

Oracle® Database

Using Oracle Sharding



18c
E87087-03
September 2021

The Oracle logo, consisting of a solid red square with the word "ORACLE" in white, uppercase, sans-serif font centered within it.

ORACLE®

Oracle Database Using Oracle Sharding, 18c

E87087-03

Copyright © 2018, 2021, Oracle and/or its affiliates.

Primary Author: Virginia Beecher

Contributors: Raihan Al-Ekram, Lance Ashdown, Nagesh Battula, David Colello, Mark Dilman, Vidhya Govindaraju, Belinda Leung, Darshan Maniyani, Joseph Meeks, Janet Stern, Nirav Vyas, Nick Wagner, Jean Zeng

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

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

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

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

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

Contents

1 Overview of Oracle Sharding

About Sharding	1-1
Benefits of Sharding	1-2
Components of the Oracle Sharding Architecture	1-3
Application Suitability for Sharding	1-6

2 Sharded Database Schema Design

Sharded Tables	2-1
Sharded Table Family	2-3
Duplicated Tables	2-7
Non-Table Objects Created on All Shards	2-9
DDL Execution in a Sharded Database	2-10
DDL Syntax Extensions for the Sharded Database	2-16

3 Physical Organization of a Sharded Database

Sharding as Distributed Partitioning	3-1
Partitions, Tablespaces, and Chunks	3-2

4 Sharding Methods

System-Managed Sharding	4-1
User-Defined Sharding	4-4
Composite Sharding	4-6
Using Subpartitions with Sharding	4-8

5 Design Considerations for Sharded Database Applications

Considerations for Sharded Database Schema Design	5-1
Developing Applications for Oracle Sharding	5-1

6 Request Routing in a Sharded Database Environment

Direct Routing to a Shard	6-1
About Direct Routing to a Shard	6-1
Sharding APIs	6-2
Queries and DMLs with Proxy Routing in a Sharded Database	6-7
About Proxy Routing in a Sharded Database	6-7
Oracle Sharding Coordinator	6-8
Resiliency of Proxy Routing	6-9
Querying and DMLs Using Proxy Routing	6-9
Proxy Routing for Single-Shard Queries	6-10
Proxy Routing for Multi-Shard Queries	6-11
Specifying Consistency Levels in a Multi-Shard Query	6-11
Supported Query Shapes in Proxy Routing	6-12
Execution Plans for Proxy Routing	6-15
Creating Affinity Between Middle-Tier Connection Pools and Shards	6-18

7 Shard-Level High Availability

About Sharding and Replication	7-1
When To Choose Oracle GoldenGate for Shard High Availability	7-1
Using Oracle Data Guard with a Sharded Database	7-2
Using Oracle GoldenGate with a Sharded Database	7-7

8 Sharded Database Deployment

Introduction to Sharded Database Deployment	8-1
Choosing a Deployment Method	8-2
Using Oracle Multitenant with Oracle Sharding	8-3
Oracle Sharding Prerequisites	8-5
Installing Oracle Database Software	8-6
Installing the Shard Director Software	8-7
Creating the Shard Catalog Database	8-7
Setting Up the Oracle Sharding Management and Routing Tier	8-9
Creating and Deploying a System-Managed Sharded Database	8-13
Deploying a System-Managed Sharded Database	8-13
Creating a Schema for a System-Managed Sharded Database	8-21
System-Managed SDB Demo Application	8-29
Creating and Deploying a User-Defined SDB	8-29
Deploying a User-Defined SDB	8-29
Creating a Schema for a User-Defined SDB	8-35
Creating and Deploying a Composite SDB	8-42

Deploying a Composite SDB	8-43
Creating a Schema for a Composite SDB	8-49
Using Transparent Data Encryption with Oracle Sharding	8-55
Creating a Single Encryption Key on All Shards	8-56

9 Migrating Data to a Sharded Database

About Migrating Data to a Sharded Database	9-1
General Guidelines for Loading Data into a Sharded Database	9-2
Migrating the Schema	9-4
Preparing the Source Database	9-7
Preparing the Target Sharded Database	9-11
Migrating Your Data	9-15
Migrating Your Application	9-22

10 Sharded Database Lifecycle Management

Managing the Sharding-Enabled Stack	10-1
Starting Up the Sharding-Enabled Stack	10-2
Shutting Down the Sharding-Enabled Stack	10-2
Monitoring a Sharded Database	10-2
Monitoring a Sharded Database with GDSCTL	10-2
Monitoring a Sharded Database with Enterprise Manager Cloud Control	10-3
Discovering Sharded Database Components	10-7
Querying System Objects Across Shards	10-8
Backing Up and Recovering a Sharded Database	10-9
Modifying a Sharded Database Schema	10-10
Managing Sharded Database Software Versions	10-11
Patching and Upgrading a Sharded Database	10-11
Upgrading Sharded Database Components	10-12
Downgrading a Sharded Database	10-13
Shard Management	10-13
About Adding Shards	10-13
Resharding and Hot Spot Elimination	10-14
Removing a Shard From the Pool	10-16
Adding Standby Shards	10-16
Managing Shards with Oracle Enterprise Manager Cloud Control	10-17
Validating a Shard	10-17
Adding Primary Shards	10-18
Adding Standby Shards	10-19
Deploying Shards	10-19

Managing Shards with GDSCTL	10-20
Validating a Shard	10-20
Adding Shards to a System-Managed SDB	10-21
Replacing a Shard	10-25
Chunk Management	10-28
About Moving Chunks	10-29
Moving Chunks	10-30
About Splitting Chunks	10-30
Splitting Chunks	10-31
Shard Director Management	10-31
Creating a Shard Director	10-32
Editing a Shard Director Configuration	10-32
Removing a Shard Director	10-33
Region Management	10-33
Creating a Region	10-33
Editing a Region Configuration	10-34
Removing a Region	10-34
Shardspace Management	10-34
Creating a Shardspace	10-35
Adding a Shardspace to a Composite Sharded Database	10-35
Shardgroup Management	10-37
Creating a Shardgroup	10-37
Services Management	10-38
Creating a Service	10-38

11 Troubleshooting Oracle Sharding

Oracle Sharding Tracing and Debug Information	11-1
Enabling Tracing for Oracle Sharding	11-1
Where to Find Oracle Sharding Alert Logs and Trace Files	11-2
Common Error Patterns and Resolutions for Sharded Databases	11-3
Issues Starting Remote Scheduler Agent	11-3
Shard Director Fails to Start	11-4
Errors From Shards Created with CREATE SHARD	11-4
Issues Using Create Shard	11-5
Issues Using Deploy Command	11-6

Index

Preface

This book provides information about using Oracle Sharding to create and maintain sharded databases. It also provides information about designing applications for a sharded database.

- [Audience](#)
- [Documentation Accessibility](#)
- [Related Documents](#)
- [Conventions](#)

Audience

This book is intended for database administrators and application developers who work with sharded databases.

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.
<i>italic</i>	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 18c](#)

Changes in Oracle Database 18c

The following are changes in *Using Oracle Sharding* for Oracle Database 18c.

- [New Features](#)
- [Other Changes](#)

New Features

The following features are new in this release:

- [User-Defined Sharding Method](#)
- [Support for PDBs as Shards](#)
- [Support for Oracle GoldenGate Replication](#)
- [Centralized Diagnostics](#)
- [Multi-Shard Query Consistency Level](#)
- [Sharding Support for JSON, LOBs and Spatial Objects](#)
- [Optimizer Enhancements for Multi-Shard Queries](#)
- [Shard Replacement](#)
- [Oracle RAC Sharding](#)
- [UCP Support for Data-Dependent Routing to Oracle Sharding Middle Tiers](#)

User-Defined Sharding Method

User-defined sharding allows you to explicitly specify 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 you must have full control moving data between shards. This method allows you to define LIST or RANGE based sharding.

See

- [User-Defined Sharding](#) for a conceptual overview of user-defined sharding
- [Using Oracle Data Guard with a Sharded Database](#) for information about replicating a user-defined sharded database Oracle Data Guard
- [Creating and Deploying a User-Defined SDB](#) for tasks related to configuring, creating and deploying a user-defined sharded database

Support for PDBs as Shards

Use a PDB in a CDB for shards or a shard catalog database. In this release Oracle Sharding supports a shard or shard catalog as a single PDB in a CDB. The GDSCTL command ADD

SHARD is extended and new commands ADD CDB, MODIFY CDB, CONFIG CDB, and REMOVE CDB are implemented so that Oracle Sharding can support a multitenant architecture.

See

- [Using Oracle Multitenant with Oracle Sharding](#) for information about how to use PDBs as shards
- *Oracle Database Global Data Services Concepts and Administration Guide* for information about the new commands

Support for Oracle GoldenGate Replication

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. The Automatic Conflict Detection and Resolution feature added in Oracle 12.2 is used to handle any conflicts due to the active-active replication

See [Shard-Level High Availability](#)

Centralized Diagnostics

The SQL `SHARDS()` clause lets you query Oracle supplied objects, such as V\$, DBA/USER/ALL views, dictionary tables, and so on, across all shards.

See [Querying System Objects Across Shards](#)

Multi-Shard Query Consistency Level

You can use the initialization parameter `MULTISHARD_QUERY_DATA_CONSISTENCY` to set different consistency levels when executing multi-shard queries.

See [Specifying Consistency Levels in a Multi-Shard Query](#)

Sharding Support for JSON, LOBs and Spatial Objects

This release enables JSON operators that generate temporary LOBs, large JSON documents (those that require LOB Storage), Spatial Objects, Index and Operators and Persistent LOBs to be used in a sharded environment.

The following interfaces are new or changed as part of this feature.

- Query and DML statements
 - Cross shard queries involving LOBs are supported.
 - DMLs involving more than one shard are not supported. This behavior is similar to scalar columns.
 - DMLs involving a single shard are supported from coordinator.
 - Locator selected from a shard can be passed as bind value to the same shard.
- OCILob
 - All non-BFILE related OCILob APIs in a sharding environment are supported. with some restrictions.

On the coordinator, the OCI_ATTR_LOB_REMOTE attribute of a LOB descriptor returns TRUE if the LOB was obtained from a sharded table.

Restrictions: For APIs that take two locators as input, OCILobAppend, OCILobCompare for example, both of the locators should be obtained from the same shard. If locators are from different shards an error is given.

- DBMS_LOB

All non-BFILE related DBMS_LOB APIs in a sharding environment are supported, with some restrictions. On the coordinator, DBMS_LOB.isremote returns TRUE if the LOB was obtained from a sharded table.

Restrictions: For APIs that take two locators as input, DBMS_LOB.append and DBMS_LOB.compare for example, both of the locators should be obtained from the same shard. If the locators are from different shards an error is given.

See [Creating a Schema for a System-Managed Sharded Database](#), [Creating a Schema for a User-Defined SDB](#), and [Creating a Schema for a Composite SDB](#) for examples of using LOBs in sharded database deployment.

Optimizer Enhancements for Multi-Shard Queries

Various enhancements were made to improve the robustness and fault tolerance of shard queries. The query explain plan is enhanced to display information for all shards participating in the query.

See [Supported Query Shapes in Proxy Routing](#) and [Execution Plans for Proxy Routing](#) for updated information about these topics.

Shard Replacement

If a shard fails and is unrecoverable, you can replace it using the `ADD SHARD -REPLACE` command in GDSCTL. You can also use the `-replace` command option to move a shard to new equipment for any reason.

See [Replacing a Shard](#)

Oracle RAC Sharding

Oracle RAC Sharding creates an affinity for table partitions to particular Oracle RAC instances, and routes database requests that specify a partitioning key to the instance that logically holds the corresponding partition. This provides better cache utilization and dramatically reduces block pings across instances. The partitioning key can only be added to the most performance critical requests. Requests that don't specify the key still work transparently and can be routed to any instance. No changes to the database schema are required to enable this feature.

See *Oracle Real Application Clusters Administration and Deployment Guide*

UCP Support for Data-Dependent Routing to Oracle Sharding Middle Tiers

A Oracle Universal Connection Pool (UCP) feature called middle-tier routing allows smart routers (F5 BigIP, for example) to route to the middle tier associated with a sharding key.

See [Creating Affinity Between Middle-Tier Connection Pools and Shards](#)

Other Changes

The following are additional changes in the release:

- Sharding Content Moved to New Book

In Oracle Database 12c Release 2 (12.2.0.2) the Oracle Sharding content was part of the *Oracle Database Administrator's Guide*. Starting in Oracle Database 18c the Oracle Sharding content is contained in its own book, *Using Oracle Sharding*.

1

Overview of Oracle Sharding

Become familiar with the concepts related to managing a sharded database with Oracle Sharding.

Oracle Sharding terminology, concepts, and benefits are described in the following topics:

- [About Sharding](#)
Sharding is a data tier architecture in which data is horizontally partitioned across independent databases.
- [Benefits of Sharding](#)
Sharding provides linear scalability and complete fault isolation for the most demanding applications.
- [Components of the Oracle Sharding Architecture](#)
Oracle Sharding is a scalability and availability feature for suitable applications. It enables distribution and replication of data across a pool of Oracle databases that share no hardware or software.
- [Application Suitability for Sharding](#)
Many customer-facing web applications, such as e-commerce, mobile, and social media are well suited to sharding. Such applications have a well defined data model and data distribution strategy (hash, range, list, or composite) and primarily access data using a sharding key.

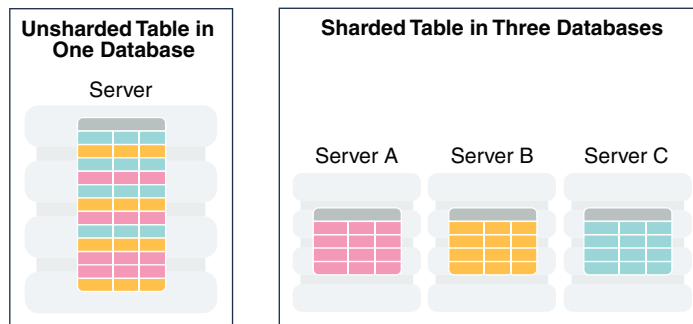
About Sharding

Sharding is a data tier architecture in which data is horizontally partitioned across independent databases.

Each database is hosted on dedicated server with its own local resources - CPU, memory, flash, or disk. Each database in such configuration is called a **shard**. All of the shards together make up a single logical database, which is referred to as a **sharded database (SDB)**.

Horizontal partitioning involves splitting a database table across 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 also known as a *sharded table*.

The following figure shows a table horizontally partitioned across three shards.

Figure 1-1 Horizontal Partitioning of a Table Across Shards

Sharding is based on shared-nothing hardware infrastructure and it eliminates single points of failure because shards do not share physical resources such as CPU, memory, or storage devices. Shards are also loosely coupled in terms of software; they do not run clusterware.

Shards are typically hosted on dedicated servers. These servers can be commodity hardware or engineered systems. The shards can run on single instance or Oracle RAC databases. They can be placed on-premises, in a cloud, or in a hybrid on-premises and cloud configuration.

From the perspective of a database administrator, an SDB consists of multiple databases that can be managed either collectively or individually. However, from the perspective of the application, an SDB looks like a single database: the number of shards and distribution of data across those shards are completely transparent to database applications.

Sharding is intended for custom applications that are suitable for a sharded database architecture. Applications that use sharding must have a well-defined data model and data distribution strategy (consistent hash, range, list, or composite) that primarily accesses data using a sharding key. Examples of a sharding key include `customer_id`, `account_no`, or `country_id`.

Benefits of Sharding

Sharding provides linear scalability and complete fault isolation for the most demanding applications.

Key benefits of sharding include:

- **Linear Scalability.** Sharding eliminates performance bottlenecks and makes it possible to linearly scale performance and capacity by adding shards.
- **Fault Containment.** Sharding is a shared nothing hardware infrastructure that eliminates single points of failure, such as shared disk, SAN, and clusterware, and provides strong fault isolation—the failure or slow-down of one shard does not affect the performance or availability of other shards.
- **Geographical Distribution of Data.** Sharding makes it possible to store particular data close to its consumers and satisfy regulatory requirements when data must be located in a particular jurisdiction.

- **Rolling Upgrades.** Applying configuration changes on one shard at a time does not affect other shards, and allows administrators to first test the changes on a small subset of data.
- **Simplicity of Cloud Deployment.** Sharding is well suited to deployment in the cloud. Shards may be sized as required to accommodate whatever cloud infrastructure is available and still achieve required service levels. Oracle Sharding supports on-premises, cloud, and hybrid deployment models.

Unlike NoSQL data stores that implement sharding, Oracle Sharding provides the benefits of sharding without sacrificing the capabilities of an enterprise RDBMS. For example, Oracle Sharding supports:

- Relational schemas
- Database partitioning
- ACID properties and read consistency
- SQL and other programmatic interfaces
- Complex data types
- Online schema changes
- Multi-core scalability
- Advanced security
- Compression
- High Availability features
- Enterprise-scale backup and recovery

Components of the Oracle Sharding Architecture

Oracle Sharding is a scalability and availability feature for suitable applications. It enables distribution and replication of data across a pool of Oracle databases that share no hardware or software.

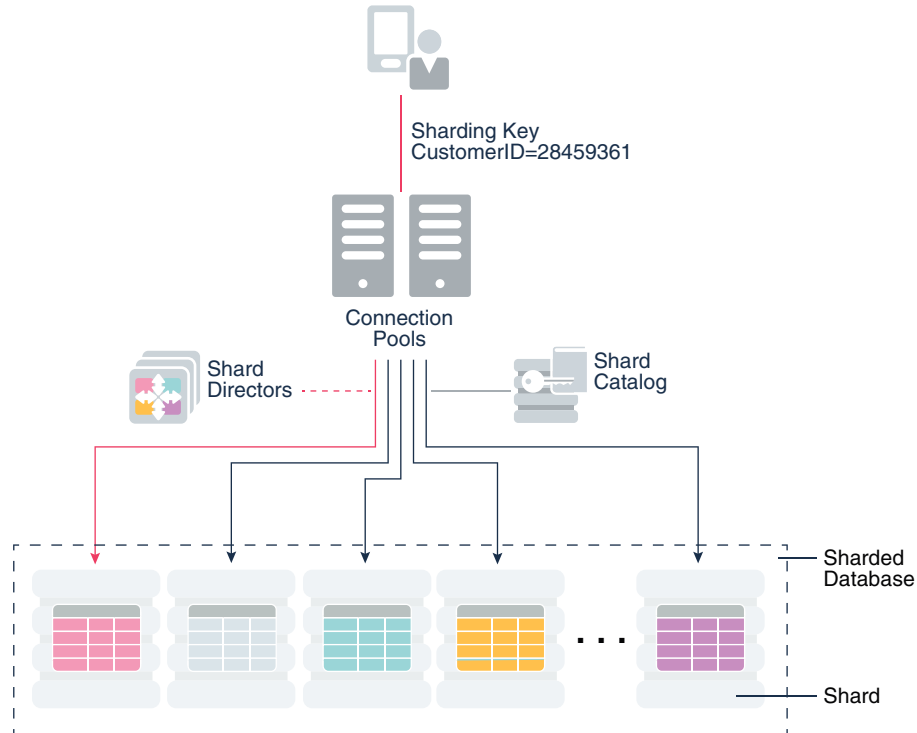
Applications perceive the pool of databases as a single logical database. Applications can elastically scale data, transactions, and users to any level, on any platform, by adding databases (shards) to the pool. Oracle Database supports scaling up to 1000 shards.

The following figure illustrates the major architectural components of Oracle Sharding:

- Sharded database (SDB) – a single logical Oracle Database that is horizontally partitioned across a pool of physical Oracle Databases (shards) that share no hardware or software
- Shards - independent physical Oracle databases that host a subset of the sharded database
- Global service - database services that provide access to data in an SDB
- Shard catalog – an Oracle Database that supports automated shard deployment, centralized management of a sharded database, and multi-shard queries
- Shard directors – network listeners that enable high performance connection routing based on a sharding key
- Connection pools - at runtime, act as shard directors by routing database requests across pooled connections

- Management interfaces - GDSCTL (command-line utility) and Oracle Enterprise Manager (GUI)

Figure 1-2 Oracle Sharding Architecture



Sharded Database and Shards

Shards are independent Oracle databases that are hosted on database servers which have their own local resources: CPU, memory, and disk. No shared storage is required across the shards.

A sharded database is a collection of shards. 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 (HA) and Disaster Recovery (DR) with Oracle replication technologies such as Data Guard. For HA, the standby shards can be placed in the same region where the primary shards are placed. For DR, the standby shards are located in another region.

Global Service

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 an SDB. For highly available shards using Active Data Guard, a separate read-only global service can be created.

Shard Catalog

The shard catalog is a special-purpose Oracle Database that is a persistent store for SDB 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 an SDB are executed by connecting to the shard catalog.

The shard catalog also contains the master copy of all duplicated tables in an SDB. 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.

Using Oracle Data Guard for shard catalog high availability is a recommended best practice. The availability of the shard catalog has no impact on the availability of the SDB. An outage of the shard catalog only affects the ability to perform maintenance operations or multi-shard queries during the brief period required to complete an automatic failover to a standby shard catalog. Transactions continue to be routed and executed by the SDB and are unaffected by a catalog outage.

Shard Director

Oracle Database 12c 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 an SDB. The director maintains a current topology map of the SDB. Based on the sharding key passed during a connection request, the director routes the connections to the appropriate shard.

For a typical SDB, a set of shard directors are installed on dedicated low-end commodity servers in each region. To achieve high availability, deploy multiple shard directors. In Oracle Database 12c Release 2, 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 SDB configuration and availability of shards
- Measure network latency between its own and other regions
- Act as a regional listener for clients to connect to an SDB
- Manage global services
- Perform connection load balancing

Connection Pools

Oracle Database supports connection-pooling in data access drivers such as OCI, JDBC, and ODP.NET. In Oracle 12c Release 2, 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 SDB.

Management Interfaces for an SDB

You can deploy, manage, and monitor Oracle Sharded databases with two interfaces: Oracle Enterprise Manager Cloud Control and GDSCTL.

Cloud Control enables life cycle management of a sharded database with a graphical user interface. You can manage and monitor an SDB for availability and performance, and you can do tasks such as add and deploy shards, services, shard directors, and other sharding components.

GDSCTL is a command-line interface that provides a simple declarative way of specifying the configuration of an SDB and automating its deployment. Only a few GDSCTL commands are required to create an SDB, for example:

- `CREATE SHARDCATALOG`
- `ADD GSM` and `START GSM` (create and start shard directors)
- `CREATE SHARD` (for each shard)
- `DEPLOY`

The GDSCTL `DEPLOY` command automatically creates the shards and their respective listeners. In addition, this command automatically deploys the replication configuration used for shard-level high availability specified by the administrator.

See Also:

- *Oracle Database Global Data Services Concepts and Administration Guide* for information about global service managers, global services, and the GDSCTL commands used with Oracle Sharding
- Oracle Sharding best practices white papers in the Oracle Database section of the [Oracle MAA web page](#)

Application Suitability for Sharding

Many customer-facing web applications, such as e-commerce, mobile, and social media are well suited to sharding. Such applications have a well defined data model and data distribution strategy (hash, range, list, or composite) and primarily access data using a sharding key.

Examples of sharding keys include customer ID, account number, and country_id. Applications will also usually require partial de-normalization of data to perform well with sharding.

Existing applications that were never intended to be sharded will require some level of redesign to achieve the benefits of a sharded architecture. In some cases it may be as simple as providing the sharding key, in other cases it may be impossible to horizontally partition data and workload as required by a sharded database.

Transactions that access data associated with a single value of the sharding key are the primary use-case for a sharded database. For example, lookup and update of a customer's records, subscriber documents, financial transactions, e-commerce transactions, and the like. Because all of the rows that have the same value of the sharding key are guaranteed to be on the same shard, such transactions are always single-shard and executed with the highest performance and provide the highest level of consistency. Multi-shard operations are supported, but with a reduced level of performance and consistency. Such transactions include simple aggregations, reporting, and the like, and play a minor role in a sharded application relative to workloads dominated by single-shard transactions.



See Also:

[Design Considerations for Sharded Database Applications](#)

2

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.

The following topics describe the objects used for this purpose:

- [Sharded Tables](#)
A **sharded table** is a table that is partitioned into smaller and more manageable pieces among multiple databases, called shards.
- [Sharded Table Family](#)
A sharded table family is a set of tables that are sharded in the same way.
- [Duplicated Tables](#)
In addition to sharded tables, an SDB can contain tables that are duplicated on all shards.
- [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.
- [DDL Execution in a Sharded Database](#)
- [DDL Syntax Extensions for the Sharded Database](#)
Oracle Sharding introduces changes to the SQL DDL statements. DDL statements with this syntax can only be executed against a sharded database.

Sharded Tables

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

Oracle Sharding is implemented based on the Oracle Database partitioning feature. Oracle Sharding is essentially distributed partitioning because it extends partitioning by supporting the distribution of table partitions 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) 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.

Example 2-1 Sharded Table

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 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 preceding 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 that global indexes on sharded tables are not supported, but local indexes are supported.

Tablespace Sets

Oracle Sharding creates and manages tablespaces as a unit called a **tablespace set**. 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.

A tablespace is a logical unit of data distribution in an SDB. 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.

 **Note:**

Only Oracle Managed Files are supported by tablespace sets.

Individual chunk tablespaces cannot be dropped or altered independently of the entire tablespace set.

`TABLESPACE SET` cannot be used with the user-defined sharding method.

Chunks

The unit of data migration between shards is a **chunk**. A chunk is a set of tablespaces that store corresponding partitions of all tables in a table family. A chunk contains a single partition from each table of a set of related tables. This guarantees that related data from different sharded tables can be moved together. The number of chunks within each shard is specified when the SDB is created.

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.

 **Note:**

In Oracle Database 12c Release 2, only a single table family is supported in an SDB.

How a Table Family Is Sharded

To illustrate sharding of a table family, consider the example of the Customers–Orders–LineItems schema. The tables in this schema may look as shown in the examples below. The three tables have a parent-child relationship, with `customers` being the root table.

Customers table:

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:

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:

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:

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

Orders table:

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

LineItems table:

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

Creating a Sharded Table Family Using CREATE TABLE

The recommended way to create a sharded table family is to specify parent-child relationships between tables using reference partitioning.

The appropriate `CREATE TABLE` statements for `Customers–Orders–LineItems` schema 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 `Orders` and `LineItems` tables which are a child and grandchild of `Customers`.

```
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.

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.

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 either be the same as the sharding key, or contain the sharding key as the leading column. 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 another column(s) prefixed by 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. For example, to link the LineItems (child) table to the Orders (parent) table, you need a primary key in Orders. The second rule implies that the primary key in Orders is prefixed by the CustNo value. (This is an existing partitioning rule not specific to Oracle Sharding.)

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 and 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.

 **Note:**

In Oracle Database 12c Release 2, only a single table family is supported in an SDB.

 **See Also:**

Oracle Database VLDB and Partitioning Guide

Duplicated Tables

In addition to sharded tables, an SDB can contain tables that are duplicated on all shards.

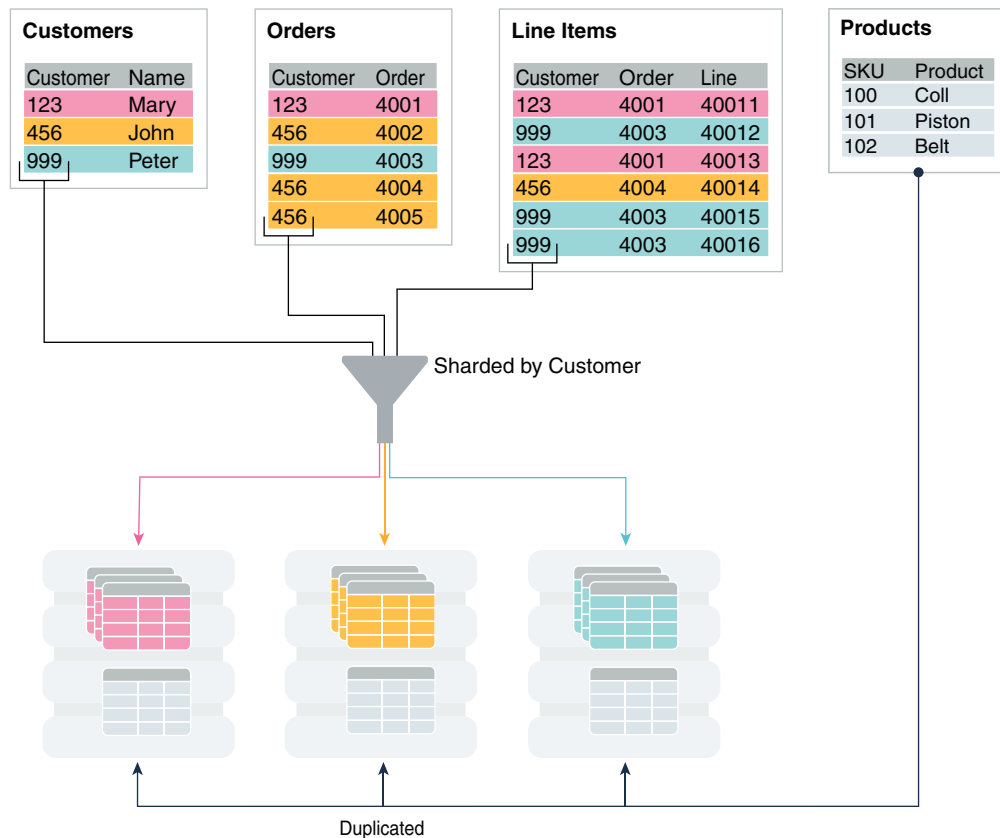
For many applications, the number of database requests handled by a single shard can be maximized by duplicating read-only or read-mostly tables across all shards. This strategy is a good choice for relatively small tables that are often accessed together with sharded tables. A table with the same contents in each shard is called a duplicated table.

An SDB 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 2-1 Sharded Tables and a Duplicated Table in an SDB



Creating a Duplicated Table Using CREATE TABLE

The duplicated Products 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 Table 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.

In Oracle Database 12c Release 2, a client must connect to the shard catalog database to update a duplicated table. In Oracle Database 18c and later, you can update a duplicated table on a shard. The update is first propagated over a dblink 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 are automatically refreshed at a configurable frequency. The refresh frequency of all duplicated tables is controlled by the database initialization parameter `SHRD_DUPL_TABLE_REFRESH_RATE`. The default value for the parameter is 60 seconds.

 **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:

- update of a LOB or a data type not supported by dblink
- update or delete of a row inserted by the same transaction

 **See Also:**

Oracle Database Administrator's Guide

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 18c, 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.

DDL Execution in a Sharded Database

To create a schema in an SDB, you must issue DDL commands on the shard catalog database, which validates the DDLs and executes them locally first. Therefore, 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 SDB schema. If the 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 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 an SDB, all of the DDLs that have been executed in the SDB are applied in the same order to the shard before it becomes accessible to clients.

There are two ways you can issue DDLs in an SDB:

- Use the `GDSCTL sql` command.

When you issue a DDL this way, `GDSCTL` waits until all of the shards have finished executing the DDL and returns the status of the execution. For example

```
GDSCTL> sql "create tablespace set tbsset"
```

- Connect to the shard catalog database using `SQL*Plus` using the `GDS$CATALOG.sdbname` service. For example

```
SQL> create tablespace set tbsset;
```

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.

Verifying DDL Propagation

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. 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 executed. Note that the user must have `GSM_ADMIN` privileges to execute these `GDSCTL` commands.

Creating Objects Locally and Globally

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 catalog database, is called an *SDB object*.

When connecting to the shard catalog using SQL*Plus, two types of objects can be created: SDB 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 catalog database, to generate and store a report, for example.

The type of object (SDB 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 SDB user – 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 SDB objects. To create a local object, the SDB user must first run `alter session disable shard ddl`. All of the objects created while `SHARD DDL` is disabled are local objects. To enable `SHARD DDL` in the session, the SDB user must run `alter session enable shard ddl`.

See [ALTER SESSION](#) for more information about the `SHARD DDL` session parameter.

Creating the SDB User to Create Schema Objects

Local users that only exist in the shard catalog database do not have the privileges to create schema objects in the SDB. Therefore, the first step of creating the SDB schema is to create the SDB user, by connecting to the shard catalog database as `SYSDBA`, enabling `SHARD DDL`, and executing the `CREATE USER` command. When the SDB user connects to the catalog database, the `SHARD DDL` mode is enabled by default.



Note:

Local users can create non-schema SDB objects, such as tablespaces, directories, and contexts, if they enable `SHARD DDL` mode; however, they cannot create schema SDB 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.

DDL Execution Examples

The following examples demonstrate the steps to issue a DDL, check its execution status, and what to do when errors are encountered. The examples are given for 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 2-2 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 IDENTIFIED BY out_standing1;
CREATE USER example_user IDENTIFIED 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
--      -
1
```

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 2-3 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 tablespace set failure, one of its tablespaces
not created
ORA-01119: error in creating database file \'/ade/b/3667445372/oracle/
rdbms/dbs/
SHARD01/datafile/ol_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 2-4 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 and 5 are applied.

DDL Syntax Extensions for the Sharded Database

Oracle Sharding introduces changes to the SQL DDL statements. DDL statements with this syntax can only be executed against a sharded database.

Note that no changes to query and DML statements are required to support Oracle Sharding, and the changes to the DDL statement 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 regular Oracle Database.

CREATE TABLESPACE SET

This is a new statement introduced 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
    }...
    )];
```

The 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.

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 and have the same properties. The properties are specified in the `USING TEMPLATE` clause. This clause is the same as `permanent_tablespace_clause` for a typical tablespace, with the exception that a datafile name cannot be specified in the `datafile_tempfile_spec` clause. The datafile 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.

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

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

The syntax and semantics for these statements are similar to `DROP` and `PURGE TABLESPACE` statements.

CREATE TABLE

This statement has been extended to allow for creation of sharded and duplicated tables and specification of 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.
- 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
}
```

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

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)
;

```

ALTER TABLE

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
- Truncate table
- Drop duplicated table materialized view log
- Drop duplicated table materialized views on shards
- Alter materialized views (of duplicated tables) on shards

ALTER SESSION

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 an SDB user (the user that exists on all shards and the catalog). To create an SDB user, the `SHARD DDL` parameter must be enabled before running `CREATE USER`.

3

Physical Organization of a Sharded Database

Learn about the physical organization of a sharded database.

The following topics describe the physical organization of a sharded database:

- [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.
- [Partitions, Tablespaces, and Chunks](#)
Distribution of partitions across shards is achieved by creating partitions in tablespaces that reside on different shards.

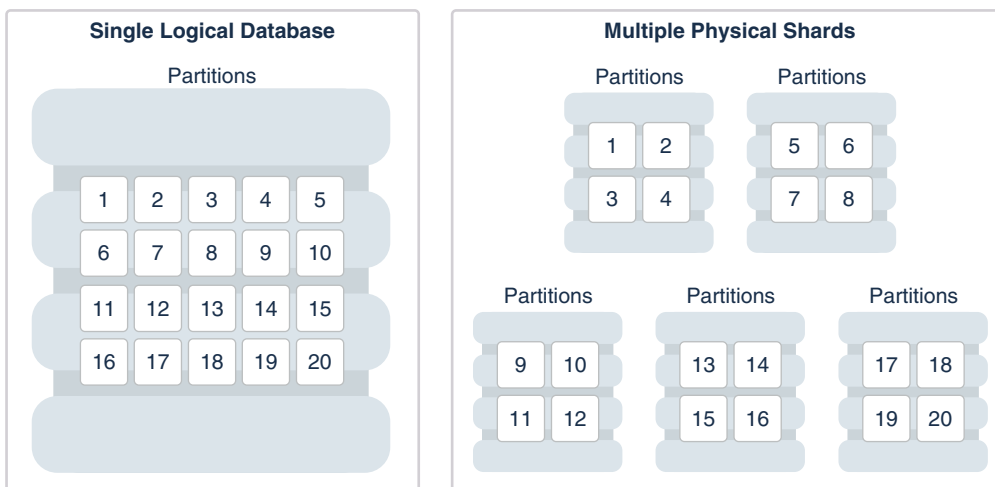
Sharding as Distributed Partitioning

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.

Even though a sharded database (SDB) looks like a single database to applications and application developers, from the perspective of a database administrator, it is a set of discrete Oracle databases, each of which is called a *shard*. A sharded table is partitioned across all shards of the SDB. 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 3-1 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.

Partitions, Tablespaces, and Chunks

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

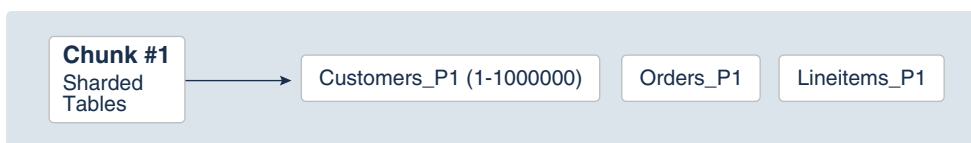
Each partition of a sharded table is stored in a separate tablespace, making the tablespace the unit of data distribution in an SDB.

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.

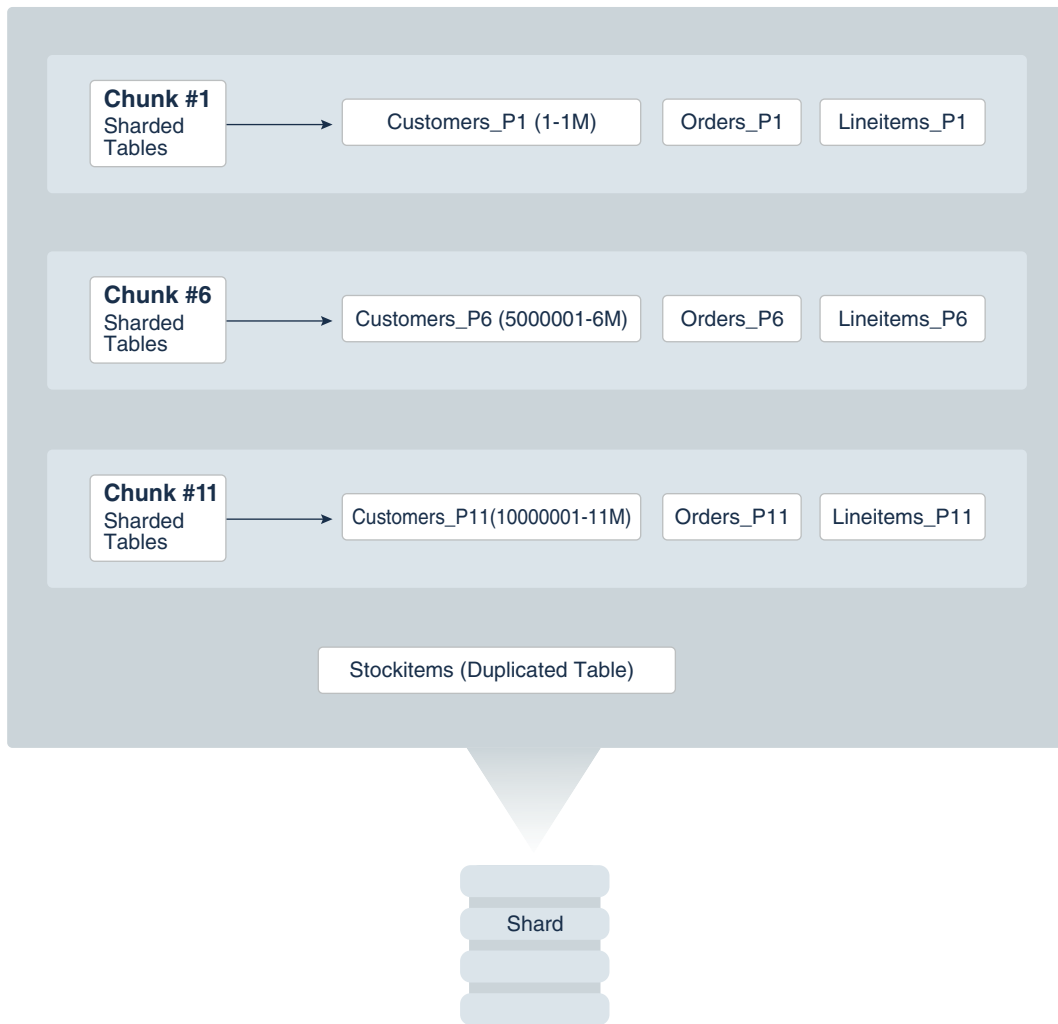
A chunk that contains corresponding partitions from the tables of Customers-Orders-LineItems schema is shown in the following figure.

Figure 3-2 Chunk as a Set of Partitions



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

Figure 3-3 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.

4

Sharding Methods

This chapter discusses the sharding methods supported by Oracle Sharding, how to choose a method, and how to use subpartitioning.

The following topics describe the sharding methods supported by Oracle Sharding:

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

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.

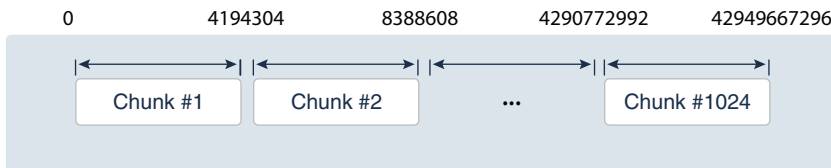
Consistent hash is a partitioning strategy commonly used in scalable distributed systems. It is different from traditional hash partitioning. With traditional hashing, the bucket number is calculated as $HF(key) \% N$ where HF is a hash function and N is the number of buckets. This approach works fine 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 4-1 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 `CREATE SHARDCATALOG` command. 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
;
```

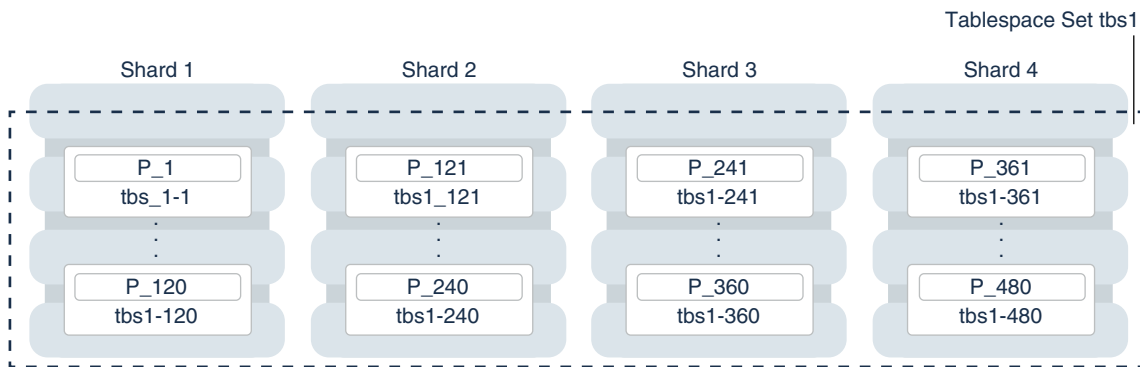
`PARTITIONS AUTO` in this statement means that the number of partitions is automatically set to the number of tablespaces in the tablespace set `ts1` (which is equal to the number of chunks) and each partition will be stored in a separate tablespace.

Each tablespace in a tablespace set belongs to a distinct chunk. In 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 4-2 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.

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 using the following GDSCTL commands:

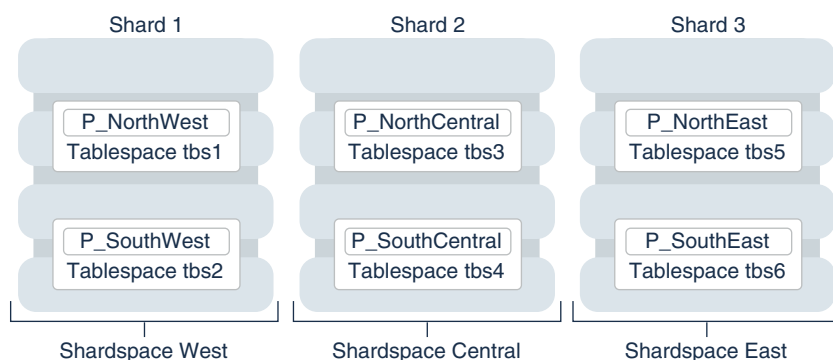
```

ADD SHARDSPACE -SHARDSPACE west, central, east;
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 4-3 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 `MOVE CHUNK` command for each chunk that needs to be migrated.

The `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.

For a user-defined sharded database, two replication schemes are supported: Oracle Data Guard or Oracle Active Data Guard.

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.

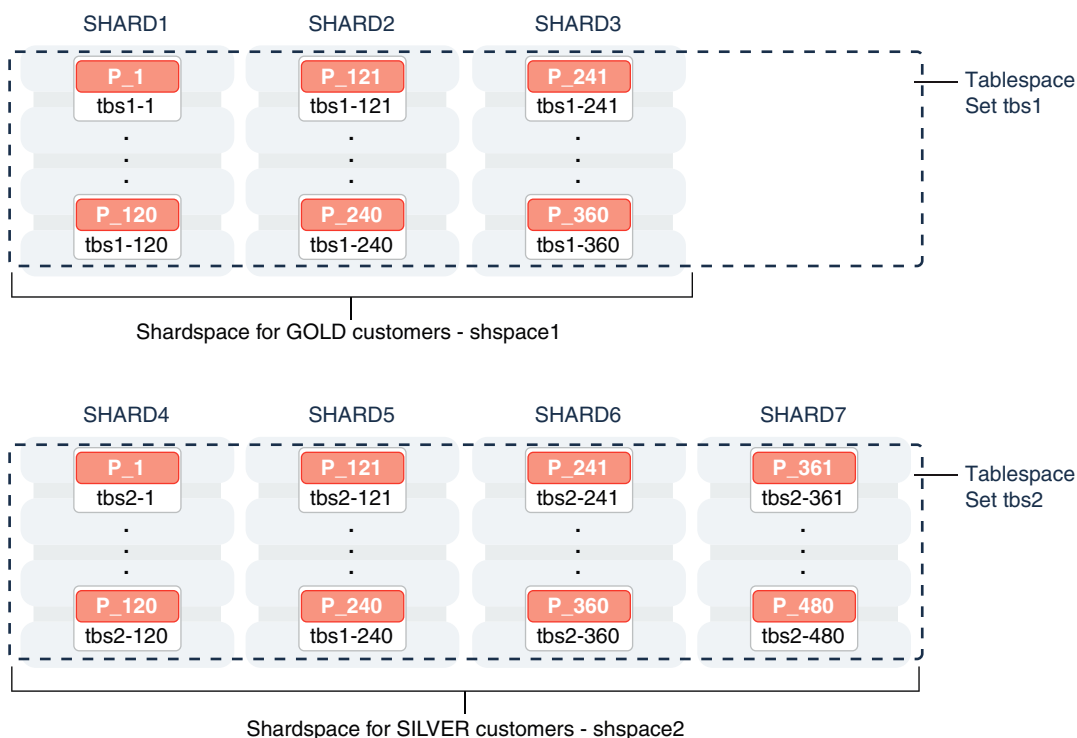
When sharding by consistent hash on a primary key, there is often a requirement to differentiate subsets of data within an SDB in order to store them in different geographic locations, allocate to them different hardware resources, or configure high availability and disaster recovery differently. Usually 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 4-4 Composite Sharding



Two shardspaces need to be created for such a configuration, using the following GDSCTL commands:

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

ADD SHARD -CONNECT shard1 -SHARDSPACE shspace1;
ADD SHARD -CONNECT shard2 -SHARDSPACE shspace1;
ADD SHARD -CONNECT shard3 -SHARDSPACE shspace1;

ADD SHARD -CONNECT shard4 -SHARDSPACE shspace2;
ADD SHARD -CONNECT shard5 -SHARDSPACE shspace2;
ADD SHARD -CONNECT shard6 -SHARDSPACE shspace2;
ADD SHARD -CONNECT shard7 -SHARDSPACE shspace2;
```

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

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

To store user-defined subsets of data in different tablespaces, Oracle 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 12c 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)
```

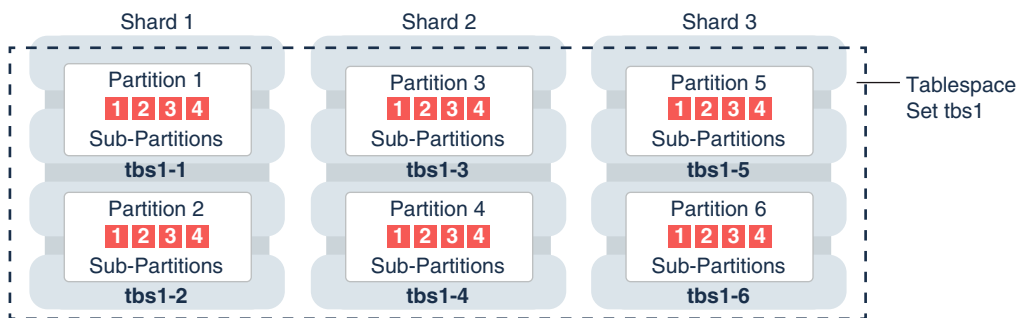
```

, 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 4-5 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)
)

```

```
)
;
```

In this example, subpartitions are stored in the tablespace of the parent partition, and the subpartition template is the same for each `PARTITIONSET`. To store subpartitions in separate tablespaces the following syntax can be used.

```
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(class, cust_id, signup_date)
)
PARTITIONSET BY LIST (class)
PARTITION BY CONSISTENT HASH (cust_id)
SUBPARTITION BY RANGE (signup_date)
PARTITIONS AUTO
(
  PARTITIONSET gold VALUES ('gld')
  SUBPARTITION TEMPLATE
  ( SUBPARTITION per1 VALUES LESS THAN (TO_DATE('01/01/2000','DD/MM/YYYY'))
    TABLESPACE SET tbs1
  , SUBPARTITION per2 VALUES LESS THAN (TO_DATE('01/01/2010','DD/MM/YYYY'))
    TABLESPACE SET tbs2
  , SUBPARTITION per3 VALUES LESS THAN (TO_DATE('01/01/2020','DD/MM/YYYY'))
    TABLESPACE SET tbs3
  , SUBPARTITION future VALUES LESS THAN (MAXVALUE))
    TABLESPACE SET tbs4
  )
, PARTITIONSET silver VALUES ('slv') TABLESPACE SET ts1
  SUBPARTITION TEMPLATE
  ( SUBPARTITION per1 VALUES LESS THAN (TO_DATE('01/01/2000','DD/MM/YYYY'))
    TABLESPACE SET tbs5
  , SUBPARTITION per2 VALUES LESS THAN (TO_DATE('01/01/2010','DD/MM/YYYY'))
    TABLESPACE SET tbs6
  , SUBPARTITION per3 VALUES LESS THAN (TO_DATE('01/01/2020','DD/MM/YYYY'))
    TABLESPACE SET tbs7
  , SUBPARTITION future VALUES LESS THAN (MAXVALUE))
    TABLESPACE SET tbs8
  )
)
;
```

5

Design Considerations for Sharded Database Applications

To obtain the benefits of sharding, a schema of an SDB should be designed in a way that maximizes the number of database requests executed on a single shard.

The following topics describe the terminology and concepts you will need to manage a sharded database schema:

- [Considerations for Sharded Database Schema Design](#)
Design of the database schema has a big impact on the performance and scalability of a sharded database (SDB). An improperly designed schema can lead to unbalanced distribution of data and workload across shards and large percentage of multi-shard operations.
- [Developing Applications for Oracle Sharding](#)
Sharding provides linear scalability and complete fault isolation for the most demanding applications without compromising on the enterprise qualities of Oracle Database: strict consistency, the full power of SQL, developer agility with JSON, security, high availability, backup and recovery, life-cycle management, and more.

Considerations for Sharded Database Schema Design

Design of the database schema has a big impact on the performance and scalability of a sharded database (SDB). An improperly designed schema can lead to unbalanced distribution of data and workload across shards and large percentage of multi-shard operations.

Once the SDB 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 an SDB:

- 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)?

Developing Applications for Oracle Sharding

Sharding provides linear scalability and complete fault isolation for the most demanding applications without compromising on the enterprise qualities of Oracle Database: strict consistency, the full power of SQL, developer agility with JSON, security, high availability, backup and recovery, life-cycle management, and more.

Sharding is a data tier architecture in which data is horizontally partitioned across independent databases. Each database in such a configuration is called a shard. All of the shards together make up a single logical database, which is referred to as a sharded database (SDB).

There are three methods of sharding available to developers.

- System-managed sharding 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 for linear scalability.
- Composite sharding allows the creation of multiple shardspaces for different subsets of data in a table partitioned by consistent hash. Composite sharding is unique to Oracle and offers developers the ability to differentiate subsets of data within an SDB in order to store them in different geographic locations, allocate to them different hardware resources, etc, and then elastically scale performance within each subset.
- 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.

Oracle Sharding is based on table partitioning; all of the subpartitioning methods provided by Oracle Database are also supported for sharding.

Sharding is intended for applications that are suitable for a sharded database architecture. Specifically:

- Applications must have a well-defined data model and data distribution strategy, system-managed (consistent hash) or composite, and must primarily access data using a sharding key. Examples of sharding keys include customer ID, account number, country_id, and so on.
- The data model should be a hierarchical tree structure with a single root table. Oracle Sharding supports any number of levels within the hierarchy.
- 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.
- 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. The sharding key must be the leading column of the primary key of the root table.
- Joins between tables in a table family should be performed using the sharding key.
- 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.
- 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.
- Applications use Oracle integrated connection pools (UCP, OCI, ODP.NET, JDBC) to connect to a sharded database.
- Separate connection pools must be used for direct routing and proxy routing. 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.



See Also:

- [Direct Routing to a Shard](#)
- [Queries and DMLs with Proxy Routing in a Sharded Database](#)

6

Request Routing in a Sharded Database Environment

Oracle Sharding supports direct, key-based, routing to a shard, routing by proxy, and routing to mid-tiers.

The following topics describe routing in an sharded database environment.

- [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.
- [Queries and DMLs with Proxy Routing in a Sharded Database](#)
Sharding supports routing for queries that do not specify a sharding key. This allows the flexibility for any database application to execute SQL statements (including `SELECT` and `DML`) in a system where tables are sharded or duplicated without the need to specify the shards where the query should be executed.
- [Creating Affinity Between Middle-Tier Connection Pools and Shards](#)
A Oracle Universal Connection Pool (UCP) feature called middle-tier routing allows smart routers (F5 BigIP, for example) to route to the middle tier associated with a sharding key. It publishes the sharded database topology to the router tier so that requests based on specific sharding keys are routed to the appropriate application middle tier, which in turn establishes connections on the given subset of shards.

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.

The following topics describe direct, key-based, routing to a shard:

- [About Direct Routing to a Shard](#)
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.
- [Sharding APIs](#)
Oracle connection pools and drivers support Oracle Sharding.

About Direct Routing to a Shard

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 workloads that perform intra-shard transactions. It is recommended that direct routing be employed for all workloads that require the highest performance and availability.

In support of Oracle Sharding, key enhancements have been made to Oracle connection pools and drivers. Starting in Oracle Database 12c Release 2, JDBC, Universal Connection Pool (UCP), OCI Session Pool (OCI), and Oracle Data Provider for .NET (ODP.NET) provide new 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.



See Also:

[Direct Routing to a Shard](#)

Sharding APIs

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.

Sharding APIs for Oracle UCP

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 pool parameters at the pool level and the shard level:
 - Initial number of connections to be created when UCP is started using `setInitialPoolSize`
 - Minimum number of connections maintained by pool at runtime using `setMinPoolSize`
 - UCP property to set maximum number of connections allowed on connection pool using `setMaxPoolSize`
 - Set max connections per shard with `setMaxConnectionsPerShard`
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 6-1 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 6-2 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.oradbccloud)
(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();
    }
}
}

```

ODP.NET Sharding APIs

A shard-aware application gets a connection to a given shard by specifying the sharding key and super sharding key using ODP.NET APIs, such as the

`SetShardingKey(OracleShardingKey shardingKey, OracleShardingKey superShardingKey)` instance method on 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 6-3 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
    }
}
```

 **See Also:**

Oracle Database JDBC Developer's Guide for information about JDBC support for Oracle Sharding

Oracle Universal Connection Pool Developer's Guide for information about UCP support for Oracle Sharding

Oracle Data Provider for .NET Developer's Guide for Microsoft Windows for information about ODP.NET support for Oracle Sharding

Oracle Call Interface Programmer's Guide for information about the OCI interface for using shards

Queries and DMLs with Proxy Routing in a Sharded Database

Sharding supports routing for queries that do not specify a sharding key. This allows the flexibility for any database application to execute SQL statements (including `SELECT` and `DML`) in a system where tables are sharded or duplicated without the need to specify the shards where the query should be executed.

The following topics describe proxy routing in detail:

- [About Proxy Routing in a Sharded Database](#)
Proxy routing is an ancillary usage pattern targeted for developer convenience. It requires a connection be established to the coordinator.
- [Oracle Sharding Coordinator](#)
The Oracle Sharding coordinator database contains the metadata of the sharded topology and provides query processing support for sharded databases.
- [Querying and DMLs Using Proxy Routing](#)
Proxy routing enables aggregation of data and reporting across shards. It also allows the flexibility for any database application to execute SQL statements (including `SELECT` and `DML`) in a system where tables are sharded or duplicated without the need to specify the sharding key (during connect) where the query should execute.
- [Proxy Routing 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.
- [Proxy Routing for Multi-Shard Queries](#)
A multi-shard query must scan data from more than one shard, and the processing on each shard is independent of any other shard.
- [Supported Query Shapes in Proxy Routing](#)
Oracle Sharding supports single and multi-shard query shapes with some restrictions.
- [Execution Plans for Proxy Routing](#)
In a multi-shard query, each shard produces an independent execution plan that is potentially different from the plans on the other shards in the query.

About Proxy Routing in a Sharded Database

Proxy routing is an ancillary usage pattern targeted for developer convenience. It requires a connection be established to the coordinator.

The shard catalog database assumes the role of the coordinator database. The coordinator uses the metadata of the sharded topology and provides query processing support for sharded databases. The SQL compiler identifies the relevant shards automatically and coordinates the query execution across all of the participating shards. Once the session is made with the coordinator, SQL queries and DMLs are executed and require no modification.

Proxy routing is suitable for the following scenarios:

- When the application cannot pass the sharding key during connect
- When the application needs to access data from sharded tables residing on multiple shards
- SQL queries typically used in reporting such as aggregates on sales data

Routing using the coordinator allows your application to submit SQL statements without a sharding key value passed during connect. The coordinator's 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. In essence, the shards act as compute nodes for the queries executed by 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 coordinator on to the shards as much as possible.

It is recommended that applications separate their workloads for direct routing and proxy routing. Separate connection pools must be created for these workloads.

Oracle Sharding Coordinator

The Oracle Sharding coordinator database contains the metadata of the sharded topology and provides query processing support for sharded databases.

Connecting to the Coordinator

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
```

Coordinator High Availability

The unavailability of the coordinator impacts proxy-routing based workloads, so it is highly recommended that the coordinator be protected with Data Guard in Maximum Availability protection mode (zero data loss failover) with fast-start failover enabled. The coordinator may optionally be Oracle RAC-enabled for additional availability and scalability.

Unavailability of the coordinator has zero impact on workloads utilizing direct routing.

Coordinator Database Sizing

The shard catalog and coordinator host the following key information:

- Metadata of the sharded database topology

- Schema of the sharded application
- Master copies of the duplicated tables

The size of the metadata and the schema is nominal; however, the number of duplicated tables and the space they occupy should be planned for when sizing the coordinator.

In addition to the above, the coordinator should also be sized to handle proxy routing, which can be CPU, I/O, and memory intensive based on the SQL queries and the amount of data being processed.

- [Resiliency of Proxy Routing](#)
It is highly recommended that the coordinator be protected with Data Guard with fast-start failover and optionally be Oracle RAC-enabled for availability and scalability

Resiliency of Proxy Routing

It is highly recommended that the coordinator be protected with Data Guard with fast-start failover and optionally be Oracle RAC-enabled for availability and scalability

Failure of the coordinator affects multi- and single-shard queries that are routed through the coordinator. The following are failure scenarios while querying and the expected behavior of proxy routing:

- If a participating shard is down, then the coordinator sends the query to another shard with same data.
- If failure happens during execution of the query on a participating shard, then the user will receive an error.

Querying and DMLs Using Proxy Routing

Proxy routing enables aggregation of data and reporting across shards. It also allows the flexibility for any database application to execute SQL statements (including SELECT and DML) in a system where tables are sharded or duplicated without the need to specify the sharding key (during connect) where the query should execute.

In both aggregation and SQL execution without a sharding key use-cases, the user accepts a reduced level of performance compared to direct, key-based, routing.

In a sharded database (SDB), there are two types of tables: sharded tables and duplicated tables.

Sharded tables are equi-partitioned on a sharding key.

$$S = S_1 \cup S_2 \cup \dots \cup S_n$$

Duplicated tables are identical on all shards.

$$R = R_1 = \dots = R_n$$

Proxy routing in an SDB provides a transparent mechanism to execute typical SQL queries that access data from sharded and duplicated tables without requiring the application to specify the relevant shards. The SQL compiler identifies the relevant shards automatically and coordinates the query execution across all the participating shards. Database links are used for the communication between the coordinator and the shards.

At a high level the coordinator rewrites each incoming query, Q, into a distributive form composed of two queries, CQ and SQ, where SQ (Shard Query) is the portion of Q that executes on each participating shard and CQ (Coordinator Query) is the portion that executes on the coordinator shard.

```
Q => CQ ( Shard_Iterator( SQ ) )
```

The following is an example of an aggregate query Q1 rewritten into Q1' for an inter shard execution:

```
Q1 : SELECT COUNT(*) FROM customers
```

```
Q1' : SELECT SUM(sc) FROM (Shard_Iterator(SELECT COUNT(*) sc FROM s1
(i) ))
```

There are two key elements in this process: (1) identifying the relevant shards (2) rewriting the query into a distributive form, and shard iteration.

During the compilation of a query on the coordinator database, the query compiler analyzes the predicates on the sharding key and extracts the ones that can be used to identify the participating shards, i.e. 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 full execution. This is termed as a Single Shard Query. If there is more than one participating shard the query is known as multi-shard query and is rewritten. The rewriting process takes into account the expressions computed by the query as well as the query shape.

Proxy Routing 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 can be SELECT, UPDATE, DELETE and INSERT. MERGE/UPSERT are not supported.

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 and INSERT

Proxy Routing for Multi-Shard Queries

A multi-shard query 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 may need to do some processing before sending the result to the client. For example, the following query fetches 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 are supported for `SELECT` statements only. A query can either access a single shard (in case of equality filter), or ALL shards (in case of no filter predicate on sharding key).

- [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.

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 Shapes in Proxy Routing

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

Restrictions

The following are restrictions on the use of query constructs in Oracle Sharding:

- Restrictions on DMLs are given with examples in the Transactions section below.
- `CONNECT BY` queries are not supported.
- IAS on sharded table on coordinator is not supported.
- `MODEL` clause is not supported.

Query Involves Only Duplicated Tables

For queries that involve only duplicated tables there are no restrictions on the query shape. The query is executed on the coordinator.

Query Involves Only Sharded Tables

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 are some examples of queries involving sharded tables.

Inner join where equi-join is only on sharding key.

```
SELECT ... FROM s1 INNER JOIN s2 ON s1.sk=s2.sk  
WHERE any_filter(s1) AND any_filter(s2)
```

Left outer join only on sharding key.

```
SELECT ... FROM s1 LEFT OUTER JOIN s2 ON s1.sk=s2.sk
```

Right outer join, same as left outer join.

```
SELECT ... FROM s1 RIGHT OUTER JOIN s2 ON s1.sk=s2.sk
```

Full outer join only on the sharding key, but only if Native is valid which requires equi-join.

```
SELECT ... FROM s1 FULL OUTER JOIN s2 ON s1.sk=s2.sk
WHERE any_filter(s1) AND any_filter(s2)
```

Query Involves 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 will contain a non sharded table which is duplicated on each shard.

Joins between a sharded table and a duplicated table can be on any column using any comparison operator (= < > <= >=) or even arbitrary join expression. The following are examples of join patterns.

Inner join

```
SELECT ... FROM s1 INNER JOIN r1 ON any_join_condition(s1,r1)
WHERE any_filter(s1) AND any_filter(r1)
```

Left/Right outer join

Where 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)
```

Where sharded table is 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)
```

Full outer join

```
SELECT ... FROM s1 FULL OUTER JOIN r1 ON s1.sk=s2.sk
WHERE any_filter(s1) AND any_filter(s2)
```

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))
```

Anti-join (NOT EXISTS)

```
SELECT ... FROM s1 NOT EXISTS  
(SELECT 1 FROM r1 WHERE r1.anykey=s1.anykey)
```

Left/Right outer join

Where 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 sharding key:

```
SELECT ... FROM r1 LEFT OUTER JOIN s1 ON any_join_condition(s1,s2)  
AND any_filter(r1) AND any_filter(s1)
```

Where 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)
```

Full Outer Join

Sharded table requiring 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)
```

Semi-join (EXISTS)

Sharded table is in a subquery that requires access of multiple shards:

```
SELECT ... FROM r1 EXISTS  
(SELECT 1 FROM s1 WHERE r1.anykey=s1.anykey)
```

Anti-join (NOT EXISTS)

Sharded table is in the sub-query:

```
SELECT ... FROM r1 NOT EXISTS  
(SELECT 1 FROM s1 WHERE r1.anykey=s1.anykey)
```

Aggregate Functions

The following aggregations are supported by proxy routing:

- COUNT
- SUM
- MIN
- MAX
- AVG

Multi-shard Queries and Global CR

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.

Transactions

A DML statement that affects only one shard is supported. For example,

```
update S1 set col = ... where sk = <constant>;
```

A DML statement that affects more than one shard is not supported. For example,

```
update S1 set col = ...;
```

Within a transaction, multiple single shard DMLs can be performed on different shards. For example,

```
insert into S1 values (...);  
Update S1 set col = ... where sk = constant;  
Delete S1 where sk = constant;  
Commit;
```

For multi-shard DML, the coordinator uses database link, starts a distributed transaction and performs two phase commit to guarantee the consistency of the distributed transaction. In the case of an in-doubt transaction, the database administrator has to recover it manually.

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.

Execution Plans for Proxy Routing

In a multi-shard query, each shard produces an independent execution plan that is potentially different from the plans on the other shards in the query.

Starting in Oracle Database 18c you no longer 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 display the plan of 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 the plans from shards use the format `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 of interest 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/X]\$SHARD_SQL

[V/X]\$SHARD_SQL 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\$SHARD_SQL view.

```
SQL> Select * from v$shard_sql 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

Creating Affinity Between Middle-Tier Connection Pools and Shards

A Oracle Universal Connection Pool (UCP) feature called middle-tier routing allows smart routers (F5 BigIP, for example) to route to the middle tier associated with a sharding key. It publishes the sharded database topology to the router tier so that requests based on specific sharding keys are routed to the appropriate application middle tier, which in turn establishes connections on the given subset of shards.

In a typical Oracle Sharding environment, middle-tier connection pools route database requests to specific shards. This can lead to a situation where each middle-tier connection pool establishes connections to each shard. This can create too many connections to the database. The issue can be solved by creating an affinity between the middle tiers and shards. In this scenario it would be ideal to dedicate a middle tier (web server, application server) for each data center or cloud, and to have client requests routed directly to the middle tier where the shard containing the client data (corresponding to the client shard key) resides. A common term used for this kind of setup is swim lanes, where each swim lane is a dedicated stack, from web server to application server all the way to the database.

Oracle Universal Connection Pool (UCP) solves this problem by providing a middle-tier routing API which can be used to route client requests to the relevant middle tier. The UCP middle tier API is exposed by the `OracleShardRoutingCache` class. An instance of this class represents the UCP internal shard routing cache, which can be created by providing connection properties such as user, password, and URL. The routing cache connects to the sharding catalog to retrieve the key to shard mapping topology and stores it in its cache.

The routing cache is used by UCP middle-tier API

`getShardInfoForKey(shardKey, superShardKey)`, which accepts a sharding key as input and returns a set of `ShardInfo` instances mapped to the input sharding key. The `ShardInfo` instance encapsulates a unique shard name and priority of the shard. An application using the middle-tier API can map the returned unique shard name value to a middle tier that has connections to a specific shard. The routing cache is automatically updated when chunks are split or moved to other shards by subscribing to respective ONS events.

The following code example illustrates the usage of Oracle UCP middle-tier routing API.

Example 6-4 Middle-Tier Routing Using UCP API

```
import java.sql.SQLException;
import java.util.Properties;
import java.util.Random;
import java.util.Set;

import oracle.jdbc.OracleShardingKey;
import oracle.jdbc.OracleType;
import oracle.ucp.UniversalConnectionPoolException;
import oracle.ucp.routing.ShardInfo;
import oracle.ucp.routing.oracle.OracleShardRoutingCache;

/**
```

```

* The code example illustrates the usage of UCP's mid-tier routing
feature.
* The API accepts sharding key as input and returns the set of ShardInfo
* instances mapped to the sharding key. The ShardInfo instance encapsulates
* unique shard name and priority. The unique shard name then can be mapped
* to a mid-tier server which connects to a specific shard.
*
*/
public class MidtierShardingExample {

    private static String user = "testuser1";
    private static String password = "testuser1";

    // catalog DB URL
    private static String url = "jdbc:oracle:thin:@//hostName:1521/
catalogServiceName";
    private static String region = "regionName";

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

    static void testMidTierRouting() throws UniversalConnectionPoolException,
        SQLException {

        Properties dbConnectProperties = new Properties();
        dbConnectProperties.setProperty(OracleShardRoutingCache.USER, user);
        dbConnectProperties.setProperty(OracleShardRoutingCache.PASSWORD,
password);
        // Mid-tier routing API accepts catalog DB URL
        dbConnectProperties.setProperty(OracleShardRoutingCache.URL, url);

        // Region name is required to get the ONS config string
        dbConnectProperties.setProperty(OracleShardRoutingCache.REGION, region);

        OracleShardRoutingCache routingCache = new OracleShardRoutingCache(
            dbConnectProperties);

        final int COUNT = 10;
        Random random = new Random();

        for (int i = 0; i < COUNT; i++) {
            int key = random.nextInt();
            OracleShardingKey shardKey = routingCache.getShardingKeyBuilder()
                .subkey(key, OracleType.NUMBER).build();
            OracleShardingKey superShardKey = null;

            Set<ShardInfo> shardInfoSet = routingCache.getShardInfoForKey(shardKey,
                superShardKey);

            for (ShardInfo shardInfo : shardInfoSet) {
                System.out.println("Sharding Key=" + key + " Shard Name="
                    + shardInfo.getName() + " Priority=" + shardInfo.getPriority());
            }
        }
    }
}

```

```
}  
}
```

Related Topics

- [Middle-Tier Routing Using UCP](#)

7

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.
- [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.
- [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.

About Sharding and Replication

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

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.

- [When To Choose Oracle GoldenGate for Shard High Availability](#)
When should Oracle GoldenGate be employed as your high availability solution for Oracle Sharding?

When To Choose Oracle GoldenGate for Shard High Availability

When should Oracle GoldenGate be employed as your high availability solution for Oracle Sharding?

Oracle GoldenGate should be your preferred high availability solution in the following cases:

- All shards read-write. With Active Data Guard the DR/backup shards are read-only.
- More flexibility in deploying shards. Each shard can be on a different operating system or a different database version.
- More than a single updatable copy of the data, even within a single shardgroup. For example, with Oracle GoldenGate, using the replication factor of 4, you can have 4 read-write copies of the data that can be updated.

 **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.

Using Oracle Data Guard with a Sharded Database

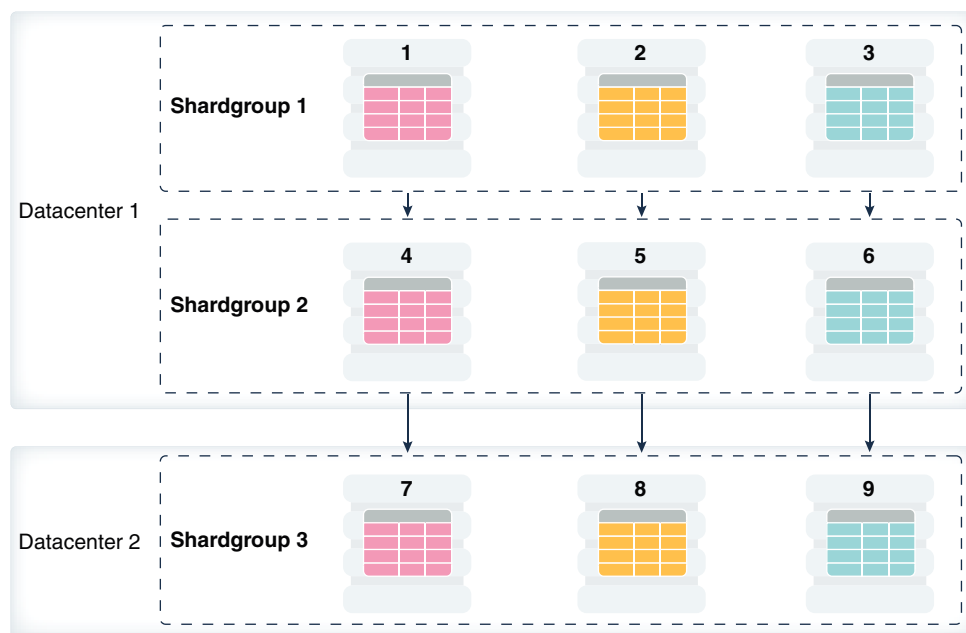
Oracle Data Guard replication maintains one or more synchronized copies (standbys) of a shard (the primary) for high availability and data protection. Standbys may be deployed locally or remotely, and when using Oracle Active Data Guard can also be open for read-only access.

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 7-1 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

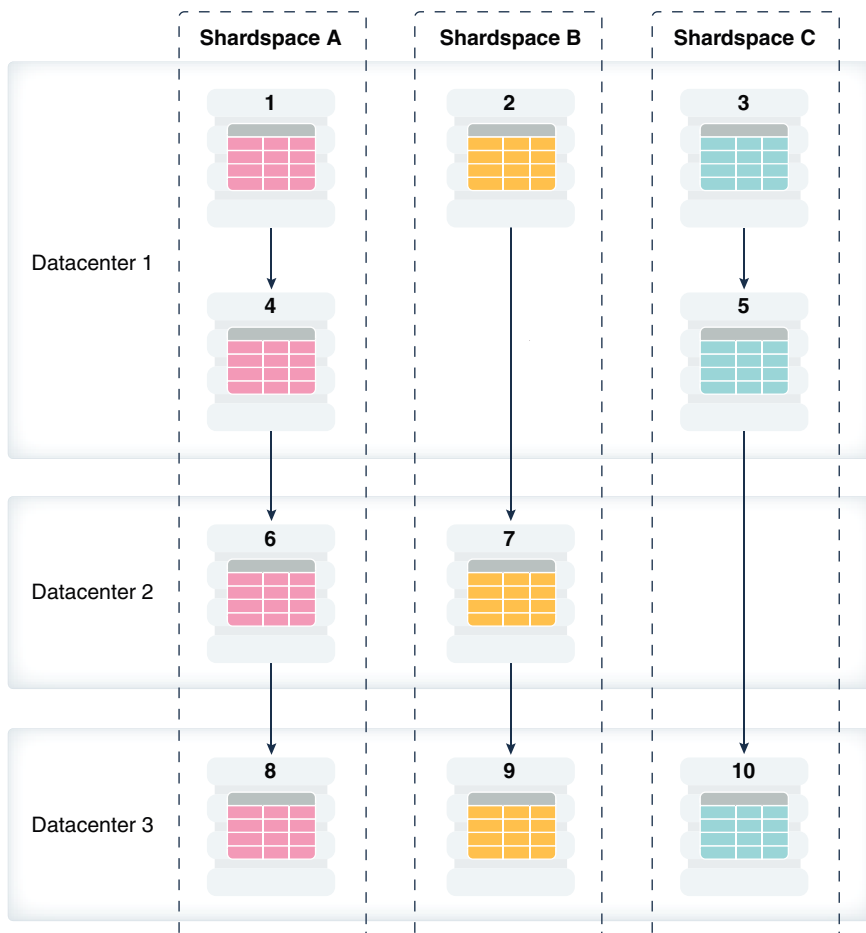
DEPLOY

```

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 7-2 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
...

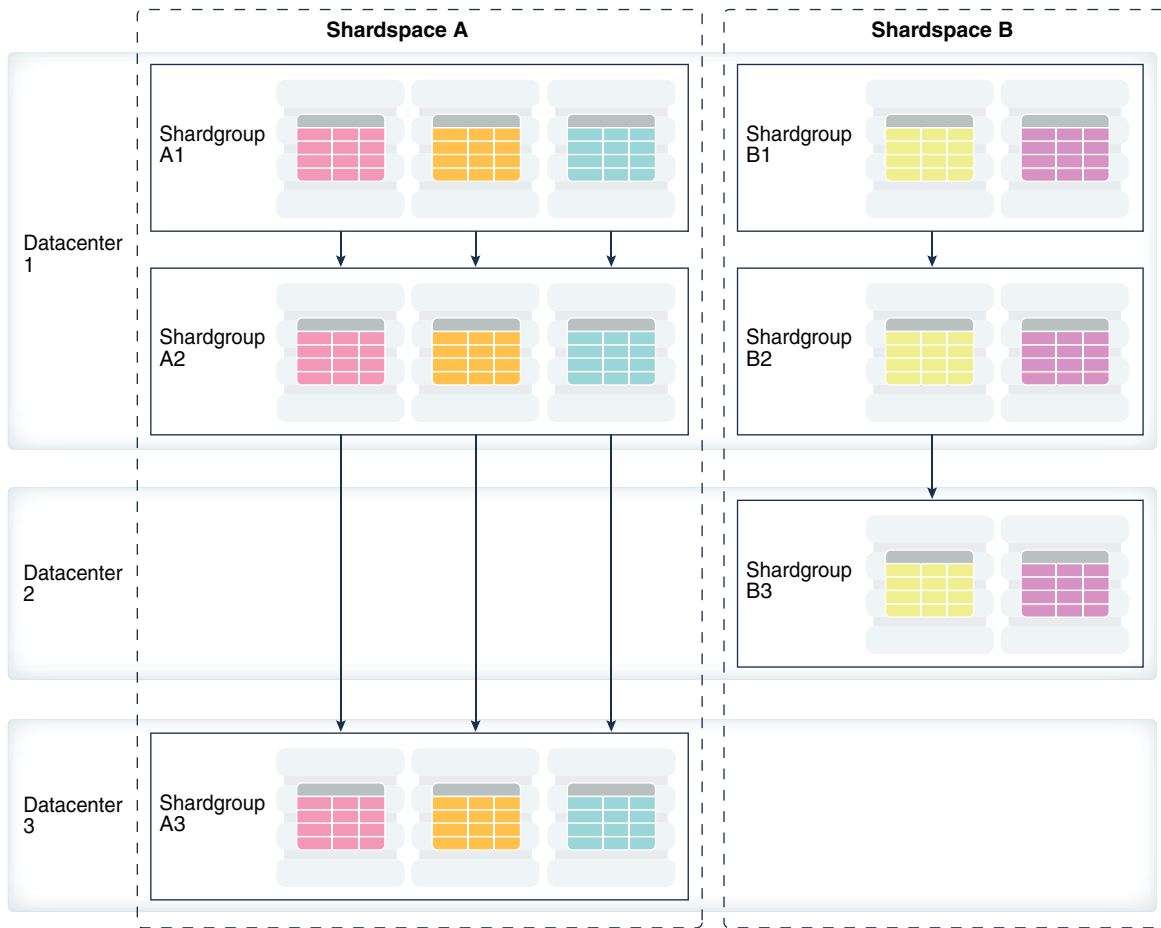
CREATE SHARD -shardspace shardspace_c -region dc3 -deploy_as standby -
destination
host10 -credential oracle_cred -netparamfile /home/oracle/netca_dbhome.rsp

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 7-3 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

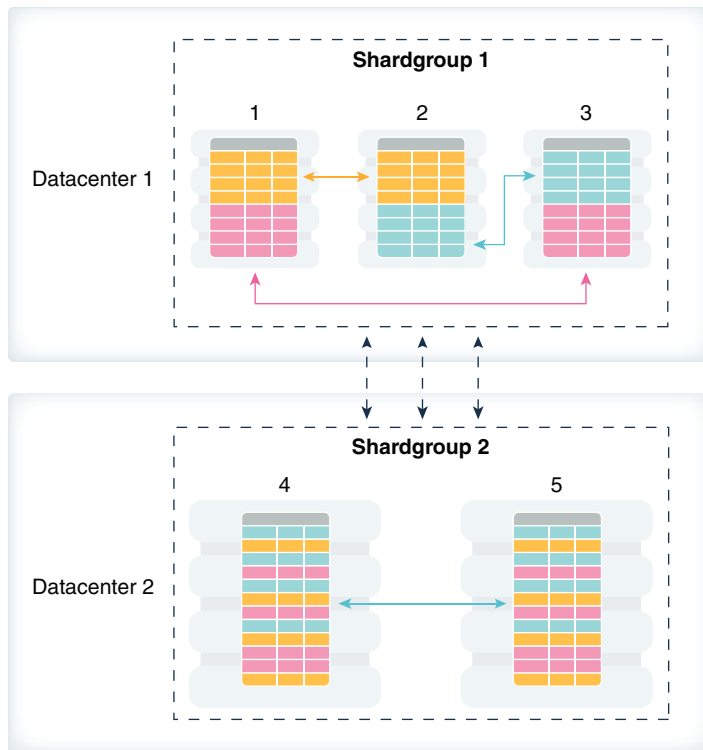
DEPLOY
```

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.

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 7-4 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.

Before creating any shards, there are some prerequisites:

- Register with scheduler (when using `GDSCTL create shard`)
- Prepare site-security wallets or client and server certificates.
- Install Oracle GoldenGate and add at least one secure deployment with sharding option, and start up GoldenGate services and servers.
- In each Oracle home, make a copy of the client wallets used to add GoldenGate deployments, and place it at `$ORACLE_BASE/admin/ggshd_wallet/`.
- Load PL/SQL packages from a GoldenGate install home. If you are creating shards using `GDSCTL CREATE SHARD`, this step is only applicable to the shard catalog. If you are using `GDSCTL ADD SHARD`, it applies to the shard catalog and all of the shards.

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

```
CREATE SHARDCATALOG -database host00:1521:shardcat -chunks 60
  -user 'gsmcatuser/gsmcatuser_password'
  -repl OGG -sharding system -sdb orasdb
ADD GSM -gsm gsm1 -listener 1571 -catalog shard-dirl:1521:shardcat -localons
3841
ADD GSM -gsm gsm2 -listener 1571 -catalog shard-dirl:1521:shardcat -localons
3841
START GSM -gsm gsm1
START GSM -gsm gsm2
CONFIGURE -timeout 900
ADD REGION -region dc1
ADD REGION -region dc2
MODIFY GSM -gsm gsm1 -region dc1
MODIFY GSM -gsm gsm2 -region dc2
ADD SHARDGROUP -shardgroup shardgroup1 -region dc1 -refactor 2
ADD SHARDGROUP -shardgroup shardgroup2 -region dc2 -refactor 2

CREATE SHARD -shardgroup shardgroup1 -destination host01 -credential
oracle_cred
  -netparam /home/oracle/netca_dbhome.rsp -gg_service host01:9900/
deployment_name
  -gg_password ggadmin_password -dbparamfile /home/oracle/dbparams01.tmp
  -dbtemplatefile /home/oracle/sharddb01.dbt

CREATE SHARD -shardgroup shardgroup1 -destination host02 -credential
oracle_cred
  -netparam /home/oracle/netca_dbhome.rsp -gg_service host02:9900/
remote_scheduler_agent
  -gg_password ggadmin_password -dbparamfile /home/oracle/dbparams02.tmp
  -dbtemplatefile /home/oracle/sharddb02.dbt

CREATE SHARD -shardgroup shardgroup1 -destination host03 -credential
oracle_cred
  -netparam /home/oracle/netca_dbhome.rsp -gg_service host03:9900/
```

```
remote_scheduler_agent
-gg_password ggadmin_password -dbparamfile /home/oracle/dbparams03.tmp
-dbtemplatefile /home/oracle/sharddb03.dbt

CREATE SHARD -shardgroup shardgroup2 -destination host04 -credential
oracle_cred
-netparam /home/oracle/netca_dbhome.rsp -gg_service host04:9900/
remote_scheduler_agent
-gg_password ggadmin_password -dbparamfile /home/oracle/dbparams04.tmp
-dbtemplatefile /home/oracle/sharddb04.dbt

CREATE SHARD -shardgroup shardgroup2 -destination host05 -credential
oracle_cred
-netparam /home/oracle/netca_dbhome.rsp -gg_service host05:9900/
remote_scheduler_agent
-gg_password ggadmin_password -dbparamfile /home/oracle/dbparams05.tmp
-dbtemplatefile /home/oracle/sharddb05.dbt
```

DEPLOY

Note that the above example uses `CREATE SHARD` to create new shards during deployment. `ADD SHARD` is the alternative to `CREATE SHARD`, and the `ADD SHARD` method assumes the pre-existence of clean slate database instances ready to be converted into database shards.

 **Note:**

Unlike sharding replication with Data Guard or Active Data Guard, you cannot deploy Oracle GoldenGate manually, it must be done using the `DEPLOY` command.

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.

8

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.
- [Oracle Sharding Prerequisites](#)
Before you install any software, review these hardware, network, and operating system requirements for Oracle Sharding.
- [Installing Oracle Database Software](#)
Install Oracle Database on each system that will host the shard catalog or database shards.
- [Installing the Shard Director Software](#)
Install the global service manager software on each system that you want to host a shard director.
- [Creating the Shard Catalog Database](#)
Create an Oracle Database using DBCA to host the shard catalog.
- [Setting Up the Oracle Sharding Management and Routing Tier](#)
The shard catalog, shard directors and shards must be configured to communicate with each other.
- [Creating and Deploying a System-Managed Sharded Database](#)
- [Creating and Deploying a User-Defined SDB](#)
- [Creating and Deploying a Composite SDB](#)
To deploy a composite SDB you must install the required Oracle Sharding software components, configure the objects for a composite SDB, and create the schema.
- [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.

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.

At a high level, the deployment steps are:

1. Set up the components.
 - Create a database that hosts the shard catalog.
 - Install Oracle Database software on the shard nodes.
 - Install shard director (GSM) software on the shard director nodes.

 **Note:**

For production deployments, it is highly recommended that you configure Data Guard for the shard catalog database.

2. Specify the topology layout using the following commands.
 - `CREATE SHARDCATALOG`
 - `ADD GSM`
 - `START GSM`
 - `ADD CREDENTIAL` (if using `CREATE SHARD`)
 - `ADD SHARDGROUP`
 - `ADD INVITEDNODE`
 - `CREATE SHARD` (or `ADD SHARD`) for each shard
3. Run `DEPLOY` and add the global service to access any shard in the sharded database.
 - `DEPLOY`
 - `ADD SERVICE`
- [Choosing a Deployment Method](#)
You can deploy a sharded database, by creating the shards at the same time for a new database, or by adding the shards from a preexisting database.
- [Using Oracle Multitenant with Oracle Sharding](#)
You can use a multitenant container database (CDB) containing a single pluggable database (PDB) as a shard in your Oracle Sharding configuration.

Choosing a Deployment Method

You can deploy a sharded database, by creating the shards at the same time for a new database, or by adding the shards from a preexisting database.

Oracle Sharding supports two deployment methods. The first method is with the `CREATE SHARD` command, where the creation of shards and the replication configuration are automatically done by the Oracle Sharding management tier. This method cannot be used in a multitenant architecture where PDBs are used as shards.

The second deployment method is with the `ADD SHARD` command. If your database creation standards require that you deploy the SDB using your own pre-created databases, the `ADD SHARD` deployment method supports this requirement by simply adding your prebuilt database shards.

Deployment Method: CREATE SHARD

The `DEPLOY` command creates the shards. This is done using the `DBMS_SCHEDULER` package (executed on the shard catalog), which communicates with the Scheduler agents on the remote shard hosts.

Agents then invoke `DBCA` and `NETCA`, and if Oracle GoldenGate replication is specified, GoldenGate Creation Assistance (GGCA) , to create the shards and the local listeners. After the primary shards are created the corresponding standby shards are built using the `RMAN DUPLICATE` command.

When Data Guard is used as the high availability solution, once the primary and standby shards are built, the `DEPLOY` command configures Data Guard Broker with Fast-Start Failover (FSFO) enabled. The FSFO observers are automatically started on the regional shard director.



Note:

The `CREATE SHARD` method is not supported for PDBs used as shards. Only the `ADD SHARD` methods can be used in a multitenant architecture.

Archivelog and flashback are enabled for all of the shards. This is required for the FSFO observer to perform standby auto-reinstantiation upon failover.

Deployment Method: ADD SHARD

Use the `ADD SHARD` command to add shards to a sharded database configuration if you have your own database creation standards and prefer to deploy the sharded database using your own pre-created databases. This method is recommended for shards that are Oracle RAC-enabled, Oracle Restart-enabled, or PDB shards. This method is recommended when ASM is being used. The `ADD SHARD` deployment method supports this requirement by adding shards, which already have database installations deployed on them, rather than creating new instances.

When the `ADD SHARD` command is used for deployment, and Data Guard is used for high availability, the `DEPLOY` command handles the configuration of Oracle GoldenGate, or Data Guard, Broker and Fast-start Failover. It also handles the scenario where you have pre-configured Data Guard for the shard that is being added.

Unlike sharding with Data Guard or Active Data Guard, you cannot deploy Oracle GoldenGate manually, it must be done using the `DEPLOY` command.

Using Oracle Multitenant with Oracle Sharding

You can use a multitenant container database (CDB) containing a single pluggable database (PDB) as a shard in your Oracle Sharding configuration.

To support consolidation of databases on under-utilized hardware, for ease of management, or geographical business requirements, you can use single PDBs in CDBs as database shards. For example, for database consolidation, you can add other, non-shard PDBs to the CDB containing a shard PDB.

To add a shard PDB to the sharded database configuration, you should first add the CDB in which that shard PDB is contained to the shard catalog. The `GDSCTL` command `ADD CDB` is

used to add a pre-created CDB to the shard catalog. Then, use the GDSCTL ADD SHARD command with the `-cdb` option to add shards which are a PDB contained within a CDB to the sharded database during deployment.

The following example adds a CDB with unique name `db11` to the shard catalog and then adds it to shardgroup `shgrp1` in a sharded database configuration.

```
GDSCTL> add cdb -connect CDB$ROOT_connect_string -pwd GSMUSER_password
GDSCTL> add shard -cdb db11 -connect PDB_connect_string -shardgroup
shgrp1
-deploy_as active_standby -pwd GSMUSER_password
```

Use `CONFIG CDB` to display information about the CDB in the shard catalog.

```
GDSCTL> config cdb
```

```
Name: tstdsbyb
Connection string: (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=cdb1host)
(PORT=1521))
(CONNECT_DATA=(SERVICE_NAME=cdb1.example.com)))
SCAN address:
ONS remote port: 0
Disk Threshold, ms: 20
CPU Threshold, %: 75
Version: 18.0.0.0
Rack:
```

**Note:**

Oracle GoldenGate replication does not support PDBs as shards.

Moving PDB Shards

You can manually unplug a shard PDB from a CDB, and plug it in to a different CDB. This can be done outside of the sharding interfaces, and then you can update the shard catalog metadata to indicate that the PDB shard has moved to another CDB. The GDSCTL command `ADD SHARD` with the `-REPLACE` option is used to update the location of the shard PDB in the shard catalog.

PDB Shard High Availability

Oracle Data Guard supports replication only at the CDB level. The existing sharding architecture allows replicated copies of the sharded data for high availability, and it can optionally configure and use Data Guard to create and maintain these copies. Data Guard does not currently support replication at the PDB level; it can only replicate an entire container.

Making Changes to CDB in Sharding Configuration

Use `MODIFY CDB` to change the metadata of the CDB in the shard catalog. Some parameters cannot be used after the CDB contains shards, or contains shards that have been deployed.

```
GDSCTL> modify cdb -shard cdb1 -pwd new_password
```

Use `REMOVE CDB` to remove a CDB from the shard catalog. Removing a CDB does not destroy it.

```
GDSCTL> remove cdb -cdb cdb1
```

Upgrading from a Non-PDB Shard to a PDB Shard

When upgrading from a non-PDB sharded environment to one which makes use of PDBs, you must back up each existing non-PDB shard and then create a new CDB, and a PDB inside it. The shard is then restored to the PDB inside the CDB, as the CDB migration guide recommends. At this point, the shard has become a PDB inside a CDB, and you use the `GDSCTL ADD CDB` command to add the new CDB, and then run `ADD SHARD -REPLACE`, specifying the connect string of the PDB, to tell the sharding infrastructure to replace the old location of the shard with new PDB location.

See Also:

Oracle Database Global Data Services Concepts and Administration Guide for information about the GDSCTL commands used with PDB shards

Oracle Multitenant Administrator's Guide for information about Oracle Multitenant

Oracle Sharding Prerequisites

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

- Hardware and operating system requirements for the shards are the same as those on which Oracle Database is supported. See your Oracle Database installation documentation for these requirements.
- Hardware, software, and operating system requirements for the shard catalog and shard directors are the same as those on which the Global Data Services catalog and global service manager are supported. See *Oracle Database Global Data Services Concepts and Administration Guide* for these requirements.
- Network requirements are Low Latency GigE
- Port communication requirements are listed below. All of the following are required for using `CREATE SHARD`. When using `ADD SHARD`, items 4 and 5 are not relevant.
 1. Each and every shard must be able to reach each and every shard director's listener and ONS ports. 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. These 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.

2. Each and every shard must be able to reach the TNS Listener port of the shard catalog (both primary and standby).
3. The TNS Listener port (default 1521) of each shard must be opened to shard directors and the shard catalog.
4. On the primary and standby shard catalog database, the port used for `-agent_port` (default 8080) in the `CREATE SHARDCATALOG` command must be visible to all of the shards.
5. The scheduler agent port on all of the shards must be visible to shard catalog node. Execute `schagent -status` on each shard to identify the port.

Installing Oracle Database Software

Install Oracle Database on each system that will host the shard catalog or database shards.

Before installing Oracle Database, create an operating system user on all of the systems where you will be hosting the sharded database, shard catalog, and shard directors, and assign them to the DBA group. Allow the user to run `su`, and make note of the credentials so that you can use them in later procedures.

See *Oracle Database Installation Guide for Linux*, or your platform's installation guide, for information about configuring operating system users.

1. Download the Oracle Database installer on all of the systems that will host the shard catalog or the database shards.
2. Install Oracle Database on all of the systems where you intend to host the shard catalog and sharded database.
 - a. Run the installer on the first system.

```
$ cd /u01/stage/database
$ ./runInstaller
```

As you step through the Oracle Database installation, be sure to select the following options on the noted screens:

- On the Installation Option page, select **Install database software only**.
- On the Grid Installation Options page, select **Single instance database installation**. Oracle RAC and Oracle RAC One Node are not supported in this release.
- On the Database Edition page, select **Enterprise Edition**.
- On the Installation Location page, use the same **Oracle base** and **Software location** values that you used when creating the environment scripts in the steps above.
- On the Create Inventory page, accept the defaults.
- On the Operating System Groups page, accept the defaults or make changes as appropriate for your environment.

- On the Summary page, you can click **Save Response File** to create a file for silent installation of the Oracle Database software on the remaining hosts.
 - During installation, execute the `oraInstRoot.sh` and `root.sh` scripts as `root` in a separate terminal when prompted.
- b. Optionally, using the response file you created in the first installation, run a silent install on each of the remaining hosts.

Note that, after performing a silent install using a response file, when you run the database `root.sh` script its execution might not prompt you interactively for any values and uses only default values (for example, for the local user `bin` directory). If any non-default values are desired, specify just the `-responseFile` location when invoking the Installer and omit the `-silent` option. Click through the Installer screens, accepting the response file values, and then run the root script(s) when prompted. During root script execution, any user prompts are presented to you and non-default values can be entered.



See Also:

Oracle Database Installation Guide for Linux for more information about using the response file for silent installation of Oracle Database

Installing the Shard Director Software

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

1. Download the Oracle Global Service Manager installer on all of the systems that will host the shard directors.
2. See *Oracle Database Global Data Services Concepts and Administration Guide* for information about installing a global service manager.
3. Optionally, using the response file you created in the first installation, run a silent install on each of the remaining shard director hosts.

See *Oracle Database Global Data Services Concepts and Administration Guide* for more information about the silent install process.

Note that, after performing a silent install using a response file, when you run the database `root.sh` script its execution might not prompt you interactively for any values and uses only default values (for example, for the local user `bin` directory). If any non-default values are desired, specify just the `-responseFile` location when invoking the Installer and omit the `-silent` option. Click through the Installer screens, accepting the response file values, and then run the root script(s) when prompted. During root script execution, any user prompts are presented to you and non-default values can be entered.

Creating the Shard Catalog Database

Create an Oracle Database using DBCA to host the shard catalog.

1. Connect to the host where you will host the shard catalog, and verify that the expected environment variables are set to the correct values.

```
$ env |grep ORA
ORACLE_BASE=/u01/app/oracle
ORACLE_HOME=/u01/app/oracle/product/18.0.0/dbhome_1
```

2. Create the oradata and fast_recovery_area directories.

```
$ mkdir /u01/app/oracle/oradata
$ mkdir /u01/app/oracle/fast_recovery_area
```

3. Run DBCA to create the shard catalog database.

```
$ dbca
```

The Database Configuration Assistant opens.

4. On the Database Operation page, select **Create a database**, and click **Next**.
5. On the Creation Mode page, select **Advanced configuration**, and click **Next**.
6. On the Deployment Type page, select the **Oracle Single Instance database** database type, select the **General Purpose or Transaction Processing** template, and click **Next**.
7. On the Database Identification page, enter the **Global Database name** and the shard catalog **SID** that you configured in the shard catalog host environment script, and click **Next**.
8. On the Storage Option page, select the **Use following for the database storage attributes** option, select **File System**, select the **Use Oracle-Managed Files (OMF)** option, and click **Next**.
9. On the Select Fast Recovery Option page, select **Specify Fast Recovery Area**, select **Enable archiving**, and click **Next**.
10. On the Specify Network Configuration Details page, select **Create a new listener**, set the listener name and port number, and click **Next**.

Make note of the listener name so that you can connect to the database later.

11. Skip the Data Vault Option page.
12. On the Configuration Options page **Memory** tab, select **Use Automatic Shared Memory Management**.
13. On the Configuration Options page **Character sets** tab, select **Use Unicode (AL32UTF8)**, and click **Next**.
14. On the Management Option page, uncheck the **Configure Enterprise Manager (EM) database express** option, and click **Next**.
15. On the User Credentials page, select the appropriate option for your business needs, enter the passwords, and click **Next**.

Make a note of the passwords you entered because you will need them later.

16. On the Creation Option page, select **Create database**, and click **Next**.
17. On the Summary page, click **Finish**.

18. After the database has been created, make a note of the Global Database Name, SID, and spfile values.
19. If you plan to use Oracle Data Guard to protect the shard catalog database, click **Password Management**, unlock the SYSDBG account, and make a note of the password you enter for this account.
20. Click **Close** to exit DBCA.

Setting Up the Oracle Sharding Management and Routing Tier

The shard catalog, shard directors and shards must be configured to communicate with each other.

Before you begin, carefully read through the port requirements prerequisites listed in [Oracle Sharding Prerequisites](#), and make any changes necessary before proceeding with the tasks in this section.

1. On the shard catalog host, verify that the expected environment values are set to the correct values.

```
$ env |grep ORA
ORACLE_SID=shardcat
ORACLE_BASE=/u01/app/oracle
ORACLE_HOME=/u01/app/oracle/product/18.0.0/dbhome_1
```

2. If the shard catalog listener is not already started, start the shard catalog listener.

```
$ lsnrctl start
```

3. Verify that `DB_CREATE_FILE_DEST` parameter is set on the shard catalog database.

If the parameter is not set, then set it as shown in the following example. Note that `open_links` and `open_links_per_instance` are set to 16 for the purposes of the Oracle Sharding demo application.

```
$ sqlplus / as sysdba

SQL> alter system set db_create_file_dest='/u01/app/oracle/oradata'
scope=both;
SQL> alter system set open_links=16 scope=spfile;
SQL> alter system set open_links_per_instance=16 scope=spfile;
```

4. Shut down and restart the shard catalog database.

```
SQL> shutdown immediate
Database closed.
Database dismounted.
```

```
SQL> startup
ORACLE instance started.
```

```
Total System Global Area 4798283776 bytes
Fixed Size                  4430760 bytes
Variable Size               1006634072 bytes
Database Buffers            3774873600 bytes
```

```
Redo Buffers          12345344 bytes
Database mounted.
Database opened.
```

5. Grant roles and privileges on the shard catalog database.

```
SQL> set echo on
SQL> set termout on
SQL> spool setup_grants_privs.lst
```

a. Unlock and set the password for the GSMCATUSER schema.

This schema is used by the shard director when connecting to the shard catalog database.

```
SQL> alter user gsmcatuser account unlock;
SQL> alter user gsmcatuser identified by gsmcatuser_password;
```

b. Create the administrator schema and grant privileges to it.

The mysdbadmin account is an account in the shard catalog database which stores information on the sharding environment. The mysdbadmin account is the database administrator schema for making administrative changes to the sharded database environment. GDSCTL connects through this user to the database when GDSCTL commands are run, and the mysdbadmin user makes the necessary changes in the database.

```
SQL> create user mysdbadmin identified by mysdbadmin_password;
SQL> grant connect, create session, gsmadmin_role to mysdbadmin;
SQL> grant inherit privileges on user SYS to GSMADMIN_INTERNAL;
SQL> spool off
```

6. Connect to a shard director host and start GDSCTL.

The commands in the following steps are executed from a shard director host because the GDSCTL command line interface is installed there.

The following example includes a sanity check that environment variables have been set properly.

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

$ gdsctl
```

7. Create the shard catalog and configure the remote scheduler agent on the shard catalog.

In this example, the sharded database is given the name `cust_sdb`, and two regions are created: `region1` and `region2`. The regions are used as local availability domains for failover of the shards from the primary to physical standby. The regions are not meant to represent geographical regions in these examples.

To create shard catalog for system-managed sharding, with Oracle GoldenGate replication:

```
GDSCTL> create shardcatalog -database
shard_catalog_host:port_number:shard_catalog_name
-user gsm_admin/mysdbadmin_password -repl OGG -repfactor 2 -sdb cust_sdb
-region region1, region2 -agent_port port_num -agent_password
rsa_password
```

 **Note:**

For production systems, it is a good practice to use the default (120 chunks per shard) , instead of specifying the `-chunks` parameter while creating the shard catalog.

Because system-managed is the default sharding method, it does not need to be specified with the `-sharding` parameter.

To create shard catalog for a composite sharded database, with Data Guard replication:

```
GDSCTL> create shardcatalog -database
shard_catalog_host:port_number:shard_catalog_name
-chunks 60 -shardspace shardspace1 -sharding composite -sdb comp_shpool
-protectmode maxavailability -user gsm_admin/mysdbadmin_password
```

To create shard catalog for a user-defined sharded database, with Data Guard replication:

```
GDSCTL> create shardcatalog -sdb udef_shpool -sharding user
-protectmode maxavailability
-database shard_catalog_host:port_number:shard_catalog_name
-user gsm_admin/mysdbadmin_password -region region1, region2
```

 **Note:**

The `-agent_port` and `-agent_password` parameters are not necessary if you are using the `ADD SHARD` deployment method.

8. While you are connected to the shard director host, create and start the shard director.

```
GDSCTL> add gsm -gsm sharddirector1 -listener listener_port -pwd
gsmcatuser_password
-catalog shard_catalog_host:1521:shardcat -region region1
```

```
GDSCTL> start gsm -gsm sharddirector1
```

Repeat steps 6 and 8 on each shard director host. Replace the shard director name and region name with appropriate values for each host.

9. Using GDSCTL, set the operating system credentials.

```
GDSCTL> add credential -credential credential_name -osaccount  
os_account_name  
-ospassword os_password  
GDSCTL> exit
```

 **Note:**

this step is not necessary if you are using the `ADD SHARD` deployment method.

These credentials are the operating system user name and password on the shard hosts (not the catalog host), and the credentials are used by the remote scheduler agent to run jobs on the hosts to set up the shards using DBCA, NETCA, and the like.

Repeat this step if a different operating system credential will be used for each host.

10. Connect to each of the shard hosts, register remote scheduler agents on them, and create directories for `oradata` and `fast_recovery_area` on them.

 **Note:**

This step is only required if you are using the `CREATE SHARD` method described in [Introduction to Sharded Database Deployment](#). If you are using the `ADD SHARD` method you can skip this step.

Execute the following statements on each of the machines that will host a shard.

Note that the `os_account_name` is the account used for Oracle software installation, `shard_host` is the host name or IP address of the machine hosting the shard, the `shard_catalog_host` is the host name or IP address of the shard catalog host, and `port_num` is the remote scheduler agent port number as specified in the `-agent_port` parameter for `create shardcatalog` in step 7 above. The remote scheduler agent prompts you for the agent registration password specified in the `-agent_password` parameter of `create shardcatalog` in step 7 above.

```
$ ssh os_account_name@shard_host  
passwd: os_password  
  
$ schagent -start  
$ schagent -status  
$ schagent -registerdatabase shard_catalog_host port_num  
  
$ mkdir /u01/app/oracle/oradata  
$ mkdir /u01/app/oracle/fast_recovery_area
```

 **See Also:**

Oracle Database Global Data Services Concepts and Administration Guide for information about usage and options for the GDSCTL commands.

Creating and Deploying a System-Managed Sharded Database

The following topics describe the tasks for creating and deploying a system-managed sharded database.

- [Deploying a System-Managed Sharded Database](#)
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.
- [Creating a Schema for a System-Managed Sharded Database](#)
Create the schema user, tablespace set, sharded tables, and duplicated tables for the sharded database. Verify that the DDLs are propagated to all of the shards, and, while connected to the shards, verify the automatic Data Guard Broker configuration with Fast-Start Failover.
- [System-Managed SDB Demo Application](#)
The system-managed sharded database (SDB) demo application simulates the workload of an online retail store. Use it to validate the setup of any system-managed (automatic sharding) SDB configuration. The demo application also provides a practical example of sharding concepts for administrators and developers new to database sharding.

Deploying a System-Managed Sharded Database

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.

System-managed sharding does not require the user to map data to shards. 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](#).

1. If you are using the `ADD SHARD` method described in [Introduction to Sharded Database Deployment](#) you must first create the databases to be used as shards on their respective hosts.

The shard databases must have the following characteristics:

- They must have an associated TNS Listener on each host
- The `GSMUSER` account must be unlocked with a known password
- `SYSDG` and `SYSBKUP` privileges must be granted to `GSMUSER`
- The primary and standby databases must be configured as such
- Redo apply should be set up between the corresponding primary and standby databases
- Flashback and force logging should be enabled

- The `compatible` parameter must be set to at least 12.2.0
- A server parameter file (SPFILE) must be in use
- A `DATA_PUMP_DIR` directory object must be created in each database and must point to a valid directory

You must then validate that a database is correctly set up for sharding. Run the following statements against each database, while logged in as SYS, before adding it to the configuration.

```
SQL> set serveroutput on
SQL> execute DBMS_GSM_FIX.validateShard
```

Screen output will include INFO, WARNING, and ERROR information that needs to be analyzed for any issues. All WARNING and ERROR messages must be resolved. Re-run `validateShard()` after making changes to confirm the configuration.

2. Connect to the shard director host.

```
$ ssh os_user@shard_director1_host
```

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

```
$ gdsctl
GDSCTL> set gsm -gsm shardedirector1
GDSCTL> connect mysdbadmin/mysdbadmin_password
```

4. Add a shardgroup for the primary shards.

In this example the shardgroup is named `primary_shardgroup`, and is part of a Data Guard replication solution.

```
GDSCTL> add shardgroup -shardgroup primary_shardgroup -deploy_as
primary
-region region1
```

The following example shows the shardgroup creation for an Oracle GoldenGate replication solution.

```
GDSCTL> add shardgroup -shardgroup shardgroup1 -region region1 -
repfactor 2
```

5. Add a shardgroup for the standby shards.

In this example the shardgroup is named `standby_shardgroup`, and is created for an Active Data Guard active standby using the `-deploy_as` parameter.

```
GDSCTL> add shardgroup -shardgroup standby_shardgroup -deploy_as
active_standby
-region region2
```

Adding a second shardgroup in an Oracle GoldenGate configuration might look like the following example.

```
GDSCTL> add shardgroup -shardgroup shardgroup2 -region region2 -refactor
2
```

6. Add each shard's host address to the valid node checking for registration (VNCR) list in the catalog, then create or add the shard in either the primary or standby shardgroup, as shown in the following examples.

 **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 may 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.

The following example shows how to create four shards using the `CREATE SHARD` command, using a Data Guard high availability solution, two of which are in the primary shardgroup and two in the standby shardgroup. The `credential_name` is the operating system credential you created in the shard catalog for the destination host.

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

There are other optional parameters for `CREATE SHARD` that allow you to customize the database parameters, storage and file locations, listener port numbers, and so on, which are documented in the *Oracle Database Global Data Services Concepts and Administration Guide* appendices.

```
GDSCTL> add invitednode shard_host_1
GDSCTL> create shard -shardgroup primary_shardgroup -destination
shard_host_1
```

```

-credential credential_name -sys_password sys_password

GDSCTL> add invitednode shard_host_2
GDSCTL> create shard -shardgroup standby_shardgroup -destination
shard_host_2
-credential credential_name -sys_password sys_password
GDSCTL> add invitednode shard_host_3
GDSCTL> create shard -shardgroup primary_shardgroup -destination
shard_host_3
-credential credential_name -sys_password sys_password
GDSCTL> add invitednode shard_host_4
GDSCTL> create shard -shardgroup standby_shardgroup -destination
shard_host_4
-credential credential_name -sys_password sys_password

```

In an Oracle GoldenGate replication solution, the shardgroups would not be designated as primary and standby because replication is handled at the chunk level and distributed among the shards within a shardgroup. However, a disaster recovery best practice is to replicate a shardgroup to one or more data centers. The following is an example of creating a shard with Oracle GoldenGate replication.

```

GDSCTL> create shard -shardgroup shardgroup -destination shard_host
-credential oracle_cred -netparam /home/oracle/netca_dbhome.rsp
-gg_service shard_host_1:$ADMINSRVR_PORT/$GGHOME
-gg_password ggadmin_password
-dbparamfile /home/oracle/dbparams01.tmp
-dbtemplatefile /home/oracle/sharddb01.dbt

```

If you are using the `ADD SHARD` method described in [Introduction to Sharded Database Deployment](#), use the following command instead of the `CREATE SHARD` commands in the example above. If the shard database to be added is a pluggable database (PDB), you must use the `-cdb` option to `ADD SHARD` to specify which container database (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 that in Oracle Database 18c, only one PDB in each CDB is allowed to be a shard.

```

GDSCTL> add shard -shardgroup shgrp1
-connect shard_host:TNS_listener_port/shard_database_name
-pwd GSMUSER_password

```

7. Check the configuration from a shard director.

Note that the shard names, sh1, sh2, sh3, and sh4, are system generated shard names when the `CREATE SHARD` method is used.

```

GDSCTL> config
Regions
-----
region1
region2

```

```

GSMs
-----
sharddirector1
sharddirector2

Sharded Database
-----
cust_sdb

Databases
-----
sh1
sh2
sh3
sh4

Shard Groups
-----
primary_shardgroup
standby_shardgroup

Shard spaces
-----
shardspaceora

Services
-----

GDSCTL pending requests
-----
Command          Object          Status
-----          -
Global properties
-----
Name: oradbcloud
Master GSM: sharddirector1
DDL sequence #: 0

GDSCTL> config shardspace
SHARDSPACE      Chunks
-----
shardspaceora   12

GDSCTL> config shardgroup
Shard Group          Chunks   Region   SHARDSPACE
-----
primary_shardgroup   12       region1  shardspaceora
standby_shardgroup   12       region2  shardspaceora

GDSCTL> config vncr
Name      Group ID
-----
shard_host_1
shard_host_2
    
```

```
shard_host_3
shard_host_4
shard_catalog_host_IP
```

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

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

The `DEPLOY` command takes some time to run, approximately 15 to 30 minutes.

```
GDSCTL> deploy
```

When the `CREATE SHARD` method is used to create the shards, the `DEPLOY` command creates the primary and standby shards using DBCA. Archivelog and flashback, which is required for the Fast-Start Failover observers to perform standby reinstantiation, are enabled for all of the shards.

Once the primary and standby shards are built, the `DEPLOY` command configures the Data Guard Broker with Fast-Start Failover enabled. The Fast-Start Failover observers are automatically started on the standby group's shard director (sharddirector2 in this example).

9. Verify that all of the 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
```

10. Verify that all of the shards are registered.

```
GDSCTL> databases
Database: "sh1" Registered: Y State: Ok ONS: N. Role: PRIMARY
Instances: 1 Region: region1
Registered instances:
cust_sdb%1
Database: "sh2" Registered: Y State: Ok ONS: N. Role: PH_STNDBY
Instances: 1 Region: region2
```



```

Registered instances:
  cust_sdb%11
Database: "sh3" Registered: Y State: Ok ONS: N. Role: PRIMARY
Instances: 1 Region: region1
Registered instances:
  cust_sdb%21
Database: "sh4" Registered: Y State: Ok ONS: N. Role: PH_STNDBY
Instances: 1 Region: region2
Registered instances:
  cust_sdb%31
  
```

11. Check the configuration of a shard.

```

GDSCTL> config shard -shard sh1
Name: sh1
Shard Group: primary_shardgroup
Status: Ok
State: Deployed
Region: region1
Connection string: shard_host_1:TNS_listener_port/sh1: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
-----
  
```

12. Add a global service that runs on all of the primary shards.

The `oltp_rw_srvc` global service is a global data service that a client can use to connect to the sharded database. The `oltp_rw_srvc` service runs the transactions on the primary shards. Likewise, the `oltp_ro_srvc` global service is created to run read-only workload on the standby shards.

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

```
GDSCTL> config service
```

```

Name                Network name                Pool        Started
Preferred all
-----
-----
oltp_rw_srvc       oltp_rw_srvc.cust_sdb.oracdbcloud cust_sdb    No        Yes
  
```

13. Start the oltp_rw_srvc global service.

```
GDSCTL> start service -service oltp_rw_srvc

GDSCTL> status service
Service "oltp_rw_srvc.cust_sdb.oradbcloud" has 2 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.
```

14. Add a global service for the read-only workload to run on the standby shards.

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

GDSCTL> config service

Name          Network name          Pool
Started Preferred all
----          -
-----
oltp_rw_srvc  oltp_rw_srvc.cust_sdb.oradbcloud cust_sdb
Yes          Yes
oltp_ro_srvc  oltp_ro_srvc.cust_sdb.oradbcloud cust_sdb
No          Yes
```

15. Start the read-only service, and verify the status of the global services.

```
GDSCTL> start service -service oltp_ro_srvc

GDSCTL> status service
Service "oltp_ro_srvc.cust_sdb.oradbcloud" has 2 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.
Service "oltp_rw_srvc.cust_sdb.oradbcloud" has 2 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.
```

**See Also:**[Creating a Schema for a System-Managed Sharded Database](#)*Oracle Database Global Data Services Concepts and Administration Guide* for more information about GDSCTL command usage

Creating a Schema for a System-Managed Sharded Database

Create the schema user, tablespace set, sharded tables, and duplicated tables for the sharded database. Verify that the DDLs are propagated to all of the shards, and, while connected to the shards, verify the automatic Data Guard Broker configuration with Fast-Start Failover.

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);
```

Specifying the shardspace is optional when creating the tablespace set. If the shardspace is not specified in the command, the default shardspace, `shardspaceora`, is used.

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 subpartition 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> ALTER SESSION ENABLE SHARD DDL;

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;

```

8. From the shard director host, verify that there were no failures during the creation of the tablespaces.

```

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"...
```

 **Note:**

The `show ddl` command output might be truncated. You can run `SELECT ddl_text FROM gsmadmin_internal.ddl_requests` on the catalog to see the full text of the statements.

9. Verify that there were no DDL errors on each of the shards.

Run the `config shard` and `config chunks` commands on each shard in your configuration.

```
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

GDSCTL> config chunks
Chunks
-----
Database                                From          To
-----
sh1                                    1           6
sh2                                    1           6
sh3                                    7           12
sh4                                    7           12
```

10. 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 is based on the number of chunks you specified in the `create shardcatalog` command.

The tablespace set with the first 6 chunks of the 12 that were specified in the shard catalog creation example, and the duplicated Products tablespace is shown in the following example.

```
$ 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.
```

Repeat this step on all of the shards in your configuration.

11. Verify that the chunks and chunk tablespaces were created on all of the shards.

```
SQL> set linesize 140
SQL> column table_name format a20
SQL> column tablespace_name format a20
SQL> column partition_name format a20
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
LINEITEM            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.

```

Repeat this step on all of the shards in your configuration.

12. Connect to the shard catalog database and verify that the chunks are uniformly distributed.

```

$ sqlplus / as sysdba

SQL> set echo off
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

```

13. 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.

The following example shows querying the tables on a database shard as the `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.
```

14. Verify that the Data Guard Broker automatic Fast-Start Failover configuration was done.

```
$ ssh os_username@shard_host_1  
$ dgmgrl  
  
DGMGRL> connect sys/password  
Connected to "sh1"  
Connected as SYSDBA.  
DGMGRL> show configuration  
  
Configuration - sh1  
  
Protection Mode: MaxPerformance  
Members:  
  sh1 - Primary database  
  sh2 - (*) Physical standby database  
  
Fast-Start Failover: ENABLED  
  
Configuration Status:  
SUCCESS (status updated 15 seconds ago)  
  
DGMGRL> show database sh1  
  
Database - sh1  
  
Role: PRIMARY  
Intended State: TRANSPORT-ON  
Instance(s):  
  sh1  
  
Database Status:  
SUCCESS  
  
DGMGRL> show database sh2  
  
Database - sh2  
  
Role: PHYSICAL STANDBY  
Intended State: APPLY-ON  
Transport Lag: 0 seconds (computed 0 seconds ago)  
Apply Lag: 0 seconds (computed 0 seconds ago)  
Average Apply Rate: 2.00 KByte/s  
Real Time Query: ON  
Instance(s):  
  sh2
```

Database Status:
SUCCESS

DGMGRL> show fast_start failover

Fast-Start Failover: **ENABLED**

```

Threshold:          30 seconds
Target:             sh2
Observer:           shard_director_host
Lag Limit:          30 seconds
Shutdown Primary:  TRUE
Auto-reinstate:    TRUE
Observer Reconnect: (none)
Observer Override:  FALSE
    
```

Configurable Failover Conditions

```

Health Conditions:
  Corrupted Controlfile      YES
  Corrupted Dictionary      YES
  Inaccessible Logfile      NO
  Stuck Archiver            NO
  Datafile Write Errors     YES
    
```

```

Oracle Error Conditions:
(none)
    
```

15. Locate the Fast-Start Failover observers.

Connect to the shard catalog database and run the following commands:

```
$ sqlplus / as sysdba
```

```
SQL> SELECT observer_state FROM gsmadmin_internal.broker_configs;
```

```
OBSERVER_STATE
```

```

-----
-----
GSM server SHARDDIRECTOR2. Observer started.
Log files at '/u01/app/oracle/product/18.0.0/gsmhome_1/network/
admin/
gsm_observer_1.log'.
    
```

```

GSM server SHARDDIRECTOR2. Observer started.
Log files at '/u01/app/oracle/product/18.0.0/gsmhome_1/
network.admin/
gsm_observer_2.log'.
    
```



See Also:

Oracle Database Global Data Services Concepts and Administration Guide
 for information about GDSCTL command usage

System-Managed SDB Demo Application

The system-managed sharded database (SDB) demo application simulates the workload of an online retail store. Use it to validate the setup of any system-managed (automatic sharding) SDB configuration. The demo application also provides a practical example of sharding concepts for administrators and developers new to database sharding.

The demo application assumes that a system-managed SDB environment was already created along with the CUSTOMER table-family. The environment may have any number of chunks and shards (database nodes). When run, the application will first populate the Products table and then start a one-hour workload that can be paused at any time by the administrator. The workload includes four types of transactions: create a customer order, lookup the list of orders, create a new product, and multi-shard query with report generation. All aspects of a sharded database configuration are exercised.

You can download the demo application, along with a README file that describes how to run and monitor it, from [My Oracle Support Document 2184500.1](#).

Creating and Deploying a User-Defined SDB

The following topics describe the tasks for creating and deploying a user-defined SDB.

- [Deploying a User-Defined SDB](#)
The following procedure describes how to deploy a user-defined sharded database using the ADD SHARD command and an Oracle Active Data Guard high availability solution.
- [Creating a Schema for a User-Defined SDB](#)
Create the schema user, tablespace set, sharded tables, and duplicated tables for the SDB. Verify that the DDLs are propagated to all of the shards, and, while connected to the shards, verify the automatic Data Guard Broker configuration with Fast-Start Failover.

Deploying a User-Defined SDB

The following procedure describes how to deploy a user-defined sharded database using the ADD SHARD command and an Oracle Active Data Guard high availability solution.

User-defined sharding allows the user to map data to shards. For more conceptual information about the user-defined sharding method, see [User-Defined Sharding](#).

1. Because this procedure describes using the ADD SHARD method as detailed in [Introduction to Sharded Database Deployment](#), you must first create the databases to be used as shards on their respective hosts.

The shard databases must have the following characteristics:

- They must have an associated TNS Listener on each host
- The GSMUSER account must be unlocked with a known password
- SYSDBG and SYSBACKUP privileges must be granted to GSMUSER
- The primary and standby databases must be configured as such
- Redo apply should be set up between the corresponding primary and standby databases
- Flashback and force logging should be enabled

- The `compatible` parameter must be set to at least 12.2.0
- A server parameter file (SPFILE) must be in use
- A `DATA_PUMP_DIR` directory object must be created in each database and must point to a valid directory

You must then validate that a database is correctly set up for sharding. Execute the following against each database before adding it to the configuration.

```
SQL> set serveroutput on
SQL> execute DBMS_GSM_FIX.validateShard
```

Screen output will include INFO, WARNING, and ERROR information that needs to be analyzed for any issues. All WARNING and ERROR messages must be resolved. Re-run `validateShard()` after making changes to confirm the configuration.

2. Connect to the shard director host.

```
$ ssh os_user@shard_director1_host
```

3. 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
```

4. Add shardspaces to the sharded database configuration for each customized grouping of shards your business case requires.

The shardspaces contain the primary shard database and one or more active standbys.

In this example the shardspaces are named `shspace1` and `shspace2`. You can choose your own names.

```
GDSCTL> add shardspace -shardspace shspace1 -protectmode
maxavailability
GDSCTL> add shardspace -shardspace shspace2 -protectmode
maxavailability
```

5. Add each shard's host address to the valid node checking for registration (VNCR) list in the catalog, then add the shard in either the primary or standby shardgroup, as shown in the following example.

 **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 may 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.

The following example shows how to add four shards using the `ADD SHARD` command, the first two of which are primary and active standby shards in shardspace `shspace1`, and the second two are primary and active standbys in shardspace `shspace2`. Note that the primaries are given a region of `region1` and the standbys are given `region2`.

```
GDSCTL> add invitednode shard_host_1
GDSCTL> add shard -connect shard_host_1:1521/shard_database_name
                 -shardspace shspace1 -deploy_as primary -pwd GSMUSER_password -region
                 region1
```

```
GDSCTL> add invitednode shard_host_2
GDSCTL> add shard -connect shard_host_2:1521/shard_database_name
                 -shardspace shspace1 -deploy_as active_standby -pwd GSMUSER_password
                 -region region2
```

```
GDSCTL> add invitednode shard_host_3
GDSCTL> add shard -connect shard_host_3:1521/shard_database_name
                 -shardspace shspace2 -deploy_as primary -pwd GSMUSER_password -region
                 region1
```

```
GDSCTL> add invitednode shard_host_4
GDSCTL> add shard -connect shard_host_4:1521/shard_database_name
                 -shardspace shspace2 -deploy_as active_standby -pwd GSMUSER_password
                 -region region2
```

If the shard database to be added is a pluggable database (PDB), you must use the `-cdb` option to `ADD SHARD` to specify which container database (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 that in Oracle Database 18c, only one PDB in each CDB is allowed to be a shard.

6. Check the configuration from a shard director.

```
GDSCTL> config
Regions
-----
region1
region2

GSMs
-----
sharddirector1
sharddirector2

Sharded Database
-----
udef_shpool

Databases
-----
sh1
sh2
sh3
sh4

Shard spaces
-----
shspace1
shspace2

Services
-----

GDSCTL pending requests
-----
Command          Object          Status
-----          -
Global properties
-----
Name: oradbcloud
Master GSM: sharddirector1
DDL sequence #: 0

GDSCTL> config vnchr
Name                Group ID
-----
shard_host_1
```

```
shard_host_2
shard_host_3
shard_host_4
shard_catalog_host_IP
```

```
GDSCTL> config shard
Name      Shard space      Status  State  Region  Availability
-----  -
sh1       shspace1         U      none  region1 -
sh2       shspace1         U      none  region2 -
sh3       shspace2         U      none  region1 -
sh4       shspace2         U      none  region2 -
```

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

The `DEPLOY` command takes some time to run, approximately 15 to 30 minutes.

```
GDSCTL> deploy
```

Once the primary and standby shards are built, the `DEPLOY` command configures the Data Guard Broker with Fast-Start Failover enabled. The Fast-Start Failover observers are automatically started on the standby group's shard director (sharddirector2 in this example).

8. Verify that all of the shards are deployed.

```
GDSCTL> config shard
Name      Shard space      Status  State  Region  Availability
-----  -
sh1       shspace1         Ok      Deployed  region1  ONLINE
sh2       shspace1         Ok      Deployed  region2  READ_ONLY
sh3       shspace2         Ok      Deployed  region1  ONLINE
sh4       shspace2         Ok      Deployed  region2  READ_ONLY
```

9. Verify that all of the shards are registered.

```
GDSCTL> databases
Database: "sh1" Registered: Y State: Ok ONS: N. Role: PRIMARY
Instances: 1 Region: region1
Registered instances:
  udef_shpool%1
Database: "sh2" Registered: Y State: Ok ONS: N. Role: PH_STNDBY
Instances: 1 Region: region2
Registered instances:
  udef_shpool%11
Database: "sh3" Registered: Y State: Ok ONS: N. Role: PRIMARY
Instances: 1 Region: region1
Registered instances:
  udef_shpool%21
Database: "sh4" Registered: Y State: Ok ONS: N. Role: PH_STNDBY
Instances: 1 Region: region2
```

```
Registered instances:
  udef_shpool%31
```

10. Check the configuration of a shard.

```
GDSCTL> config shard -shard sh1
Name: sh1
Shard space: shspace1
Status: Ok
State: Deployed
Region: region1
Connection string: shard_host_1:1521/sh1: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
-----            -
```

11. Add a global service that runs on all of the primary shards.

The `oltp_rw_srvc` global service is a global data service that a client can use to connect to the sharded database. The `oltp_rw_srvc` service runs the transactions on the primary shards. Likewise, the `oltp_ro_srvc` global service is created to run read-only workload on the standby shards.

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

```
GDSCTL> config service
```

```
Name                Network name        Pool
Started Preferred all
-----            -
-----            -
oltp_rw_srvc      oltp_rw_srvc.cust_sdb.oradbcloud udef_shpool
No                Yes
```

12. Start the `oltp_rw_srvc` global service.

```
GDSCTL> start service -service oltp_rw_srvc
```

```
GDSCTL> status service
Service "oltp_rw_srvc.cust_sdb.oradbcloud" has 2 instance(s).
Affinity: ANYWHERE
  Instance "udef_shpool%1", name: "sh1", db: "sh1", region:
  "region1",
  status: ready.
```



```
Instance "udef_shpool%21", name: "sh3", db: "sh3", region: "region1",
status: ready.
```

13. Add a global service for the read-only workload to run on the standby shards.

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

```
GDSCTL> config service
```

Name	Network name	Pool	Started
Preferred all			
----	-----	----	-----

oltp_rw_srvc	oltp_rw_srvc.cust_sdb.oradbcloud	cust_sdb	Yes Yes
oltp_ro_srvc	oltp_ro_srvc.cust_sdb.oradbcloud	cust_sdb	No Yes

14. Start the read-only service, and verify the status of the global services.

```
GDSCTL> start service -service oltp_ro_srvc
```

```
GDSCTL> status service
```

```
Service "oltp_ro_srvc.cust_sdb.oradbcloud" has 2 instance(s). Affinity:
ANYWHERE
  Instance "udef_shpool%11", name: "sh2", db: "sh2", region: "region2",
status: ready.
  Instance "udef_shpool%31", name: "sh4", db: "sh4", region: "region2",
status: ready.
Service "oltp_rw_srvc.cust_sdb.oradbcloud" has 2 instance(s). Affinity:
ANYWHERE
  Instance "udef_shpool%1", name: "sh1", db: "sh1", region: "region1",
status: ready.
  Instance "udef_shpool%21", name: "sh3", db: "sh3", region: "region1",
status: ready.
```

See Also:

Oracle Database Global Data Services Concepts and Administration Guide for more information about GDSCTL command usage

Creating a Schema for a User-Defined SDB

Create the schema user, tablespace set, sharded tables, and duplicated tables for the SDB. Verify that the DDLs are propagated to all of the shards, and, while connected to the shards, verify the automatic Data Guard Broker configuration with Fast-Start Failover.

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 c1_tsp DATAFILE SIZE 100M autoextend on next
10M maxsize
unlimited extent management local segment space management auto in
shardspace shspace1;
```

```
SQL> CREATE TABLESPACE c2_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 Lineltems 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;
```

8. From the shard director host, verify that there were no failures during the creation of the tablespaces.

```
GDSCTL> show ddl
id      DDL Text                                     Failed shards
--      -
3       grant create table, create procedure,...
4       grant unlimited tablespace to app_schema
5       grant select_catalog_role to app_schema
6       create tablespace c1_tsp DATAFILE SIZ...
7       Create tablespace c2_tsp DATAFILE SIZ...
8       CREATE SHARDED TABLE Customers ( Cu...
9       CREATE SHARDED TABLE Orders ( Order...
10      CREATE SHARDED TABLE LineItems ( Or...
11      create tablespace products_tsp datafi...
12      CREATE MATERIALIZED VIEW "APP_SCHEMA"...
```

 **Note:**

The `show ddl` command output might be truncated. You can run `SELECT ddl_text FROM gsmadmin_internal.ddl_requests` on the catalog to see the full text of the statements.

9. Verify that there were no DDL errors on each of the shards.

Run the `config shard` and `config chunks` commands on each shard in your configuration.

```
GDSCTL> config shard -shard sh1
```

```
Name: sh1
Shard space: shspace1
Status: Ok
State: Deployed
Region: region1
Connection string: shard_host_1:1521/sh1: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
Rack:
```

```
Supported services
```

```
-----
```

Name	Preferred	Status
oltp_ro_srvc	Yes	Enabled
oltp_rw_srvc	Yes	Enabled

```
GDSCTL> config chunks
```

```
Chunks
```

```
-----
```

Database	From	To
sh1	1	1
sh2	1	1
sh3	2	2
sh4	2	2

- Verify that the tablespaces 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 is based on the number of chunks you specified in the `create shardcatalog` command.

The tablespace set with the first 6 chunks of the 12 that were specified in the shard catalog creation example, and the duplicated Products tablespace is shown in the following example.

```
$ sqlplus / as sysdba
```

```
SQL> select TABLESPACE_NAME, BYTES/1024/1024 MB
       from sys.dba_data_files
       order by tablespace_name;
```

```
TABLESPACE_NAME          MB
```

```

-----
C1_TSP                100
PRODUCTS_TSP         10
SYSAUX                722.1875
SYSEXT                39
SYSTEM                782.203125
SYS_SHARD_TS         100
UD1                   470
  
```

7 rows selected.

Repeat this step on all of the shards in your configuration.

11. Verify that the chunks and chunk tablespaces were created on all of the shards.

```

SQL> set linesize 140
SQL> column table_name format a20
SQL> column tablespace_name format a20
SQL> column partition_name format a20
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
CUSTOMERS	CK1	C1_TSP
ORDERS	CK1	C1_TSP
LINEITEMS	CK1	C1_TSP

Repeat this step on all of the shards in your configuration.

12. 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.

The following example shows querying the tables on a database shard as the app_schema user.

```

$ sqlplus app_schema/app_schema_password
Connected.
  
```

```

SQL> select table_name from user_tables;
  
```

TABLE_NAME
CUSTOMERS

```
ORDERS
LINEITEMS
PRODUCTS
USLOG$_PRODUCTS
```

13. Verify that the Data Guard Broker automatic Fast-Start Failover configuration was done.

```
$ ssh os_username@shard_host_1
$ dgmgrl

DGMGRL> connect sys/password
Connected to "sh1"
Connected as SYSDBA.
DGMGRL> show configuration

Configuration - sh1

Protection Mode: MaxPerformance
Members:
  sh1 - Primary database
  sh2 - (*) Physical standby database

Fast-Start Failover: ENABLED

Configuration Status:
SUCCESS (status updated 15 seconds ago)

DGMGRL> show database sh1

Database - sh1

Role:                PRIMARY
Intended State:      TRANSPORT-ON
Instance(s):
  sh1

Database Status:
SUCCESS

DGMGRL> show database sh2

Database - sh2

Role:                PHYSICAL STANDBY
Intended State:      APPLY-ON
Transport Lag:       0 seconds (computed 0 seconds ago)
Apply Lag:           0 seconds (computed 0 seconds ago)
Average Apply Rate: 2.00 KByte/s
Real Time Query:    ON
Instance(s):
  sh2

Database Status:
SUCCESS
```

```
DGMGRL> show fast_start failover

Fast-Start Failover: ENABLED

Threshold:          30 seconds
Target:             sh2
Observer:           shard_director_host
Lag Limit:          30 seconds
Shutdown Primary:  TRUE
Auto-reinstate:     TRUE
Observer Reconnect: (none)
Observer Override:  FALSE

Configurable Failover Conditions
Health Conditions:
  Corrupted Controlfile      YES
  Corrupted Dictionary      YES
  Inaccessible Logfile      NO
  Stuck Archiver            NO
  Datafile Write Errors     YES

Oracle Error Conditions:
(none)
```

14. Locate the Fast-Start Failover observers.

Connect to the shard catalog database and run the following commands:

```
$ ssh oracle@shard6

$ ps -ef |grep dgmgrl
oracle   8210  8089  0 22:18 pts/4    00:00:00 grep dgmgrl
oracle  20189    1  0 02:57 ?          00:02:40 dgmgrl -
delete_script
@/u01/app/oracle/product/18.0.0/gsmhome_1/network/admin/
gsm_observer_1.cfg
oracle  20193    1  0 02:57 ?          00:02:43 dgmgrl -
delete_script
@/u01/app/oracle/product/18.0.0/gsmhome_1/network/admin/
gsm_observer_2.cfg
```

See Also:

Oracle Database Global Data Services Concepts and Administration Guide
for information about GDSCTL command usage

Creating and Deploying a Composite SDB

To deploy a composite SDB you must install the required Oracle Sharding software components, configure the objects for a composite SDB, and create the schema.

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.

The following topics describe the tasks for deploying a composite SDB.

- [Deploying a Composite SDB](#)
To deploy a composite SDB you create shardgroups and shards, execute the `DEPLOY` command, and create role-based global services.
- [Creating a Schema for a Composite SDB](#)
Create the schema user, tablespace set, sharded tables, and duplicated tables for the SDB. Verify that the DDLs are propagated to all of the shards, and, while connected to the shards, verify the automatic Data Guard Broker configuration with Fast-Start Failover.

Deploying a Composite SDB

To deploy a composite SDB you create shardgroups and shards, execute the `DEPLOY` command, and create role-based global services.

The examples used in this deployment procedure are based on a global distribution scenario where separate shardspaces and shardgroups are created for America and Europe.

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
GDSTCL> set gsm -gsm sharddirector1
GDSTCL> connect mysdbadmin/mysdbadmin_password
```

3. Add shardspaces and shardgroups for each customized grouping of shards your business case requires.

In this example the shardspaces and shardgroups are created for the America and Europe customers. You can choose your own names.

```
GDSTCL> add shardspace -shardspace cust_america
GDSTCL> add shardgroup -shardspace cust_america -shardgroup america_shgrp1
    -deploy_as primary -region region1

GDSTCL> add shardspace -shardspace cust_europe
GDSTCL> add shardgroup -shardspace cust_europe -shardgroup europe_shgrp1
    -deploy_as primary -region region2
```

 **Note:**

For production deployments, additional shardgroups must be created for high availability using the `add shardgroup` command

4. Verify the shardspace and shardgroup configurations.

```
GDSCTL> config shardspace
SHARDSPACE                               Chunks
-----                               -
cust_america                             12
cust_europe                              12
shardspaceora                            12

GDSCTL>config shardgroup
Shard Group          Chunks Region          SHARDSPACE
-----
america_shgrp1      12    region1         cust_america
europe_shgrp1       12    region2         cust_europe
```

5. Verify the sharded database configuration.

```
GDSCTL> config

Regions
-----
region1
region2

GSMs
-----
sharddirector1
sharddirector2

Sharded Database
-----
cust_sdb_comp

Databases
-----

Shard Groups
-----
america_shgrp1
europe_shgrp1

Shard spaces
-----
cust_america
cust_europe
shardspaceora

Services
```

```

-----
GDSCTL pending requests
-----
Command                Object                Status
-----                -
Global properties
-----
Name: oradbcloud
Master GSM: sharddirector1
DDL sequence #: 0

```

6. Add each shard's host address to the valid node checking for registration (VNCR) list in the catalog, then create the shard in either the primary or standby shardgroup, as shown in the following example.

 **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 may 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.

The example shows how to create four shards, two of which are in the America shardgroup and two in the Europe shardgroup. The `os_credential` is the operating system credential you created on each host.

While creating the shards, you can also set the SYS password in the `CREATE SHARD` using `-sys_password` as shown in the following example. This sets the SYS password after the shards are created when running `DEPLOY`. There are other optional parameters for `CREATE SHARD` that allow you to customize the database parameters, storage and file locations,

listener port numbers, and so on, which are documented in the *Oracle Database Global Data Services Concepts and Administration Guide* appendices.

```
GDSCTL> add invitednode shard_host_1
GDSCTL> create shard -shardgroup america_shgrp1 -destination
shard_host_1
-credential os_credential-sys_password

GDSCTL> add invitednode shard_host_2
GDSCTL> create shard -shardgroup america_shgrp1 -destination
shard_host_2
-credential os_credential-sys_password

GDSCTL> add invitednode shard_host_3
GDSCTL> create shard -shardgroup europe_shgrp1 -destination
shard_host_3
-credential os_credential-sys_password

GDSCTL> add invitednode shard_host_4
GDSCTL> create shard -shardgroup europe_shgrp1 -destination
shard_host_4
-credential os_credential-sys_password
```

If you are using the `ADD SHARD` method described in [Introduction to Sharded Database Deployment](#), use the following command instead of the `CREATE SHARD` commands in the example above. If the shard database to be added is a pluggable database (PDB), you must use the `-cdb` option to `ADD SHARD` to specify which container database (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 that in Oracle Database 18c, only one PDB in each CDB is allowed to be a shard.

```
GDSCTL> add shard -shardgroup america_shgrp1
-connect shard_host:TNS_listener_port/shard_database_name
-pwd GSMUSER_password
```

7. Check the configuration from a shard director.

Note that the shard names, sh1, sh2, sh3, and sh4, are system generated shard names.

```
GDSCTL> config shard
Name          Shard Group          Status   State   Region
Availability
-----
-----
sh1           america_shgrp1       U        none   region1
-
sh2           america_shgrp1       U        none   region1
-
sh3           europe_shgrp1        U        none   region2
-
sh4           europe_shgrp1        U        none   region2
```

```
-
GDSCTL> config vncr
Name      Group ID
-----  -
shard_host_1
shard_host_2
shard_host_3
shard_host_4
shard_catalog_host_IP
```

8. Run the `DEPLOY` command to create the shards.

```
GDSCTL> deploy
```

The `DEPLOY` command takes some time to run, approximately 15 to 30 minutes. The `DEPLOY` command creates the shards using DBCA.

9. Verify that all of the shards are deployed.

```
GDSCTL> config shard
Name      Shard Group      Status   State   Region
Availability
-----  -
sh1      america_shgrp1    Ok      Deployed  region1  ONLINE
sh2      america_shgrp1    Ok      Deployed  region1  ONLINE
sh3      europe_shgrp1     Ok      Deployed  region2  ONLINE
sh4      europe_shgrp1     Ok      Deployed  region2  ONLINE
```

10. Verify that all of the shards are registered.

```
GDSCTL> databases
Database: "sh1" Registered: Y State: Ok ONS: N. Role: PRIMARY Instances: 1
Region: region1
Registered instances:
cust_sdb_comp%1
Database: "sh2" Registered: Y State: Ok ONS: N. Role: PRIMARY Instances: 1
Region: region1
Registered instances:
cust_sdb_comp%11
Database: "sh3" Registered: Y State: Ok ONS: N. Role: PRIMARY Instances: 1
Region: region2
Registered instances:
cust_sdb_comp%21
Database: "sh4" Registered: Y State: Ok ONS: N. Role: PRIMARY Instances: 1
Region: region2
Registered instances:
cust_sdb_comp%31
```

11. Check the configuration of a shard.

```
GDSCTL> config shard -shard sh1
```

```
Name: sh1
Shard Group: america_shgrp1
Status: Ok
State: Deployed
Region: region1
Connection string: shard1:1521/sh1: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
----
-----
```

12. Add a global service that runs on all of the primary shards.

The `oltp_rw_srvc` global service is the global data services listener that helps route a connection from the client to the actual database. The `oltp_rw_srvc` service runs the transactions on the primary shards.

```
GDSCTL> add service -service oltp_rw_srvc
GDSCTL> config service
```

Name	Network name	Pool	Started
Preferred all			
-----	-----	----	-----
oltp_rw_srvc	oltp_rw_srvc.cust_sdb_comp.or	cust_sdb_comp	No
Yes	adbcloud		

13. Start the `oltp_rw_srvc` global service.

```
GDSCTL> start service -service oltp_rw_srvc

GDSCTL> status service
Service "oltp_rw_srvc.cust_sdb_comp.oradbcloud" has 4 instance(s).
Affinity: ANYWHERE
Instance "cust_sdb_comp%1", name: "sh1", db: "sh1", region:
"region1",
status: ready.
Instance "cust_sdb_comp%11", name: "sh2", db: "sh2", region:
"region1",
status: ready.
Instance "cust_sdb_comp%21", name: "sh3", db: "sh3", region:
```

```
"region2",
status: ready.
  Instance "cust_sdb_comp%31", name: "sh4", db: "sh4", region: "region2",
status: ready.
```



See Also:

Oracle Database Global Data Services Concepts and Administration Guide

Creating a Schema for a Composite SDB

Create the schema user, tablespace set, sharded tables, and duplicated tables for the SDB. Verify that the DDLs are propagated to all of the shards, and, while connected to the shards, verify the automatic Data Guard Broker configuration with Fast-Start Failover.

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 );
```

Specifying the shardspace is optional when creating the tablespace set. If the shardspace is not specified in the command, the default shardspace is used.

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 subpartition 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;

```

9. From the shard director host, verify that there were no failures during the creation of the tablespaces.

```

GDSCTL> show ddl
id      DDL Text                                     Failed shards
--      -
11      CREATE TABLESPACE SET TSP_SET_2 in s...
12      CREATE TABLESPACE products_tsp datafi...
13      CREATE SHARDED TABLE Customers ( Cu...
14      CREATE SEQUENCE Orders_Seq;
15      CREATE SHARDED TABLE Orders ( Order...
16      CREATE SHARDED TABLE LineItems ( Or...
17      create database link "PRODUCTSDBLINK@...
18      CREATE MATERIALIZED VIEW "PRODUCTS" ...
19      CREATE OR REPLACE FUNCTION PasswCreat...
20      CREATE OR REPLACE FUNCTION PasswCheck...

```

10. Verify that there were no DDL errors on each of the shards.

Run the `config shard` and `config chunks` commands on each shard in your configuration.

```

GDSCTL> config shard -shard sh1

Name: sh1
Shard Group: america_shgrp1
Status: Ok
State: Deployed
Region: region1
Connection string: shard1:1521/sh1: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_rw_srvc                             Yes
Enabled
```

GDSCTL> config chunks

Chunks

```
-----
Database                                From      To
-----
sh1                                     1         6
sh2                                     7         12
sh3                                     1         6
sh4                                     7         12
```

11. 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 is based on the number of chunks you specified in the `create shardcatalog` command.

The tablespace set with the first 6 chunks of the 12 that were specified in the shard catalog creation example, and the duplicated Products tablespace is shown in the following example on the `shard_host_1`.

```
$ 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
-----
TSP_SET_2                100
UNDOTBS1                 110
USERS                    5
```

14 rows selected.

Repeat this step on all of the shards in your configuration.

12. Verify that the chunks and chunk tablespaces were created on all of the shards.

```
SQL> set linesize 140
SQL> column table_name format a20
SQL> column tablespace_name format a20
SQL> column partition_name format a20
SQL> show parameter db_unique_name
NAME                TYPE        VALUE
-----
db_unique_name      string      sh2
```

```
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
LINEITEMS	CUSTOMERS_P7	C007TSP_SET_1
CUSTOMERS	CUSTOMERS_P7	C007TSP_SET_1
ORDERS	CUSTOMERS_P7	C007TSP_SET_1
CUSTOMERS	CUSTOMERS_P8	C008TSP_SET_1
LINEITEMS	CUSTOMERS_P8	C008TSP_SET_1
ORDERS	CUSTOMERS_P8	C008TSP_SET_1
LINEITEMS	CUSTOMERS_P9	C009TSP_SET_1
CUSTOMERS	CUSTOMERS_P9	C009TSP_SET_1
ORDERS	CUSTOMERS_P9	C009TSP_SET_1
CUSTOMERS	CUSTOMERS_P10	C00ATSP_SET_1
LINEITEMS	CUSTOMERS_P10	C00ATSP_SET_1
ORDERS	CUSTOMERS_P10	C00ATSP_SET_1
CUSTOMERS	CUSTOMERS_P11	C00BTSP_SET_1
LINEITEMS	CUSTOMERS_P11	C00BTSP_SET_1
ORDERS	CUSTOMERS_P11	C00BTSP_SET_1
CUSTOMERS	CUSTOMERS_P12	C00CTSP_SET_1
LINEITEMS	CUSTOMERS_P12	C00CTSP_SET_1
ORDERS	CUSTOMERS_P12	C00CTSP_SET_1

18 rows selected.

Repeat this step on all of the shards in your configuration.

13. Connect to the shard catalog database and verify that the chunks are uniformly distributed.

```
$ sqlplus / as sysdba
```

```
SQL> set echo off
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;
```

SHARD	NUMBER_OF_CHUNKS
sh1	6
sh2	6
sh3	6
sh4	6

14. 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.

The following example shows querying the tables on a database shard as the `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.

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.
- [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.

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`.

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));
```


9

Migrating Data to a Sharded Database

You can migrate data from a non-sharded database to an Oracle Sharding sharded database using the methods described here.

These data loading methods assume that you are using a non-sharded Oracle database at the time you want to migrate to a sharded one. These methods proposed also apply to migrating data from other database systems, as well as the first time database users.

- [About Migrating Data to a Sharded Database](#)
After Oracle Sharding software installation, and sharded database configuration and creation, you can migrate your data to a sharded database.
- [General Guidelines for Loading Data into a Sharded Database](#)
Transitioning from non-sharded to a sharded database involves moving the data from non-sharded tables to sharded and duplicated tables. Moving data from non-sharded tables to duplicated tables does not introduce any complexity, but moving data from non-sharded tables to sharded tables requires special attention.
- [Migrating the Schema](#)
Before the existing database can be migrated to the sharded database, you must decide how to organize the sharded database, including the number of shards and the replication strategy, and you must decide which tables in the application are sharded and which tables are duplicated tables. For the sharded tables, you must decide the sharding method as well as the parent-child relationships between the sharded tables in the table family.
- [Preparing the Source Database](#)
To make the transition to a sharded database schema smoother, you can modify the source, non-sharded database so that it matches the table definitions in the target sharded database.
- [Preparing the Target Sharded Database](#)
Before you start data migration to the sharded database, you must create the sharded database schema according to your design.
- [Migrating Your Data](#)
After you create the target sharded database with a single shard, or multiple shards, and you have your sharded and duplicated tables defined, you can start migrating your data into the sharded database.
- [Migrating Your Application](#)
The sharded database operating environment empowers applications with direct access to shards. This feature provides true linear scalability, but it comes with a small price—a slight change to the application code.

About Migrating Data to a Sharded Database

After Oracle Sharding software installation, and sharded database configuration and creation, you can migrate your data to a sharded database.

The following are the high level steps to migrate to a sharded database environment.

1. Design and create the sharded database schema.
2. Migrate the data.
3. Migrate the application.

 **See Also:**

[Application Suitability for Sharding](#) to familiarize yourself with the constraints of migration to a sharded database applications.

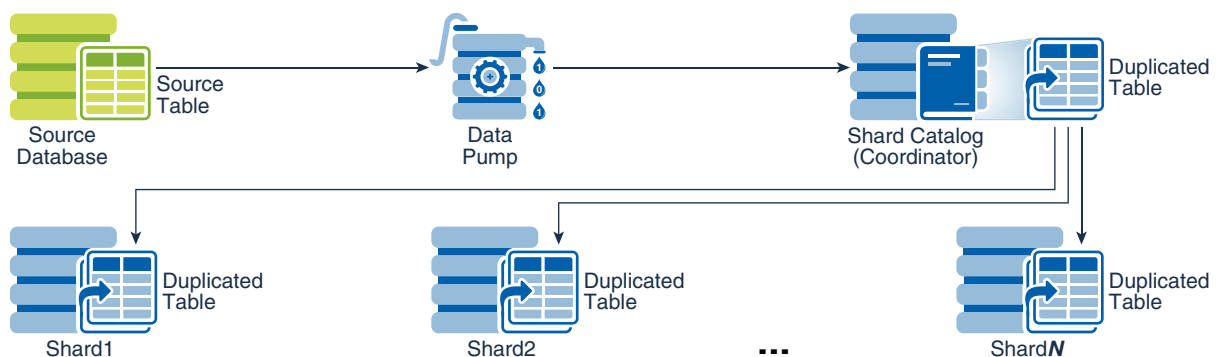
General Guidelines for Loading Data into a Sharded Database

Transitioning from non-sharded to a sharded database involves moving the data from non-sharded tables to sharded and duplicated tables. Moving data from non-sharded tables to duplicated tables does not introduce any complexity, but moving data from non-sharded tables to sharded tables requires special attention.

Loading Data into Duplicated Tables

Loading data into a duplicated table can be accomplished using any existing database tools: Data Pump, SQL Loader, or plain SQL. The data must be loaded using the shard catalog (coordinator) database node. In other words, the entire contents of the duplicated table is contained in the shard catalog database. 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 non-sharded table.

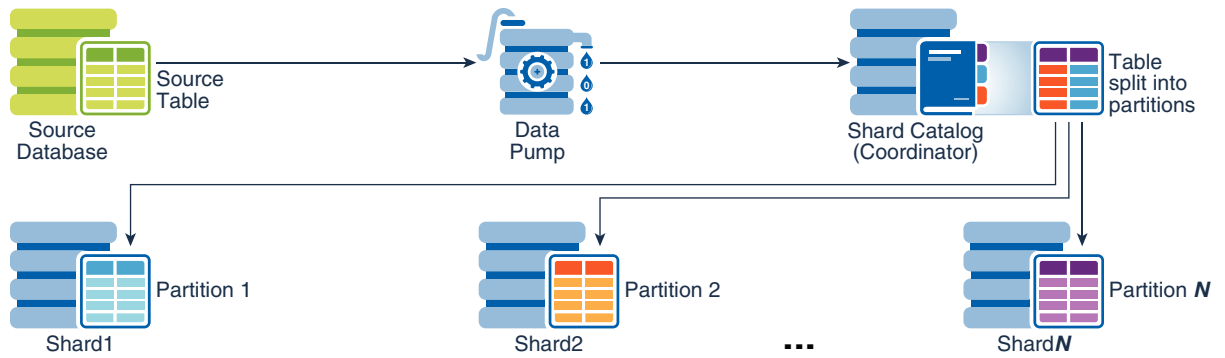
Figure 9-1 Loading Duplicated Tables



Loading Data into Sharded Tables

When loading a sharded table, each database shard accommodates a distinct subset (shard) of the entire data set, so you must split (shuffle) the data before loading each subset to a particular shard.

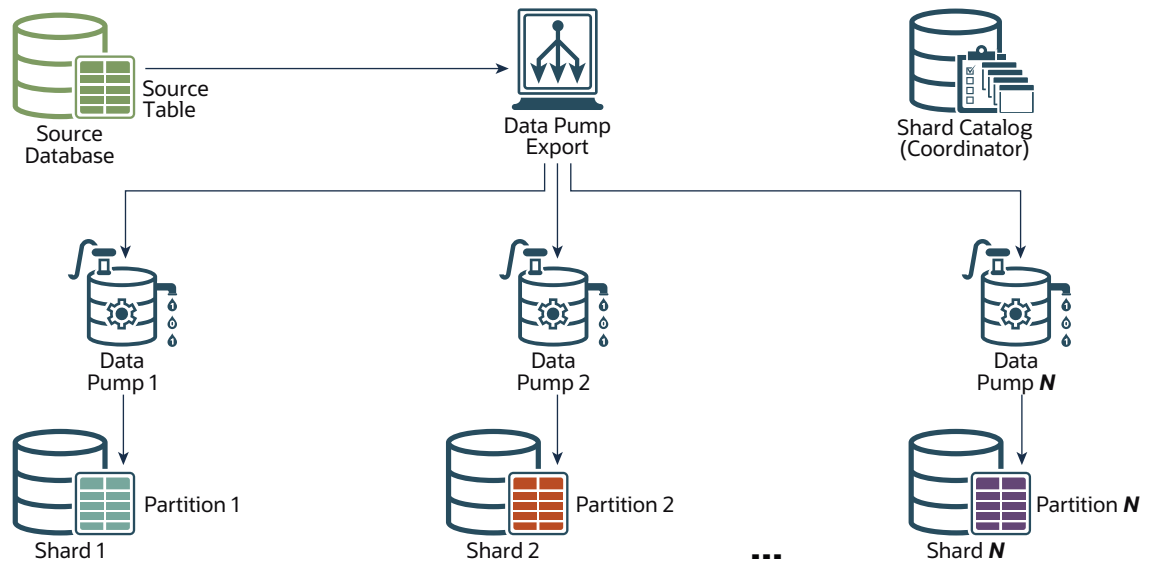
Figure 9-2 Loading Sharded Tables Using the Shard Catalog



You must use the Oracle Data Pump utility to load the data across database shards in subsets. Consider the following two options:

- Load the data through the sharding coordinator (catalog) node, as illustrated above.
- Load the data directly to the database shards, as illustrated below.

Figure 9-3 Loading Sharded Tables Directly to the Database Shards



Loading the data into a sharded database using the sharding coordinator is slower than loading the entire data set into a non-sharded table, because of the splitting logic running on the sharding coordinator (catalog) node and additional overhead of pushing the data to the shards.

Loading the data directly into the database shards is much faster, because each shard is loaded separately. That is, by running Data Pump on each shard, you can complete the data loading operation within the period of time needed to load the shard with the maximum subset of the entire data set. On average, the loading time can be approximated as the time needed to load the entire data set into a non-sharded database, divided by the number of shards in the sharded database.

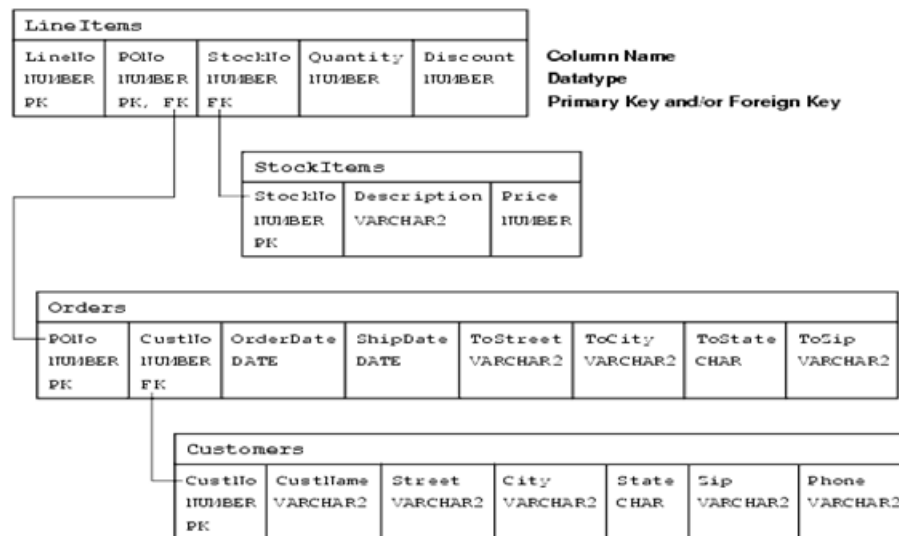
Rather than relying on the Oracle Data Pump utility to split the load data set in distinct subsets, you can use an open source shard splitting library that integrates the splitting (shuffling) logic into your application. The shard splitting library source code, as well as sample use, is available in the Oracle Sharding Tools Library at <https://github.com/oracle/db-sharding/>. Based on this shard splitting library, Oracle develops a generic streaming load library for the use in the Oracle Cloud. At this time the streaming load library is only available upon request.

Migrating the Schema

Before the existing database can be migrated to the sharded database, you must decide how to organize the sharded database, including the number of shards and the replication strategy, and you must decide which tables in the application are sharded and which tables are duplicated tables. For the sharded tables, you must decide the sharding method as well as the parent-child relationships between the sharded tables in the table family.

The schema migration to a sharded database environment is illustrated using a sample application, which is defined over a data model and imposed constraints. We analyze how migration to a sharded database affects the application using sample program code. The following figure shows the sample application data model

Figure 9-4



The data model comprises four tables: Customers, Orders, StockItems, and LineItems, and the 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 sample application database 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 StockItems (  
  StockNo     NUMBER(4) PRIMARY KEY  
  Description  VARCHAR2(20),  
  Price       NUMBER(6,2)  
);
```

```
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)  
);
```

Sharding Key

Sharding is a database scaling technique based on horizontal partitioning across multiple independent Oracle databases. The database requests are routed to appropriate shard database based on the value of sharding key column. The sharding design goal is to select a sharding key which maximizes single shard operations and minimizes or eliminates cross shard operations.

Based on the primary key to foreign key functional dependencies identified in the sample application data model, the following table family is formed.

- Customers – parent table
- Orders – child table
- Lineitems – grandchild table

The remaining StockItems table is simply a lookup table mapping stock item number to stock item description and price (StockNo -> (Description, Price)).

Sharded database definitions require the following table DDL statements for members of the table family using reference partitioning, plus the additional DDL statement defining the StockItems lookup table:

```
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 DUPLICATED TABLE StockItems (
  StockNo     NUMBER(4) PRIMARY KEY
  Description  VARCHAR2(20),
  Price       NUMBER(6,2)
);

CREATE SHARDED TABLE LineItems (
  LineNo      NUMBER(2) NOT NULL,
  PoNo        NUMBER(5) NOT NULL,
  StockNo     NUMBER(4) REFERENCES StockItems,
  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)  
;
```

After comparing the sharded database DDL, shown above, to the original (non-sharded) database tables, you notice the following structural table changes (appearing in large bold italic).

- The `CREATE TABLE` statement for tables in the table family includes the additional `SHARDED` keyword.
- The `CREATE TABLE` statement for the lookup table includes additional keyword, `DUPLICATED`.
- All tables in the table family contain the sharding key column, `CustNo`, as the leading column of the primary key. This is the sharding-specific de-normalization, which expands a composite primary key at every level of the table family hierarchy to include the immediate parent key, also known as the level key, as the leading component.
- Sharded tables are `PARTITIONED BY` the sharding key column. In this particular case, the root table of the family is partitioned by `CONSISTENT HASH`. This partitioning schema propagates to lower hierarchy levels `BY REFERENCE` (reference partitioning). The data partitioning by `CONSISTENT HASH` is called the system-managed sharding method (as opposed to user-defined sharding).
- In system-managed sharding, tablespace sets are defined for sharded tables. The first set of tablespaces is used for `SHARDED` tables. A tablespace set is used in a sharded database as a logical storage unit for one or more sharded tables and indexes. A tablespace set consists of multiple tablespaces distributed across shards in a shardspace. The database automatically creates the tablespaces in a tablespace set. The number of tablespaces is determined automatically and is equal to the number of chunks in the corresponding shardspace.

```
CREATE TABLESPACE SET tbs1; for the sharded tables
```

In our example, `Customers`, `Orders`, and `LineItems` are placed in tablespace set `tbs1`. That means that corresponding partitions of the three tables in the table family are stored in the same tablespace set, `tbs1` (partition by reference). However, it is possible to specify separate tablespace sets for each table.

Related Topics

- [Partitions, Tablespaces, and Chunks](#)
Distribution of partitions across shards is achieved by creating partitions in tablespaces that reside on different shards.

Preparing the Source Database

To make the transition to a sharded database schema smoother, you can modify the source, non-sharded database so that it matches the table definitions in the target sharded database.

Ideally, the table definitions (table name, column names, and their data types) in the target sharded database and the table definitions in the source database would be exactly the same. However, as part of the transition to a sharded database, you might need to modify the table definitions for use in the sharded database. If that is the case, you can modify the source, non-sharded database so that it matches the table definitions in the new sharded database. Depending on the extent of changes, this can also require changes to the

application code. By modifying the source database schema, so that it matches the target sharded database schema ahead of the migration process, you provide conditions for uninterrupted transition from original non-sharded to the new sharded database. These preparations are prerequisites for minimum downtime, if downtime would be incurred at all. Also, as illustrated by the example application, the preparation for migration is a seamless and easily reversible process. The activities you undertake to prepare the source database are highly desirable, but not required. If you are, for whatever reason, not able to modify your source database operating environment, you can skip this topic.

The steps shown here follow the same sample schema that was defined in the previous topic.

In order to migrate the sample database, you must add the sharding key column, `CustNo`, to the `LineItems` table in the source database using `ALTER TABLE`, as shown in this example.

```
ALTER TABLE LineItems ADD (CustNo NUMBER(3));
```

With this additional column, the row data layout in the source table and the desired layout in the target sharded table are identical. Now you are ready to prime this new column with the matching data. Values in the additional sharding key column, `CustNo`, in the `LineItems` table must be derived from `Orders` joining `LineItems` in a parent-child relationship.

```
SELECT Orders.CustNo FROM Orders JOIN LineItems ON Orders.PONo =  
LineItems.PONo;
```

In this example, use the `MERGE` statement to populate the `CustNo` column. You could also use the standard SQL to accomplish the same goal. In the example shown here, the `MERGE` statement would look as follows.

```
SQL> BEGIN  
  2  MERGE INTO LineItems l  
  3  USING Orders o  
  4  ON (l.PONo = o.PONo)  
  5  WHEN MATCHED THEN  
  6  UPDATE SET l.CustNo = o.CustNo;  
  7  END;  
  8  /
```

You may discover at this point that there is a referential integrity to be maintained for the `CustNo` column. To make sure the new column is populated correctly, you should add a `NOT NULL` constraint after executing the `MERGE` statement., as shown here.

```
ALTER TABLE LineItems MODIFY CustNo NOT NULL;
```

By running the above `MERGE` statement you bring the `LineItems` table row layout and the row data to the desired state. The additional `CustNo` column makes the `LineItems` table sharded the same way as the root of the table family (`Customers`). You might consider using this change as the one of the last actions before the actual migration. Otherwise, you must maintain this new column within your application. Consequently, you must also maintain the referential integrity for the added sharding key column in

your existing database. The referential integrity constraint for the matching CustNo columns is defined in the LineItems table as shown here.

```
ALTER TABLE LineItems ADD CONSTRAINT LineFk FOREIGN KEY (CustNo, PONO)
REFERENCES Orders (CustNo, PONO);
```

Before changing the referential integrity constraint on the LineItems table you must drop the existing FOREIGN KEY constraint referencing the Orders table. This could be accomplished safely by enclosing the DROP CONSTRAINT followed by ADD CONSTRAINT statements within ALTER TABLE LineItems READ ONLY; and ALTER TABLE LineItems READ WRITE; or simply by locking the table with LOCK TABLE LineItems IN SHARE MODE for the duration of the constraint modifications.

As a result of adding the CustNo column as a part of the foreign key definition in the LineItems table, you must modify the primary key on the Orders table. Changing the primary key, in turn, requires rebuilding indexes, and this may take some time to complete. This effort makes sense only if you plan to run your application against this new schema for some period of time before migrating to the sharded database.

The following example illustrates changing the LineItems and Orders schemata as a result of adding the sharding key to the LineItems table. Prior to dropping the existing foreign key constraint on the LineItems table, and primary key constraint on the Orders table, you must retrieve the respective constraint names as shown here.

```
SQL> SELECT a.table_name, a.column_name, a.constraint_name
  2  FROM ALL_CONS_COLUMNS A, ALL_CONSTRAINTS C
  3  WHERE A.CONSTRAINT_NAME = C.CONSTRAINT_NAME
  4  and a.table_name='LINEITEMS' and C.CONSTRAINT_TYPE = 'R';
```

```
LINEITEMS
PONO
SYS_C009087
```

```
LINEITEMS
STOCKNO
SYS_C009088
```

```
SQL> SELECT cols.table_name, cols.column_name, cols.constraint_name,
cols.position
  2  FROM all_constraints cons, all_cons_columns cols
  3  WHERE cons.constraint_type = 'P'
  4  AND cons.constraint_name = cols.constraint_name
  5  AND cols.table_name = 'ORDERS'
  6  ORDER BY cols.table_name, cols.position;
```

```
ORDERS
ORDER_ID
ORDER_PK
      1
ORDERS
PONO
SYS_C009148
      1
```

```
SQL> ALTER TABLE LineItems READ ONLY;
```

Table altered.

```
SQL> ALTER TABLE Orders READ ONLY;
```

Table altered.

```
SQL> ALTER TABLE LineItems DROP CONSTRAINT SYS_C009087;
```

Table altered.

```
SQL> ALTER TABLE ORDERS DROP CONSTRAINT SYS_C009148;
```

Table altered.

```
SQL> ALTER TABLE ORDERS ADD CONSTRAINT order_pk PRIMARY KEY (CustNo,  
PONO);
```

Table altered.

```
SQL> ALTER TABLE LineItems ADD CONSTRAINT LineFk FOREIGN KEY (CustNo,  
PONO) REFERENCES Orders (CustNo, PONO);
```

Table altered.

```
SQL> ALTER TABLE Orders READ WRITE;
```

Table altered.

```
SQL> ALTER TABLE LineItems READ WRITE;
```

Table altered.

Similarly, you should extend the `PRIMARY KEY` definition for the `LineItems` table to a full level key by including `CustNo` as the leading column, as shown here.

```
ALTER TABLE LineItems ADD CONSTRAINT LinePK PRIMARY KEY (CustNo, PONO,  
LineNo);
```

Again, you must drop the existing `PRIMARY KEY` constraint before introducing the new one. To preserve the data integrity, modify the `PRIMARY KEY` and `FOREIGN KEY` constraints using one of the two transaction isolation strategies suggested earlier. In the following example, the `LineItems` table is locked while the constraint modifications take place. Afterward, `COMMIT` releases the lock.

```
SQL> SELECT cols.table_name, cols.column_name, cols.constraint_name,  
cols.position  
2 FROM all_constraints cons, all_cons_columns cols  
3 WHERE cons.constraint_type = 'P'  
4 AND cons.constraint_name = cols.constraint_name  
5 AND cols.table_name = 'LINEITEMS'  
6 ORDER BY cols.table_name, cols.position;
```

LINEITEMS

```
LINENO  
SYS_C009086  
    1  
LINEITEMS  
PONO  
SYS_C009086  
    2
```

```
SQL> LOCK TABLE LineItems IN SHARE MODE;
```

Table(s) Locked.

```
SQL> ALTER TABLE LINEITEMS DROP CONSTRAINT SYS_C009086;
```

Table altered.

```
SQL> ALTER TABLE LineItems ADD CONSTRAINT LinePK PRIMARY KEY (CustNo, PONO,  
LineNo);
```

Table altered.

```
SQL> COMMIT;
```

Commit complete.

The referential integrity related modifications are optional. The proposed modifications bring the source database very close to resembling the sharded target database. This further facilitates the transition process.

In some cases, referential integrity cannot be imposed or it is undesirable to create. If this is the case, then the reference partitioning cannot be defined. In that situation you can use the PARENT clause instead.

Finally, the additional CustNo column in the LineItems table might affect the existing queries, such as `SELECT * FROM LineItems`. To avoid this problem you can modify the CustNo column to become invisible, as shown here.

```
SQL> ALTER TABLE LineItems MODIFY CustNo INVISIBLE;
```

With these modifications to the source database tables, you have prepared the existing sample database for the migration.

Preparing the Target Sharded Database

Before you start data migration to the sharded database, you must create the sharded database schema according to your design.

The data migration from a non-sharded to a sharded database environment can be accomplished in two distinct ways:

- **Two-step approach:** This is a more cautious, incremental approach to data migration. Create a sharded database with only one shard. As long as your sharded database is contained within one shard, your application, as well as your database maintenance procedures could be used without application code changes and/or a negligible amount

of related modifications. In other words, your sharded database would behave the same way as non-sharded database upon migration to the sharded environment. Then, once you modify your applications and operating procedures for sharding, you can proceed with scaling out the database to the appropriate number of shards.

- **Single-step approach:** Create a sharded database with the appropriate number of shards initially. In this case your application and operating procedures should be fully prepared for sharding operations upon migration.

The more cautious, two-step approach allows a smooth, but significantly longer transition to a sharded database environment. Running your application against a single shard gives you time to gradually modify your application for direct routing. Only after you modify your existing applications to use the shard director to connect to the correct shard, the remaining shards can be instantiated.

The first step of the process is creating a sharded database with only one shard. Then you modify your existing application as suggested in one of the following sections. The last step is to scale-out your sharded database to the number of shards you need. This method also provides an opportunity to split and rebalance the data chunks across all of the shards before scaling out.

This two-step migration approach not only requires more time, but it also requires more space. If you migrate all of your data to a single shard, you can load your sharded table data directly to the single shard without restrictions. After you scale out to multiple shards, you should strictly use the Data Pump utility if you want to correctly load sharded tables directly into multiple database shards. Duplicated tables reside in the catalog database, and you should always load duplicated tables using the catalog database.

Whether you decide to use the one-step or two-step approach, you must export your data and create your sharded database schema before you load the data into the sharded database. In the examples below, it is assumed that the software installation is complete and that you have created a sharded database environment, including at least a sharding catalog and one or more databases for the shards. You can create a new shard catalog or use an existing one. To illustrate the migration process, the examples in this procedure use the following database instances:

- `orignode` – site hosting the original, non-sharded database instance, `SID=orig`
- `catnode` – catalog node hosting shard catalog database instance, `SID=ctlg`
- `shrdnodeN` – shard node(s) hosting the database shard instance(s), `SID=shrdN`, where `N` could be 1, 2, and so on
- `gsmnode` – catalog node hosting the shard director (GSM) instance, `SID=gsm1`

Whether you have modified the source database in preparation for migration or not, your migration process requires modifications to DDL definitions including, at least, the `CREATE SHARDED TABLE` and `CREATE DUPLICATED TABLE` statements. In order to migrate the database schema to the target sharded database, you must extract the DDL definitions from your source database and modify the `SHARDED` and `DUPLICATED` table metadata. A convenient way to extract the DDL statements from the source database is to create a Data Pump extract file. Then use the Data Pump import utility against the database export file, as shown here.

```
impdp uname/pwd@orignode directory=expdir dumpfile=sample.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. This database export file is created as shown here.

```
expdp uname/pwd@orignode full=Y directory=expdir dumpfile=sample.dmp
logfile=sample.log
```

The full database export file has the entire database contents: data and metadata. Exporting the entire database may take a long time. If you want to do this step quickly, you can export only the metadata, or only the part containing the set of tables you are interested in, as shown in this example.

```
expdp uname/pwd directory=DATA_PUMP_DIR dumpfile=sample_mdt.dmp
logfile=sample_mdt.log INCLUDE=TABLE:"IN \( \'CUSTOMERS\' , \'ORDERS\' ,
\'STOCKITEMS\' , \'LINEITEMS\' \) \ " CONSISTENT=Y CONTENT=METADATA_ONLY
```

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 and perform the DDL modifications as required by your sharded database schema design. If you decided to export the entire database, you would likely also use it as an input for loading your data.

Data Pump provides a secure way to transport your data. 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 the `EXP_FULL_DATABASE` role to the user, `uname`. No additional role is required for a table level export. For more information about the Data Pump utility see the Database Utilities documentation in the references below.

If you modified your source (non-sharded) database so that the row layout matches the target (sharded) database, no full database or table level export is required. The data can be efficiently transferred as is, without an intermediate dump file.

After finalizing your sharded database schema, run the prepared DDL against the sharding catalog database (`ctlg`) using database administrator credentials. All DDL statements must be executed in a session with the `SHARD DDL` setting enabled to ensure that all DDL statements are propagated from the catalog database (`ctlg`) to the shard databases (`shrd1, 2, ..., N`).

```
ALTER SESSION ENABLE SHARD DDL;
```

With the sharded and duplicated tables defined, your sharded database is ready for data loading. It is recommended that you validate the sharding configuration using the `GDSCTL VALIDATE` command, before loading the data.

```
gdsctl validate
```

After successful validation your sharded database is ready for data loading. If you see inconsistencies or errors, you must correct the problem using the GDSCTL commands `SHOW DDL` and `RECOVER`.

Data Pump export utility files are, by default, consistent on a per table basis. If you want all of the tables in the export to be consistent to the same point in time, you must use the `FLASHBACK_SCN` or `FLASHBACK_TIME` parameters. Having a consistent “as of” point in time database export file is recommended. This is especially important if you opt for uninterrupted migration from a non-sharded to a sharded database, that is, if you want to provide continuous database operations during the migration. Migration to a sharded database using Data Pump during continuous operations is complemented with Oracle GoldenGate. An export command producing a consistent database snapshot would look like the following.

```
expdp uname/pwd@orignode full=Y directory=expdir dumpfile=sample.dmp
logfile=sample.log CONSISTENT=Y FLASHBACK_TIME=SYSTIMESTAMP
```

The consistent snapshot database image requires additional `CONSISTENT` or `FLASHBACK_TIME` parameters. When you run the command you notice that both parameters, `CONSISTENT` and `FLASHBACK_TIME`, mean the same thing. Note that the timestamp converts to a system change number (SCN) as shown here.

```
SQL> SELECT TIMESTAMP_TO_SCN(SYSTIMESTAMP) FROM dual;
TIMESTAMP_TO_SCN(SYSTIMESTAMP)
-----
                        1559981
```

If you prefer using `FLASHBACK_SCN` over `FLASHBACK_TIME`, you can obtain the current SCN by selecting it from `V$DATABASE` as shown here.

```
SQL> SELECT current_scn FROM v$database;
CURRENT_SCN
-----
      1560005
```

Alternatively, you can declare it as shown here.

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE SCN NUMBER;
2 BEGIN
3   SCN := DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER;
4   DBMS_OUTPUT.PUT_LINE(SCN);
5 END;
6 /
1560598
```

You might need to ask a database administrator for authorization to access `DBMS_FLASHBACK`.

You can make Data Pump run faster by using the `PARALLEL` parameter. This parameter should be used in conjunction with the `%U` wildcard in the `DUMPFILE` parameter to allow multiple dump files be created, as shown in this example.

```
expdp uname/pwd@orignode full=Y directory=expdir dumpfile=samp_%U.dmp  
logfile=samp.log CONSISTENT=Y PARALLEL=3
```

The above command uses four parallel workers and creates four dump files suffixed with `_01`, `_02`, `_03`. The same wildcard can be used during the import to allow you to reference multiple input files. Note that the three dump files, `samp_01.dmp`, `samp_02.dmp`, and `samp_03.dmp`, are created by parallel export rather than the single output file, `sample.dmp` created in a previous example. Also, the elapsed time of the parallel export is less than third of the elapsed time with serial execution, that is, a single dump file output.



See Also:

[Sharded Database Deployment](#)

Oracle Database Utilities

Oracle Database Global Data Services Concepts and Administration Guide

Migrating Your Data

After you create the target sharded database with a single shard, or multiple shards, and you have your sharded and duplicated tables defined, you can start migrating your data into the sharded database.

Make sure you understand the following data loading considerations before you start migrating your data from your source to the sharded database:

- Differences between migrating duplicated and sharded tables
Duplicated tables reside in the shard catalog, they are always loaded into the shard catalog database using any of available data loading utilities, or plain SQL. You have two options for loading sharded tables, however. The sharded tables can be loaded using the sharding coordinator (catalog), or they can be loaded directly into the shards, using the Data Pump utility.
- Migrating sharded tables using the coordinator or direct loading to the shards
Loading your sharded tables directly to the shard databases is always faster because you can load multiple shards simultaneously.
- Migrating to multiple shards or migrating to a single shard
Migrating a non-sharded database to a sharded database with a single shard does not require major changes to your application and database maintenance procedures. You can continue with your current operations largely unchanged until you are ready to split the data into shards. However, migrating a non-sharded database to sharded database with multiple shards involves non-trivial preparation processes, with modifications to the application source code being the most time consuming prerequisite. If you intend to migrate to sharded database with a single shard, you must also set up a plan to scale your database to multiple shards, that is, distribute the database chunks across multiple shards, at some later time. Meanwhile, you can work on the application and other changes required to run with a sharded database with multiple shards.

- Migration with downtime or uninterrupted migration
Uninterrupted migration from a non-sharded to a sharded database with only one shard is much easier and simpler than migration to a sharded database with multiple shards. The subsequent scale-out from one shard to multiple shards is preformed while the database is running.

If you want to migrate from your non-sharded database to the target multi-shard database in a single step, it is strongly recommended that you try this out in a test environment first, and start migrating your production environment only after you make sure that the test environment migrates without issues.

Whether the target row data layout is prepared in the source database or not, there are various methods to choose from for efficient data migration. Choose the method that best fits your conditions: available disk space, remote file access, network throughput, and so on.

Consider Downtime During Migration

If you want to eliminate database down time, your migration plan must include Oracle GoldenGate. To keep your target (sharded) database in sync with the source (non-sharded) database, you must use Oracle GoldenGate to process the changes that are made to the source database during the migration process.

In addition to the benefit of a zero downtime migration, you might also choose to use Oracle GoldenGate for active-active replication of your sharded database. If you have defined Oracle GoldenGate active-active replication for your sharded database then all of the data migration activity should be restricted to the shard catalog database.

Migrating Data to Sharded Tables With Downtime

If you do not use Oracle GoldenGate, take measures to keep your database as available as possible during the migration; you should plan for the downtime.

If you decide to modify your source, non-sharded database tables before migration, your table schema would match your target sharded database schema, and your data migration process will be smoother than if you did not take this pre-migration step. This approach results in an identical row data layout in the source database and corresponding layout in your target sharded database, so you can copy over the database content directly.

1. Export the data from your database tables.

```
expdp uname/pwd@non_sharded_db directory=file_dir
      dumpfile=original_tables.dmp logfile=original_table.log
      INCLUDE=TABLE:"IN \( \'CUSTOMERS\', \'ORDERS\',
\'STOCKITEMS\',
      \'LINEITEMS\' \) \'" CONSISTENT=Y CONTENT=DATA_ONLY
```

This Data Pump export example is limited to the tables used by the Oracle Sharding sample application. Because the SHARDED and DUPLICATED tables have been already created in the sample, you only export the table content (DATA_ONLY).

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. When the export is complete, you can start importing the content to your sharded database.
The DUPLICATED table (StockItems) must be loaded using the shard catalog. The following is an example of the import command.

```
impdp uname/pwd@catnode:1521/ctlg directory=data_pump_dir
      dumpfile=original_tables.dmp logfile=imp.log tables=StockItems
      content=DATA_ONLY
```

4. Load the shards directly, or use the shard catalog to load them.
The best way to load the SHARDED tables (Customers, Orders, and LineItems) 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=data_pump_dir
      dumpfile=original_tables.dmp logfile=imp.log tables=Customers,
      Orders, LineItems content=DATA_ONLY
```

Alternatively, you can run Data Pump on the shard catalog to load all of the tables. The following example shows the import command.

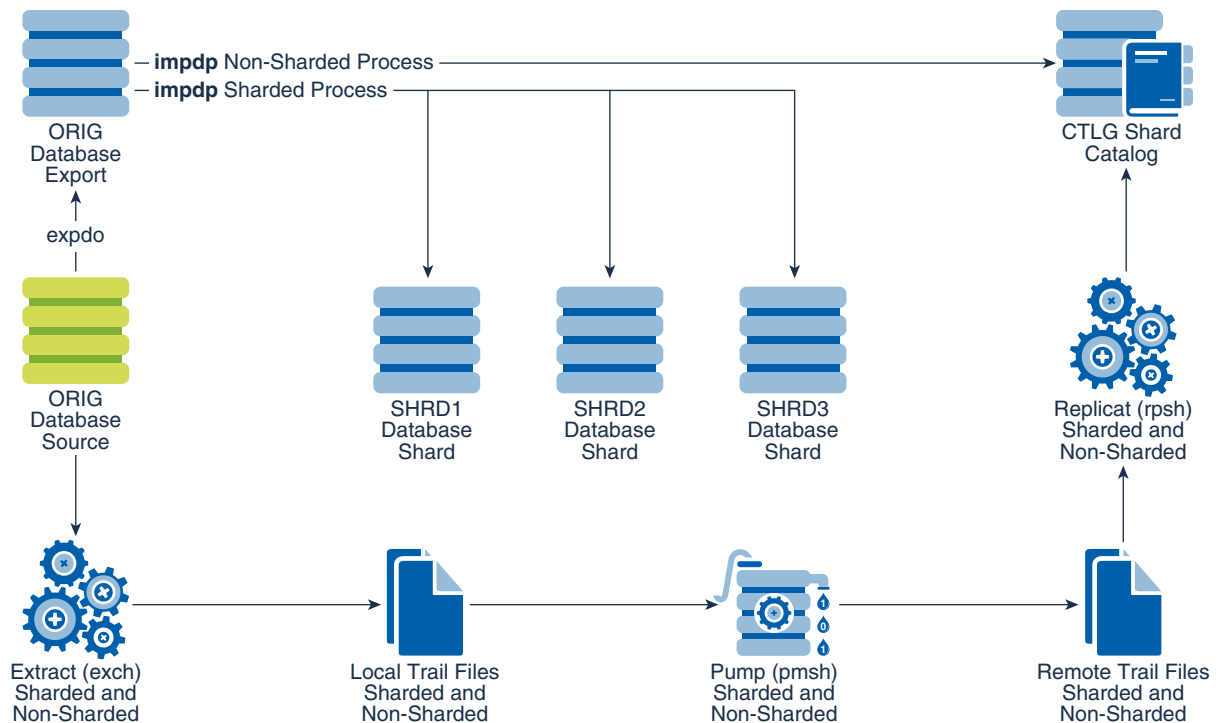
```
impdp uname/pwd@catnode:1521/ctlg directory=data_pump_dir
      dumpfile=original_tables.dmp logfile=imp.log
      tables=Customers, Orders, LineItems, StockItems
      content=DATA_ONLY
```

Migrating Data to Sharded Tables Without Downtime

Oracle Sharding provides tight integration between Oracle GoldenGate replication and Data Pump export and import utilities. If Oracle GoldenGate is not currently present on either the source database or target sharded database, then before you install it you should upgrade your database to the latest release of Oracle Database on both the source and target databases. Upgrading the databases provides the maximum available functionality and simplifies the setup. With Oracle Database 12c Release 2 and later you can use integrated capture on the non-sharded source and integrated replicat on the target sharded database.

As long as Data Pump export and import are used to recreate the entire database, Oracle GoldenGate ensures that the migration of database changes during and after the export and import takes place. It is up to you to decide what needs to be replicated by Oracle GoldenGate during the migration process. It is not recommended that you replicate database changes that are not required by the application.

When Oracle GoldenGate is configured for the source and target databases, it is recommended that you do testing using the live data before scheduling the production migration. If the source database is cloned you can use the clone for testing the migration without affecting your production environment.

Figure 9-5 Migration Without Downtime

The migration process using Data Pump combined with Oracle GoldenGate is illustrated above. The bulk of the data migration is performed using Data Pump, moving data directly to the shard catalog and shards. The database changes during the Data Pump run are collected in Oracle GoldenGate local trail files, and moved to the shard catalog database. The changes to sharded tables are propagated from the shard catalog database to the shards.

From the Data Pump perspective, the source database (ORIG) is split into the shard catalog (CTLG) and the shard databases (SHRD1, SHRD2, and SHRD3) using impdp non-sharded and impdp sharded processes. The impdp non-sharded process migrates duplicated tables to the shard catalog database. The three impdp sharded parallel processes migrate the sharded tables directly to shards SHRD1, SHRD2, and SHRD3.

From the Oracle GoldenGate perspective, all of the databases share extract (exsh), pump (pmsh), and replicat (rpsh) process pipelines forked from the Extract root process.

Assuming that you have prepared the `obey` files corresponding to the diagram above, a GGSCI terminal session for Oracle GoldenGate pipeline from the source database node (orignode) to shard catalog node (catnode) would look like the following example.

```
view params ./dirprm/add_exsh_2pumps.oby
-- add a change data extract process group named exsh
-- exsh reads DUPLICATED and SHARDED tables from orig database redo logs
add extract exsh, tranlog, begin now
-- associate the trail file as output from exsh process group
add exttrail ./dirdat/et, extract exsh
-- add SHARDED and DUPLICATE change data extract pump process pmsh
```

```
-- pmsb copies local trail data to ctlgnode remote trail location
add extract pmsb, exttrailsource ./dirdat/et
-- associate the remote trail with pmsb
add rmttrail ./dirdat/et, extract pmsb
-- connect to the database and add table level supplemental logging for:
-- Customers, Orders, LineItems, and StockItems tables
add trandata uname.Customers
add trandata uname.Orders
add trandata uname.LineItems
add trandata uname.StockItems
```

Run the GGSCI obey command for the pipeline, followed by info all.

```
obey ./dirprm/add_exsh_2pumps.oby
info all
```

You should see the extract processes for the non-sharded and sharded pipeline initialized, and waiting in stopped status.

```
...
EXTRACT STOPPED exsh ...
EXTRACT STOPPED pmsb ...
...
```

Check the process group parameter, EXTRACT, for the sharded and catalog tables.

```
view params exsh
-- first line must be extract followed the name
extract exsh
-- login info to get metadata
userid uname@orignode, password pwd
-- export is writing to trail info
exttrail ./dirdat/exsh
-- checkpoint time interval with source
checkpointsecs 1
-- source table
table uname.Customers
table uname.Orders
table uname.LineItems
table uname.StockItems
```

View parameters for the Pump process group for all tables, pmsb.

```
view parms pmsb
-- first line must be extract followed the name
extract pmsb
-- no need to log into the database
passthru
-- connect to remote host, write and talk to the manager there
rmthost shrdnode, mgrport 7810
-- where is the trail on remote host
rmttrail ./dirdat/rt
```

```
-- checkpoint time interval with target
checkpointsecs 1
-- tables
table uname.Customers
table uname.Orders
table uname.LineItems
table uname.StockItems
```

The Oracle GoldenGate Pump process is created under the assumption that the manager process on the Shard catalog node (catnode) uses port number 7810. This is a fair assumption because the shard catalog and shard databases run on separate machines.

Check the definition on the shard catalog node (catnode).

```
view params mgr
PORT 7810
-- used by the PUMP process on the source side for the collector on the
target
DYNAMICPORTLIST 8000-8010
SYSLOG NONE
```

On the shard catalog node (catnode), look at the prepared Replicat process obey command file.

```
view params ./dirprm/add_rpsh.oby
-- connect to the database
dblogin userid uname@catnode, password pwd
-- add checkpoint table
add checkpointtable uname.gg_checkpoint
-- add replicat process rpsh that will convert remote trail into SQL
continuously
add replicat rpsh, exttrail ./dirdat/rt, checkpointtable
uname.gg_checkpoint
```

If you modified the source database to match the sharded row data layout in preparation for migration, you might have introduced invisible columns. Invisible columns can be preserved during the replication process by adding `MAPINVISIBLECOLUMNS` as a replicat process parameter.

Run the obey file `./dirprm/add_rpsh.oby` on the shard catalog node (catnode).

At this point the system is configured for data migration without downtime. The bulk of the load is performed by the Data Pump export (expdp) and import (impdp) database utilities. The eventual database changes during the export and import processes are synchronized by Oracle GoldenGate processes. Note that the Oracle GoldenGate Pump process has nothing to do with any of the Data Pump processes.

Before starting the Data Pump export, the Oracle GoldenGate replication process must be provided with instantiation Commit Sequence Numbers (CSNs) for each table that is a part of the Data Pump export. As described earlier, this can be done by

running expdp with `CONSISTENT=Y` and `FLASHBACK_SCN=scn_num`. The `FLASHBACK_SCN` can be obtained with the following statement.

```
SELECT current_scn from v$database;
```

Because expdp is run with `CONSISTENT=Y`, all table images appear “as of `scn_num`”, so that the replication process can be started from the same, in this case, CSN number.

For the replicat on the shard catalog node, the appropriate GGSCI command might look like this.

```
START REPLICAT rpsh, AFTERCSN scn_num
```

The simpler way to do this is to use the `ADD SCHEMATRANDATA` GGSCI command on the source node. This command populates the system tables and views so that the instantiation CSNs can be used on the import. This way the Oracle GoldenGate CSN becomes synchronized with the Data Pump SCN by matching two stamps representing the committed version of the database. In other words, `FLASHBACK_SCN` for export process and `AFTERCSN` for replicat process are defined automatically, as shown here.

```
ADD SCHEMATRANDATA uname PREPARESCN ALLCOLS
```

The `ADD SCHEMATRANDATA` command enables schema-level supplemental logging for all of the current and future tables in a given ‘uname’ schema of the ORIG database to automatically log a superset of keys that Oracle GoldenGate uses for row identification. The `PREPARESCN` parameter instructs the Data Pump export (expdp) to automatically generate actions to set instantiation CSN (GGSCI command, `SET_INSTANTIATION_CSN`) for each table at target upon import (impdp). The `ALLCOLS` parameter enables the unconditional supplemental logging of all supported key and non-key columns for all current and future tables in the given schema. This option enables the logging of the keys required to compute dependencies, plus columns that are required for filtering, conflict resolution, or other purposes. It is important to note that the sharding related data migration is limited to the specific set of tables, and other database (incremental) changes are not propagated to the shard catalog.

The replication on the target database should be stopped before starting the Data Pump export. This should always be the case because the target database is created from scratch. The GGSCI command to stop the replicat process on the shard catalog node is shown here.

```
STOP REPLICAT rpsh
```

This command preserves the state of synchronization for the next time the replicat process starts, and it ensures that the Oracle GoldenGate manager processes do not automatically start the replicat process.

The extract process on the source database should be active before starting the Data Pump export. Check if extract is already active by using the `INFO EXTRACT` or `STATUS EXTRACT` command. The following command starts the extract process, if it is not already started.

```
START EXTRACT exsh, BEGIN NOW
```

From this point on, the extract process collects the changes to the source database for all of the tables involved in the sharding process, and you can safely initiate Data Pump export.

```
expdp uname/pwd@orignode full=Y directory=expdir dumpfile=sample.dmp
logfile=sample.log
```

It is a good practice to verify that the export utility expdp `FLASHBACK_SCN` parameter is added automatically. You should be able to find the message “FLASHBACK automatically enabled to preserve database integrity” in the expdp command output. After the export completes, immediately continue with the Data Pump import on the target databases as described earlier for the shard catalog.

```
impdp uname/pwd@catnode:1521/ctlg directory=data_pump_dir
dumpfile=sample.dmp logfile=imp.log
tables=Customers,Orders,LineItems,StockItems content=DATA_ONLY
```

At this point it is safe to start replicat processes on the shard catalog node.

```
START REPLICAT rpsl
```

From this point on, the selected set of tables in the source database (Customers, Orders, LineItems, and StockItems) will be in synch with the Customers, Orders, and LineItems tables in the shards and the StockItems duplicated table in the shard catalog database. It is a good practice, after starting replicat processes, to take a look at the report file for the replicat processes. Verify that the replicat process was aware of the SCN or CSN number existing in the database while the export was in progress, and it knows that any changes after that SCN now need to be applied on the target table, look for “Instantiation CSN filtering is enabled on table uname.Customers, uname,Orders, ...” to verify.

Migrating Your Application

The sharded database operating environment empowers applications with direct access to shards. This feature provides true linear scalability, but it comes with a small price—a slight change to the application code.

The examples that follow show you how to migrate your application. The examples are a skeleton of the sample application for which we migrated the database schema and data. The key parts of the sample application include order management and reporting functionality, contained in Java class, `POManager`. The first static method in this class introduces a new row into the Customers table using the `addCustomer()` method. The column values are passed in to this function using a parameter list, as shown here.

```
import java.sql.*;
import java.io.*;
import oracle.jdbc.driver.*;

public class POManager {
    public static void addCustomer (int custNo, String custName,
        String street, String city, String state, String zipCode,
        String phoneNo) throws SQLException {
```

```
String sql = "INSERT INTO Customers VALUES (?, ?, ?, ?, ?, ?, ?)";
try {
    Connection conn =
        DriverManager.getConnection("jdbc:default:connection:");
    PreparedStatement pstmt = conn.prepareStatement(sql);
    pstmt.setInt(1, custNo);
    pstmt.setString(2, custName);
    pstmt.setString(3, street);
    pstmt.setString(4, city);
    pstmt.setString(5, state);
    pstmt.setString(6, zipCode);
    pstmt.setString(7, phoneNo);
    pstmt.executeUpdate();
    pstmt.close();
} catch (SQLException e) {System.err.println(e.getMessage());}
}
```

The second static method in the `POManager` class, `addStockItem()`, adds a row in the `StockItem` table. The column values are passed as parameter values, as shown in the following example.

```
public static void addStockItem (int stockNo, String description,
float price) throws SQLException {
    String sql = "INSERT INTO StockItems VALUES (?, ?, ?)";
    try {
        Connection conn =
            DriverManager.getConnection("jdbc:default:connection:");
        PreparedStatement pstmt = conn.prepareStatement(sql);
        pstmt.setInt(1, stockNo);
        pstmt.setString(2, description);
        pstmt.setFloat(3, price);
        pstmt.executeUpdate();
        pstmt.close();
    } catch (SQLException e) {System.err.println(e.getMessage());}
}
```

The third static method in the `POManager` class, `enterOrder()`, adds a row into the `Orders` table. The column values are provided in a parameter list, as shown here.

```
public static void enterOrder (int orderNo, int custNo,
String orderDate, String shipDate, String toStreet,
String toCity, String toState, String toZipCode)
throws SQLException {
    String sql = "INSERT INTO Orders VALUES (?, ?, ?, ?, ?, ?, ?, ?)";
    try {
        Connection conn =
            DriverManager.getConnection("jdbc:default:connection:");
        PreparedStatement pstmt = conn.prepareStatement(sql);
        pstmt.setInt(1, orderNo);
        pstmt.setInt(2, custNo);
        pstmt.setString(3, orderDate);
        pstmt.setString(4, shipDate);
        pstmt.setString(5, toStreet);
        pstmt.setString(6, toCity);
```

```
        pstmt.setString(7, toState);
        pstmt.setString(8, toZipCode);
        pstmt.executeUpdate();
        pstmt.close();
    } catch (SQLException e) {System.err.println(e.getMessage());}
}
```

The next static method in the `POManager` class, `addLineItem()`, adds a row in the `LineItems` table. The column values are passed in as parameter values, as shown in the following example.

```
public static void addLineItem (int lineNo, int orderNo,
    int stockNo, int quantity, float discount) throws SQLException {
    String sql = "INSERT INTO LineItems VALUES (?, ?, ?, ?, ?)";
    try {
        Connection conn =
            DriverManager.getConnection("jdbc:default:connection:");
        PreparedStatement pstmt = conn.prepareStatement(sql);
        pstmt.setInt(1, lineNo);
        pstmt.setInt(2, orderNo);
        pstmt.setInt(3, stockNo);
        pstmt.setInt(4, quantity);
        pstmt.setFloat(5, discount);
        pstmt.executeUpdate();
        pstmt.close();
    } catch (SQLException e) {System.err.println(e.getMessage());}
}
```

The next static method in the `POManager` class, `totalOrders()`, produces the total order value for every order in the `Orders` table. The result set relation is printed out using the `printResult()` method, as shown here.

```
public static void totalOrders () throws SQLException {
    String sql =
        "SELECT O.PONo, ROUND(SUM(S.Price * L.Quantity)) AS TOTAL " +
        "FROM Orders O, LineItems L, StockItems S " +
        "WHERE O.PONo = L.PONo AND L.StockNo = S.StockNo " +
        "GROUP BY O.PONo";
    try {
        Connection conn =
            DriverManager.getConnection("jdbc:default:connection:");
        PreparedStatement pstmt = conn.prepareStatement(sql);
        ResultSet rset = pstmt.executeQuery();
        printResults(rset);
        rset.close();
        pstmt.close();
    } catch (SQLException e) {System.err.println(e.getMessage());}
}
```


The helper method, `printResults()`, shown below, is used to print out the result set relations produced by the `totalOrders()` method. A reference to the result set relation is passed in as a parameter.

```
static void printResults (ResultSet rset) throws SQLException {
    String buffer = "";
    try {
        ResultSetMetaData meta = rset.getMetaData();
        int cols = meta.getColumnCount(), rows = 0;
        for (int i = 1; i <= cols; i++) {
            int size = meta.getPrecision(i);
            String label = meta.getColumnLabel(i);
            if (label.length() > size) size = label.length();
            while (label.length() < size) label += " ";
            buffer = buffer + label + " ";
        }
        buffer = buffer + "\n";
        while (rset.next()) {
            rows++;
            for (int i = 1; i <= cols; i++) {
                int size = meta.getPrecision(i);
                String label = meta.getColumnLabel(i);
                String value = rset.getString(i);
                if (label.length() > size) size = label.length();
                while (value.length() < size) value += " ";
                buffer = buffer + value + " ";
            }
            buffer = buffer + "\n";
        }
        if (rows == 0) buffer = "No data found!\n";
        System.out.println(buffer);
    } catch (SQLException e) {System.err.println(e.getMessage());}
}
```

The `checkStockItem()` static method, shown below, retrieves all orders, customers, and line item details for the specified stock item. The stock item is passed in as a parameter. The helper method, `printResults()`, detailed above, is used to print out the result set relations produced by the `checkStockItem()` method.

```
public static void checkStockItem (int stockNo)
    throws SQLException {
    String sql = "SELECT O.PONo, O.CustNo, L.StockNo, " +
        "L.LineNo, L.Quantity, L.Discount " +
        "FROM Orders O, LineItems L " +
        "WHERE O.PONo = L.PONo AND L.StockNo = ?";
    try {
        Connection conn =
            DriverManager.getConnection("jdbc:default:connection:");
        PreparedStatement pstmt = conn.prepareStatement(sql);
        pstmt.setInt(1, stockNo);
        ResultSet rset = pstmt.executeQuery();
        printResults(rset);
        rset.close();
        pstmt.close();
    }
```

```
    } catch (SQLException e) {System.err.println(e.getMessage());}  
}
```

The `changeQuantity()` static method updates the line item quantity for the given order and the stock item. The specific order number and the stock item are provided as input parameters, as shown here.

```
public static void changeQuantity (int newQty, int orderNo,  
int stockNo) throws SQLException {  
    String sql = "UPDATE LineItems SET Quantity = ? " +  
        "WHERE PONo = ? AND StockNo = ?";  
    try {  
        Connection conn =  
            DriverManager.getConnection("jdbc:default:connection:");  
        PreparedStatement pstmt = conn.prepareStatement(sql);  
        pstmt.setInt(1, newQty);  
        pstmt.setInt(2, orderNo);  
        pstmt.setInt(3, stockNo);  
        pstmt.executeUpdate();  
        pstmt.close();  
    } catch (SQLException e) {System.err.println(e.getMessage());}  
}
```

The last static method, `deleteOrder()`, removes the specified order from the Orders table and all associated line items. The order that is to be deleted is specified as the input parameter, as shown in the following example.

```
public static void deleteOrder (int orderNo) throws SQLException {  
    String sql = "DELETE FROM LineItems WHERE PONo = ?";  
    try {  
        Connection conn =  
            DriverManager.getConnection("jdbc:default:connection:");  
        PreparedStatement pstmt = conn.prepareStatement(sql);  
        pstmt.setInt(1, orderNo);  
        pstmt.executeUpdate();  
        sql = "DELETE FROM Orders WHERE PONo = ?";  
        pstmt = conn.prepareStatement(sql);  
        pstmt.setInt(1, orderNo);  
        pstmt.executeUpdate();  
        pstmt.close();  
    } catch (SQLException e) {System.err.println(e.getMessage());}  
}
```

Now, look at the sample application program code that is modified for sharding. The first thing to note are the additional imports of sharding related Java packages, starting with `OracleShardingKey`.

```
import java.sql.*;  
import java.io.*;  
import oracle.jdbc.driver.*;  
  
//
```

```
// import sharding and related packages
//
import oracle.jdbc.OracleShardingKey;
import oracle.jdbc.OracleType;
import oracle.jdbc.pool.OracleDataSource;

//
// Sample App: order management and reporting
// modified for sharding
//
public class PManager
{
    // Connection factory for the sharded database used by Sample App
    private OracleDataSource ods;

    //
    // Construct PManager class using Sharded database connection properties.
    // Use service name when connecting to sharded database, something like:
    // "jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(HOST=myhost)(PORT=3216)"
    // "(PROTOCOL=tcp))(CONNECT_DATA=(SERVICE_NAME=myservice)(REGION=east))"
    //
    public PManager(String yourURL, String yourUser, String yourPwd)
        throws SQLException
    {
        ods = new OracleDataSource();
        ods.setURL(yourURL);
        ods.setUser(yourUser);
        ods.setPassword(yourPwd);
    } // PManager
}
```

As shown above, the PManager class now contains the OracleDataSource factory for Connection objects. The following two methods show you how to use the connection factory to produce direct and proxy routing connections. The getCatConn() method returns a connection to the sharding catalog. The getShardConn() method returns the connection to the shard that matches the sharding key provided as a parameter.

```
//
// Connect to the Sharding Catalog database.
//
public static Connection getCatConn() throws SQLException
{
    Connection catConn = ods.getConnection();
} // getCatConn

//
// Connect to Shard database using sharding key.
//
public static Connection getShardConn(int custNo) throws SQLException
{
    OracleShardingKey shardKey =
        ods.createShardingKeyBuilder().subkey(custNo, JDBCType.NUMERIC).build();
    OracleConnection shardConn =
        ods.createConnectionBuilder().shardingKey(shardKey);
}
```

```
    return shardConn;  
} // getShardConn
```

With all of this in mind, you would rewrite the `addCustomer()` method for the sharding environment, using `getShardConn()`, as shown here.

```
//  
// Connect to Shard database to add a customer.  
//  
public static void addCustomer (int custNo, String custName,  
    String street, String city, String state, String zipCode,  
    String phoneNo) throws SQLException {  
    String sql = "INSERT INTO Customers VALUES (?, ?, ?, ?, ?, ?, ?)";  
    try {  
        Connection conn = getShardConn(custNo);  
        PreparedStatement pstmt = conn.prepareStatement(sql);  
        pstmt.setInt(1, custNo);  
        pstmt.setString(2, custName);  
        pstmt.setString(3, street);  
        pstmt.setString(4, city);  
        pstmt.setString(5, state);  
        pstmt.setString(6, zipCode);  
        pstmt.setString(7, phoneNo);  
        pstmt.executeUpdate();  
        pstmt.close();  
    } catch (SQLException e) {System.err.println(e.getMessage());}  
} // addCustomer
```

In `addStockItem()`, you use `getCatConn()` to connect to the shard catalog database and insert into the duplicated table, `StockItems`.

```
//  
// Connect to Sharding Catalog to add a stock item.  
//  
public static void addStockItem (int stockNo, String description,  
    float price) throws SQLException {  
    String sql = "INSERT INTO StockItems VALUES (?, ?, ?)";  
    try {  
        Connection conn = getCatConn();  
        PreparedStatement pstmt = conn.prepareStatement(sql);  
        pstmt.setInt(1, stockNo);  
        pstmt.setString(2, description);  
        pstmt.setFloat(3, price);  
        pstmt.executeUpdate();  
        pstmt.close();  
    } catch (SQLException e) {System.err.println(e.getMessage());}  
} // addStockItem
```

The Orders tables is a child of the Customers table. To insert into the Orders table, connect to the shard database based on the provided sharding key, as shown in the following example.

```
//
// Connect to Shard database to add an order for a customer.
//
public static void enterOrder (int orderNo, int custNo,
    String orderDate, String shipDate, String toStreet,
    String toCity, String toState, String toZipCode)
    throws SQLException {
    String sql = "INSERT INTO Orders VALUES (?, ?, ?, ?, ?, ?, ?, ?)";
    try {
        Connection conn = getShardConn(custNo);
        PreparedStatement pstmt = conn.prepareStatement(sql);
        pstmt.setInt(1, orderNo);
        pstmt.setInt(2, custNo);
        pstmt.setString(3, orderDate);
        pstmt.setString(4, shipDate);
        pstmt.setString(5, toStreet);
        pstmt.setString(6, toCity);
        pstmt.setString(7, toState);
        pstmt.setString(8, toZipCode);
        pstmt.executeUpdate();
        pstmt.close();
    } catch (SQLException e) {System.err.println(e.getMessage());}
} // enterOrder
```

Notice that for the root of the table family, Customers, and the immediate child level, Orders, the sharding key is provided as a parameter. For levels below immediate child you might not have a full level key, so you must retrieve it from the immediate parent. In the following example there is a helper method, `getCustomerFromOrder()`, for retrieving the sharding key, `custNo`, from the Order table, based on the Order table key value, `orderNo`. Every child table row is supposed to have a single parent row. That is why an integrity violation exception is raised for children without a parent, or children with more than one parent.

```
//
// Determine which shard order is in
//
static int getCustomerFromOrder(int orderNo) throws SQLException
{
    String sql = "SELECT O.CustNo FROM Orders O " +
        "WHERE O.PONo = ?";
    int custNo;
    int rsSize = 0;
    Exception exception;
    try {
        Connection conn = getCatConn();
        PreparedStatement pstmt = conn.prepareStatement(sql);
        pstmt.setInt(1, orderNo);
        PreparedStatement pstmt = conn.prepareStatement(sql);
        ResultSet rset = pstmt.executeQuery();
        while(rset.next() && rsSize < 3) {
            custNo = rs.getInt("CustNo");
            rsSize++;
        }
    }
}
```

```

    }
    rset.close();
    pstmt.close();
} catch (SQLException e) {System.err.println(e.getMessage());}
if (rsSize == 0) {
    throw new
        SQLIntegrityConstraintViolationException(
            "No matching parent level key");
}
if (rsSize == 2)
{
    throw new
        SQLIntegrityConstraintViolationException(
            "More than one parent level key");
}
return custNo;
} // getCustomerFromOrder

```

**Note:**

Your code should not make any assumptions about the number of shards in the sharded database.

The following example shows the rewritten `addLineItem()` method, that was provided in the original `POManager` class, now using the `getCustomerFromOrder()` helper method. For a given `orderNo` it queries the shard catalog for the matching `custNo` values. This part of the query is propagated to all of the shards using proxy routing. The helper function returns a single `custNo` value. Based on this value, use the direct route to the shard to insert the `LineItems` table row.

```

//
// Get customer (parent) from the catalog, then insert into a shard!
//
public static void addLineItem (int lineNo, int orderNo,
    int stockNo, int quantity, float discount) throws SQLException {
    String sql = "INSERT INTO LineItems VALUES (?, ?, ?, ?, ?, ?)";
    try {
        int custNo = getCustomerFromOrder(orderNo);
        Connection conn = getShardConn(custNo);
        PreparedStatement pstmt = conn.prepareStatement(sql);
        pstmt.setInt(1, custNo);
        pstmt.setInt(2, lineNo);
        pstmt.setInt(3, orderNo);
        pstmt.setInt(4, stockNo);
        pstmt.setInt(5, quantity);
        pstmt.setFloat(6, discount);
        pstmt.executeUpdate();
        pstmt.close();
    } catch (SQLException e) {System.err.println(e.getMessage());}
} // addLineItem

//
// You've got whole level key handy, insert into a shard directly.

```

```
//
public static void addLineItemWithinParent (int lineNo, int orderNo,
    int custNo, int stockNo, int quantity, float discount)
    throws SQLException
{
    String sql = "INSERT INTO LineItems VALUES (?, ?, ?, ?, ?, ?)";
    try {
        Connection conn = getShardConn(custNo);
        PreparedStatement pstmt = conn.prepareStatement(sql);
        pstmt.setInt(1, custNo);
        pstmt.setInt(2, lineNo);
        pstmt.setInt(3, orderNo);
        pstmt.setInt(4, stockNo);
        pstmt.setInt(5, quantity);
        pstmt.setFloat(6, discount);
        pstmt.executeUpdate();
        pstmt.close();
    } catch (SQLException e) {System.err.println(e.getMessage());}
} // addLineItemWithinParent
```

 **Note:**

Most of the time the full level key context will be available in the application context. That is why we introduce the additional method `addLineItemWithinParent()` which connects directly to the shard based on the leading column `custNo` in the `LineItems` table composite level key. This eliminates the round-trip to the shard catalog. Avoid using expensive sharding programming practices similar to our helper function: `getCustomerFromOrder()` whenever possible.

A majority of aggregate queries must be executed using the shard catalog connection. The shard catalog database uses proxy routing to collect partial results from the shards. The final aggregation is produced based on the partial results produced by the shards. This is why the `totalOrders()` method introduced in the original `POManager` class is rewritten to connect to the shard catalog database, as shown here.

```
//
// xshard aggregate connects to the shard catalog
//
public static void totalOrders () throws SQLException {
    String sql =
        "SELECT O.PONo, ROUND(SUM(S.Price * L.Quantity)) AS TOTAL " +
        "FROM Orders O, LineItems L, StockItems S " +
        "WHERE O.PONo = L.PONo AND L.StockNo = S.StockNo " +
        "GROUP BY O.PONo";
    try {
        Connection conn = getCatConn();
        PreparedStatement pstmt = conn.prepareStatement(sql);
        ResultSet rset = pstmt.executeQuery();
        printResults(rset);
        rset.close();
        pstmt.close();
    }
```

```

    } catch (SQLException e) {System.err.println(e.getMessage());}
} // totalOrders

```

The `printResults()` helper function, introduced previously, does not depend on the database structure, so no modifications are necessary.

```

//
// helper function - no change required
//
static void printResults (ResultSet rset) throws SQLException {
    String buffer = "";
    try {
        ResultSetMetaData meta = rset.getMetaData();
        int cols = meta.getColumnCount(), rows = 0;
        for (int i = 1; i <= cols; i++) {
            int size = meta.getPrecision(i);
            String label = meta.getColumnLabel(i);
            if (label.length() > size) size = label.length();
            while (label.length() < size) label += " ";
            buffer = buffer + label + " ";
        }
        buffer = buffer + "\n";
        while (rset.next()) {
            rows++;
            for (int i = 1; i <= cols; i++) {
                int size = meta.getPrecision(i);
                String label = meta.getColumnLabel(i);
                String value = rset.getString(i);
                if (label.length() > size) size = label.length();
                while (value.length() < size) value += " ";
                buffer = buffer + value + " ";
            }
            buffer = buffer + "\n";
        }
        if (rows == 0) buffer = "No data found!\n";
        System.out.println(buffer);
    } catch (SQLException e) {System.err.println(e.getMessage());}
} // printResults

```

The values in `StockItem` table key column, `stockNo`, could potentially match `Order` table rows on all of the shards. That is why you must modify the `checkStockItem()` method, introduced in the original `POManager` class, to connect to the shard catalog. The shard catalog database returns a union of all rows returned as a result of local joins performed in each shard.

```

//
// xshard query matching duplicated table
//
public static void checkStockItem (int stockNo)
    throws SQLException {
    String sql = "SELECT O.PONo, O.CustNo, L.StockNo, " +
        "L.LineNo, L.Quantity, L.Discount " +
        "FROM Orders O, LineItems L " +
        "WHERE O.PONo = L.PONo AND L.StockNo = ?";
}

```



```

    try {
        Connection conn = getCatConn();
        PreparedStatement pstmt = conn.prepareStatement(sql);
        pstmt.setInt(1, stockNo);
        ResultSet rset = pstmt.executeQuery();
        printResults(rset);
        rset.close();
        pstmt.close();
    } catch (SQLException e) {System.err.println(e.getMessage());}
} // checkStockItem

```

The `changeQuantity()` method, introduced in the original `POManager` class, updates the grandchild table, `LineItems`. Again, use the `getCustomerFromOrder()` helper method to obtain the sharding key so your application can connect to the correct shard to perform the update. Similar to the `addLineItem()` method modification, you should expect that the `custNo` column value is available in the application context. That is why you should use `changeQuantityWithinParent()` in `changeQuantity()`, saving the round trip to the shard catalog.

```

//
// Get customer (parent) from the catalog then update a shard!
//
public static void changeQuantity (int newQty,
                                   int orderNo, int stockNo)
    throws SQLException
{
    String sql = "UPDATE LineItems SET Quantity = ? " +
        "WHERE CustNo = ? AND PONo = ? AND StockNo = ?";
    try {
        int custNo = getCustomerFromOrder(orderNo);
        Connection conn = getShardConn(custNo);
        PreparedStatement pstmt = conn.prepareStatement(sql);
        pstmt.setInt(1, newQty);
        pstmt.setInt(2, custNo);
        pstmt.setInt(3, orderNo);
        pstmt.setInt(4, stockNo);
        pstmt.executeUpdate();
        pstmt.close();
    } catch (SQLException e) {System.err.println(e.getMessage());}
} // changeQuantity

//
// You've got the full level key handy, update shard directly.
//
public static void changeQuantityWithinParent (int newQty,
                                               int custNo, int orderNo, int stockNo)
    throws SQLException
{
    String sql = "UPDATE LineItems SET Quantity = ? " +
        "WHERE CustNo = ? AND PONo = ? AND StockNo = ?";
    try {
        Connection conn = getShardConn(custNo);
        PreparedStatement pstmt = conn.prepareStatement(sql);
        pstmt.setInt(1, newQty);
        pstmt.setInt(2, custNo);
    }
}

```

```

        pstmt.setInt(3, orderNo);
        pstmt.setInt(4, stockNo);
        pstmt.executeUpdate();
        pstmt.close();
    } catch (SQLException e) {System.err.println(e.getMessage());}
} // changeQuantityWithinParent

```

The modification to the last method, `deleteOrder()`, follows the same guidelines applied to `addLineItem()`, and `changeQuantity()`. Before deleting a row from the `LineItems` table, your application should first look in the shard catalog for the `custNo` sharding key value corresponding to the requested order. Once you have the sharding key, connect to the shard and perform the delete. Again, use the `deleteOrderWithinParent()` method, with the expectation that the sharding key value is available in the application context.

```

//
// Get customer (parent) first from the catalog, delete in a shard!
//
public static void deleteOrder (int orderNo) throws SQLException
{
    String sql = "DELETE FROM LineItems WHERE CustNo = ? AND PONo = ?";
    try {
        int custNo = getCustomerFromOrder(orderNo);
        Connection conn = getShardConn(custNo);
        PreparedStatement pstmt = conn.prepareStatement(sql);
        pstmt.setInt(1, orderNo);
        pstmt.executeUpdate();
        sql = "DELETE FROM Orders WHERE PONo = ?";
        pstmt = conn.prepareStatement(sql);
        pstmt.setInt(1, custNo);
        pstmt.setInt(2, orderNo);
        pstmt.executeUpdate();
        pstmt.close();
    } catch (SQLException e) {System.err.println(e.getMessage());}
} // deleteOrder

//
// You've got whole level key handy, delete in shard directly
//
public static void deleteOrderWithinParent (int custNo,
                                           int orderNo)
    throws SQLException
{
    String sql = "DELETE FROM LineItems WHERE CustNo = ? AND PONo = ?";
    try {
        Connection conn = getShardConn(custNo);
        PreparedStatement pstmt = conn.prepareStatement(sql);
        pstmt.setInt(1, orderNo);
        pstmt.executeUpdate();
        sql = "DELETE FROM Orders WHERE PONo = ?";
        pstmt = conn.prepareStatement(sql);
        pstmt.setInt(1, custNo);
        pstmt.setInt(2, orderNo);
        pstmt.executeUpdate();
    }
}

```

```
        pstmt.close();  
    } catch (SQLException e) {System.err.println(e.getMessage());}  
    } // deleteOrderWithinParent  
  
} // PManager
```

10

Sharded Database Lifecycle Management

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](#)
- [Monitoring a Sharded Database](#)
Sharded databases can be monitored using Enterprise Manager Cloud Control or GDSCTL.
- [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.
- [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.
- [Managing Sharded Database Software Versions](#)
- [Shard Management](#)
You can manage shards in your Oracle Sharding deployment with Oracle Enterprise Manager Cloud Control and GDSCTL.
- [Chunk Management](#)
You can manage chunks in your Oracle Sharding deployment with Oracle Enterprise Manager Cloud Control and GDSCTL.
- [Shard Director Management](#)
You can add, edit, and remove shard directors in your Oracle Sharding deployment with Oracle Enterprise Manager Cloud Control.
- [Region Management](#)
You can add, edit, and remove regions in your Oracle Sharding deployment with Oracle Enterprise Manager Cloud Control.
- [Shardspace Management](#)
You can add, edit, and remove shardspaces in your Oracle Sharding deployment with Oracle Enterprise Manager Cloud Control.
- [Shardgroup Management](#)
You can add, edit, and remove shardgroups in your Oracle Sharding deployment with Oracle Enterprise Manager Cloud Control.
- [Services Management](#)
You can manage services in your Oracle Sharding deployment with Oracle Enterprise Manager Cloud Control.

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](#)
- [Shutting Down the Sharding-Enabled Stack](#)

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.

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.

- [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 Sharded Database with Enterprise Manager Cloud Control](#)
Oracle Enterprise Manager Cloud Control lets you discover, monitor, and manage the components of a sharded database.
- [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.

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

GDCTL 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:**

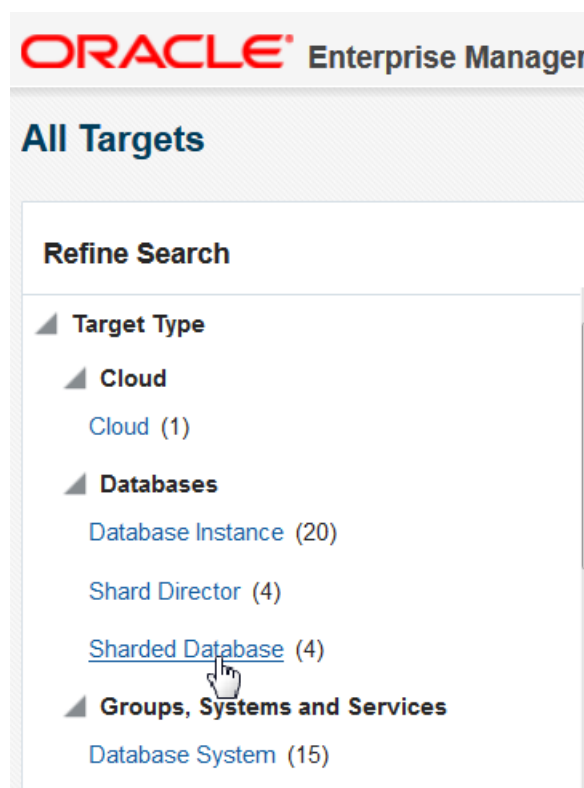
Oracle Database Global Data Services Concepts and Administration Guide for information about using the GDCTL CONFIG commands

Monitoring a Sharded Database with Enterprise Manager Cloud Control

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

Sharded database targets are found in the All Targets page.

Figure 10-1 Sharded Databases in the All Targets Refine Search pane



The target home page for a sharded database shows you a summary of the sharded database components and their statuses.

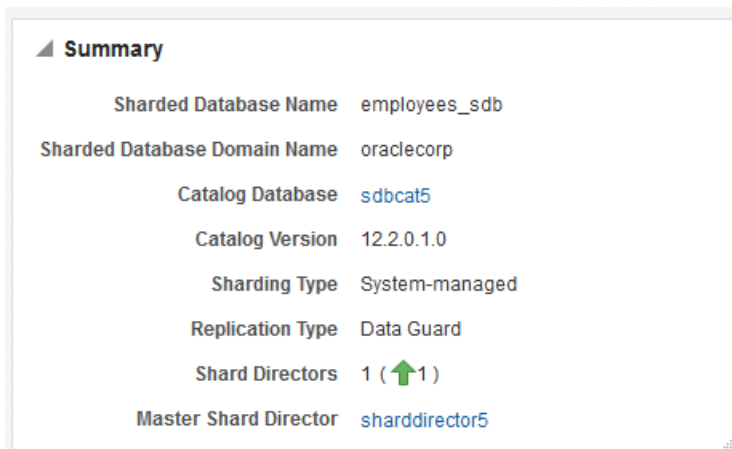
To monitor sharded database components you must first discover them. See [Discovering Sharded Database Components](#) for more information.

Summary

The Summary pane, in the top left of the page, shows the following information:

- Sharded database name
- Sharded database domain name
- Shard catalog name. You can click the name to view more information about the shard catalog.
- Shard catalog database version
- Sharding method used to shard the database
- Replication technology used for high availability
- Number and status of the shard directors
- Master shard director name. You can click the name to view more information about the master shard director.

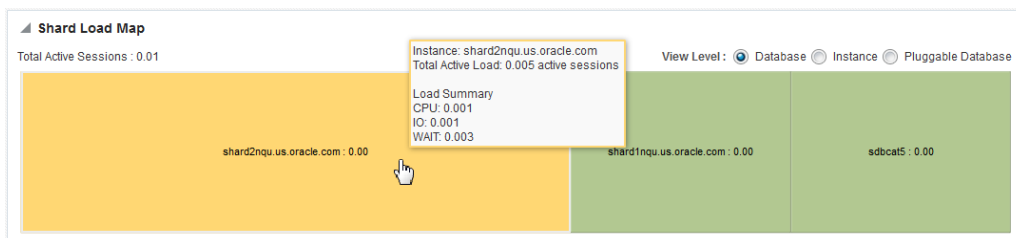
Figure 10-2 Sharded Database Summary pane



Shard Load Map

The Shard Load Map, in the upper right of the page, shows a pictorial graph illustrating how transactions are distributed among the shards.

Figure 10-3 Sharded Database Shard Load Map



You can select different View Levels above the graph.

- **Database**
The database view aggregates database instances in Oracle RAC cluster databases into a single cell labeled with the Oracle RAC cluster database target name. This enables you to easily compare the total database load in Oracle RAC environments.
- **Instance**
The instance view displays all database instances separately, but Oracle RAC instances are grouped together as sub-cells of the Oracle RAC database target. This view is essentially a two-level tree map, where the database level is the primary division, and the instance within the database is the secondary division. This allows load comparison of instances within Oracle RAC databases; for instance, to easily spot load imbalances across instances.
- **Pluggable Database**
Although the PDB option is shown, PDB is not supported for Oracle Sharding in the current release.

Notice that the cells of the graph are not identical in size. Each cell corresponds to a shard target, either an instance or a cluster database. The cell size (its area) is proportional to the target database's load measured in average active sessions, so that targets with a higher load have larger cell sizes. Cells are ordered by size from left to right and top to bottom. Therefore, the target with the highest load always appears as the upper leftmost cell in the graph.

You can hover your mouse pointer over a particular cell of the graph to view the total active load (I/O to CPU ration), CPU, I/O, and wait times. Segments of the graph are colored to indicate the dominant load:

- Green indicates that CPU time dominates the load
- Blue indicates that I/O dominates the load
- Yellow indicates that WAIT dominates the load

Members

The Members pane, in the lower left of the page, shows some relevant information about each of the components.

Figure 10-4 Sharded Database Members pane

▲ Members Refresh Configuration

Shardspaces **Shardgroups** Shard Directors **Shards**

Name	Status	Shardspace	Shardgroup	Data Guard Role	Region	Deployed
▲ shard1nqu.us.oracle.com	↑	shardspaceora	shardgroup_east	Primary	east	✓
shard2nqu.us.oracle.co	↑	shardspaceora	shardgroup_west	Standby	west	✓

The pane is divided into tabs for each component: Shardspaces, Shardgroups, Shard Directors, and Shards. Click on a tab to view the information about each type of component

- **Shardspaces**
The Shardspaces tab displays the shardspace names, status, number of chunks, and Data Guard protection mode. The shardspace names can be clicked to reveal more details about the selected shardspace.
- **Shardgroups**
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 and shardspace names to reveal more details about the selected component.
- **Shard Directors**
The Shard Directors tab displays the shard director names, status, region, host, and Oracle home. You can click the shard director names can be clicked to reveal more details about the selected shard director.
- **Shards**
The Shards tab displays the shard names, deploy status, status, the shardspaces and shardgroups to which they belong, Data Guard roles, and the regions to which they belong. In the Names column, you can expand the Primary shards to display the information about its corresponding Standby shard. 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.

Services

The Services pane, in the lower right of the page, shows the names, status, and Data Guard role of the sharded database 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.

Figure 10-5 Sharded Database Services pane

▲ **Services**

Services 2 (↑2)

Name	Status	Data Guard Role
employees_report_year_2015	↑	Physical Stan...
employees_report_year_2016	↑	Primary

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 administrate 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 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.

- [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.

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, you must use Cloud Control to discover the shard director hosts and the shard catalog database. Because the catalog database and each of the shards is a database itself, you can use standard database discovery procedures.

Monitoring 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 SDB 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 catalog database to get the configuration information. The monitoring user is granted GDS_CATALOG_SELECT role and has read only privileges on the catalog repository tables.

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.

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.

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` and `CREATE TRIGGER`. 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

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.
- [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.
- [Downgrading a Sharded Database](#)
Oracle Sharding does not support downgrading 18c versions to 12c.

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.

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 SDB. 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 SDB.

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.

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.

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 18c versions to 12c.

Sharded database catalogs and shards cannot be downgraded from 18c to 12c.

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.
- [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.
- [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.
- [Adding Standby Shards](#)
You can add Data Guard standby shards to an Oracle Sharding environment; however there are some limitations.
- [Managing Shards with Oracle Enterprise Manager Cloud Control](#)
You can manage database shards using Oracle Enterprise Manager Cloud Control
- [Managing Shards with GDSCTL](#)
You can manage shards in your Oracle Sharding deployment using the GDSCTL command-line utility.

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:

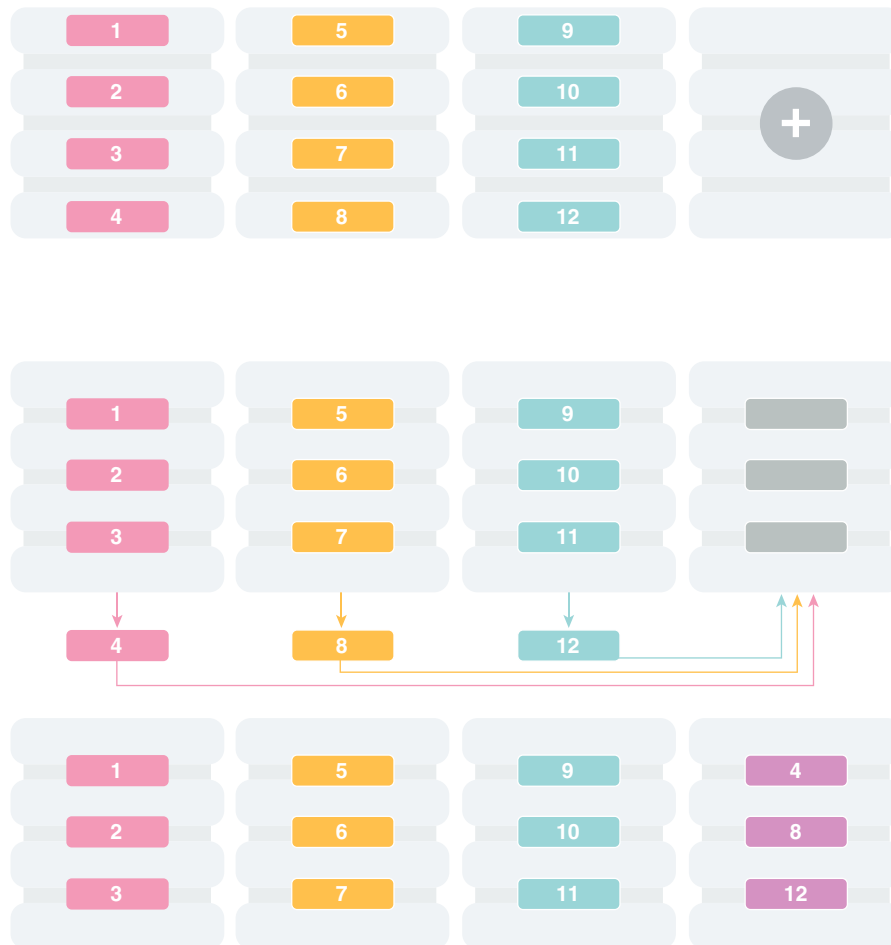
- When one or multiple shards are added to or removed from an SDB
- When there is skew in the data or workload distribution across shards

The unit of data migration between shards is the chunk. Migrating data in chunks guarantees 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 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.
- [Adding Primary Shards](#)
Use Oracle Enterprise Manager Cloud Control to add a primary shards to your Oracle Sharding deployment.
- [Adding Standby Shards](#)
Use Oracle Enterprise Manager Cloud Control to add a standby shards to your Oracle Sharding deployment.
- [Deploying Shards](#)
Use Oracle Enterprise Manager Cloud Control to deploy shards that have been added to your Oracle Sharding environment.

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.
- [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.
- [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.

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:

- [Installing Oracle Database Software](#)
 - Registering remote scheduler agents as described in [Setting Up the Oracle Sharding Management and Routing Tier](#)
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 shardedirector1
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:**

See this [note](#) for information about automatic VNCR registration.

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.oradbccloud" 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.oradbccloud" 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 accidentally 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 `GDSCCTL`. 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.
- [Moving Chunks](#)
You can move chunks from one shard to another in your Oracle Sharding deployment using Oracle Enterprise Manager Cloud Control.
- [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.
- [Splitting Chunks](#)
You can split chunks in your Oracle Sharding deployment using Oracle Enterprise Manager Cloud Control.

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 are 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

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.
- [Editing a Shard Director Configuration](#)
Use Oracle Enterprise Manager Cloud Control to edit a shard director configuration in your Oracle Sharding deployment.

- [Removing a Shard Director](#)
Use Oracle Enterprise Manager Cloud Control to remove shard directors from your Oracle Sharding deployment.

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.
- [Editing a Region Configuration](#)
Edit sharded database region configurations in your Oracle Sharding deployment using Oracle Enterprise Manager Cloud Control.
- [Removing a Region](#)
Remove sharded database regions in your Oracle Sharding deployment using Oracle Enterprise Manager Cloud Control.

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.
- [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.

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.

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:

```
GDSCCTL 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.

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

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:

- [Oracle Sharding Tracing and Debug Information](#)
The following topics explain how to enable tracing and find the logs.
- [Common Error Patterns and Resolutions for Sharded Databases](#)
See the following topics for information about troubleshooting common errors in Oracle Sharding.

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.
- [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.

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:
- [Shard Director Fails to Start](#)
If you encounter issues starting the shard director, try the following:
- [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:
- [Issues Using Create Shard](#)
The following are solutions to some issues that occur when using the GDSCTL CREATE SHARD command..
- [Issues Using Deploy Command](#)

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=tcip)
(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 directory 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
```


Index

A

about, [1-3](#)
Active Data Guard, [7-1](#)
advantages, [1-3](#)
application migration, [9-22](#)
architecture, [1-3](#)

B

benefits, [1-2](#)

C

candidates for sharding, [1-6](#)
CDB, [8-3](#)
chunk management, [10-28](#)
chunks
 moving, [10-30](#)
 splitting, [10-31](#)
connection pools, [6-18](#)
consistency levels in multi-shard queries, [6-11](#)

D

data encryption, [8-55](#), [8-56](#)
Data Guard
 in Oracle Sharding, [10-16](#)
data migration, [9-1](#), [9-15](#)
data migration, preparing the source, [9-7](#)
data routing, [6-1](#)
data, loading, [9-2](#)
deployment
 add shard, [8-2](#)
 create shard, [8-2](#)
 system-managed, [8-13](#)
direct routing, [6-1](#)
discovering sharded database, [10-7](#)
downgrade, [10-13](#)
duplicated table, [2-7](#)

E

encrypted data, [8-55](#), [8-56](#)

Enterprise Manager Cloud Control
 monitoring Oracle Sharding with, [10-3](#)
 sharded database discovery, [10-7](#)
external table, [9-2](#)

F

features, [1-3](#)

G

GDSCCTL
 ADD SHARD, [10-25](#)
GoldenGate, [7-7](#)

H

high availability, [7-1](#), [7-7](#)

I

installation
 system-managed, [8-13](#)

K

key-based routing, [6-1](#)

L

loading data, [9-2](#)

M

middle-tier routing, [6-18](#)
migrating data, [9-1](#)
migration of data, preparing the source, [9-7](#)
migration, application, [9-22](#)
migration, data, [9-15](#)
migration, preparing, [9-11](#)
migration, schema, [9-4](#)
moving a shard, [10-25](#)
multi-shard query consistency, [6-11](#)
multi-shard query consistency level, [6-11](#)

MULTISHARD_QUERY_DATA_CONSISTENCY,
6-11
multitenant, 8-3

O

Oracle Data Guard, 7-1

Oracle Enterprise Manager Cloud Control

chunk management, 10-28

chunks

moving, 10-30

splitting, 10-31

region

creating, 10-33

editing, 10-34

removing, 10-34

region management, 10-33

services

create, 10-38

services management, 10-38

shard

adding, 10-18, 10-19

deploy, 10-19

validate, 10-17

shard director

creating, 10-32

removing, 10-33

updating, 10-32

shard director management, 10-31

shard management, 10-13

shardgroup

creating, 10-37

shardgroup management, 10-37

shardspace

creating, 10-35

shardspace management, 10-34

Oracle GoldenGate, 7-1, 7-7

Oracle Multitenant, 8-3

Oracle Sharding

>NET support for, 6-2

about, 1-1

add shard, 10-13

application development, 5-1

backup and recovery, 10-9

chunk, 2-1

chunk management, 10-28

chunks, 3-2

moving, 10-30

splitting, 10-31

Cloud Control, 10-3

composite sharding

method

composite sharding, 4-6

configure, 8-9

coordinator, 6-8

Oracle Sharding (*continued*)

Data Guard standby, 10-16

data routing

proxy routing, 6-7

DDL, 2-16

DDL execution, 2-10

deployment, 8-1

composite SDB, 8-43

user-defined SDB, 8-29

discovery in Cloud Control, 10-7

distributed partitioning, 3-1

duplicated objects, 2-9

elastic scaling, 10-21

hardware, 8-5

high availability, 7-1, 7-2

hot spots, 10-14

installation

composite SDB, 8-43

user-defined SDB, 8-29

installing

shard director, 8-7

JDBC support for, 6-2

method

system-managed, 4-1

monitoring, 10-2, 10-8

monitoring with Cloud Control, 10-3

moving chunks, 10-29

multi-shard queries, 6-11

networking, 8-5

Oracle Call Interface APIs for, 6-2

Oracle Database install, 8-6

Oracle UCP APIs for, 6-2

partitions, 3-2

prerequisites, 8-5

proxy routing, 6-9–6-11

region

creating, 10-33

editing, 10-34

removing, 10-34

region management, 10-33

remove shard, 10-16

replication, 7-1, 7-2

request routing

statement-level, 6-9

requirements, 8-5

resharding, 10-14

root table, 2-3

scaling, 10-21

schema changes, 10-10

schema creation

composite SDB, 8-49

user-defined SDB, 8-35

schema design, 2-1

schema design considerations, 5-1

Oracle Sharding (*continued*)

- services
 - create, [10-38](#)
 - services management, [10-38](#)
 - shard
 - adding, [10-13](#), [10-18](#), [10-19](#)
 - deploy, [10-19](#)
 - standby, [10-16](#)
 - validate, [10-17](#)
 - shard catalog
 - creating, [8-7](#)
 - shard director
 - creating, [10-32](#)
 - removing, [10-33](#)
 - updating, [10-32](#)
 - shard director management, [10-31](#)
 - shard management, [10-13](#)
 - shard validation, [10-20](#)
 - sharded table, [2-1](#)
 - shardgroup
 - creating, [10-37](#)
 - shardgroup management, [10-37](#)
 - shards
 - adding, [10-21](#)
 - shardspace
 - creating, [10-35](#)
 - shardspace management, [10-34](#)
 - single-shard queries, [6-10](#)
 - splitting chunks, [10-30](#)
 - SQL, [2-16](#)
 - standby shard, [10-16](#)
 - subpartitions, [4-8](#)
 - system-managed, [4-1](#)
 - table family, [2-3](#)
 - tablespace set, [2-1](#)
 - tablespaces, [3-2](#)
 - user-defined
 - method
 - user-defined, [4-4](#)
 - validate shard, [10-20](#)
 - with Oracle Data Guard, [7-1](#), [7-2](#)
 - with Oracle GoldenGate, [7-1](#)
- Oracle Universal Connection Pool, middle-tier routing, [6-18](#)
- overview, [1-3](#)

P

- patching, [10-11](#)
- PDB, [8-3](#)
- proxy routing
 - queries shapes supported in, [6-12](#)

R

- reference tables, [2-7](#)
- region
 - creating, [10-33](#)
 - editing, [10-34](#)
 - removing, [10-34](#)
- region management, [10-33](#)
- replacing s shard, [10-25](#)
- replication, [7-7](#)
- routing
 - direct, [6-1](#)
 - key-based, [6-1](#)

S

- schema creation
 - system-managed, [8-21](#)
- schema migration, [9-4](#)
- security, [8-55](#), [8-56](#)
- services
 - create, [10-38](#)
- services management, [10-38](#)
- shard
 - adding, [10-18](#), [10-19](#)
 - deploy, [10-19](#)
 - validate, [10-17](#)
- shard director
 - creating, [10-32](#)
 - removing, [10-33](#)
 - updating, [10-32](#)
- shard director management, [10-31](#)
- shard management, [10-13](#)
- shard replacement, [10-25](#)
- sharded database (SDB), [1-1](#)
- sharded database discovery, [10-7](#)
- sharded tables, [1-1](#)
- shardgroup
 - creating, [10-37](#)
- shardgroup management, [10-37](#)
- shards, [1-1](#)
- shards as PDBs, [8-3](#)
- SHARDS clause, [10-8](#)
- shardspace
 - creating, [10-35](#)
- shardspace management, [10-34](#)
- shardspace, adding, [10-35](#)
- swim lanes, [6-18](#)
- system-managed
 - deployment, [8-13](#)
 - schema, [8-21](#)

T

- tables, reference, [2-7](#)

Transparent Data Encryption, [8-55](#), [8-56](#)

upgrading, [10-11](#)

U

upgrade

order, [10-12](#)