving Data Sovereignty with Oracle Sharding

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 Sharding

Oracle Sharding 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.

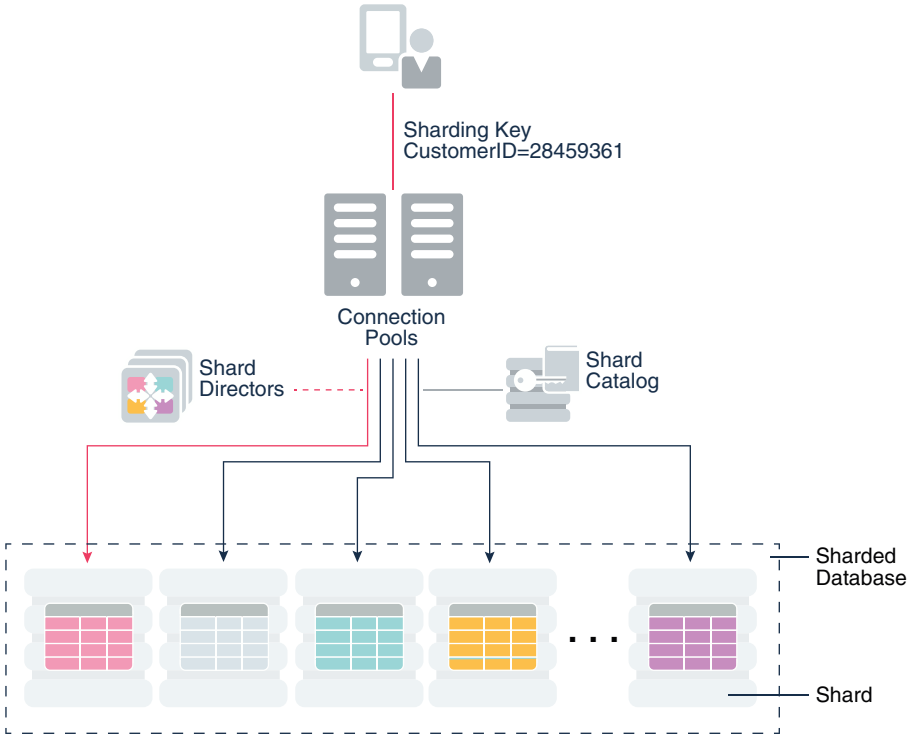
- It also supports storing particular data closer to its consumers. Oracle Sharding automates the entire lifecycle of a sharded database – deployment, schema creation, data-dependent routing with superior run-time performance, elastic scaling, and life-cycle management.
- It also provides the advantages 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 Sharding

Oracle Sharding 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 Sharding 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 11-1 Oracle Sharding Architecture



Additionally, Oracle Database security features such as Real Application Security (RAS) 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 by using Oracle Data Guard and Oracle GoldenGate for such replication.

Oracle Sharding 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 Sharding 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 Sharding pushes queries to the nodes where the data resides.

Oracle Sharding 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 Sharding can be used with Oracle Data Guard or Oracle GoldenGate to replicate data within the same Data Sovereignty region.

Use Case of Achieving Data Sovereignty with Oracle Sharding

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.

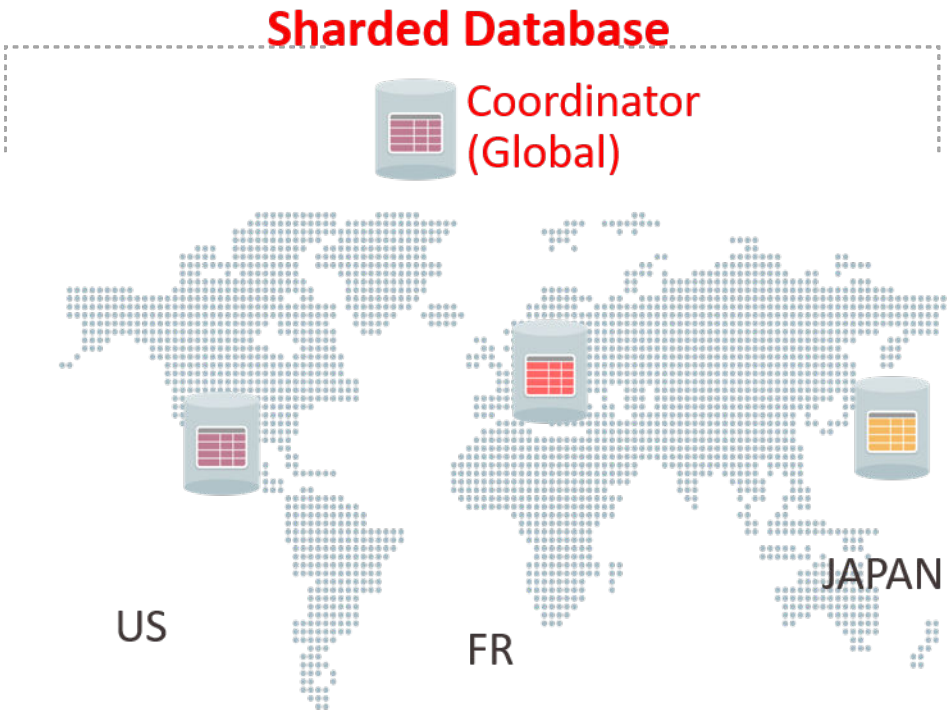
The access to the data has to be limited and data administrators in one country cannot see data in others. The solution for this use case is user-defined Sharding with shards configured in different countries and Real Application Security (RAS) for data access control.

Overview of Oracle Sharding Solution

Oracle Sharding 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 Sharding user-defined deployment between OCI data centers and on-premises across multiple regions. In this Oracle Sharding 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 Sharding allows you to query data in one shard (within one country), and Oracle Sharding supports multi-shard queries (that can query data from all the countries).

Figure 11-2 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 Sharding 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.

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.

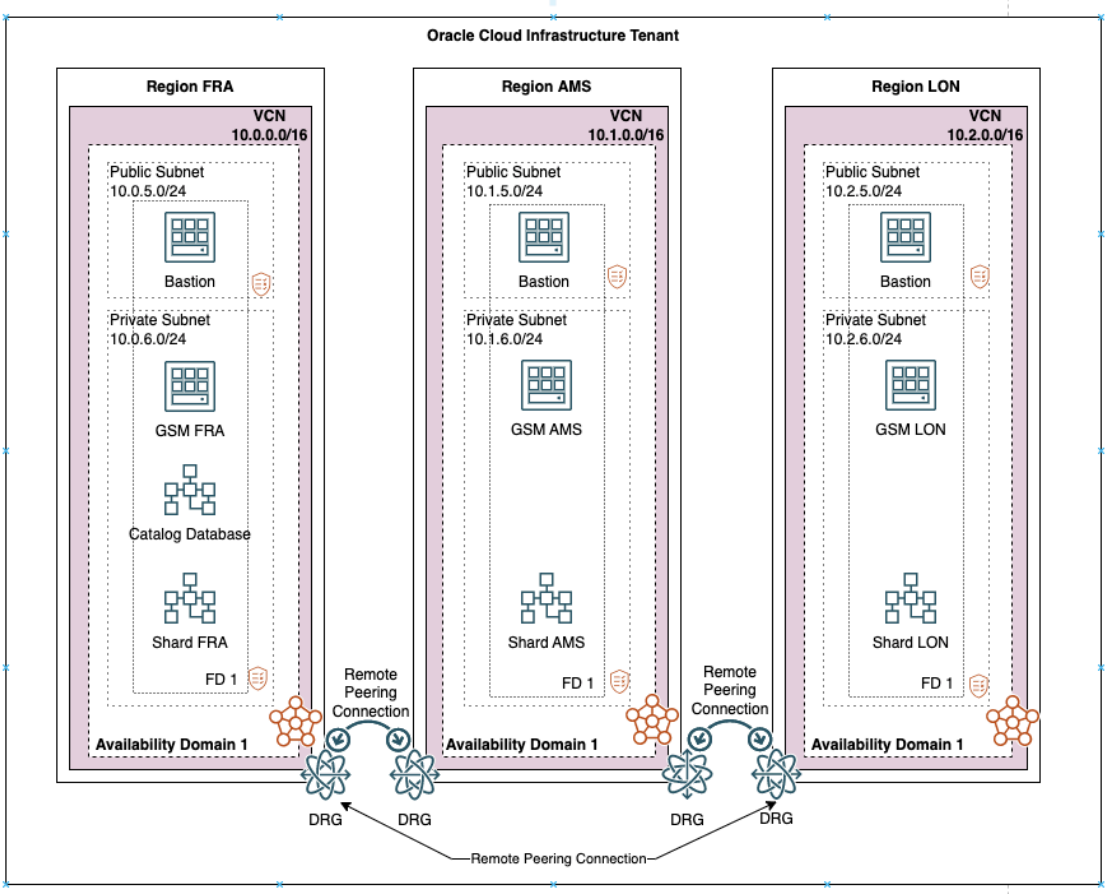
- Single Multimodel Big Data store with reduced volume of data duplication.
- Better fault isolation as planned/unplanned down time within one region/LOB does not impact other regions/LOBs.
- Ability to split busy partitions and shards as needed.
- Support for full ACID properties is critical for transactional applications.

Deployment Topology of Data Sovereignty with Oracle Sharding

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 11-3 Deployment Topology of Data Sovereignty with Oracle Sharding



Configuring Data Sovereignty with Oracle Sharding

Configure Data Sovereignty with Oracle Sharding 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: Oracle 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

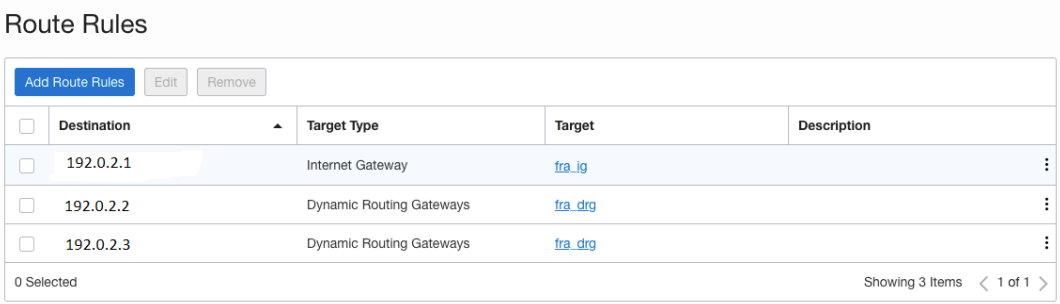
> **Note:**
>
> Repeat the steps in all regions used in the sharding deployment. The subnet CIDER must be different in each region and you must provide region prefix in the VCN/ subnet name.

Configuring Remote VCN Peering Between All Three Regions

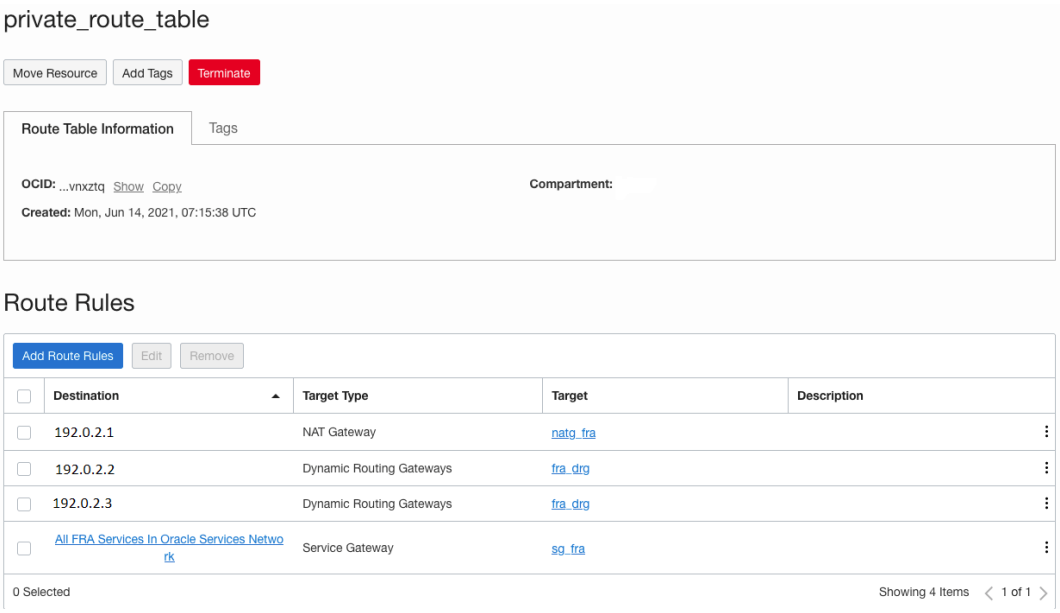
Remote VCN peering is the process of connecting two VCNs in different regions, which allows the VCNs' resources to communicate using private IP addresses without routing the traffic over the internet.

Configure two remote peering connections (RPCs) in each region to connect with the other two regions in the topology.

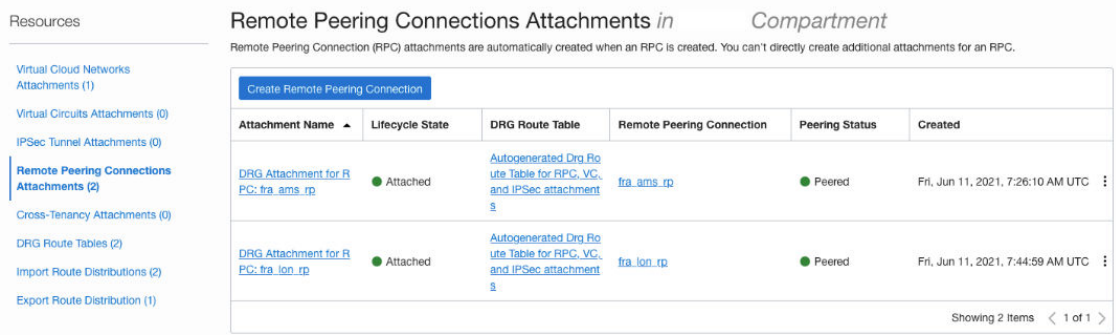
1. See Remote VCN Peering using an RPC for the steps to configure an RPC.
2. Configure routing rules for the public subnet/VCN.



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 in Oracle Sharding 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 Sharding. 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 Sharding.

```
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 Oracle Sharding 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 Oracle Sharding Global Database

Configure global service manager listener, create shard catalog database, and add all shards to configuration. The deployment step configures all shards as a single global database.

1. Configure the shard catalog in Oracle Sharding.

> **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. Execute 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

12

Troubleshooting Oracle Sharding

You can enable tracing, locate log and trace files, and troubleshooting common issues.

The following topics describe Oracle Sharding troubleshooting in detail:

Troubleshooting Tips

Use these tips to discover information about the sharded database that you need to help you troubleshoot issues.

Topics:

- Checking the Sharding Method
- Checking the Replication Type
- Checking the Oracle Data Guard Protection Mode
- Checking Which Shards Are Mapped to a Key
- Checking Shard Operation Mode (Read-Only or Read-Write)
- Checking DDL Text
- Checking Chunk Migration Status
- Checking Table Type (Sharded or Duplicated)
- Checking User Type (Local or ALL_SHARD)
- Identifying Tables Created as Sharded Tablespaces
- Checking if Shard DDL is Enabled or Disabled
- Filtering Data by Sharding Key

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 12-1 Replication types in config sdb output

Replication Type	Value Shown in Output
Oracle Data Guard	Data Guard
Oracle GoldenGate	Golden Gate

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
```

Setting the Duplicated Table Refresh Rate

You can modify the refresh rate for duplicated tables by setting the `SHRD_DUPL_TABLE_REFRESH_RATE` database parameter.

By default duplicated tables are refreshed every 60 seconds. The example below shows increasing the refresh interval to 100 seconds.

```
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
```

Oracle Sharding Tracing and Debug Information

The following topics explain how to enable tracing and find the logs.

Enabling Tracing for Oracle Sharding

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 Oracle Sharding 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 Oracle Sharding Alert Logs and Trace Files

There are several places to look for trace and alert logs in the Oracle Sharding 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
```

Master 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 execution, 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 for Sharded Databases

See the following topics for information about troubleshooting common errors in Oracle Sharding.

Issues Starting Remote Scheduler Agent

If you encounter issues starting Remote Scheduler Agent on all the shard hosts, try the following:

To start Scheduler you must be inside ORACLE_HOME on each shard server.

```
[oracle@shard2 ~]$ echo welcome | schagent -registerdatabase 192.0.2.24 8080
Agent Registration Password?
Failed to get agent Registration Info from db: No route to host

Solution: Disable firewall

service ipchains stop
service iptables stop
chkconfig ipchains off
chkconfig iptables off
```

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)))';
   ```

Errors From Shards Created with CREATE SHARD

For errors that occur during a DEPLOY from shards created with the GDSCTL CREATE SHARD command check the following:

- Remote Scheduler Agent logs on shard hosts
- DBA_SCHEDULER_JOB_RUN_DETAILS view on shard catalog
- NETCA/DBCA output files in $ORACLE_BASE/cfgtoollogs on shard hosts

Issues Using Create Shard

The following are solutions to some issues that occur when using the GDSCTL CREATE SHARD command..

Make sure to create $ORACLE_BASE/oradata and $ORACLE_BASE/fast_recovery_area directories to avoid the following errors

```
GDSCTL> create shard -shardgroup primary_shardgroup -destination che -
osaccount
 oracle -ospassword oracle
GSM-45029: SQL error
ORA-03710: directory does not exist or is not writeable at destination:
 $ORACLE_BASE/oradata
ORA-06512: at "GSMADMIN_INTERNAL.DBMS_GSM_POOLADMIN", line 6920
ORA-06512: at "SYS.DBMS_SYS_ERROR", line 86
ORA-06512: at "GSMADMIN_INTERNAL.DBMS_GSM_POOLADMIN", line 4730
ORA-06512: at line 1
```

```
GDSCTL>create shard -shardgroup primary_shardgroup -destination che -
osaccount oracle
 -ospassword oracle
GSM-45029: SQL error
ORA-03710: directory does not exist or is not writeable at destination:
 $ORACLE_BASE/fast_recovery_area
ORA-06512: at "GSMADMIN_INTERNAL.DBMS_GSM_POOLADMIN", line 6920
ORA-06512: at "SYS.DBMS_SYS_ERROR", line 86
ORA-06512: at "GSMADMIN_INTERNAL.DBMS_GSM_POOLADMIN", line 4755
ORA-06512: at line 1
```

Solution: Create oradata,fast_recovery_area under $ORACLE_BASE on all the shard hosts.

Privilege issues

```
GDSCTL>create shard -shardgroup primary_shardgroup -destination blr -
credential cred
GSM-45029: SQL error
ORA-02610: Remote job failed with error:
EXTERNAL_LOG_ID="job_79126_3",
USERNAME="oracle",
STANDARD_ERROR="Launching external job failed: Login executable not setuid-
root"
ORA-06512: at "GSMADMIN_INTERNAL.DBMS_GSM_POOLADMIN", line 6920
ORA-06512: at "SYS.DBMS_SYS_ERROR", line 86
ORA-06512: at "GSMADMIN_INTERNAL.DBMS_GSM_POOLADMIN", line 4596
ORA-06512: at line 1
```

Solution: Make sure to have root privilege on following directories,

```
chown root $ORACLE_HOME/bin/extjob
chmod 4750 $ORACLE_HOME/bin/extjob
chown root $ORACLE_HOME/rdbms/admin/externaljob.ora
chmod 640 $ORACLE_HOME/rdbms/admin/externaljob.ora
chown root $ORACLE_HOME/bin/jssu
chmod 4750 $ORACLE_HOME/bin/jssu
```

Error on create shard

```
GDSCTL>create shard -shardgroup primary_shardgroup -destination mysql02 -
osaccount
 oracle -ospassword oracle
GSM-45029: SQL error
ORA-03719: Shard character set does not match catalog character set.
ORA-06512: at "GSMADMIN_INTERNAL.DBMS_GSM_POOLADMIN", line 7469
ORA-06512: at "SYS.DBMS_SYS_ERROR", line 79
ORA-06512: at "GSMADMIN_INTERNAL.DBMS_GSM_POOLADMIN", line 5770
ORA-06512: at line 1
```

Solution: Check the JAVA version, it must be the same on the shard catalog and all shard servers.

```
rpm -qa|grep java
```

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
   ```

13

Oracle Sharding Reference

The following topics provide you with reference information to help you plan, configure, deploy, and manage your Oracle Sharding sharded database configuration.

Using GDSCTL with Oracle Sharding

Several of the Global Data Services GDSCTL commands are used for setting up and deploying an Oracle Sharding configuration. Learn how to use the GDSCTL command-line tool and the Oracle Sharding-related GDSCTL commands in the following topics.

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 Oracle Sharding 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 Sharding

A subset of GDSCTL commands is used with Oracle Sharding.

- add cdb
- add credential
- add file
- add gsm
- add invitednode (add invitedsubnet)
- add region
- add service
- add shard
- add shardgroup
- add shardspace
- config
- 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 shard
- create shardcatalog
- delete catalog
- deploy
- disable service
- enable service
- modify catalog
- modify cdb
- modify credential
- modify file
- modify gsm
- modify region
- modify service
- modify shard
- modify shardgroup
- modify shardspace
- move chunk
- relocate service
- remove cdb
- remove credential
- remove file
- remove gsm
- remove invitednode (remove invitedsubnet)
- remove region
- remove service
- remove shard
- remove shardgroup
- remove shardspace
- services
- set gsm
- set inbound_connect_level
- set log_level
- set outbound_connect_level
- set trace_level
- split chunk
- sql

- start gsm
- start service
- status gsm
- status service
- stop gsm
- stop service
- validate catalog