Oracle® Database

Using Oracle Sharding

20c
F16352-04
Apri 2020

Oracle Database Using Oracle Sharding, 20c

F16352-04

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

Contents

3 Physical Organization of a Sharded Database

4 Sharding Methods

5 Developing Applications for the Sharded Database

6 Shard-Level High Availability

7 Sharded Database Deployment

8 Migrating to a Sharded Database

9 Sharded Database Administration

11 Oracle Sharding Solutions

12 Oracle Sharding Reference

Preface

Review the following topics to:

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

Audience

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

Documentation Accessibility

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

Access to Oracle Support

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

Related Documents

The following publications may be of particular interest to you:

Install related Oracle products:

- Oracle Database Install and Upgrade
- Oracle GoldenGate

Configuration and administration:

- *Oracle Database Administrator's Guide*
- *Oracle Data Guard Concepts and Administration*
- *Oracle Data Guard Broker*
- *Using the Oracle GoldenGate Microservices Architecture*
- *Oracle Database Global Data Services Concepts and Administration Guide*

Application development:

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

1

Oracle Sharding Overview

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

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

- What is Sharding
  Hyperscale computing is a computing architecture that can scale up or down quickly to meet increased demand on the system. This architecture innovation was originally driven by internet giants that run distributed sites and has been adopted by large-scale cloud providers.
- About Oracle Sharding
  Oracle Sharding is a feature of Oracle Database that lets you distribute and replicate data across a pool of Oracle databases that share no hardware or software. Oracle Sharding provides the best features of mature RDBMS capabilities and NoSQL databases, as described here.
- Benefits of Oracle Sharding
  Oracle Sharding provides linear scalability and complete fault isolation for the most demanding applications.
- Example Applications using Database Sharding
  Oracle Sharding provides benefits for a variety of use cases.
- Flexible Deployment Models
  The shared-nothing architecture of Oracle Sharding lets you keep your data on-premises, in the cloud, or on a hybrid of cloud and on-premises systems. Because the database shards do not share any resources, the shards can exist anywhere on a variety of on-premises and cloud systems.
- High Availability in Oracle Sharding
  Oracle Sharding is tightly integrated with Oracle Database replication technologies, Oracle Data Guard and Oracle GoldenGate, to provide high availability and disaster recovery. Replication is automatically configured and deployed when the sharded database is created.
- Sharding Methods
  You can shard database tables by consistent hash (*system-managed* sharding), by range or list (*user-defined* sharding), or a combination (*composite* sharding).
- Client Request Routing
  Oracle Sharding supports direct, key-based, routing (JDBC, OCI, UCP, ODP.NET) to a shard, routing by proxy, and affinity with middle tiers, such as application containers, web containers, and so on.
- Query Execution
  No changes to query and DML statements are required to support Oracle Sharding. Most existing DDL statements will work the same way on a sharded database with the same syntax and semantics as they do on a non-sharded Oracle Database.
- High Speed Data Ingest
  SQL*Loader enables direct data loading into the database shards for a high speed data ingest.

- Deployment Automation
  Sharded database deployment is highly automated with Terraform, Kubernetes, and Ansible scripts.
- Migration Automation
  Migrating a non-sharded database to a sharded database is made simpler using the Sharding Advisor to help you design an optimal sharded database configuration.
- Lifecycle Management of Shards
  The Oracle Sharding command-line interface and Oracle Enterprise Manager help you manage your sharded database.
- Federated Sharding
  Unify multiple existing databases into one sharded database architecture.
- Components of the Oracle Sharding Architecture
  The following figure illustrates the major architectural components of Oracle Sharding.
- Where To Go From Here
  Planning and deploying a sharded database configuration that best fits your requirements can be a daunting task. The following roadmap can guide you through the process, from initial planning to life cycle management of a sharded database.

What is Sharding

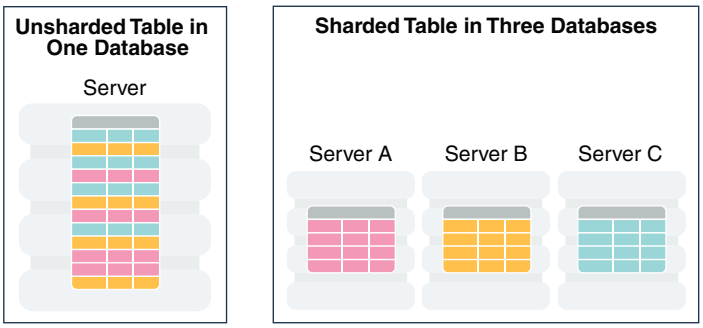
Hyperscale computing is a computing architecture that can scale up or down quickly to meet increased demand on the system. This architecture innovation was originally driven by internet giants that run distributed sites and has been adopted by large-scale cloud providers.

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

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

The following figure compares a table in a database that has not been sharded, shown on the left, with that same table, horizontally partitioned by color and distributed across three databases, on Server A, Server B, and Server C, which serve as the shards. The row colors could indicate any characteristic by which you want to distribute the data, such as geographical area, service level, or any other characteristic.

Figure 1-1 Distribution of a Table Across Database Shards



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

From the perspective of a database administrator, a sharded database consists of multiple databases that can be managed collectively.

About Oracle Sharding

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

- Strict data consistency, complex joins, ACID transaction properties, distributed transactions, relational data store, security, encryption, robust performance optimizer, backup and recovery, and patching with Oracle Database
- Oracle innovations and enterprise-level features, including Advanced Security, Automatic Storage Management (ASM), Advanced Compression, partitioning, high-performance storage engine, SMP scalability, Oracle RAC, Exadata, in-memory columnar, online redefinition, JSON document store, and so on
- Sharding-aware Oracle Database tools, such as SQL Developer, Enterprise Manager Cloud Control, Recovery Manager (RMAN), and Data Pump, for sharded database application development and management
- Programmatic interfaces, such as Java Database Connectivity (JDBC), Oracle Call Interface (OCI), Universal Connection Pool (UCP), Oracle Data Provider for .NET (ODP.NET), and PL/SQL, including extensions for sharded application development
- Extreme availability with Oracle Data Guard, Active Data Guard, and Oracle GoldenGate
- SQL language used for deployment, object creation, and run time
- Support for multi-model data like relational, text, and JSON
- Easier application maintenance because the schema is in the database instead of the application
- Leverage in-house and world-wide Oracle database administrator skill set
- Enterprise-level support

- Extreme scalability and availability of NoSQL databases

Benefits of Oracle Sharding

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

Key benefits of Oracle Sharding include:

- **Linear Scalability**

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

- **Extreme Availability and Fault Isolation**

  Single points of failure are eliminated because shards do not share physical resources such as CPU, memory, or storage devices—the failure or slow-down of one shard does not affect the performance or availability of other shards. Additionally any unplanned outage resulting a potential brownout impacts only a subset of data.

  Planned downtime is similarly transparent because maintenance on one shard does not affect the other shards, and is further mitigated by using Oracle Data Guard for rolling upgrades and switchovers.

- **Geographical Distribution of Data**

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

Example Applications using Database Sharding

Oracle Sharding provides benefits for a variety of use cases.

Real Time OLTP

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

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

Global Applications

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

- Strict enforcement of data sovereignty, where data privacy regulations require data to stay in a certain geographic location, region, country, or even state.
- Reduction of data replication across locations
- Better application SLAs, because planned and unplanned outages in one region do not impact other regions
- Simplified management, because it is easier to manage a smaller set of databases rather than a single monolithic database

Internet of Things and Data Streaming Applications

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

Machine Learning

Many machine learning applications require training and scoring of models in realtime. Model training and scoring for many applications using algorithms like anomaly detection, and clustering is specific to a given entity (for example, a given user's financial transaction patterns or specific device metrics at a certain time of the day). This kind of data can easily be shared by using a sharding key specific to the user or devices. Additionally, Oracle Database Machine Learning algorithms can be applied directly in the database obviating the need for a separate data pipeline and machine learning processing infrastructure.

Big Data Analytics

When you have terabytes of data, sharding means you don't have to warehouse data to do analytics on it. With up to 1000 shards in capacity, Oracle Sharding can turn a relational database into a warehouse-sized data store. With Federated Sharding in Oracle Database 20c, multiple database installations in different locations that run the same application can be converted into a federated sharded database so that you can run data analytics without moving the data.

NoSQL Alternative

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

Flexible Deployment Models

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

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

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

High Availability in Oracle Sharding

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

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

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

Optionally, you can use Oracle RAC for shard-level high availability, complemented by replication, to maintain shard-level data availability in the event of a cluster outage. Each shard can be deployed on an Oracle RAC cluster to give it instant protection from node failure. For example, each shard could be a two node Oracle RAC cluster.

Sharding Methods

You can shard database tables by consistent hash (*system-managed* sharding), by range or list (*user-defined* sharding), or a combination (*composite* sharding).

- **System-Managed sharding** does not require you to map data to shards. The data is automatically distributed across shards using partitioning by consistent hash. The partitioning algorithm uniformly and randomly distributes data across shards.
- **User-defined sharding** lets you explicitly specify the mapping of data to individual shards. It is used when, because of performance, regulatory, or other reasons, certain data needs to be stored on a particular shard, and the administrator needs to have full control over moving data between shards.
- **Composite sharding** allows you to create multiple subsets of data that correspond to a range or list of key values, in a table partitioned by consistent hash. In many use cases, especially Data Sovereignty and Data Proximity, the composite sharding method offers the best of both system-managed, and user-defined., giving you the automation you want and the control you need. No other sharding provider has composite sharding capability.

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

Client Request Routing

Oracle Sharding supports direct, key-based, routing (JDBC, OCI, UCP, ODP.NET) to a shard, routing by proxy, and affinity with middle tiers, such as application containers, web containers, and so on.

- **Key-based routing.** Oracle clients can recognize sharding keys specified in the connection string for high performance data dependent routing. A shard routing cache in the connection layer is used to route database requests directly to the shard where the data resides.
- **Routing by proxy.** Oracle Sharding supports routing for queries that do not specify a sharding key, giving any database application the flexibility to run SQL statements without specifying the shards where the query should be executed. Proxy routing can handle single-shard queries and multi-shard queries.
- **Middle-tier routing.** In addition to sharding the data tier, you can shard the web tier and application tier, distributing the shards of those middle tiers to service a particular set of database shards, creating a pattern known as a *swim lane*. A smart router can route client requests based on specific sharding keys to the appropriate swim lane, which in turn establishes connections on its subset of shards.

Query Execution

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

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

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

High Speed Data Ingest

SQL*Loader enables direct data loading into the database shards for a high speed data ingest.

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

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

Deployment Automation

Sharded database deployment is highly automated with Terraform, Kubernetes, and Ansible scripts.

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

Migration Automation

Migrating a non-sharded database to a sharded database is made simpler using the Sharding Advisor to help you design an optimal sharded database configuration.

The Sharding Advisor is a tool provided with Oracle Sharding which you can use to analyze your current database schema and workload, and plan your migration using one of the recommended configurations.

Sharding Advisor bases recommendations on key goals such as paralellism (distributing query execution evenly among shards), minimizing cross-shard join operations, and minimizing duplicated data.

Lifecycle Management of Shards

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

Using the tools provided you can:

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

Federated Sharding

Unify multiple existing databases into one sharded database architecture.

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

- Queries can be seamlessly executed against a single federated database using multi-shard query coordinator

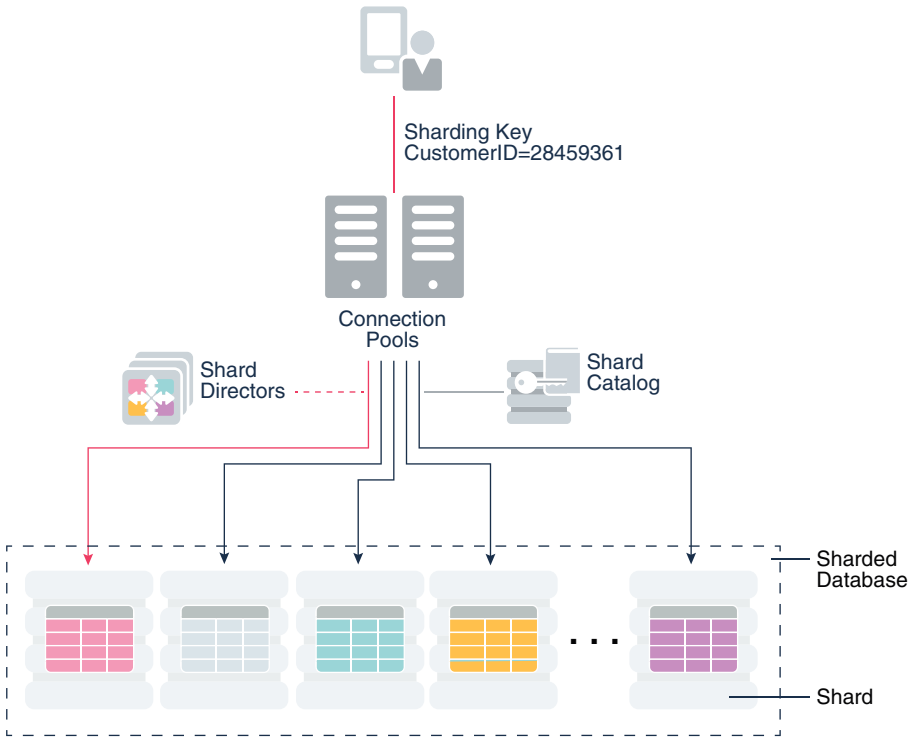
- Removes the need to replicate data for reporting and analytics purposes
- Tolerance for differences in schema and database versions

Components of the Oracle Sharding Architecture

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

- Sharded database – 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 a sharded database
- 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 and Disaster Recovery with Oracle replication technologies such as Oracle Data Guard and Oracle GoldenGate. For high availability, Data Guard standby shards can be placed in the same region where the primary shards are placed. For disaster recovery, the standby shards 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 a sharded database. 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 sharded database configuration data and plays a key role in centralized management of a sharded database. All configuration changes, such as adding and removing shards and global services, are initiated on the shard catalog. All DDLs in a sharded database are executed by connecting to the shard catalog.

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

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 sharded database. 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 sharded database and are unaffected by a catalog outage.

Shard Director

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

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

For a typical sharded database, a set of shard directors are installed on dedicated low-end commodity servers in each region. To achieve high availability, deploy multiple shard directors. You can deploy up to 5 shard directors in a given region.

The following are the key capabilities of shard directors:

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

Connection Pools

Oracle Database supports connection-pooling in data access drivers such as OCI, JDBC, and ODP.NET. These drivers can recognize sharding keys specified as part of a connection request. Similarly, the Oracle Universal Connection Pool (UCP) for JDBC clients can recognize sharding keys specified in a connection URL. Oracle UCP also enables non-Oracle application clients such as Apache Tomcat and WebSphere to work with Oracle Sharding.

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

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

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

Management Interfaces for a Sharded Database

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 a sharded database for availability and performance, and you can make changes to the sharding configuration, 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 a sharded database and automating its deployment. Only a few GDSCTL commands are required to create a sharded database.

Where To Go From Here

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

- **Deploy** a sharded database topology configuration, as explained, with examples, in Sharded Database Deployment
- **Design** a sharded database schema for balanced distribution of data and workload across shards in Sharded Database Schema Design
- **Develop** a high performance, efficient sharded database application using the concepts and APIs described in Developing Applications for the Sharded Database
- **Migrate** your existing database and application to a sharded database, as explained in Migrating to a Sharded Database
- **Manage** your sharded database with the procedures described in Sharded Database Administration

2
Sharded Database Schema Design

Learn sharded database schema concepts and design a database schema for 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.
- 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.
- 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.
- DDL Execution in a Sharded Database
- PL/SQL Procedure Execution in a Sharded Database
  In the same way that DDL statements can be executed on all shards in a configuration, so too can certain Oracle-provided PL/SQL procedures. These specific procedure calls behave as if they were sharded DDL statements, in that they are propogated to all shards, tracked by the catalog, and run whenever a new shard is added to a configuration.
- Generating Unique Sequence Numbers Across Shards
  Oracle Sharding allows you to generate globally unique sequence numbers across shards for non-primary key columns, and it is handled by the sharded database.
- Create a Sharded Database User
  Local users that only exist in the shard catalog database do not have the privileges to create schema objects in the sharded database. The first step of creating the sharded database schema is to create the sharded database user.
- 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.

- 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.
- 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.

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)VARCHAR2`
- `(N)CHAR`
- `DATE`
- `TIMESTAMP`

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

Even though the partitions of a sharded table reside in multiple shards, to the application, the table looks and behaves exactly the same as a partitioned table in a single database. SQL statements issued by an application never have to refer to shards or depend on the number of shards and their configuration.

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 tablespaces cannot be dropped or altered independently of the entire tablespace set.
>
> `TABLESPACE SET` cannot be used with the user-defined sharding method.

Sharded Table Family

A sharded table family is a set of tables that are sharded in the same way.

Often there is a parent-child relationship between database tables with a referential constraint in a child table (foreign key) referring to the primary key of the parent table. Multiple tables linked by such relationships typically form a tree-like structure where every child has a single parent. A set of such tables is referred to as a table family. A table in a table family that has no parent is called the root table. There can be only one root table in a table family.

How a Table Family Is Sharded

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 the Orders and LineItems tables, which are a child and grandchild of the Customers table.

```
CREATE SHARDED TABLE Orders
( OrderNo   NUMBER NOT NULL
, CustNo    NUMBER NOT NULL
, OrderDate DATE
, CONSTRAINT OrderPK PRIMARY KEY (CustNo, OrderNo)
, CONSTRAINT CustFK  FOREIGN KEY (CustNo) REFERENCES Customers(CustNo)
)
PARTITION BY REFERENCE (CustFK)
;


CREATE SHARDED TABLE LineItems
( CustNo    NUMBER NOT NULL
, LineNo    NUMBER(2) NOT NULL
, OrderNo   NUMBER(5) NOT NULL
, StockNo   NUMBER(4)
, Quantity  NUMBER(2)
, CONSTRAINT LinePK  PRIMARY KEY (CustNo, OrderNo, LineNo)
, CONSTRAINT LineFK  FOREIGN KEY (CustNo, OrderNo) REFERENCES
Orders(CustNo, OrderNo)
)
PARTITION BY REFERENCE (LineFK)
;
```

In the example statements above, corresponding partitions of all tables in the family are stored in the same tablespace set – TS1. However, it is possible to specify separate tablespace sets for each table.

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. 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 the Orders table. The second rule implies that the primary key in the Orders table 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.

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.

Multiple Table Families in a Sharded Database

> **Note:**
>
> In Oracle Database 19c, Oracle Sharding includes support for multiple table families. This feature applies to system-managed sharded databases only. Composite and user-defined sharded databases only support one table family.

A sharded database can have multiple table families, where all data from different table families reside in the same chunks, which contain partitions from different table families sharing the same hash key range. Cross-table family queries should be minimal and only carried out on the sharding coordinator. Each table family is associated with a different global service. Applications from different table families each have their own connection pool and service, and use their own sharding key for routing to the correct shard.

Each table family is identified by its root table. Tables in the different table families should not be related to each other. Each table family should have its own sharding key definition, while the same restriction on having the same sharding key columns in child tables still holds true within each table family. This means that all tables from

different table families are sharded the same way with consistent hash into the same number of chunks, with each chunk containing data from all the table families.

The following example shows you how to create multiple table families using the PARENT clause.

```
CREATE SHARDED TABLE Customers <=== Table Family #1
( CustId NUMBER NOT NULL
, Name VARCHAR2(50)
, Address VARCHAR2(250)
, region VARCHAR2(20)
, class VARCHAR2(3)
, signup DATE
)
PARTITION BY CONSISTENT HASH (CustId)
PARTITIONS AUTO
TABLESPACE SET ts1
;

CREATE SHARDED TABLE Orders
( OrderNo NUMBER
, CustId NUMBER
, OrderDate DATE
)
PARENT Customers
PARTITION BY CONSISTENT HASH (CustId)
PARTITIONS AUTO
TABLESPACE SET ts1
;

CREATE SHARDED TABLE LineItems
( LineNo NUMBER
, OrderNo NUMBER
, CustId NUMBER
, StockNo NUMBER
, Quantity NUMBER
)
)
PARENT Customers
PARTITION BY CONSISTENT HASH (CustId)
PARTITIONS AUTO
TABLESPACE SET ts1
;

CREATE SHARDED TABLE Products <=== Table Family #2
( ProdId NUMBER NOT NULL,
  CONSTRAINT pk_products PRIMARY KEY (ProdId)
)
PARTITION BY CONSISTENT HASH (ProdId)
PARTITIONS AUTO
TABLESPACE SET ts_2
;
```

ORA-3850 is thrown if a tablespace set has already been used by an existing table family and you try to use it for another table family.

When you create the first root table (that is, the first table family) all of the existing global services are automatically associated with it. You can use the GDSCTL command `MODIFY SERVICE` to change the table family service association after more table families are created. For example,

```
GDSCTL> MODIFY SERVICE -GDSPOOL shdpool -TABLE_FAMILY sales.customer -
SERVICE sales
```

> **Note:**
>
> Joins across table families may not be efficient, and if you have many such joins, or if they are performance-critical, you should use duplicated tables instead of multiple table families.

> **See Also:**
>
> *Oracle Database VLDB and Partitioning Guide*
>
> *Oracle Database Global Data Services Concepts and Administration Guide* for GDSCTL command reference

Duplicated Tables

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

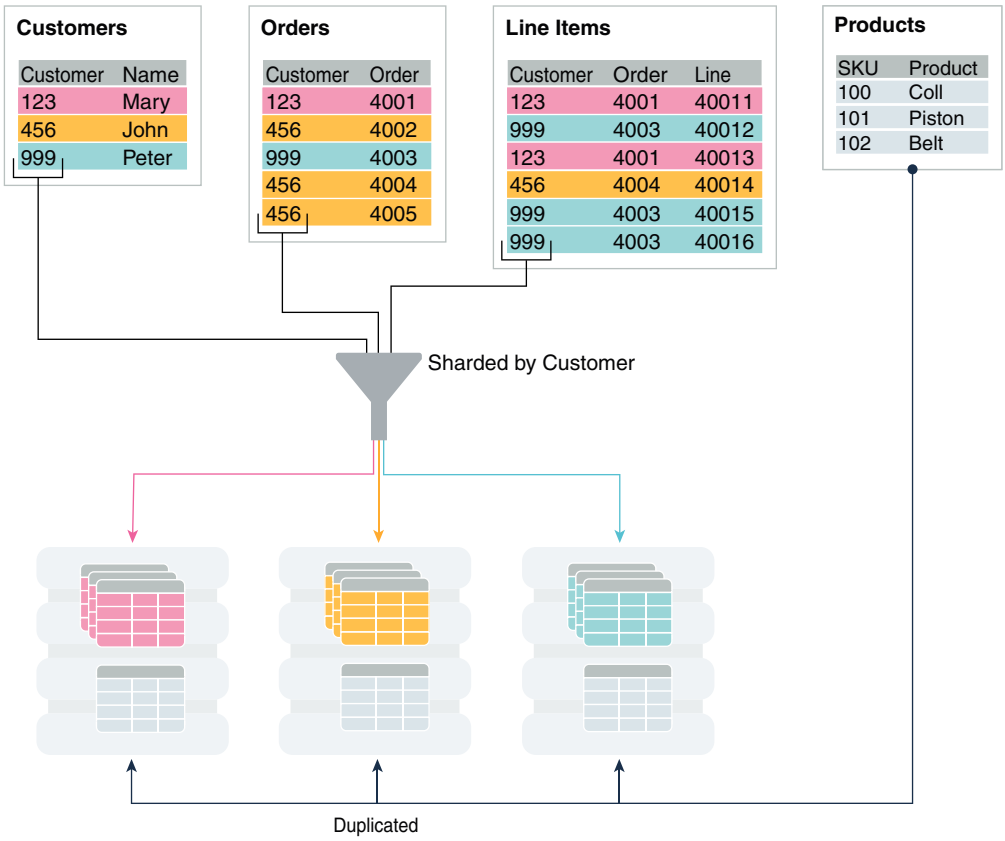
For many applications, the number of database requests handled by a single shard can be maximized by duplicating read-only or read-mostly tables across all shards. This strategy is a good choice for relatively small tables that are often accessed together with sharded tables. A table with the same contents 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 dblinks
> - 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 18*c*, with the following restrictions:

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

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)?

DDL Execution in a Sharded Database

To create a schema in a sharded database, you must issue DDL commands on the shard catalog database, which validates the DDLs and executes them locally, prior to their execution on the shards. 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 sharded database 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 a sharded database, 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 a sharded database:

- 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.

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 IDENTRIFIED BY out_standing1;
CREATE USER example_user IDENTRIFIED BY out_Standing1
                         *
ERROR at line 1:
ORA-00922: missing or invalid option
```

The DDL fails to execute on the shard catalog and, as expected, the GDSCTL `show ddl` command shows that no DDL was executed on any of the shards:

```
GDSCTL> show ddl
id      DDL Text                          Failed shards
--      --------                          -------------
```

Then the user repeats the command with the correct spelling. Note that there is no need to run `alter session enable shard ddl`again because the same session is used.

```
SQL> CREATE USER example_user IDENTIFIED BY out_Standing1;
User created.
```

Now `show ddl` shows that the DDL has been successfully executed on the shard catalog database and it did not fail on any shards that are online.

```
GDSCTL> show ddl
id      DDL Text                                      Failed shards
--      --------                                      -------------
1       create user example_user identified by *****
```

> **Note:**
>
> For any shard that is down at the time of the DDL execution, the DDL is automatically applied when the shard is back up.

Example 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 tablepsace set failure, one of its
tablespaces not created
ORA-01119: error in creating database file \'/ade/b/3667445372/oracle/
rdbms/dbs/
SHARD01/datafile/o1_mf_tbsset_%u_.dbf\'
ORA-27040: file create error, unable to create file
Linux-x86_64 Error: 13: Permission denied
Additional information: 1 \(ngsmoci_execute\)
Failed DDL id: 2
Availability: ONLINE
```

The text beginning with “Failed DDL:” indicates the problem. To resolve it, the user must log in to the shard database host and make the directory writable.

Display the permissions on the directory:

```
cd $ORACLE_HOME/rdbms/dbs
 ls –l ../ | grep dbs
dr-xr-xr-x  4 oracle dba    102400 Jul 20 15:41 dbs/
```

Change the directory to writable:

```
chmod +w .
ls –l ../ | grep dbs
drwxrwxr-x  4 oracle dba    102400 Jul 20 15:41 dbs/
```

Go back to the GDSCTL console and issue the `recover shard` command:

```
GDSCTL> recover shard -shard shard01
```

Check the status again:

```
GDSCTL> show ddl
id      DDL Text                                        Failed shards
--      --------                                        -------------
1       create user example_user identified by *****
2       create tablespace set tbsset

GDSCTL> config shard -shard shard01

Conversion = ':'Name: shard01
Shard Group: dbs1
Status: Ok
State: Deployed
Region: east
Connection string: (DESCRIPTION=(ADDRESS=(HOST=shard01-host)(PORT=1521)
(PROTOCOL=tcp))
(CONNECT_DATA=(SID=shard01)))
SCAN address:
ONS remote port: 0
Disk Threshold, ms: 20
CPU Threshold, %: 75
Version: 18.0.0.0
Last Failed DDL:
DDL Error: ---
DDL id:
Availability: ONLINE
```

As shown above, the failed DDL error no longer appears.

Example 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.

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

PL/SQL Procedure Execution in a Sharded Database

In the same way that DDL statements can be executed on all shards in a configuration, so too can certain Oracle-provided PL/SQL procedures. These specific procedure calls behave as if they were sharded DDL statements, in that they are propogated to all shards, tracked by the catalog, and run whenever a new shard is added to a configuration.

All of the following procedures can act as if they were a sharded DDL statement.

- Any procedure in the DBMS_FGA package
- Any procedure in the DBMS_RLS package
- The following procudures from the DBMS_STATS package:
  - GATHER_INDEX_STATS
  - GATHER_TABLE_STATS
  - GATHER_SCHEMA_STATS
  - GATHER_DATABASE_STATS

  - GATHER_SYSTEM_STATS
- The following procedures from the DBMS_GOLDENGATE_ADM package:
  - ADD_AUTO_CDR
  - ADD_AUTO_CDR_COLUMN_GROUP
  - ADD_AUTO_CDR_DELTA_RES
  - ALTER_AUTO_CDR
  - ALTER_AUTO_CDR_COLUMN_GROUP
  - PURGE_TOMBSTONES
  - REMOVE_AUTO_CDR
  - REMOVE_AUTO_CDR_COLUMN_GROUP
  - REMOVE_AUTO_CDR_DELTA_RES

To run one of the procedures in the same way as sharded DDL statements, do the following steps.

1. Connect to the shard catalog database using SQL*Plus as a database user with the gsm_pooladmin_role.
2. Enable sharding DDL using `alter session enable shard ddl`.
3. Run the target procedure using a sharding-specific PL/SQL procedure named `SYS.EXEC_SHARD_PLSQL`.
   This procedure takes a single CLOB argument, which is a character string specifying a fully qualified procedure name and its arguments. Note that running the target procedure without using `EXEC_SHARD_PLSQL` causes the procedure to only be run on the catalog and it is not propogated to all of the shards. Running the procedure without specifying the fully qualified name (for example, `SYS.DBMS_RLS.ADD_POLICY`) will result in an error.

For example, to run `DBMS_RLS.ADD_POLICY` on all shards, do the following from SQL*Plus after enabling shard DLL.

```
exec
sys.exec_shard_plsql('sys.dbms_rls.add_policy(object_schema
=>
          ''testuser1'',

          object_name     => ''DEPARTMENTS'',

          policy_name     => ''dept_vpd_pol'',

          function_schema => ''testuser1'',

          policy_function => ''authorized_emps'',

          statement_types => ''INSERT, UPDATE, DELETE, SELECT, INDEX'',

          update_check    => TRUE)'

          ) ;
```

Take careful note of the need for double single-quotes inside the target procedure call specification, because the call specification itself is a string parameter to `exec_shard_plsql`.

If the target procedure executes correctly on the shard catalog database, it will be queued for processing on all the currently deployed shards. Any error in the target procedure execution on the catalog is returned to the SQL*Plus session. Errors during execution on the shards can be tracked in the same way they are for DDLs.

Generating Unique Sequence Numbers Across Shards

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

Customers often need to generate unique IDs for non-primary key columns, for example order_id, when the customer_id is the sharding key. For this case among others, this feature lets you generate unique sequence numbers across shards, while not requiring you to manage the global uniqueness of a given non-primary key column in your application.

This functionality is supported by a new object, `SHARDED SEQUENCE`. A sharded sequence is created on the shard catalog but has an instance on each shard. Each instance generates monotonically increasing numbers that belong to a range which does not overlap with ranges used on other shards. Therefore, every generated number is globally unique.

A sharded sequence can be used, for example, to generate a unique order number for a table sharded by a customer ID. An application that establishes a connection to a shard using the customer ID as a key can use a local instance of the sharded sequence to generate a globally unique order number.

Note that the number generated by a sharded sequence cannot be immediately used as a sharding key for a new row being inserted into this shard, because the key value may belong to another shard and the insert will result in an error. To insert a new row, the application should first generate a value of the sharding key and then use it to connect to the appropriate shard. A typical way to generate a new value of the sharding key would be use a regular (non-sharded) sequence on the shard catalog.

If a single sharding key generator becomes a bottleneck, a sharded sequence can be used for this purpose. In this case, an application should connect to a random shard (using the global service without specifying the sharding key), get a unique key value from a sharded sequence, and then connect to the appropriate shard using the key value.

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

```
CREATE | ALTER SEQUENCE [ schema. ]sequence
   [ { INCREMENT BY | START WITH } integer
   | { MAXVALUE integer | NOMAXVALUE }
   | { MINVALUE integer | NOMINVALUE }
   | { CYCLE | NOCYCLE }
   | { CACHE integer | NOCACHE }
   | { ORDER | NOORDER }
   | { SCALE {EXTEND | NOEXTEND} | NOSCALE}
```

```
| { SHARD {EXTEND | NOEXTEND} | NOSHARD}
]
```

NOSHARD is the default for a sequence. If the SHARD clause is specified, this property is registered in the sequence object's dictionary table, and is shown using the DBA_SEQUENCES, USER_SEQUENCES, and ALL_SEQUENCES views.

When SHARD is specified, the EXTEND and NOEXTEND clauses define the behavior of a sharded sequence. When EXTEND is specified, the generated sequence values are all of length *(x+y)*, where *x* is the length of a SHARD offset of size 4 (corresponding to the width of the maximum number of shards, that is, 1000) affixed at beginning of the sequence values, and *y* is the maximum number of digits in the sequence MAXVALUE/MINVALUE.

The default setting for the SHARD clause is NOEXTEND. With the NOEXTEND setting, the generated sequence values are at most as wide as the maximum number of digits in the sequence MAXVALUE/MINVALUE. This setting is useful for integration with existing applications where sequences are used to populate fixed width columns. On invocation of NEXTVAL on a sequence with SHARD NOEXTEND specified, a user error is thrown if the generated value requires more digits of representation than the sequence's MAXVALUE/MINVALUE.

If the SCALE clause is also specified with the SHARD clause, the sequence generates scalable values within a shard for multiple instances and sessions, which are globally unique. When EXTEND is specified with both the SHARD and SCALE keywords, the generated sequence values are all of length *(x+y+z)*, where *x* is the length of a prepended SHARD offset of size 4, *y* is the length of the scalable offset (default 6), and *z* is the maximum number of digits in the sequence MAXVALUE/MINVALUE.

> **Note:**
>
> When using the SHARD clause, do not specify ORDER on the sequence. Using SHARD generates globally unordered values. If ORDER is required, create the sequences locally on each node.
>
> The SHARD keyword will work in conjunction with CACHE and NOCACHE modes of operation.

> **See Also:**
>
> *Oracle Database SQL Language Reference*

Create a Sharded Database User

Local users that only exist in the shard catalog database do not have the privileges to create schema objects in the sharded database. The first step of creating the sharded database schema is to create the sharded database user.

Create the sharded database user by connecting to the shard catalog database as SYSDBA, enabling `SHARD DDL`, and executing the `CREATE USER` command. When the sharded database user connects to the shard catalog database, the `SHARD DDL` mode is enabled by default.

> **Note:**
>
> Local users can create non-schema sharded database objects, such as tablespaces, directories, and contexts, if they enable `SHARD DDL` mode; however, they cannot create schema objects, such as tables, views, indexes, functions, procedures, and so on.
>
> Sharded objects cannot have any dependency on local objects. For example, you cannot create an all shard view on a local table.

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 subpartitition level in system-managed sharding.

4. Create a tablespace for the duplicated tables.

   In this example the duplicated table is the Products table in the sample Customers-Orders-Products schema.

```
SQL> CREATE TABLESPACE products_tsp datafile size 100m
 autoextend on next 10M maxsize unlimited
 extent management local uniform size 1m;
```

5. Create a sharded table for the root table.

   In this example, the root table is the Customers table in the sample Customers-Orders-Products schema.

```
SQL> CONNECT app_schema/app_schema_password

SQL> 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 SYSDG.
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

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 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
--    --------                                  -------------

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
```

10. 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 SYSDG.
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 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 subpartitition level in composite sharding.

5. Create a tablespace for the duplicated tables.

In this example the duplicated table is the Products table in the sample Customers-Orders-Products schema.

```
CREATE TABLESPACE products_tsp datafile size 100m autoextend on next 10M
 maxsize unlimited extent management local uniform size 1m;
```

6. Create a sharded table for the root table.

   In this example, the root table is the Customers table in the sample Customers-Orders-Products schema.

```
connect app_schema/app_schema_password
alter session enable shard ddl;

CREATE SHARDED TABLE Customers
(
  CustId      VARCHAR2(60) NOT NULL,
  FirstName   VARCHAR2(60),
  LastName    VARCHAR2(60),
  Class       VARCHAR2(10),
  Geo         VARCHAR2(8),
  CustProfile VARCHAR2(4000),
  Passwd      RAW(60),
  CONSTRAINT pk_customers PRIMARY KEY (CustId),
  CONSTRAINT json_customers CHECK (CustProfile IS JSON)
) partitionset by list(GEO)
partition by consistent hash(CustId)
partitions auto
(partitionset america values ('AMERICA') tablespace set tsp_set_1,
partitionset europe values ('EUROPE') tablespace set tsp_set_2
);
```

> **Note:**
>
> If any columns in the sharded tables contain LOBs, the CREATE SHARDED TABLE statement can include the LOB tablespace set, as shown here.

```
CREATE SHARDED TABLE Customers
(
  CustId      VARCHAR2(60)  NOT NULL,
  FirstName   VARCHAR2(60),
  LastName    VARCHAR2(60),
  Class       VARCHAR2(10),
  Geo         VARCHAR2(8)   NOT NULL,
  CustProfile VARCHAR2(4000),
  Passwd      RAW(60),
  image       BLOB,
  CONSTRAINT pk_customers PRIMARY KEY (CustId),
  CONSTRAINT json_customers CHECK (CustProfile IS JSON)
) partitionset by list(GEO)
partition by consistent hash(CustId)
partitions auto
(partitionset america values ('AMERICA') tablespace set
tsp_set_1
 lob(image) store as (tablespace set lobts1),
partitionset europe values ('EUROPE') tablespace set tsp_set_2
 lob(image) store as (tablespace set lobts2));
```

7. Create a sharded table for the other tables in the table family.

   In this example, sharded tables are created for the Orders and LineItems tables in the sample Customers-Orders-Products schema.

   Create the sequence used for the OrderId column.

   ```
   CREATE SEQUENCE Orders_Seq;
   ```

   The Orders sharded table is created first:

   ```
   CREATE SHARDED TABLE Orders
   (
     OrderId     INTEGER NOT NULL,
     CustId      VARCHAR2(60) NOT NULL,
     OrderDate   TIMESTAMP NOT NULL,
     SumTotal    NUMBER(19,4),
     Status      CHAR(4),
     constraint  pk_orders primary key (CustId, OrderId),
     constraint  fk_orders_parent foreign key (CustId)
       references Customers on delete cascade
   ) partition by reference (fk_orders_parent);
   ```

Create a sharded table for LineItems

```
CREATE SHARDED TABLE LineItems
(
  OrderId     INTEGER NOT NULL,
  CustId      VARCHAR2(60) NOT NULL,
  ProductId   INTEGER NOT NULL,
  Price       NUMBER(19,4),
  Qty         NUMBER,
  constraint  pk_items primary key (CustId, OrderId, ProductId),
  constraint  fk_items_parent foreign key (CustId, OrderId)
    references Orders on delete cascade
) partition by reference (fk_items_parent);
```

8. Create any required duplicated tables.

   In this example, the Products table is a duplicated object.

```
CREATE DUPLICATED TABLE Products
(
  ProductId  INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  Name       VARCHAR2(128),
  DescrUri   VARCHAR2(128),
  LastPrice  NUMBER(19,4)
) tablespace products_tsp;
```

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

TABLE_NAME         PARTITION_NAME      TABLESPACE_NAME
------------------- ------------------- --------------------
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.
```

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.

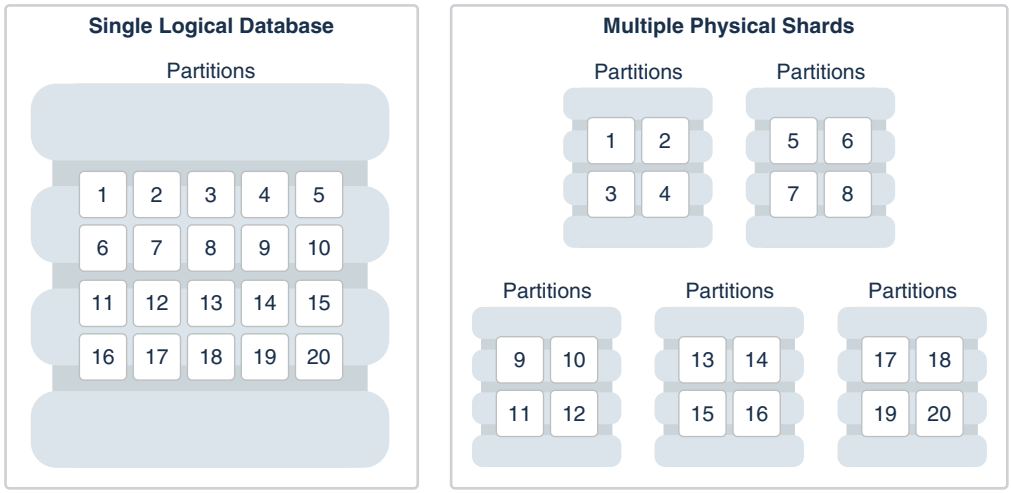
Sharding as Distributed Partitioning

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.

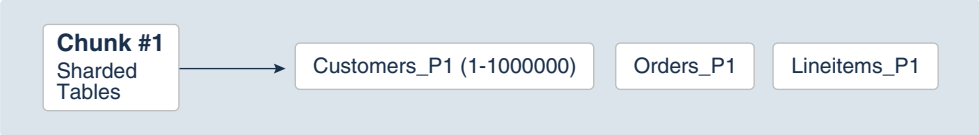
Each partition of a sharded table is stored in a separate tablespace, making the tablespace the unit of data distribution in an SDB.

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.

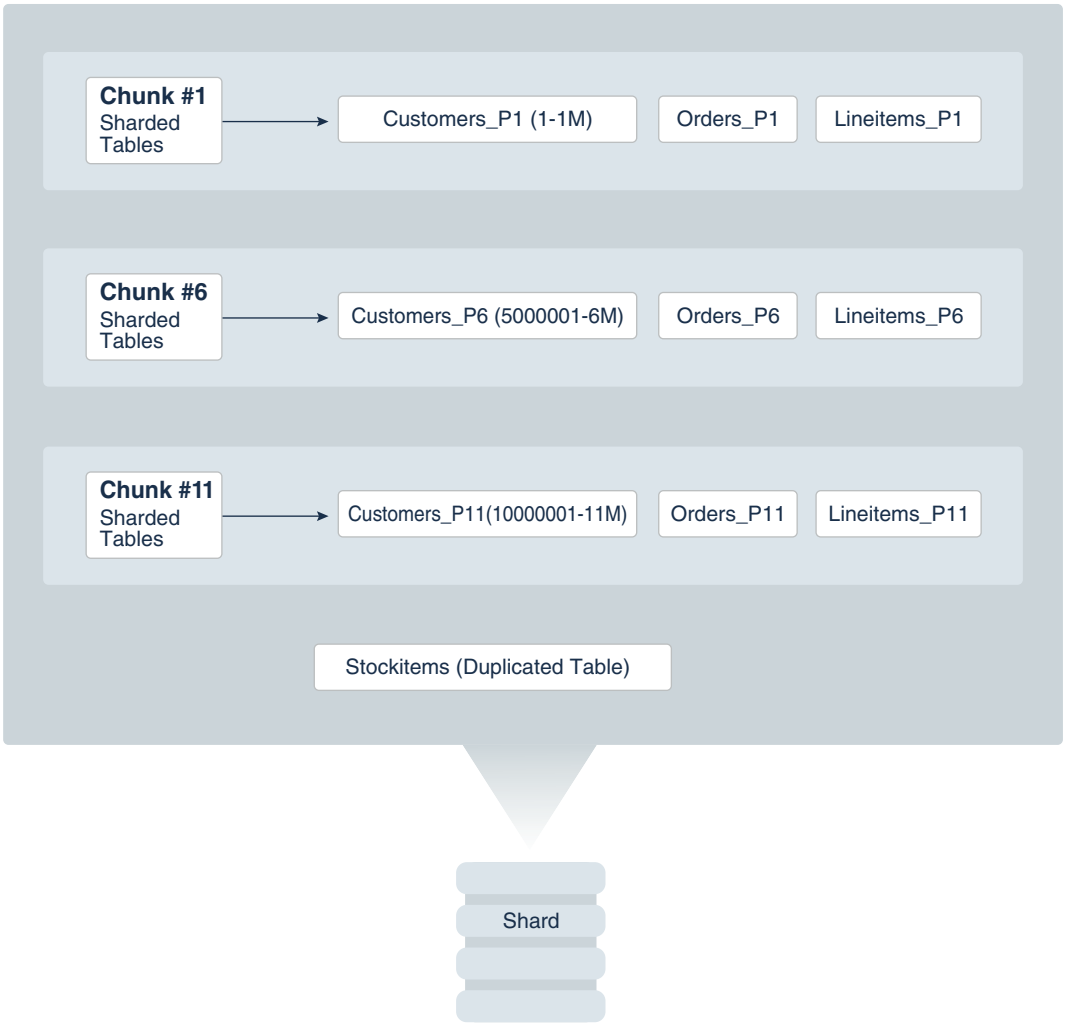
A chunk that contains corresponding partitions from the tables of Cutomers-Orders-LineItems schema is shown in the following figure.

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

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

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

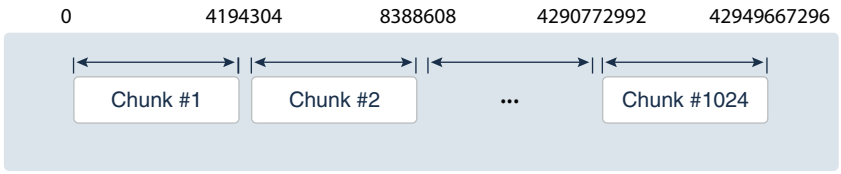
Consistent hash is a partitioning strategy commonly used in scalable distributed systems. It is different from traditional hash partitioning. With traditional hashing, the bucket number is calculated as `HF(key) % N` where HF is a hash function and N is the number of buckets. This approach works fine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

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

The implementation of consistent hashing used in Oracle 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 sharded database contains 1024 chunks, and each chunk gets assigned a range of 2^{22} hash values. Therefore partitioning by consistent hash is essentially partitioning by the range of hash values.

Figure 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 sharded database. For example, if 1024 chunks are created in a sharded database that contains 16 shards, each shard will contain 64 chunks.

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

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

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

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

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

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

```
CREATE TABLESPACE SET ts1;
```

In this case each tablespace in the set contains a single OMF file with default attributes. To customize tablespace attributes, the `USING TEMPLATE` clause (shown in

the example below) is added to the statement. The `USING TEMPLATE` clause specifies attributes that apply to each tablespace in the set.

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

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

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

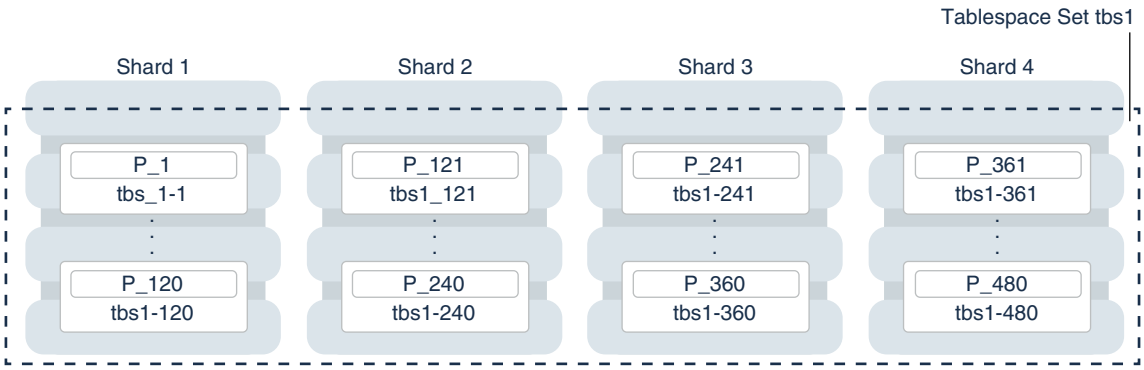
`PARTITIONS AUTO` in this statement means that the number of partitions is automatically set to the number of tablespaces in the tablespace set `ts1` (which is equal to the number of chunks) and each partition will be stored in a separate tablespace.

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

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

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

Figure 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.

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

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

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

```
 CREATE SHARDED TABLE accounts
( id             NUMBER
, account_number NUMBER
, customer_id    NUMBER
, branch_id      NUMBER
, state          VARCHAR(2) NOT NULL
, status         VARCHAR2(1)
)
PARTITION BY LIST (state)
( PARTITION p_northwest VALUES ('OR', 'WA') TABLESPACE ts1
, PARTITION p_southwest VALUES ('AZ', 'UT', 'NM') TABLESPACE ts2
```

```
, PARTITION p_northcentral VALUES ('SD', 'WI') TABLESPACE ts3
, PARTITION p_southcentral VALUES ('OK', 'TX') TABLESPACE ts4
, PARTITION p_northeast VALUES ('NY', 'VM', 'NJ') TABLESPACE ts5
, PARTITION p_southeast VALUES ('FL', 'GA') TABLESPACE ts6
)
;
```

There is no tablespace set for user-defined sharding. Each tablespace has to be created individually and explicitly associated with a shardspace. A *shardspace* is set of shards that store data that corresponds to a range or list of key values.

In user-defined sharding, a shardspace consists of a shard or a set of fully replicated shards. See Shard-Level High Availability for details about replication with user-defined sharding. For simplicity, assume that each shardspace consists of a single shard.

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

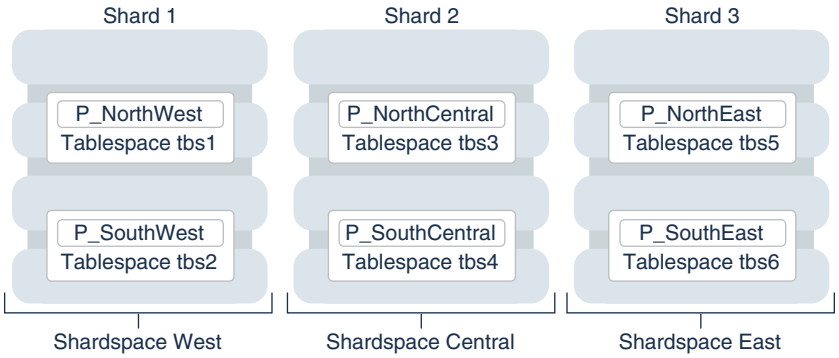
```
CREATE TABLESPACE tbs1 IN SHARDSPACE west;
CREATE TABLESPACE tbs2 IN SHARDSPACE west;

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

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

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

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

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

Figure 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 GDSCTL `MOVE CHUNK` command for each chunk that needs to be migrated.

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

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

Composite Sharding

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

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

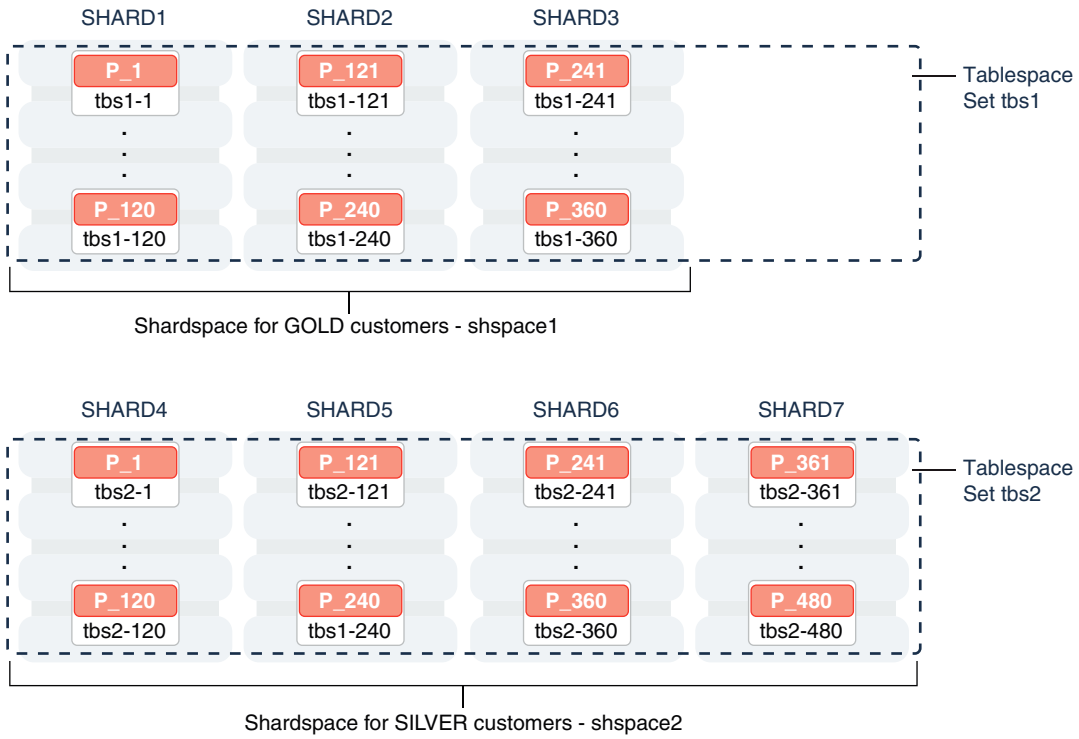
When sharding by consistent hash on a primary key, there is often a requirement to differentiate subsets of data within an SDB in order to store them in different geographic locations, allocate to them different hardware resources, or configure high availability and disaster recovery differently. Usually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

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

data across shards in each shardspace, and, at the same time, partition data across shardspaces.

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

Figure 4-4 Composite Sharding



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

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

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

Using Subpartitions with Sharding

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

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

From a manageability perspective, subpartitioning makes it possible to support the tiered storage approach by putting subpartitions into separate tablespaces and moving

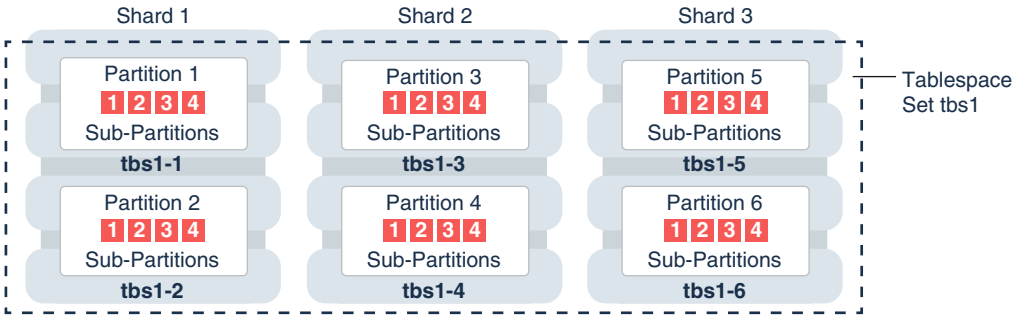
them between storage tiers. Migration of subpartitions between storage tiers can be done without sacrificing the scalability and availability benefits of sharding and the ability to perform partition pruning and partition-wise joins on a primary key.

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

```
CREATE SHARDED TABLE customers
( cust_id     NUMBER NOT NULL
, name        VARCHAR2(50)
, address     VARCHAR2(250)
, location_id VARCHAR2(20)
, class       VARCHAR2(3)
, signup_date DATE
, CONSTRAINT cust_pk PRIMARY KEY(cust_id, signup_date)
)
TABLESPACE SET ts1
PARTITION BY CONSISTENT HASH (cust_id)
SUBPARTITION BY RANGE (signup_date)
SUBPARTITION TEMPLATE
( SUBPARTITION per1 VALUES LESS THAN (TO_DATE('01/01/2000','DD/MM/YYYY')),
  SUBPARTITION per2 VALUES LESS THAN (TO_DATE('01/01/2010','DD/MM/YYYY')),
  SUBPARTITION per3 VALUES LESS THAN (TO_DATE('01/01/2020','DD/MM/YYYY')),
  SUBPARTITION future VALUES LESS THAN (MAXVALUE)
)
PARTITIONS AUTO
;
```

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

Figure 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’)
   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

Developing Applications for the Sharded Database

Application Suitability for 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.

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.

Transactions that access data associated with a single value of the sharding key are the primary use-case for a sharded database, such as lookup and update of a customer's records, subscriber documents, financial transactions, e-commerce transactions, and the like. Because all the rows in a sharded schema 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:**
>
> Developing Applications for the Sharded Database

About 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.

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, and must primarily accesses 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.

- 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.

JDBC Sharding Data Source

Oracle Java Database Connectivity (JDBC) sharding data source enables Java connectivity to a sharded database without requiring the application to provide a sharding key.

Using the JDBC sharding data source, you do not need to identify and build the sharding key and the super sharding key to establish a connection. The sharding data source scales out to sharded databases transparently because it does not involve any change to the application code.

To use the JDBC sharding data source, set the connection property `oracle.jdbc.useShardingDriverConnection` to `true` as shown here.

```
Properties prop = new Properties();
prop.setProperty("oracle.jdbc.useShardingDriverConnection", "true");
```

The default value of `oracle.jdbc.useShardingDriverConnection` is `false`.

See the *Oracle Database JDBC Developer's Guide* for more information and examples.

Related Topics

- Overview of the Sharding Data Source in *Oracle Database JDBC Developer's Guide*

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

In support of Oracle Sharding, key enhancements have been made to Oracle connection pools and drivers. JDBC, Universal Connection Pool (UCP), OCI Session Pool (OCI), and Oracle Data Provider for .NET (ODP.NET) provide APIs to pass sharding keys during the connection creation. Apache Tomcat, IBM Websphere, Oracle WebLogic Server, and JBOSS can leverage JDBC/UCP support and use sharding. PHP, Python, Perl, and Node.js can leverage OCI support.

A shard topology cache is a mapping of the sharding key ranges to the shards. Oracle Integrated Connection Pools maintain this shard topology cache in their memory. Upon the first connection to a given shard (during pool initialization or when the pool connects to newer shards), the sharding key range mapping is collected from the shards to dynamically build the shard topology cache.

Caching the shard topology creates a fast path to the shards and expedites the process of creating a connection to a shard. When a connection request is made with a sharding key, the connection pool looks up the corresponding shard on which this particular sharding key exists (from its topology cache). If a matching connection is available in the pool then the pool returns a connection to the shard by applying its internal connection selection algorithm.

A database connection request for a given sharding key that is in any of the cached topology map, goes directly to the shard (that is, bypassing the shard director). Connection Pool also subscribes to RLB notifications from the SDB and dispenses the best connection based on runtime load balancing advisory. Once the connection is established, the client executes transactions directly on the shard. After all transactions for the given sharding key have been executed, the application must return the connection to the pool and obtain a connection for another key.

If a matching connection is not available in the pool, then a new connection is created by forwarding the connection request with the sharding key to the shard director.

Once the pools are initialized and the shard topology cache is built based on all shards, a shard director outage has no impact on direct routing.

> **See Also:**
>
> Direct Routing to a Shard

Sharding APIs Supporting Direct Routing

Oracle connection pools and drivers support Oracle Sharding.

JDBC, UCP, OCI, and Oracle Data Provider for .NET (ODP.NET) recognize sharding keys as part of the connection check. Apache Tomcat, Websphere, and WebLogic leverage UCP support for sharding and PHP, Python, Perl, and Node.js leverage OCI support.

- Oracle JDBC APIs for Oracle Sharding
  Oracle Java Database Connectivity (JDBC) provides APIs for connecting to database shards in an Oracle Sharding configuration.
- Oracle Call Interface for Oracle Sharding
  Oracle Call Interface (OCI) provides an interface for connecting to database shards in an Oracle Sharding configuration.
- Oracle Universal Connection Pool APIs for Oracle Sharding
  Oracle Universal Connection Pool (UCP) provides APIs for connecting to database shards in an Oracle Sharding configuration.
- Oracle Data Provider for .NET APIs for Oracle Sharding
  Oracle Data Provider for .NET (ODP.NET) provides APIs for connecting to database shards in an Oracle Sharding configuration.

Oracle JDBC APIs for Oracle Sharding

Oracle Java Database Connectivity (JDBC) provides APIs for connecting to database shards in an Oracle Sharding configuration.

The JDBC driver recognizes the specified sharding key and super sharding key and connects to the relevant shard that contains the data. Once the connection is established to a shard, then any database operations, such as DMLs, SQL queries and so on, are supported and executed in the usual way.

A shard-aware application gets a connection to a given shard by specifying the sharding key using the database sharding APIs.

- The `OracleShardingKey` interface indicates that the current object represents an Oracle sharding key that is to be used with Oracle sharded database.
- The `OracleShardingKeyBuilder` interface builds the compound sharding key with subkeys of various supported data types. This interface uses the new JDK 8 builder pattern for building a sharding key.
- The `OracleConnectionBuilder` interface builds connection objects with additional parameters other than user name and password.
- The `OracleDataSource` class provides database sharding support with the `createConnectionBuilder` and `createShardingKeyBulider` methods.
- The `OracleXADataSource` class provides database sharding support with the `createConnectionBuilder` method
- The `OracleConnection` class provides database sharding support with the `setShardingKeyIfValid` and `setShardingKey` methods.
- The `OracleXAConnection` class provides database sharding support with the `setShardingKeyIfValid` and `setShardingKey` methods.

See the *Oracle Database JDBC Developer's Guide* for more information and examples.

Example 5-1 Sample Shard-Aware Application Code Using JDBC

The following code snippet shows how to use JDBC sharding APIs

```
OracleDataSource ods = new OracleDataSource();
   ods.setURL("jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(HOST=myhost)
(PORT=1521)(PROTOCOL=tcp))
(CONNECT_DATA=(SERVICE_NAME=myorcldbservicename)))");
   ods.setUser("hr");
   ods.setPassword("hr");

  // Employee name is the sharding Key in this example.
  // Build the Sharding Key using employee name as shown below.

   OracleShardingKey employeeNameShardKey = ods.createShardingKeyBuilder()
                                              .subkey("Mary",
JDBCType.VARCHAR)// First Name
                                              .subkey("Claire",
JDBCType.VARCHAR)// Last Name
                                              .build();

   OracleShardingKey locationSuperShardKey =
ods.createShardingKeyBuilder() // Building a super sharding key using
location as the key
                                               .subkey("US",
JDBCType.VARCHAR)
                                               .build();

   OracleConnection connection = ods.createConnectionBuilder()
                                    .shardingKey(employeeNameShardKey)
                                    .superShardingKey(locationSuperShardKey
```

```
)
                                                .build();
```

Related Topics

- JDBC Support for Database Sharding in *Oracle Database JDBC Developer's Guide*

Oracle Call Interface for Oracle Sharding

Oracle Call Interface (OCI) provides an interface for connecting to database shards in an Oracle Sharding configuration.

To make requests that read from or write to a chunk, your application must be routed to the appropriate database (shard) that stores that chunk during the connection initiation step. This routing is accomplished by using a data key. The data key enables routing to the specific chunk by specifying its sharding key or to a group of chunks by specifying its super sharding key.

In order to get a connection to the correct shard containing the chunk you wish to operate on, you must specify a key in your application before getting a connection to a sharded Oracle database for either stand-alone connections or connections obtained from an OCI Session pool. For an OCI Session pool, you must specify a data key before you check out connections from the pool.

At a high-level, the following steps have to be followed to form sharding keys and shard group keys and get a session with an underlying connection:

1. Allocate the sharding key descriptor by calling `OCIDescriptorAlloc()` and specifying the descriptor type parameter as `OCI_DTYPE_SHARDING_KEY` to form the sharding key.
2. Allocate the shard group key descriptor by calling `OCIDescriptorAlloc()` and specifying the descriptor type parameter as `OCI_DTYPE_SHARDING_KEY` to form the shard group key.
3. Call `OCISessionGet()` using the initialized authentication handle from the previous step containing the sharding key and shard group key information to get the database connection to the shard and chunk specified by the sharding key and group of chunks as specified by the shard group key.

See *Oracle Call Interface Programmer's Guide* for information about creating connections to OCI Session pools, stand-alone connections, and custom pool connections.

Related Topics

- OCI Interface for Using Shards in *Oracle Call Interface Programmer's Guide*

Oracle Universal Connection Pool APIs for Oracle Sharding

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

A shard-aware application gets a connection to a given shard by specifying the sharding key using the enhanced sharding API calls `createShardingKeyBuilder` and `createConnectionBuilder`.

At a high-level, the following steps have to be followed in making an application work with a sharded database:

1. Update the URL to reflect the shard directors and global service.
2. Set the following pool parameters at the pool level and the shard level.
   - `setInitialPoolSize` sets the initial number of connections to be created when UCP is started
   - `setMinPoolSize` sets the minimum number of connections maintained by pool at runtime
   - `setMaxPoolSize` sets maximum number of connections allowed on connection pool
   - `setMaxConnectionsPerShard` sets max connections per shard
3. Build a sharding key object with `createShardingKeyBuilder`.
4. Establish a connection using `createConnectionBuilder`.
5. Execute transactions within the scope of the given shard.

Example 5-2 Establishing a Connection Using UCP Sharding API

The following is a code fragment which illustrates how the sharding keys are built and connections established using UCP Sharding API calls.

```
...

PoolDataSource pds =
     PoolDataSourceFactory.getPoolDataSource();

  // Set Connection Pool properties
pds.setURL(DB_URL);
pds.setUser("hr");
pds.setPassword("****");
pds.setInitialPoolSize(10);
pds.setMinPoolSize(20);
pds.setMaxPoolSize(30);

// build the sharding key object

OracleShardingKey shardingKey =
    pds.createShardingKeyBuilder()
      .subkey("mary.smith@example.com", OracleType.VARCHAR2)
      .build();

  // Get an UCP connection for a shard
Connection conn =
    pds.createConnectionBuilder()
     .shardingKey(shardingKey)
     .build();
...
```

Example 5-3 Sample Shard-Aware Application Code Using UCP Connection Pool

In this example the pool settings are defined at the pool level and at the shard level.

```
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import oracle.jdbc.OracleShardingKey;
import oracle.jdbc.OracleType;
import oracle.jdbc.pool.OracleDataSource;
import oracle.ucp.jdbc.PoolDataSource;
import oracle.ucp.jdbc.PoolDataSourceFactory;

public class MaxConnPerShard
{
  public static void main(String[] args) throws SQLException
  {
    String url = "jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(HOST=shard-dir1)
(PORT=3216)
 (PROTOCOL=tcp))(CONNECT_DATA=(SERVICE_NAME=shsvc.shpool.oradbcloud)
(REGION=east)))";
    String user="testuser1", pwd = "testuser1";

    int maxPerShard = 100, initPoolSize = 20;

    PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
    pds.setConnectionFactoryClassName(OracleDataSource.class.getName());
    pds.setURL(url);
    pds.setUser(user);
    pds.setPassword(pwd);
    pds.setConnectionPoolName("testpool");
    pds.setInitialPoolSize(initPoolSize);

    // set max connection per shard
    pds.setMaxConnectionsPerShard(maxPerShard);
    System.out.println("Max-connections per shard is:
"+pds.getMaxConnectionsPerShard());

    // build the sharding key object
    int shardingKeyVal = 123;
    OracleShardingKey sdkey = pds.createShardingKeyBuilder()
        .subkey(shardingKeyVal, OracleType.NUMBER)
        .build();

    // try to build maxPerShard connections with the sharding key
    Connection[] conns = new Connection[maxPerShard];
    for (int i=0; i<maxPerShard; i++)
    {
      conns[i] = pds.createConnectionBuilder()
          .shardingKey(sdkey)
          .build();
```

```
Statement stmt = conns[i].createStatement();
      ResultSet rs = stmt.executeQuery("select sys_context('userenv',
'instance_name'),
       sys_context('userenv', 'chunk_id') from dual");
      while (rs.next()) {
        System.out.println((i+1)+" - inst:"+rs.getString(1)+",
chunk:"+rs.getString(2));
      }
      rs.close();
      stmt.close();
    }

    System.out.println("Try to build "+(maxPerShard+1)+" connection ...");
    try {
      Connection conn = pds.createConnectionBuilder()
          .shardingKey(sdkey)
          .build();

      Statement stmt = conn.createStatement();
      ResultSet rs = stmt.executeQuery("select sys_context('userenv',
'instance_name'),
       sys_context('userenv', 'chunk_id') from dual");
      while (rs.next()) {
        System.out.println((maxPerShard+1)+" - inst:"+rs.getString(1)+",
         chunk:"+rs.getString(2));
      }
      rs.close();
      stmt.close();

      System.out.println("Problem!!! could not build connection as max-
connections per
        shard exceeded");
      conn.close();
    } catch (SQLException e) {
      System.out.println("Max-connections per shard met, could not build
connection
        any more, expected exception: "+e.getMessage());
    }
    for (int i=0; i<conns.length; i++)
    {
      conns[i].close();
    }
  }
}
```

Related Topics

- UCP APIs for Database Sharding Support in *Oracle Universal Connection Pool Developer's Guide*

Oracle Data Provider for .NET APIs for Oracle Sharding

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

Using ODP.NET APIs, a shard-aware application gets a connection to a given shard by specifying the sharding key and super sharding key with APIs such as the `SetShardingKey(OracleShardingKey shardingKey, OracleShardingKey superShardingKey)` instance method in the `OracleConnection` class.

At a high level, the following steps are necessary for a .NET application to work with a sharded database:

1. Use ODP.NET, Unmanaged Driver.

   Sharding is supported with or without ODP.NET connection pooling. Each pool can maintain connections to different shards of the sharded database.

2. Use an `OracleShardingKey` class to set the sharding key and another instance for the super sharding key.

3. Invoke the `OracleConnection.SetShardingKey()` method prior to calling `OracleConnection.Open()` so that ODP.NET can return a connection with the specified sharding key and super sharding key.

   These keys must be set while the `OracleConnection` is in a Closed state, otherwise an exception is thrown.

Example 5-4 Sample Shard-Aware Application Code Using ODP.NET

```
using System;
using Oracle.DataAccess.Client;

class Sharding
{
  static void Main()
  {
    OracleConnection con = new OracleConnection
      ("user id=hr;password=hr;Data Source=orcl;");
    //Setting a shard key
    OracleShardingKey shardingKey = new
OracleShardingKey(OracleDbType.Int32, 123);
    //Setting a second shard key value for a composite key
    shardingKey.SetShardingKey(OracleDbType.Varchar2, "gold");
    //Creating and setting the super shard key
    OracleShardingKey superShardingKey = new OracleShardingKey();
    superShardingKey.SetShardingKey(OracleDbType.Int32, 1000);

    //Setting super sharding key and sharding key on the connection
    con.SetShardingKey(shardingKey, superShardingKey);
    con.Open();

    //perform SQL query
  }
}
```

Related Topics

- Database Sharding in *Oracle Data Provider for .NET Developer's Guide for Microsoft Windows*

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
  Queries that need data from multiple shards, and queries that do not specify a sharding key, cannot be routed directly by the application. Those queries require a proxy to route requests between the application and the shards.
- Multi-Shard Query Coordinator
  The Oracle Sharding **multi-shard query coordinator**, also known as "the coordinator," is part of the shard catalog. The coordinator contains the metadata of the sharded database topology and provides query processing support for sharded databases.
- Connecting to the Multi-Shard Query Coordinator
  The Oracle Sharding multi-shard query coordinator database, a component of the shard catalog, contains the metadata of the sharded topology and provides query processing support for sharded databases.
- Querying and DMLs Using Proxy Routing
  Proxy routing enables data aggregation and reporting across shards. It also provides 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 a sharding key.
- 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 is a query that must scan data from more than one shard, and the processing on each shard is independent of any other shard.
- Execution Plans for Proxy Routing
  In a multi-shard query, each shard produces an independent execution plan which is optimized for the data size and compute resources available on the shard.

About Proxy Routing in a Sharded Database

Queries that need data from multiple shards, and queries that do not specify a sharding key, cannot be routed directly by the application. Those queries require a proxy to route requests between the application and the shards.

Proxy routing is suitable for the following scenarios:

- The application cannot pass the sharding key during connect
- 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

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

How Proxy Routing Works

Multi-shard queries and queries that do not specify a sharding key are routed through the **multi-shard query coordinator** which acts as a proxy for the application requesting the data. The multi-shard query coordinator (or, "the coordinator") runs on the shard catalog or its replicas.

The coordinator uses the sharded database topology metadata 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.

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.

Proxy Routing Resiliency

Failure of the coordinator affects multi-shard 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.

Multi-Shard Query Coordinator

The Oracle Sharding **multi-shard query coordinator**, also known as "the coordinator," is part of the shard catalog. The coordinator contains the metadata of the sharded database topology and provides query processing support for sharded databases.

Multi-shard queries and queries that do not specify a sharding key are routed through the multi-shard query coordinator, which acts as a proxy for the application requesting the data. The multi-shard query coordinator runs on the shard catalog or its replicas. When a query requires proxy routing by the coordinator, the shard catalog database assumes the role of the coordinator database.

The coordinator uses the sharded database topology metadata 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.

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.

Connecting to the Multi-Shard Query Coordinator

The Oracle Sharding multi-shard query coordinator database, a component of the shard catalog, contains the metadata of the sharded topology and provides query processing support for sharded databases.

- To perform multi-shard queries, connect to the multi-shard coordinator using the `GDS$CATALOG` service on the shard catalog database.

```
sqlplus app_schema/app_schema@shardcatvm:1521/GDS\$CATALOG.oradbcloud
```

Querying and DMLs Using Proxy Routing

Proxy routing enables data aggregation and reporting across shards. It also provides 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 a sharding key.

Proxy routing in a sharded database provides a transparent mechanism to run 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 of 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 part of Q that executes on each participating shard, and CQ (Coordinator Query) is the part of Q that executes on the coordinator shard.

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

The following is an example of an aggregate query Q1 rewritten into Q1'.

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

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

There are two main elements in this process.

1. The relevant shards are identified.
2. The query is rewritten into a distributive form and iterated across the relevant shards.

During the query compilation on the coordinator database, the query compiler analyzes the predicates on the sharding key, and extracts the predicates that can be used to identify the participating shards, that is, the shards that will contribute rows for the sharded tables referenced in the query. The rest of the shards are referred to as **pruned** shards.

In the case where only one participating shard was identified, the full query is routed to that shard for execution. This is called a **single-shard query**.

If there is more than one participating shard, the query called a **multi-shard query** and is rewritten. The rewriting process takes into account the expressions computed by the query as well as the query shape.

> **Note:**
>
> In cases of data aggregation or SQL execution where a sharding key is not provided, you must accept a reduced level of performance, when compared with direct, key-based, routing.

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 are supported for:

- Equality and In-list, such as `Area = 'West'`
- Conditions containing literal, bind, or expression of literals and binds, such as

  ```
  Area = :bind
  ```

  ```
  Area = CASE :bind <10 THEN 'West' ELSE 'East' END
  ```

- `SELECT`, `UPDATE`, `DELETE`, `INSERT`, `FOR UPDATE`, and `MERGE`. `UPSERT` is not supported.

Proxy Routing for Multi-Shard Queries

A multi-shard query is a query that must scan data from more than one shard, and the processing on each shard is independent of any other shard.

A **multi-shard query** maps to more than one shard and the coordinator might need to do some processing before sending the result to the client. For example, the following query gets the number of orders placed by each customer.

```
SELECT count(*), c.custno FROM customers c, orders o WHERE c.custno =
o.custno
 GROUP BY c.custno;
```

The query is transformed to the following by the coordinator.

```
SELECT sum(count_col), custno FROM (SELECT count(*) count_col, c.custno
 FROM customers c, orders o
 WHERE c.custno = o.custno GROUP BY c.custno) GROUP BY custno;
```

The inline query block is mapped to every shard just as a remote mapped query block. The coordinator performs further aggregation and `GROUP BY` on top of the result set from all shards. The unit of execution on every shard is the inline query block.

Multi-Shard Queries and Global Read Consistency

A multi-shard query must maintain global read consistency (CR) by issuing the query at the highest common SCN across all the shards. See Specifying Consistency Levels in a Multi-Shard Query for information about how to set consistency levels.

Passing Hints in Multi-Shard Queries

Any hint specified in the original query on the coordinator is propagated to the shards.

Tracing and Troubleshooting Slow Running Multi-Shard Queries

Set the trace event `shard_sql` on the coordinator to trace the query rewrite and shard pruning. One of the common performance issues observed is when the GROUP BY is not pushed to the shards because of certain limitations of the sharding. Check if all of the possible operations are pushed to the shards and the coordinator has minimal work to consolidate the results from shards.

- Limitations in Multi-Shard DML Support
  The following DML features are not supported by multi-shard DML in Oracle Sharding.
- 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.

Limitations in Multi-Shard DML Support

The following DML features are not supported by multi-shard DML in Oracle Sharding.

- **Parallel DML** Parallel DML is not supported by multi-shard DML. The DML will always run on one shard at a time (serially) in multi-shard DML.
- **Error Logging** The `ERROR LOG` clause with DML is not supported by multi-shard DML. A user error is raised in this case.
- **Array DML** Array DML is not supported by multi-shard DML. ORA-2681 is raised in this cases.
- **RETURNING Clause** The `RETURNING INTO` clause is not supported by regular distributed DMLs; therefore, it is not supported by Oracle Sharding. ORA-22816 is raised if you try to use the `RETURNING INTO` clause in multi-shard DMLs.
- **MERGE and UPSERT** The `MERGE` statement is partially supported by Oracle Sharding, that is, a `MERGE` statement affecting only single shard is supported. ORA error is raised if a `MERGE` statement requires the modification of multiple shards.
- **Multi-Table INSERT** Multi-table inserts are not supported by database links; therefore, multi-table inserts are not supported by Oracle Sharding.
- **Updatable Join View** ORA-1779 is thrown when the updatable join view has a join on a sharded table on sharding keys. The reason for this error is that the primary key defined on a sharded table is combination of internal column `SYS_HASHVAL` + sharding key and you cannot specify `SYS_HASHVAL` in the updatable join view. Because of this restriction you cannot establish the key-preserved table resulting in raising ORA-1779.
- **Triggers**

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.

Execution Plans for Proxy Routing

In a multi-shard query, each shard produces an independent execution plan which is optimized for the data size and compute resources available on the shard.

You do not need to connect to individual shards to see the explain plan for SQL fragments. Interfaces provided in `dbms_xplan.display_cursor()` display on the coordinator the plans for the SQL segments executed on the shards, and `[V/X]$SHARD_SQL` uniquely maps a shard SQL fragment of a multi-shard query to the target shard database.

SQL Segment Interfaces for `dbms_xplan.display_cursor()`

Two interfaces can display the plan for a SQL segment executed on shards. The interfaces take shard IDs as the argument to display the plans from the specified shards. The `ALL_SHARDS` format displays the plans from all of the shards.

To print all of the plans from all shards use the `format` value `ALL_SHARDS` as shown here.

```
select * from table(dbms_xplan.display_cursor(sql_id=>:sqlid,
                                              cursor_child_no=>:childno,
                                              format=>'BASIC +ALL_SHARDS',
                                              shard_ids=>shard_ids))
```

To print selective plans from the shards, pass shard IDs in the `display_cursor()` function. For plans from multiple shards, pass an array of numbers containing shard IDs in the `shard_ids` parameter as shown here.

```
select * from table(dbms_xplan.display_cursor(sql_id=>:sqlid,
                                               cursor_child_no=>:childno,
                                               format=>'BASIC',
                                               shard_ids=>ids))
```

To return a plan from one shard pass the shard ID directly to the `shard_id` parameter, as shown here.

```
select * from table(dbms_xplan.display_cursor(sql_id=>:sqlid,
                                              cursor_child_no=>:childno,
                                              format=>'BASIC',
                                              shard_id=>1))
```

V$SQL_SHARD

`V$SQL_SHARD` uniquely maps a shard SQL fragment of a multi-shard query to the target shard database. This view is relevant only for the shard coordinator database to store a list of shards accessed for each shard SQL fragment for a given multi-shard query. Every execution of a multi-shard query can execute a shard SQL fragment on different set of shards, so every execution updates the shard IDs. This view maintains the SQL

ID of a shard SQL fragment for each REMOTE node and the SHARD IDs on which the shard SQL fragment was executed.

```
Name                                      Null?    Type
----------------------------------------- --------
----------------------------
 SQL_ID                                            VARCHAR2(13)
 CHILD_NUMBER                                      NUMBER
 NODE_ID                                           NUMBER
 SHARD_SQL_ID                                      VARCHAR2(13)
 SHARD_ID                                          NUMBER
 SHARD_CHILD_NUMBER                                NUMBER
```

- **SQL_ID** – SQL ID of a multi-shard query on coordinator
- **CHILD_NUMBER** – cursor child number of a multi-shard query on coordinator
- **NODE_ID** – ID of REMOTE node for a shard SQL fragment of a multi-shard query
- **SHARD_SQL_ID** – SQL ID of the shard SQL fragment for given remote NODE ID
- **SHARD_ID** – IDs of shards where the shard SQL fragment was executed
- **SHARD _CHILD_NUMBER**– cursor child number of a shard SQL fragment on a shard (default 0)

The following is an example of a multi-shard query on the sharded database and the execution plan.

```
SQL> select count(*) from departments a where exists (select distinct
department_id
 from departments b where b.department_id=60);
------------------------------------------------
| Id  | Operation          | Name              |
------------------------------------------------
|   0 | SELECT STATEMENT   |                   |
|   1 |  SORT AGGREGATE    |                   |
|   2 |   FILTER           |                   |
|   3 |    VIEW            | VW_SHARD_377C5901 |
|   4 |     SHARD ITERATOR |                   |
|   5 |      REMOTE        |                   |
|   6 |    VIEW            | VW_SHARD_EEC581E4 |
|   7 |     SHARD ITERATOR |                   |
|   8 |      REMOTE        |                   |
------------------------------------------------
```

A query of SQL_ID on the V$SQL_SHARD view.

```
SQL> Select * from v$sql_shard where SQL_ID = '1m024z033271u';
SQL_ID        NODE_ID   SHARD_SQL_ID  SHARD_ID
------------- -------  -------------- --------
1m024z033271u       5   5z386yz9suujt        1
1m024z033271u       5   5z386yz9suujt       11
1m024z033271u       5   5z386yz9suujt       21
1m024z033271u       8   8f50ctj1a2tbs         11
```

> **See Also:**
>
> *Oracle Database PL/SQL Packages and Types Reference*
>
> *Oracle Database Reference*

Transaction Handling in Oracle Sharding

An `UPDATE` or `DELETE` DML statement might affect only one, or it can involve multiple shards.

Shown here is a DML statement that affects the `EMPLOYEES` table.

```
UPDATE employees SET salary = salary *1.1;
```

The above `UPDATE` statement affects all of the rows in the `EMPLOYEES` table because it does not have a `WHERE` clause.

To run this `UPDATE` statement on all shards, the shard coordinator iterates over all primary shard databases, and invokes remote execution of the `UPDATE` statement. The coordinator starts a distributed transaction and performs two phase commit to guarantee the consistency of the distributed transaction. In the case of an in-doubt transaction, you must recover it manually.

Supported Query Constructs and Example Query Shapes

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

The following are restrictions on query constructs in Oracle Sharding.

- **CONNECT BY Queries** `CONNECT BY` queries are not supported.
- **MODEL Clause** The `MODEL` clause is not supported.
- **User-Defined PL/SQL in the WHERE Clause** User-defined PL/SQL is allowed in multi-shard queries only in the `SELECT` clause. If it is specified in the `WHERE` clause then an error is thrown.
- **XLATE and XML Query type** XLATE and XML Query type columns are not supported.
- **Object types** You can include object types in `SELECT` lists, `WHERE` clauses, and so on, but custom constructors and member functions of type object type are not permitted in `WHERE` clauses.

  Furthermore, for duplicated tables, non-final types, that is, object types that are created with the `NOT FINAL` keyword, cannot be used as a column data type. For sharded tables, non-final types can be used as a column data type but the column must be created with keywords `NOT SUBSTITUTABLE AT ALL LEVELS`.

> **Note:**
>
> Queries involving only duplicated tables are run on the coordinator.

The following topics show several examples of query shapes supported in Oracle Sharding.

- Queries on Sharded Tables Only
  For a single-table query, the query can have an equality filter on the sharding key that qualifies a shard. For join queries, all of the tables should be joined using equality on the sharding key.
- Queries Involving Both Sharded and Duplicated Tables
  A query involving both sharded and duplicated tables can be either a single-shard or multi-shard query, based on the predicates on the sharding key. The only difference is that the query contains a non-sharded table.

Queries on Sharded Tables Only

For a single-table query, the query can have an equality filter on the sharding key that qualifies a shard. For join queries, all of the tables should be joined using equality on the sharding key.

The following examples show queries where only sharded tables participate.

Example 5-5 Inner Join

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

Example 5-6 Left Outer Join

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

Example 5-7 Right Outer Join

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

Example 5-8 Full Outer Join

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

Queries Involving Both Sharded and Duplicated Tables

A query involving both sharded and duplicated tables can be either a single-shard or multi-shard query, based on the predicates on the sharding key. The only difference is that the query contains a non-sharded table.

> **Note:**
>
> Joins between a sharded table and a duplicated table can be on any column, using any comparison operator, = < > <= >=, or arbitrary join expressions.

Example 5-9 Inner Join

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

Example 5-10 Left or Right Outer Join

In this case, the sharded table is the first table in `LEFT OUTER JOIN`.

```
SELECT … FROM s1 LEFT OUTER JOIN r1 ON any_join_condition(s1,r1)
WHERE any_filter(s1) AND any_filter(r1)

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

In this case, the sharded table is the second table in `RIGHT OUTER JOIN`.

```
SELECT … FROM r1 RIGHT OUTER JOIN s1 ON any_join_condition(s1,r1)
WHERE any_filter(s1) AND any_filter(r1)

SELECT … FROM s1 RIGHT OUTER JOIN r1 ON any_join_condition(s1,s2)
AND filter_one_shard(s1) AND any_filter(r1)
```

In some cases, the duplicated table is the first table in `LEFT OUTER JOIN`, or the sharded table is first and it maps to a single shard, based on filter predicate on the sharding key.

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

In some cases, the duplicated table is the second table in `RIGHT OUTER JOIN`, or the sharded table is second and it maps to a single shard based on filter predicate on sharding key.

```
SELECT … FROM s1 RIGHT OUTER JOIN r1 ON any_join_condition(s1,s2)
AND any_filter (s1) AND any_filter(r1)
```

Example 5-11 Full Outer Join

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

In this case, the sharded table requires access to multiple shards:

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

Example 5-12 Semi-Join (EXISTS)

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

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

In this case, the sharded table is in a subquery that requires the participation of multiple shards.

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

Example 5-13 Anti-Join (NOT EXISTS)

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

In this case, the sharded table is in the sub-query.

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

Aggregate Functions Supported by Oracle Sharding

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

- COUNT
- SUM
- MIN
- MAX
- AVG

6

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 availablility solution for Oracle Sharding?

When To Choose Oracle GoldenGate for Shard High Availability

When should Oracle GoldenGate be employed as your high availablility 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.

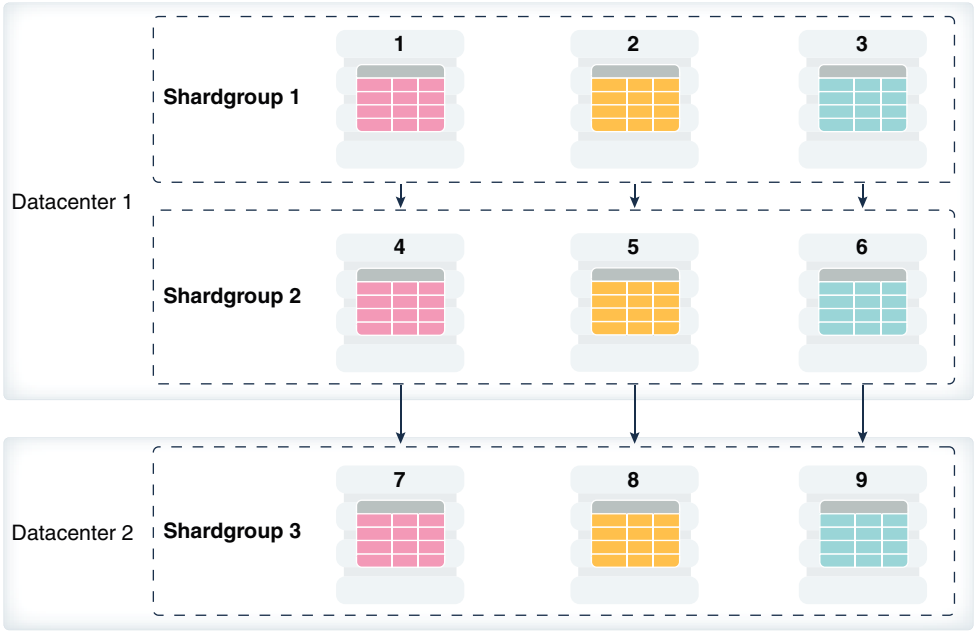
Oracle Data Guard can be used as the replication technology for sharded databases using the system-managed, user-defined, or composite method of sharding.

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 6-1 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.

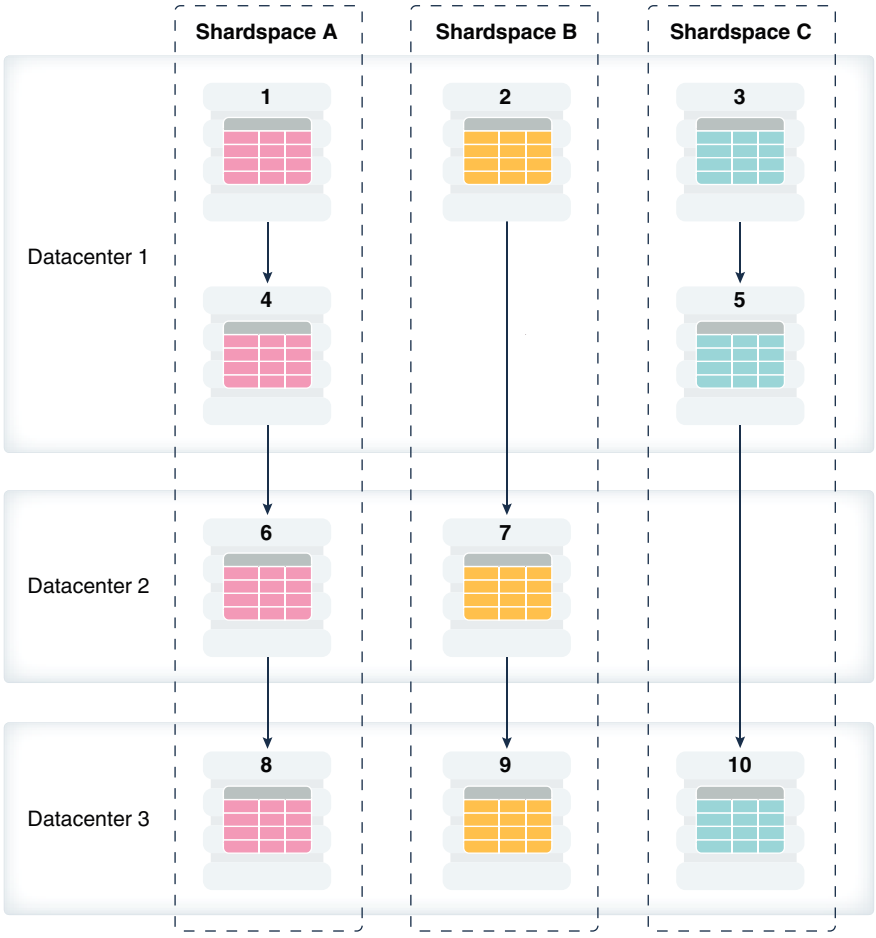
```
CREATE SHARDCATALOG –database host00:1521:shardcat –region dc1,dc2

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 6-2 User-Defined Sharding with Data Guard Replication



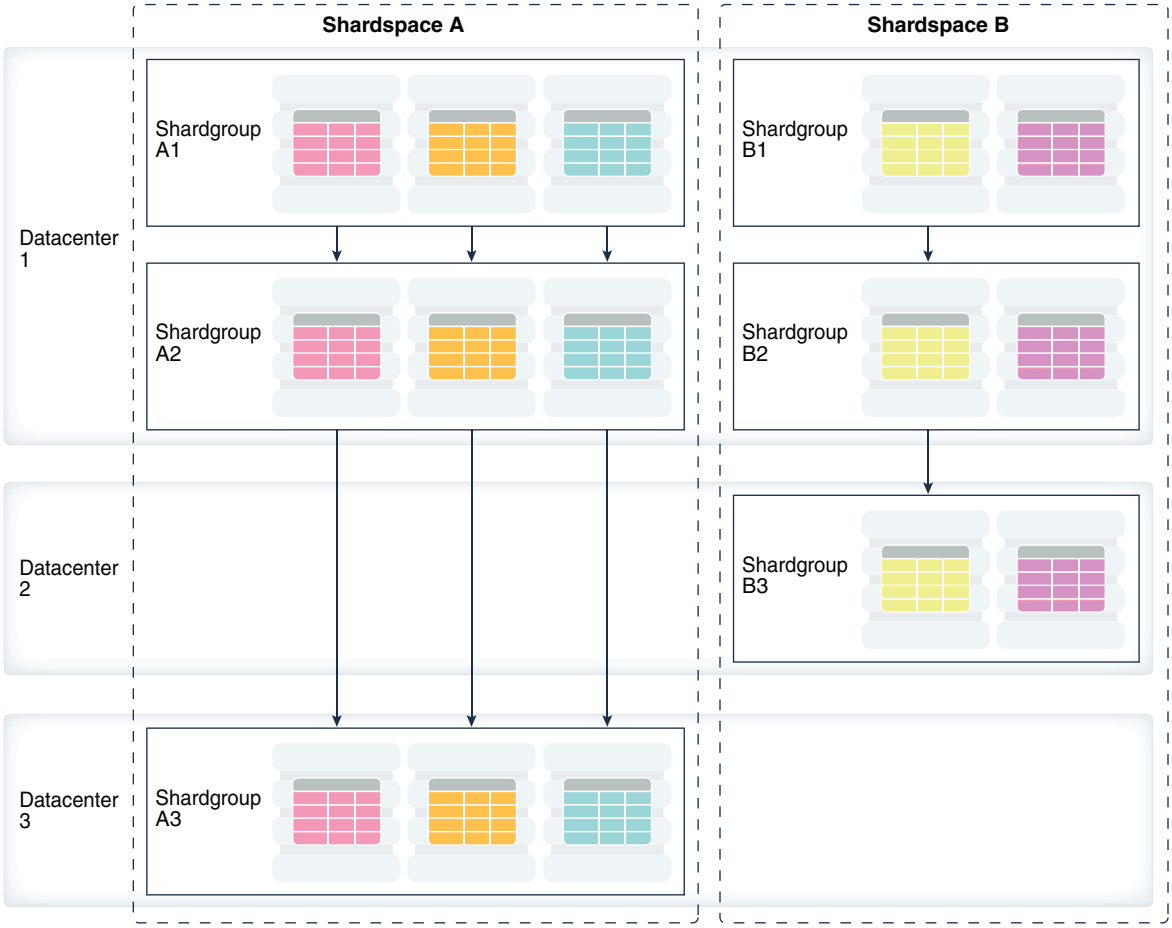
Run the following GDSCTL commands to deploy the example user-defined sharded database with Data Guard replication shown in the figure above.

```
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 6-3 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
```

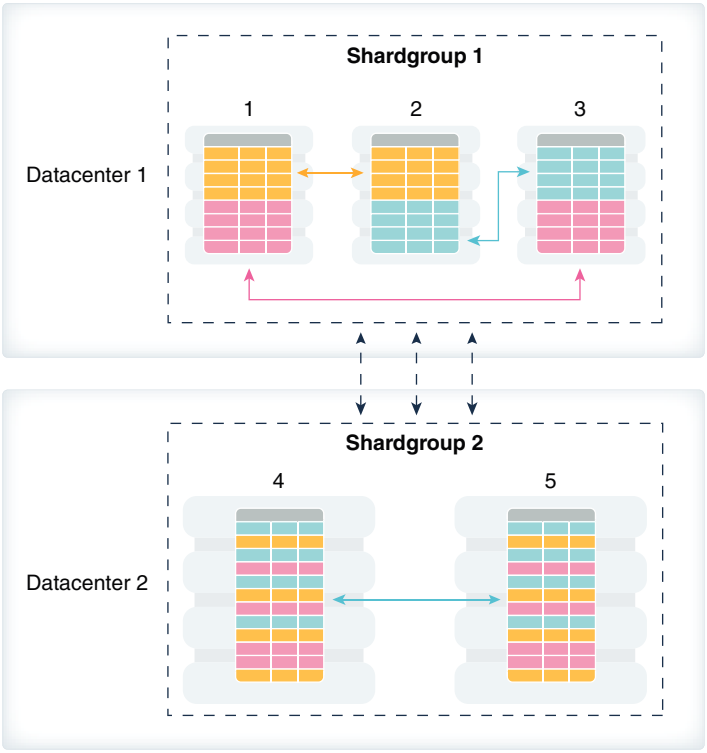
Using Oracle GoldenGate with a Sharded Database

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.

> **Note:**
>
> Oracle Database 20c supports only multitenant architecture (CDB). Oracle GoldenGate versions 12.3-19.1 only support Oracle Sharding with single-instance Oracle databases (release 11g through 19c.)

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

Figure 6-4 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:

- 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. This step 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-dir1:1521:shardcat -
localons 3841
ADD GSM -gsm gsm2 -listener 1571 –catalog shard-dir1:1521: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 -repfactor 2
ADD SHARDGROUP -shardgroup shardgroup2 -region dc2 -repfactor 2

ADD SHARD -shardgroup shardgroup1 -pwd gsmuser -connect inst2 -gg_service
    https://host01:10000/deployment_name
.
.
.
DEPLOY
```

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

See Also:

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

7

Sharded Database Deployment

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

The following topics explain the concepts and tasks 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.
- Provision and Configure Hosts and Operating Systems
  Before you install any software, review these hardware, network, and operating system requirements for Oracle Sharding.
- Install the Oracle Database Software
  Install Oracle Database on each system that will host the shard catalog, a database shard, or their replicas.
- Install the Shard Director Software
  Install the global service manager software on each system that you want to host a shard director.
- Create the Shard Catalog Database
  Use the following information and guidelines to create the shard catalog database.
- Create the Shard Databases
  The databases that will be used as shards should be created on their respective hosts.
- Configure the Sharded Database Topology
  After the databases for the shard catalog and all of the shards are configured, along with corresponding TNS listeners, you can add the sharding metadata to the shard catalog database using `GDSCTL`. The sharding metadata describes the topology used for the sharded database.
- Deploy the Sharding Configuration
  When the sharded database topology has been fully configured with `GDSCTL` commands, run the `GDSCTL DEPLOY` command to deploy the sharded database configuration.
- Post-Deployment Tasks
  After a successful deployment, complete the following tasks to make sure your new sharded database environment is ready for use.
- Example Sharded Database Deployment
  This example explains how to deploy a typical system-managed sharded database with multiple replicas, using Oracle Data Guard for high availability.
- 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.

Before You Begin

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

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

Sharded Database Deployment Roadmap

At a high level, the deployment steps are:

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

- `ADD INVITEDNODE`

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

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

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

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

Provision and Configure Hosts and Operating Systems

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

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

    The default listener port of the shard director is 1522, and the default ONS ports on most platforms are 6123 for the local ONS and 6234 for remote ONS.
  - Each and every shard must be able to reach the TNS Listener port (default 1521) of the shard catalog (both primary and standbys).
  - The TNS Listener port of each shard must be opened to all shard directors and the shard catalog.

  - All of the port numbers listed above are modifiable during the deployment configuration. However, the port numbers to be used must be known before setting up the host software.

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

Number and Sizing of Host Systems

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

- **Shard catalog host.** The shard catalog host runs the Oracle Database that serves as the shard catalog. This database contains a small amount of sharding topology metadata and any duplicated tables that are created for your application. In addition, this database acts as a query coordinator for cross-shard queries and services connections for applications that have not been written to be sharding-aware. In general, the transaction workload and size of this database are not particularly large.
- **Shard catalog database standbys (replicas).** At least one more host to contain a replica or standby of the primary shard catalog database is recommended. This host is necessary in case of a failure of the primary catalog host. In addition, while acting as a standby database, this host can also be configured to be a query coordinator for cross-shard queries.
- **Shard director host.** The shard director (global service manager) software can reside on a separate host, or it can be co-located on the same host as the shard catalog. This component of the sharding system is comprised of a network listener and several background processes used to monitor and configure a sharded configuration. If co-located on the same host as the catalog database, it must be installed in a separate Oracle Home from the catalog database, because the installation package is different than the one used for Oracle Database.
- **Multiple shard directors.** For high-availability purposes, it is recommended thaht you have more than one shard director running in a sharded system. Any additional shard directors can run on their own hosts or on the hosts running the standby shard catalog databases.
- **Shards.** In addition to the above hosts, each shard that is configured in the system should also run on its own separate host. The hosts and their configurations chosen for this task should be sized in the same way as a typical Oracle Database host depending on how much load is put on each particular shard.
- **Shard standbys (replicas).** Again, for high-availability and disaster recovery purposes, replication technology such as Oracle Data Guard or Oracle Golden Gate should be used, and replicas created for all sharded data. Additional hosts will be needed to run these replica or standby databases.

Once the number of hosts and capacity requirements for each host have been determined, provision your hardware resources as appropriate for your environment using whatever methodologies you choose. Before installing any software, you must confirm that the hosts can communicate with each other though the ports as described above. Because a sharding configuration is inherently a distributed system, it is crucial that this connectivity between and among all of the hosts is confirmed before moving on to the next steps in the deployment process. Failure to set up port access correctly will lead to failures in subsequent commands.

Install the Oracle Database Software

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

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

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

Install the Shard Director Software

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

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

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

Create the Shard Catalog Database

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

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

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

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

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

```
$ sqlplus / as sysdba

SQL> show parameter spfile
```

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

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

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

  ```
  $ sqlplus / as sysdba

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

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

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

  ```
  $ sqlplus / as sysdba

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

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

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

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

  ```
  $ sqlplus / as sysdba

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

  NAME                                 TYPE        VALUE
  ```

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

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

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

  ```
  $ sqlplus / as sysdba

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

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

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

  Note that `GSMCATUSER` is a common user in the container database. As a result, its password is the same for `CDB$ROOT` and all PDBs in the CDB. If multiple PDBs in a single CDB are to be used as catalog databases for different sharding configurations, they will all share the same `GSMCATUSER` password which can be a security concern. To avoid this, host only one shard catalog PDB per CDB, and do not unlock the `GSMCATUSER` account in any other PDBs.

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

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

  ```
  $ sqlplus / as sysdba

  SQL> alter user gsmcatuser account unlock;

  User altered.

  SQL> alter user gsmcatuser identified by gsmcatuser_password;

  User altered.

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

  User altered.
  ```

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

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

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

  ```
  $ sqlplus / as sysdba

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

  User created.

  SQL> grant gsmadmin_role to mysdbadmin;

  Grant succeeded.
  ```

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

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

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

  ```
  $ sqlplus mysdbadmin/mysdbadmin_password@catalog_connect_string

  SQL> show con_name

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

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

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

For high availability and disaster recovery purposes, it is highly recommended that you also create one or more standby shard catalog databases. From a sharding perspective, as long as the above requirements are also met on the standby

databases, and all changes to the primary shard catalog database are consistently applied to the standbys, there are no further sharding-specific configuration steps required.

Create the Shard Databases

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

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

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

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

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

  ```
  $ sqlplus / as sysdba

  SQL> alter user gsmrootuser account unlock;

  User altered.

  SQL> alter user gsmrootuser identified by gsmrootuser_password;

  User altered.

  SQL> grant SYSDG, SYSBACKUP to gsmrootuser;

  Grant succeeded.
  ```

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

  ```
  $ sqlplus / as sysdba

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

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

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

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

  ```
  $ sqlplus / as sysdba

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

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

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

  ```
  $ sqlplus / as sysdba

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

  NAME                 TYPE        VALUE
  -------------------- ----------- -----------------
  compatible           string      20.0.0
  ```

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

  ```
  $ sqlplus / as sysdba

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

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

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

  ```
  $ sqlplus / as sysdba

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

```
SQL> select force_logging from v$database;

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

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

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

  ```
  $ sqlplus / as sysdba

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

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

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

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

  ```
  $ sqlplus / as sysdba

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

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

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

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

  ```
  $ sqlplus / as sysdba

  SQL> alter session set container=shard_pdb_name;
  SQL> create or replace directory DATA_PUMP_DIR as '/u01/app/oracle/
  oradata';
  ```

```
Directory created.

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

Grant succeeded.
```

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

  ```
  $ sqlplus / as sysdba

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

  NAME TYPE VALUE
  -------------------- --------- ----------------------------
  db_file_name_convert string /u01/app/oracle/dbs/
  ```

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

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

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

  ```
  $ sqlplus / as sysdba

  SQL> alter user gsmuser account unlock;

  User altered.

  SQL> alter user gsmuser identified by gsmuser_password;

  User altered.

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

  User altered.
  ```

```
SQL> grant SYSDG, SYSBACKUP to gsmuser;

Grant succeeded.
```

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

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

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

  ```
  $ sqlplus gsmuser/gsmuser_password@shard_connect_string

  SQL> show con_name

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

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

Validate the Shard Database

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

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

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

```
$ sqlplus / as sysdba

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

This procedure will produce output similar to the following:

```
INFO: Data Guard shard validation requested.
INFO: Database role is PRIMARY.
INFO: Database name is SHARD1.
INFO: Database unique name is shard1.
INFO: Database ID is 4183411430.
```

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

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

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

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

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

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

Configure the Sharded Database Topology

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

The sharded database topology consists of the sharding method, replication (high availability) technology, the default number of chunks to be present in the sharded database, the location and number of shard directors, the numbers of shardgroups,

shardspaces, regions, and shards in the sharded database, and the global services that will be used to connect to the sharded database.

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

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

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

- Create the Shard Catalog
  Use the `GDSCTL CREATE SHARDCATALOG` command to create metadata describing the sharded database topology in the shard catalog database.
- Add and Start Shard Directors
  Add to the configuration the shard directors, which will monitor the sharding system and run background tasks in response to `GDSCTL` commands and other events, and start them.
- Add Shardspaces If Needed
  If you are using composite or user-defined sharding, and you need to add more shardspaces to complete your desired sharding topology, use the `ADD SHARDSPACE` command to add additional shardspaces.
- Add Shardgoups If Needed
  If your sharded database topology uses the system-managed or composite sharding method, you can add any necessary additional shardgroups for your application.
- Verify the Sharding Topology
  Before adding information about your shard databases to the catalog, verify that your sharding topology is correct before proceeding by using the various `GDSCTL CONFIG` commands.
- Add the Shard CDBs
  Add the CDBs containing the shard PDBs to the sharding configuration with the `ADD CDB` command.
- Add the Shard PDBs
  Use the `ADD SHARD` command to add the shard PDB information to the shard catalog, then verify it with the `CONFIG SHARD` command.
- Add Host Metadata
  Add all of the host names and IP addresses of your shard hosts to the shard catalog.

Create the Shard Catalog

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

Note that once you run `CREATE SHARDCATALOG`, and the rest of the sharding metadata has been created, there are several metadata properties that cannot be modified without recreating the entire sharded database from scratch. These include the sharding method (system-managed, composite, user-defined), replication technology

(Oracle Data Guard, Oracle GoldenGate), default number of chunks in the database, and others. Make sure that you consult the GDSCTL reference documentation for the complete list of possible command options and their defaults.

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

Shard Catalog Connect String

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

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

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

The following is a simple tnsnames.ora entry.

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

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

  System-Managed Sharding Method

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

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

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

  Composite Sharding Method

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

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

User-Defined Sharding Method

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

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

Future Connections to the Shard Catalog

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

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

Add and Start Shard Directors

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

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

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

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

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

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

   (Optional) Enter the result of the step here.

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

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

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

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

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

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

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

Add Shardspaces If Needed

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

- Run ADD SHARDSPACE as shown here.

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

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

Add Shardgoups If Needed

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

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

- Run ADD SHARDGROUP to add shardgroups to the configuration.

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

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

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

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

Verify the Sharding Topology

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

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

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

```
GDSCTL> config

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

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

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

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

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

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

Services
```

```
-----------------------

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

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

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

Add the Shard CDBs

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

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

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

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

   ```
   GDSCTL> config cdb
   ```

Add the Shard PDBs

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

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

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

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

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

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

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

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

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

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

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

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

Add Host Metadata

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

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

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

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

(Optional) Enter task prerequisites here.

1. View a list of trusted hosts.

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

   ```
   GDSCTL> config vncr
   ```

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

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

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

3. Run the REMOVE INVITEDNODE command to manually remove any host names that are not necessary and cannot be resolved from all of the hosts.

4. Run the ADD INVITEDNODE command to manually add all host names and IP addresses of your shard hosts to the shard catalog metadata.

   ```
   GDSCTL> add invitednode 127.0.0.1
   ```

Deploy the Sharding Configuration

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

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

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

What Happens During Deployment

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

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

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

What Does a Successful Deployment Look Like?

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

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

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

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

What To Do If Something Is Not Right

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

Post-Deployment Tasks

After a successful deployment, complete the following tasks to make sure your new sharded database environment is ready for use.

- Create and Start Global Database Services
  After the shards are successfully deployed, and the correct status has been confirmed, create and start global database services on the shards to service incoming connection requests from your application.
- Verify Shard Status
  Once you complete the DEPLOY step in your sharding configuration deployment, verify the detailed status of a shard
- Where to Go Next
  Once the services are started, and you have verified that the sharded database component configurations are correct, your sharded database is ready for application schema creation and incoming client connection requests.

Create and Start Global Database Services

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

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

Example 7-1 Add and start a global service that runs on all of the primary shards

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

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

Example 7-2 Add and start a global service for the read-only workload to run on the standby shards

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

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

Example 7-3 Verify the status of the global services

```
GDSCTL> config service

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


GDSCTL> status service
Service "oltp_rw_srvc.orasdb.oradbcloud" has 2 instance(s). Affinity:
ANYWHERE
   Instance "orasdb%1", name: "cdb1_pdb1", db: "cdb1_pdb1", region:
"region1", status: ready.
   Instance "orasdb%21", name: "cdb3_pdb2", db: "cdb3_pdb2", region:
"region1", status: ready.
Service "oltp_ro_srvc.orasdb.oradbcloud" has 2 instance(s). Affinity:
ANYWHERE
   Instance "orasdb%11", name: "cdb2_pdb1", db: "cdb2_pdb1", region:
"region2", status: ready.
   Instance "orasdb%31", name: "cdb4_pdb2", db: "cdb4_pdb2", region:
"region2", status: ready.
```

Verify Shard Status

Once you complete the DEPLOY step in your sharding configuration deployment, verify the detailed status of a shard

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

```
GDSCTL> config shard -shard cdb1_pdb1
Name: cdb1_pdb1
Shard Group: shardgroup_primary
Status: Ok
State: Deployed
Region: region1
Connection string:shard_connect_string
SCAN address:
ONS remote port: 0
Disk Threshold, ms: 20
CPU Threshold, %: 75
Version: 20.0.0.0
Failed DDL:
DDL Error: ---
Management error:
Failed DDL id:
Availability: ONLINE
Rack:


Supported services
------------------------
Name Preferred Status
```

```
---- --------- ------
oltp_ro_srvc Yes Enabled
oltp_rw_srvc Yes Enabled
```

Where to Go Next

Once the services are started, and you have verified that the sharded database component configurations are correct, your sharded database is ready for application schema creation and incoming client connection requests.

1. To begin creating users and other schema objects, go to Sharded Database Schema Design.
2. To learn how to make your application sharding-aware, go to Developing Applications for the Sharded Database.
3. To move existing data into your sharded database, go to Migrating Data to a Sharded Database

Example Sharded Database Deployment

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

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

You are not required to map data to shards in system-managed sharding, because the data is automatically distributed across shards using partitioning by consistent hash. The partitioning algorithm evenly and randomly distributes data across shards. For more conceptual information about the system-managed sharded Database, see System-Managed Sharding.

- Example Sharded Database Topology
  Consider the following system-managed sharded database configuration, where shardgroup 1 contains the primary shards, while shardgroups 2 and 3 contain standby replicas.
- Deploy the Example Sharded Database
  Do the following steps to deploy the example system-managed sharded database with multiple replicas, using Oracle Data Guard for high availability.

Example Sharded Database Topology

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

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

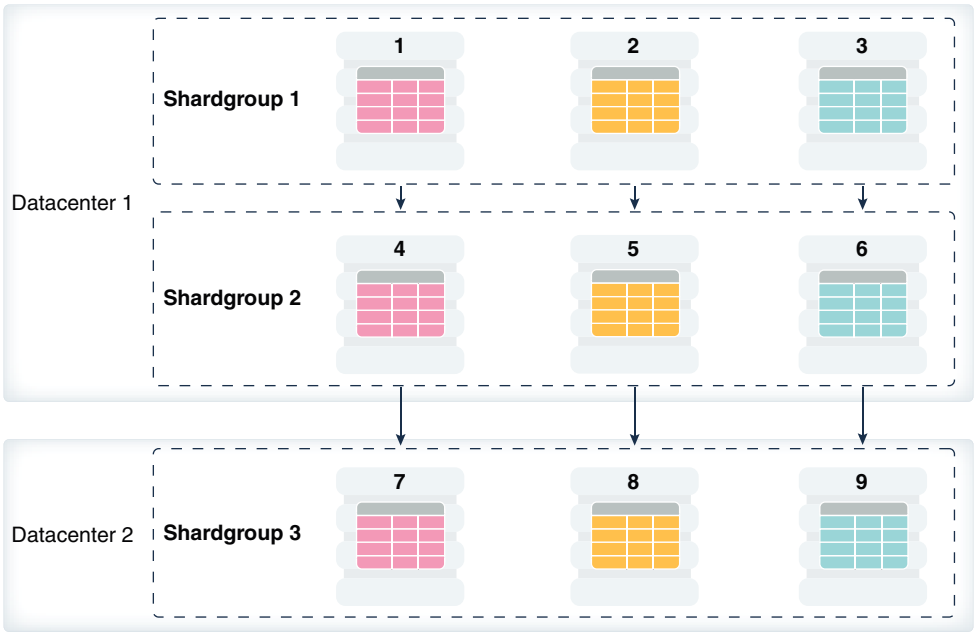


Table 7-1 Example System-Managed Topology Host Names

Topology Object	Description
Shard Catalog Database	Every sharded database topology requires a shard catalog. In our example, the shard catalog database has 2 standbys, one in each data center. Primary • Data center = 1 • Host name = cathost • DB_UNIQUE_NAME = catcdb • PDB name = catpdb • Connect service name = catpdb Active Standby • Data center = 1 • Host name = cathost1 Standby • Data center = 2 • Host name = cathost2
Regions	Because there are two datacenters involved in this configuration, there are two corresponding regions created in the shard catalog database. Data center 1 • Region name = dc1 Data center 2 • Region name = dc2

Table 7-1 (Cont.) Example System-Managed Topology Host Names

Topology Object	Description
Shard Directors (global service managers)	Each region requires a shard director running on a host within that data center. Data center 1 • Shard director host name = gsmhost1 • Shard director name = gsm1 Data center 2 • Shard director hast name = gsmhost2 • Shard director name = gsm2
Shardgroups	Data center 1 • sg1 • sg2 Data center 2 • sg3
Shards	• Host names = shardhost1, …, shardhost9 • DB_UNIQUE_NAME = cdb1, …, cdb9 • PDB names = pdb1, pdb2, pdb3 PDB names on standby replicas are the same as the PDB names on their corresponding primaries

Deploy the Example Sharded Database

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

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

   See Provision and Configure Hosts and Operating Systems for details.

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

   See Install the Oracle Database Software for details.

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

   See Install the Shard Director Software for details.

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

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

   See Create the Shard Catalog Database for details.

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

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

   - shardhost1 (cdb1/pdb1) replicas on shardhost4 (cdb4) and shardhost7 (cdb7)
   - shardhost2 (cdb2/pdb2) replicas on shardhost5 (cdb5) and shardhost8 (cdb8)

- shardhost3 (cdb3/pdb3) replicas on shardhost6 (cdb6) and shardhost9 (cdb9)

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

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

See Create the Shard Databases for details.

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

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

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

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

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

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

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

   See Add and Start Shard Directors for details.

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

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

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

```
add shard -connect shardhost1.yourdomain.com:1521/
pdb1.yourdomain.com -pwd gsmuser_password -shardgroup sg1 -cdb cdb1
add shard -connect shardhost2.yourdomain.com:1521/
pdb2.yourdomain.com -pwd gsmuser_password -shardgroup sg1 -cdb cdb2
add shard -connect shardhost3.yourdomain.com:1521/
```

```
pdb3.yourdomain.com -pwd gsmuser_password -shardgroup sg1 -cdb cdb3
add shard -connect shardhost4.yourdomain.com:1521/
pdb1.yourdomain.com -pwd gsmuser_password -shardgroup sg2 -cdb cdb4
add shard -connect shardhost5.yourdomain.com:1521/
pdb2.yourdomain.com -pwd gsmuser_password -shardgroup sg2 -cdb cdb5
add shard -connect shardhost6.yourdomain.com:1521/
pdb3.yourdomain.com -pwd gsmuser_password -shardgroup sg2 -cdb cdb6
add shard -connect shardhost7.yourdomain.com:1521/
pdb1.yourdomain.com -pwd gsmuser_password -shardgroup sg3 -cdb cdb7
add shard -connect shardhost8.yourdomain.com:1521/
pdb2.yourdomain.com -pwd gsmuser_password -shardgroup sg3 -cdb cdb8
add shard -connect shardhost9.yourdomain.com:1521/
pdb3.yourdomain.com -pwd gsmuser_password -shardgroup sg3 -cdb cdb9
```

See Add Shardgoups If Needed, Add the Shard CDBs, and Add the Shard PDBs for details.

d. Use the `CONFIG VNCR` and `ADD INVITEDNODE` commands to validate that all of the VNCR entries are valid and sufficient for a successful deployment.

   See Add Host Metadata for details.

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

   See Deploy the Sharding Configuration for details.

f. Add and start services for read-write and read-only access to the sharded database.

```
add service -service oltp_rw_srvc -role primary
start service -service oltp_rw_srvc
add service -service oltp_ro_srvc -role physical_standby
start service -service oltp_ro_srvc
```

   See Create and Start Global Database Services for details.

7. You can use the `GDSCL CONFIG`, `CONFIG SHARD`, and `CONFIG SERVICE` commands to validate that all of the shards and services are online and running.

   See Verify Shard Status for details.

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.
   >
   > `ENCRYPTION_WALLET_LOCATION` parameter is being deprecated. You are advised to use the `WALLET_ROOT` static initialization and `TDE_CONFIGURATION` dynamic initialization parameter instead.

   With shard DDL disabled, issue the following statement.

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

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

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

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

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

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

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

   (Optional) Enter the result of the step here.

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

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

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

   On the catalog with shard DDL enabled

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

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

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

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

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

8
Migrating to a Sharded Database

Oracle Sharding provides some tools and guidelines for migrating your database to an Oracle Sharding configuration.

- Using the Sharding Advisor Database Migration Tool
  Sharding Advisor simplifies the migration of your existing, non-sharded Oracle database to a sharded database, by analyzing your workload and database schema, and recommending the most effective Oracle Sharding configurations.
- 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.

Using the Sharding Advisor Database Migration Tool

Sharding Advisor simplifies the migration of your existing, non-sharded Oracle database to a sharded database, by analyzing your workload and database schema, and recommending the most effective Oracle Sharding configurations.

Jump to

- About Sharding Advisor
  Learn what Sharding Advisor is, why you need it, and how it works.
- Run Sharding Advisor
  Run the Sharding Advisor command-line tool against your existing, non-sharded Oracle Database to obtain recommended Oracle Sharding configurations.
- Run Sharding Advisor on a Non-Production System
  To minimize the impact on a live production system, you can run the Sharding Advisor on a copy of the database schema and workload, located on a different server than the production system.
- Review Sharding Advisor Output
  Sharding Advisor discovers the table families for each potential sharding column that it extracts from the query workload, and ranks the table families based on query classification rules and a ranking algorithm.
- Choose a Sharding Advisor Recommended Configuration
  There are some aspects of database sharding to take into consideration when deciding which configuration to choose for your sharded database.

About Sharding Advisor

Learn what Sharding Advisor is, why you need it, and how it works.

The Sharding Advisor analysis provides you with the information you need to

- Maximize query workload performance
- Minimize multi-shard operations requiring cross-shard joins

- Maximize parallelism for complex queries (spread query execution across all shards)
- Minimize the amount of duplicated data on each shard

The Sharding Advisor is a client-side, command-line tool that you run against any non-sharded, production, 10g or later release, Oracle Database that you are considering migrating to an Oracle Sharding environment.

Sharding Advisor is installed as a standalone tool, and connects to your database using authenticated OCI connections.

To get an understanding of your schema and other preferences, Sharding Advisor asks you questions as part of an interactive dialog.

Sharding Advisor then connects to the existing non-sharded database, also called the **source**, analyzes its schema and query workload, and produces a set of alternative designs for the sharded database, including recommendations for an effective sharding key, which tables to shard, and which tables to duplicate on all shards.

Sharding configurations are ranked in terms of query performance, with the ranking favoring configurations that maximize single and multi-shard queries that do not require cross-shard joins, while minimizing multi-shard queries that require cross-shard joins.

You choose the design that best fits your requirements. The designs are ranked by the advisor, so if you don't have specific preferences you can choose the highest ranked design by default.

> **Note:**
>
> There are restrictions to Sharding Advisor capabilities:
> The source database must be Oracle Database 10g or later release.
>
> If you cannot run the Sharding Advisor against the live production database, you can run the Sharding Advisor on a different server that has the schema and workload imported from the production database.
>
> Sharding Advisor discovers the table families based on primary key-foreign key relationships. If the schema does not have any primary key-foreign key constraints, sharding by `PARENT` clause is recommended.
>
> Currently, Sharding Advisor recommends only single-table family, system-managed sharding (sharding by reference) configurations if the source database has foreign key constraints; otherwise, Sharding Advisor recommends sharding using the `PARENT` clause.

Run Sharding Advisor

Run the Sharding Advisor command-line tool against your existing, non-sharded Oracle Database to obtain recommended Oracle Sharding configurations.

Run the Sharding Advisor from the command line, as shown here.

```
$ gwsadv -u username -p password -c –w sch=(schema1,schema2
```

Where `-u` and `-p` are the user name and password of the user that runs the Sharding Advisor.

Use the capture workload parameter, `-c`, the first time you run Sharding Advisor against an existing query workload, to capture the predicate information from the source's `GV$SQL_PLAN_STATISTICS_ALL` view. You don't need to use `-c` in subsequent queries on the same workload.

The required `-w` flag indicates that Sharding Advisor uses the query workload for sharding configuration generation and ranking.

In this case, the `sch` parameter specifies a list of schemas to run Sharding Advisor against. There are several other options you can use with Sharding Advisor, detailed in Sharding Advisor Usage and Options.

Run Sharding Advisor on a Non-Production System

To minimize the impact on a live production system, you can run the Sharding Advisor on a copy of the database schema and workload, located on a different server than the production system.

To get the same results as if it were the live production system, the production database schema and workload can be exported using the Oracle Data Pump utilities and copied to a different server. Then you can run Sharding Advisor on the imported schema.

You only export the database schema and system tables. There is no need to export the actual data.

1. The following procedure uses the HR schema as an example.

   Do the following steps on the source (production) database server.

1. Export the schema using Data Pump Export.

   ```
   > expdp system/password SCHEMAS=HR DIRECTORY=HR_DIR
   CONTENT=METADATA_ONLY
    DUMPFILE=hr_metadata.dmp LOGFILE=hr_exp.lst
   ```

2. Export the Automatic Work Repository (AWR) snapshot.

   ```
   SQL> @$ORACLE_HOME/rdbms/admin/awrextr.sql
   ```

2. Do the following steps on the target database server.
3. Copy the dump files from the source to the target.

   For example, copy the dump files to `/scratch/dump`.

4. Create a user that can run Sharding Advisor on the schema.

   ```
   SQL> CREATE USER hr IDENTIFIED BY password;
   ```

5. Create (or replace) the dump file directory variable that Data Pump Import can reference.

```
SQL> CREATE DIRECTORY HR_DIR AS '/scratch/dump'

SQL> REPLACE DIRECTORY  HR_DIR AS '/scratch/dump'
```

6. Import the schema.

```
> impdp system/password DIRECTORY=HR_DIR DUMPFILE=hr.dmp
LOGFILE=imp.lst SCHEMAS=HR
```

7. Load the AWR data.

```
SQL> @$ORACLE_HOME/rdbms/admin/awrload.sql
```

8. Now you can run Sharding Advisor on the target, non-production, copy of the database with the user you created.

```
> gwsadv –u hr –p password –c -awr_snap_begin begin_timestamp –
awr_snap_end end_timestamp -w
```

Review Sharding Advisor Output

Sharding Advisor discovers the table families for each potential sharding column that it extracts from the query workload, and ranks the table families based on query classification rules and a ranking algorithm.

To review the sharding configurations and related information that is owned by the user running Sharding Advisor, you can query the following output database tables, which are stored in the same schema as your source database.

- `SHARDINGADVISOR_CONFIGURATIONS` has one row for each table in a ranked sharded configuration, and provides details for each table, such as whether to shard or duplicate it, and if sharded, its level in a table family hierarchy, its parent table, root table sharding key, foreign key reference constraints, and the estimated size per shard.
- `SHARDINGADVISOR_CONFIGDETAILS` has one row for each ranked sharding configuration, and provides details for each ranked sharding configuration, such as the number and collective size, per shard, of the sharded tables, and the number and collective size of the duplicated tables. It also provides the number of single shard and multi-shard queries to expect in production, as well as the number of multi-shard queries requiring cross-shard joins, based on your source database's current workload, and an estimated cost.
- `SHARDINGADVISOR_QUERYTYPES`, for each query in the workload, lists the query type for each sharding configuration. Note that the same query can be of a different query type depending on the sharding configuration.

Because the Sharding Advisor output is contained in regular database tables, you can run many kinds of SQL queries against them to look at the output from different perspectives.

For example, to display the sharding configurations in ranking order, run

```
SELECT rank, tableName as tname, tabletype as type,
           tablelevel as tlevel, parent, shardby as shardBy,
           shardingorreferencecols as cols, unenforceableconstraints,
           sizeoftable
FROM SHARDINGADVISOR_CONFIGURATIONS
ORDER BY rank, tlevel, tname, parent;
```

For details about the Sharding Advisor output tables and more example queries see Sharding Advisor Output Tables and Sharding Advisor Output Review SQL Examples

Choose a Sharding Advisor Recommended Configuration

There are some aspects of database sharding to take into consideration when deciding which configuration to choose for your sharded database.

Increasing the number of shards will result in higher availability and scalability of the sharded database.

Minimizing duplicated data can conflict with your desire to minimize multi-shard queries that require joins across multiple shards. Because joins in a sharded database are usually performed on related data, storing related data in the same shard can dramatically speed up execution of such joins.

The overall cost, in terms of query workload, of the recommended sharding configurations is based on the number of each query type (single shard, multi-shard, and multi-shard with cross-shard joins) in the workload, where multi-shard queries with cross-shard joins have the highest cost, and single shard queries have the lowest cost. The cost information is in the `COST` column of the Sharding Advisor `SHARDINGADVISOR_CONFIGDETAILS` output table.

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.

- 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.

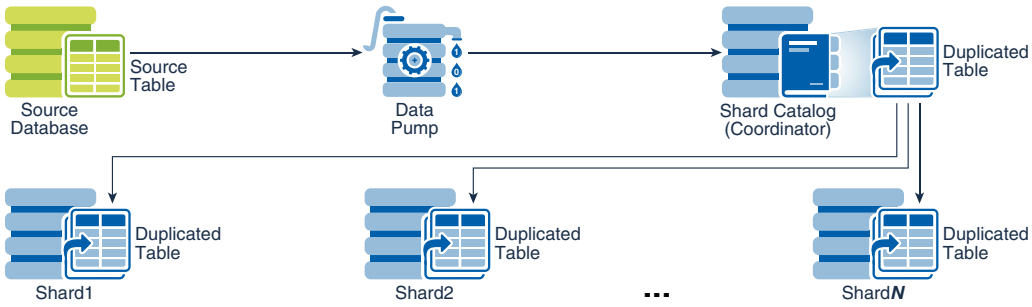
General Guidelines for Loading Data into a Sharded Database

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.

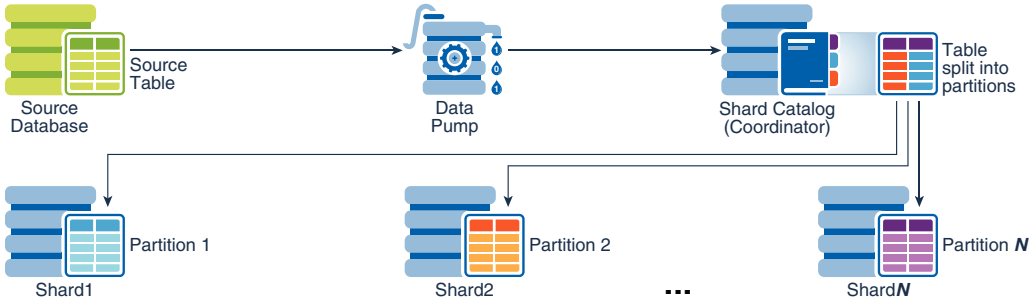
Figure 8-1 Loading Duplicated Tables



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.

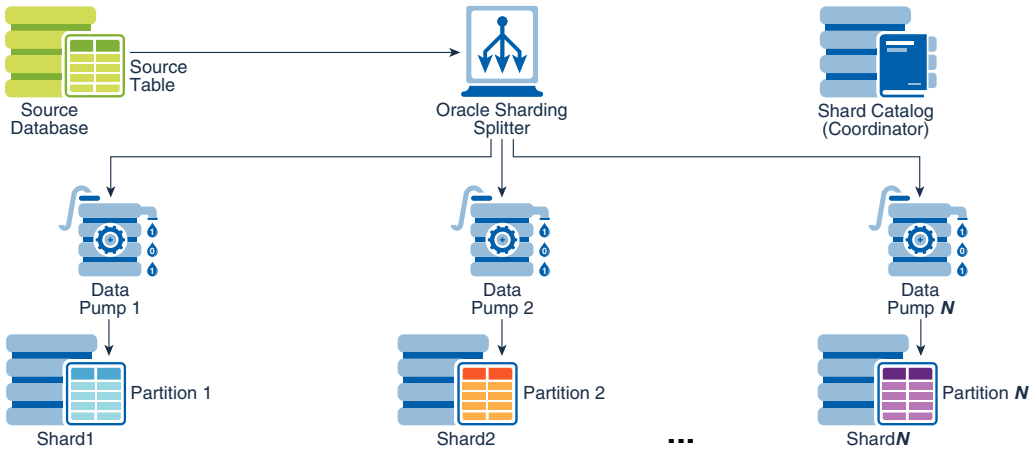
Figure 8-2 Loading Sharded Tables Using the Shard Catalog



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 8-3 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.

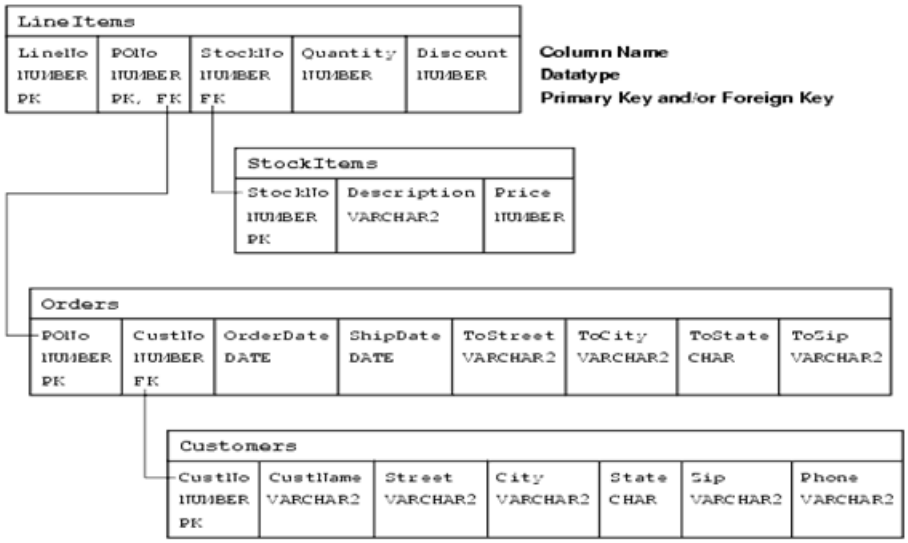
Migrating the Schema

Before the existing database can be migrated to the sharded database, you must decide how to organize the sharded database.

You must decide on 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 8-4



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 constrains.

- 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 Cusomters (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.

> **See Also:**
>
> Partitions, Tablespaces, and Chunks

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 you 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 SESSTION 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
```

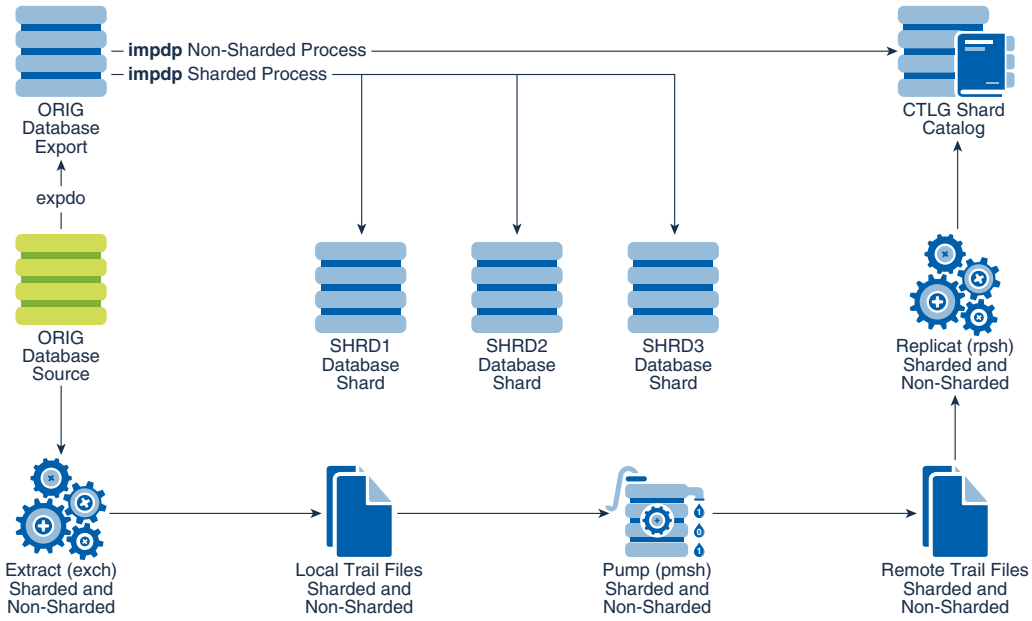
Migrating Data to Sharded Tables Without Downtime

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 8-5 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 GoldeGate 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
-- pmsh copies local trail data to ctlgnode remote trail location
add extract pmsh, exttrailsource ./dirdat/et
-- associate the remote trail with pmsh
```

```
add rmttrail ./dirdat/et, extract pmsh
-- 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 pmsh …
…
```

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, pmsh.

```
view parms pmsh
-- first line must be extract followed the name
extract pmsh
-- 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 `FLASBACK_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 sahrd 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 rpsh
```

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 POManager
{
  // Connection factory for the sharded database used by Sample App
  private OracleDataSource ods;

  //
  // Construct POManager 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 POManager(String yourURL, String yourUser, String yourPwd)
    throws SQLException
  {
      ods = new OracleDataSource();
      ods.setURL(yourURL);
      ods.setUser(yourUser);
      ods.setPassword(yourPwd);
 } // POManager
```

As shown above, the POManager 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(shardingKey);
  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 kay 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 changeQuantifyWithinParent() 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


} // POManager
```

9
Sharded Database Administration

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

> **Note:**
>
> A multitenant container database is the only supported architecture in Oracle Database 20c. While the documentation is being revised, legacy terminology may persist. In most cases, "database" and "non-CDB" refer to a CDB or PDB, depending on context. In some contexts, such as upgrades, "non-CDB" refers to a non-CDB from a previous release.

The following topics describe sharded database administration in detail:

- Managing the Sharding-Enabled Stack
- Managing Oracle Sharding Database Users
- 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.
- Propagation of Parameter Settings Across Shards
  When you configure system parameter settings at the shard catalog, they are automatically propagated to all shards of the sharded database.
- Migrating a Non-PDB Shard to a PDB
  Do the following steps if you want to migrate shards from a traditional single-instance database to Oracle multitenant architecture. Also, you must migrate to a multitenant architecture before upgrading to Oracle Database 20c.
- 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.

Managing Oracle Sharding Database Users

This section describes the database users specific to Oracle Sharding. It contains the following topics:

- About the GSMUSER Account
  The GSMUSER account is used by GDSCTL and global service managers to connect to databases in a GDS configuration.
- About the GSMROOTUSER Account
  GSMROOTUSER is a database account specific to Oracle Sharding that is only used when pluggable database (PDB) shards are present. The account is used by GDSCTL and global service managers to connect to the root container of container databases (CDBs) to perform administrative tasks.

About the GSMUSER Account

The GSMUSER account is used by GDSCTL and global service managers to connect to databases in a GDS configuration.

GSMUSER exists by default on any Oracle database. In an Oracle Sharding configuration, the account is used to connect to shards instead of pool databases, and it must be granted both the SYSDG and SYSBACKUP system privileges after the account has been unlocked.

The password given to the GSMUSER account is used in the `gdsctl add shard` command. Failure to grant SYSDG and SYSBACKUP to GSMUSER on a new shard causes `gdsctl add shard` to fail with an ORA-1031: insufficient privileges error.

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

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

About the GSMROOTUSER Account

GSMROOTUSER is a database account specific to Oracle Sharding that is only used when pluggable database (PDB) shards are present. The account is used by GDSCTL and global service managers to connect to the root container of container databases (CDBs) to perform administrative tasks.

If PDB shards are not in use, the GSMROOTUSER user should not by unlocked nor assigned a password on any database. However, in sharded configurations containing

PDB shards, GSMROOTUSER must be unlocked and granted the SYSDG and SYSBACKUP privileges before a successful `gdsctl add cdb` command can be executed. The password for the GSMROOTUSER account can be changed after deployment if desired using the `alter user` SQL command in the root container of the CDB in combination with the `gdsctl modify cdb -pwd` command.

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

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 GDSCTL commands can also be scripted and through the use of a scheduler and can be done at regular intervals to help ensure that everything is running smoothly. When using Oracle GoldenGate for replication it is also important to monitor the lag of each replication stream.

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

Monitoring a Sharded Database with Enterprise Manager Cloud Control

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.

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.

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.

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

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.

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.

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

Propagation of Parameter Settings Across Shards

When you configure system parameter settings at the shard catalog, they are automatically propagated to all shards of the sharded database.

Before Oracle Database 19c, you had to configure `ALTER SYSTEM` parameter settings on each shard in a sharded database. In Oracle Database 19c, Oracle Sharding provides centralized management by allowing you to set parameters on the shard catalog. Then the settings are automatically propagated to all shards of the sharded database.

Propagation of system parameters happens only if done under `ENABLE SHARD DDL` on the shard catalog, then include `SHARD=ALL` in the `ALTER` statement.

```
SQL>alter session enable shard ddl;
SQL>alter system set enable_ddl_logging=true shard=all;
```

> **Note:**
>
> Propagation of the `enable_goldengate_replication` parameter setting is not supported.

Migrating a Non-PDB Shard to a PDB

Do the following steps if you want to migrate shards from a traditional single-instance database to Oracle multitenant architecture. Also, you must migrate to a multitenant architecture before upgrading to Oracle Database 20c.

1. Back up each existing non-PDB shard, and then create a new CDB, and a PDB inside it.
2. Restore each shard to the PDB inside the CDB.
3. Run the `GDSCTL ADD CDB` command to add the new CDB.

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

4. Run the `GDSCTL ADD SHARD -REPLACE` command, specifying the connect string of the PDB, *shard_connect_string*, which tells the sharding infrastructure to replace the old location of the shard with new PDB location.

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

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

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

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

Managing Sharded Database Software Versions

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

- Patching and Upgrading a Sharded Database
  Applying an Oracle patch to a sharded database environment can be done on a single shard or all shards; however, the method you use depends on the replication option used for the environment and the type of patch being applied.
- 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.
- Compatibility and Migration from Oracle Database 18c
  When upgrading from an Oracle Database 18c installation which contains a single PDB shard for a given CDB, you must update the shard catalog metadata for any PDB.

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.
   - If you are upgrading an Oracle Database 18c sharded database configuration containing pluggable database (PDB) shards, follow the PDB-specific upgrade instructions in Compatibility and Migration from Oracle Database 18c.
2. Upgrade the shard catalog database.

   For best results the catalog should be upgraded using a rolling database upgrade; however, global services will remain available during the upgrade if the catalog is unavailable, although service failover will not occur.
3. Upgrade any shard directors that are used to run GDSCTL clients, and which do not also run a global service manager server.

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

- On Linux, Solaris64, and Solaris Sparc64:

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

- On AIX:

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

- On HPI:

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

- On Windows no error messages are expected.

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

   To ensure zero downtime, at least one shard director server should always be running. Shard director servers at an earlier version than the catalog will continue to operate fully until catalog changes are made.

> **See Also:**
>
> *Oracle Data Guard Concepts and Administration* for information about using DBMS_ROLLING to perform a rolling upgrade.
>
> *Oracle Data Guard Concepts and Administration* for information about patching and upgrading databases in an Oracle Data Guard configuration.

Downgrading a Sharded Database

Oracle Sharding does not support downgrading.

Sharded database catalogs and shards cannot be downgraded.

Compatibility and Migration from Oracle Database 18c

When upgrading from an Oracle Database 18c installation which contains a single PDB shard for a given CDB, you must update the shard catalog metadata for any PDB.

Specifically, in 18c, the name of a PDB shard is the `DB_UNIQUE_NAME` of its CDB; however, in Oracle Database 19c, the shard names are *db_unique_name_of_CDB_pdb_name*.

To update the catalog metadata to reflect this new naming methodology, and to also support the new GSMROOTUSER account as described in About the GSMROOTUSER Account, perform the following steps during the upgrade process as described in Upgrading Sharded Database Components.

1. After upgrading any CDB that contains a PDB shard, ensure that the GSMROOTUSER account exists, is unlocked, has been assigned a password, and has been granted SYSDG, SYSBACKUP, and gsmrootuser_role privileges.

   The following SQL statements in SQL*Plus will successfully set up GSMROOTUSER while connected to the root container (CDB$ROOT) of the CDB.

   ```
   SQL> alter session set "_oracle_script"=true;
   Session altered.

   SQL> create user gsmrootuser;
   User created.

   SQL> alter user gsmrootuser identified by new_GSMROOTUSER_password
     account unlock;
   User altered.

   SQL> grant sysdg, sysbackup, gsmrootuser_role to gsmrootuser
   container=current;
   Grant succeeded.

   SQL> alter session set "_oracle_script"=false;
   Session altered.
   ```

2. After upgrading the catalog database to the desired Oracle Database version, run the following PL/SQL procedure to update the catalog metadata to reflect the new name for the PDB shards present in the configuration.

   This procedure must be executed for each Oracle Database 18c PDB shard.

   The first parameter to pdb_fixup is the value of db_unique_name in the CDB that contains the PDB shard. In Oracle Database 18c, this is the same as the shard name as shown by gdsctl config shard.

   The second parameter is the PDB name of the shard PDB as shown by show con_name in SQL*Plus when connected to the shard PDB.

   The pdb_fixup procedure will update the catalog metadata to make it compatible with the new naming method for PDB shards.

   ```
   SQL> connect sys/password as sysdba
   Connected.
   SQL> set serveroutput on
   SQL> execute gsmadmin_internal.dbms_gsm_pooladmin.pdb_fixup('cdb1',
   'pdb1');
   ```

3. After upgrading all of the shard directors to the desired version, run the following GDSCTL command once for each CDB in the configuration to inform the shard directors of the password for the GSMROOTUSER in each CDB.

   ```
   GDSCTL> modify cdb -cdb CDB_name -pwd new_GSMROOTUSER_password
   ```

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:

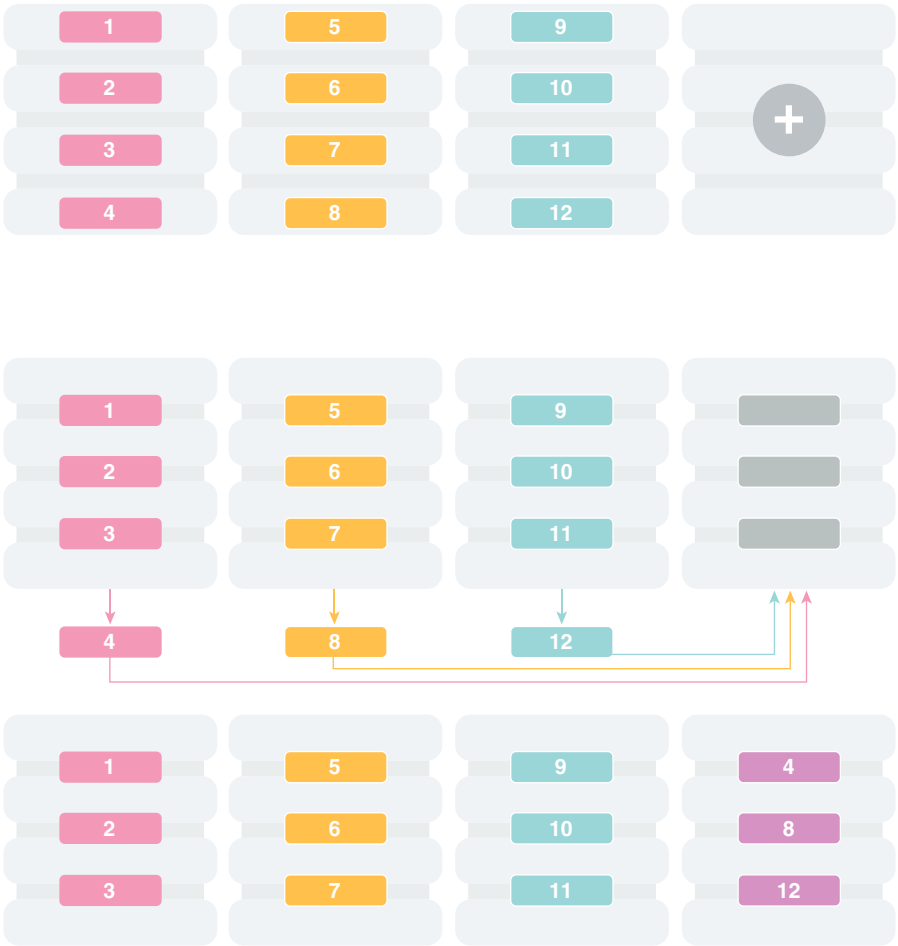
- When one or multiple shards are added to or removed from an SDB
- When there is skew in the data or workload distribution across shards

The unit of data migration between shards is the chunk. Migrating data in chunks guaranties that related data from different sharded tables are moved together.

When a shard is added to or removed from an SDB, multiple chunks are migrated to maintain a balanced distribution of chunks and workload across shards.

Depending on the sharding method, resharding happens automatically (system-managed) or is directed by the user (composite). The following figure shows the stages of automatic resharding when a shard is added to an SDB with three shards.

Figure 9-1 Resharding an SDB



A particular chunk can also be moved from one shard to another, when data or workload skew occurs, without any change in the number of shards. In this case, chunk migration can be initiated by the database administrator to eliminate the hot spot.

RMAN Incremental Backup, Transportable Tablespace, and Oracle Notification Service technologies are used to minimize impact of chunk migration on application availability. A chunk is kept online during chunk migration. There is a short period of time (a few seconds) when data stored in the chunk is available for read-only access only.

FAN-enabled clients receive a notification when a chunk is about to become read-only in the source shard, and again when the chunk is fully available in the destination shard on completion of chunk migration. When clients receive the `chunk read-only` event, they can either repeat connection attempts until the chunk migration is completed, or access the read-only chunk in the source chunk. In the latter case, an attempt to write to the chunk will result in a run-time error.

Note:

Running multi-shard queries while a sharded database is resharding can result in errors, so it is recommended that you do not deploy new shards during multi-shard workloads.

See Also:

Adding Shards to a System-Managed SDB

Sharding Methods

Removing a Shard From the Pool

It may become necessary to remove a shard from the sharded database environment, either temporarily or permanently, without losing any data that resides on that shard.

For example, removing a shard might become necessary if a sharded environment is scaled down after a busy holiday, or to replace a server or infrastructure within the data center. Prior to decommissioning the shard, you must move all of the chunks from the shard to other shards that will remain online. As you move them, try to maintain a balance of data and activity across all of the shards.

If the shard is only temporarily removed, keep track of the chunks moved to each shard so that they can be easily identified and moved back once the maintenance is complete.

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

Adding Standby Shards

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

When using Data Guard as the replication method for a sharded database, Oracle Sharding supports only the addition of a primary or physical standby shard; other types of Data Guard standby databases are not supported when adding a new standby to the sharded database. However, a shard that is already part of the sharded database can be converted from a physical standby to a snapshot standby. When converting a physical standby to a snapshot standby, the following steps should be followed:

1. Stop all global services on the shard using the GDSCTL command `STOP SERVICE`.
2. Disable all global services on the shard using the GDSCTL command `DISABLE SERVICE`.
3. Convert the shard to a snapshot standby using the procedure described in the Data Guard documentation.

   At this point, the shard remains part of the sharded database, but will not accept connections which use the sharding key.

If the database is converted back to a physical standby, the global services can be enabled and started again, and the shard becomes an active member of the sharded database.

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

Managing Shards with Oracle Enterprise Manager Cloud Control

You can manage database shards using Oracle Enterprise Manager Cloud Control

To manage shards using Cloud Control, they must first be discovered. Because each database shard is a database itself, you can use standard Cloud Control database discovery procedures.

The following topics describe shard management using Oracle Enterprise Manager Cloud Control:

- Validating a Shard
  Validate a shard prior to adding it to your Oracle Sharding deployment.

- 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:

- Install the Oracle Database Software

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

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

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

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

3. Verify the current shard configuration.

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

```
sh3             primary_shardgroup  Ok        Deployed    region1
ONLINE
sh4             standby_shardgroup  Ok        Deployed    region2
READ_ONLY
```

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

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

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

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

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

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

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

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

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

```
GDSCTL> deploy
```

6. Verify that the new shards are deployed.

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

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

```
$ gdsctl config chunks -show_Reshard

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

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

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

```
$ gdsctl databases

Database: "sh1" Registered: Y State: Ok ONS: N. Role: PRIMARY
Instances: 1
 Region: region1
   Service: "oltp_ro_srvc" Globally started: Y Started: N
            Scan: N Enabled: Y Preferred: Y
   Service: "oltp_rw_srvc" Globally started: Y Started: Y
            Scan: N Enabled: Y Preferred: Y
   Registered instances:
     cust_sdb%1
Database: "sh2" Registered: Y State: Ok ONS: N. Role: PH_STNDBY
Instances: 1
 Region: region2
   Service: "oltp_ro_srvc" Globally started: Y Started: Y
            Scan: N Enabled: Y Preferred: Y
   Service: "oltp_rw_srvc" Globally started: Y Started: N
            Scan: N Enabled: Y Preferred: Y
   Registered instances:
     cust_sdb%11
Database: "sh3" Registered: Y State: Ok ONS: N. Role: PRIMARY
Instances: 1
 Region: region1
   Service: "oltp_ro_srvc" Globally started: Y Started: N
            Scan: N Enabled: Y Preferred: Y
   Service: "oltp_rw_srvc" Globally started: Y Started: Y
            Scan: N Enabled: Y Preferred: Y
   Registered instances:
     cust_sdb%21
Database: "sh4" Registered: Y State: Ok ONS: N. Role: PH_STNDBY
```

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

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

```
$ gdsctl services

Service "oltp_ro_srvc.cust_sdb.oradbcloud" has 3 instance(s). Affinity:
ANYWHERE
   Instance "cust_sdb%11", name: "sh2", db: "sh2", region: "region2",
status: ready.
   Instance "cust_sdb%31", name: "sh4", db: "sh4", region: "region2",
status: ready.
   Instance "cust_sdb%51", name: "sh6", db: "sh6", region: "region2",
status: ready.
Service "oltp_rw_srvc.cust_sdb.oradbcloud" has 3 instance(s). Affinity:
ANYWHERE
   Instance "cust_sdb%1", name: "sh1", db: "sh1", region: "region1",
status: ready.
   Instance "cust_sdb%21", name: "sh3", db: "sh3", region: "region1",
status: ready.
   Instance "cust_sdb%41", name: "sh5", db: "sh5", region: "region1",
status: ready.
```

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

Replacing a Shard

If a shard fails and is unrecoverable, or if you just want to move a shard to a new host for other reasons, you can replace it using the ADD SHARD -REPLACE command in GDSCTL.

When a shard database fails and the database can be recovered on the same host (using RMAN backup/restore or other methods), there is no need to replace the shard using the -replace parameter. If the shard cannot be recovered locally, or for some other reason you want to relocate the shard to another host or CDB, it is possible to create its replica on the new host. The sharding configuration can be updated with the new information by specifying the -replace option in GDSCTL command ADD SHARD.

The following are some cases where replacing a shard using ADD SHARD -REPLACE is useful.

- The server (machine) where the shard database was running suffered irreparable damage and has to be replaced
- You must replace a working server with another (more powerful, for example) server
- A shard in a PDB was relocated from one CDB to another

In all of these cases the number of shards and data distribution across shards does not change after ADD SHARD is executed; a shard is replaced with another shard that holds the same data. This is different from ADD SHARD used without the -replace option when the number of shards increases and data gets redistributed.

Upon running ADD SHARD -REPLACE, the old shard parameters, such as connect_string, db_unique_name, and so on, are replaced with their new values. A new database can have different db_unique_name than the failed one. When replacing a standby in a Data Guard configuration, the DBID of the new database must match the old one, as Data Guard requires all of the members of the configuration to have same DBID.

Before Using Replace

Before you use ADD SHARD -REPLACE, verify the following:

- You have restored the database correctly (for example, using RMAN restore or other method). The new database shard must have the same sharding metadata as the failed one. Perform basic validation to ensure that you do not accidently provide a connect string to the wrong shard.
- The shard that failed must have been in a deployed state before failure happened.
- The shard that failed must be down when executing the ADD SHARD -REPLACE command.
- Fast-start failover observer must be running, if fast-start failover is enabled (which it is by default).

Replacing a Shard in a Data Guard Environment

The `ADD SHARD -REPLACE` command can only be used to replace a standby shard if the primary is still alive. In order to replace a primary shard that failed, wait for one of the remaining standbys to switch over to the primary role before trying to replace the failed shard.

When a switchover is not possible (primary and all the standbys are down), you must run ADD SHARD -REPLACE for each member starting with the primary. This creates a new broker configuration from scratch.

In MAXPROTECTION mode with no standbys alive, the primary database shuts down to maintain the protection mode. In this case, the primary database cannot be opened if the standby is not alive. To handle the replace operation in this scenario, you must first downgrade Data Guard protection mode using DGMGRL (to MAXAVAILABILITY or MAXPERFORMANCE) by starting up the database in mounted mode. After the protection mode is set, open the primary database and perform the replace operation using GDSCTL. After the replace operation finishes you can revert the protection mode back to the previous level using DGMGRL.

When replacing a standby in a Data Guard configuration, the DBID of the new database must match the old one, as Data Guard requires all of the members of the configuration to have same DBID.

Example 9-1 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 9-2 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 a removing a shard from a shardgroup.
- After splitting a chunk it is often advisable to move one of the resulting chunks to a new shard.

When moving shards to maintain scalability, the ideal targets of the chunks are shards that are less active, or have a smaller portion of data. Oracle Enterprise Manager and AWR reports can help you identify the distribution of activity across the shards, and help identify shards that are good candidates for chunk movement.

> **Note:**
>
> Any time a chunk is moved from one shard to another, you should make a full backup of the databases involved in the operation (both the source of the chunk move, and the target of the chunk move.)

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

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:

```
GDSCTL Command: ADD SHARDGROUP -SHARDGROUP 'north' -SHARDSPACE
'shardspaceora'
 -REGION 'north' -DEPLOY_AS 'STANDBY'
```

When the shardgroup is created successfully it appears in the **Manage Shardgroups** list. You might need to refresh the page to see the updates.

Services Management

You can manage services in your Oracle Sharding deployment with Oracle Enterprise Manager Cloud Control.

To manage Oracle Sharding services, open the **Sharded Database** menu, located in the top left corner of the Sharded Database target page, and choose **Services**. On the Services page, using the controls at the top of the list of services, you can start, stop, enable, disable, create, edit, and delete services.

Selecting a service opens a service details list which displays the hosts and shards on which the service is running, and the status, state, and Data Guard role of each of those instances. Selecting a shard in this list allows you to enable, disable, start, and stop the service on the individual shards.

The following topics describe services management tasks:

- Creating a Service
  Create services in your Oracle Sharding deployment using Oracle Enterprise Manager Cloud Control.

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.

10

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 Moving Chunks
  If you encounter issues with `MOVE CHUNK`, try the following:

Issues Starting Remote Scheduler Agent

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

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

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

```
Solution: Disable firewall

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

Shard Director Fails to Start

If you encounter issues starting the shard director, try the following:

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

```
GDSCTL>start gsm -gsm shardDGdirector
GSM-45054: GSM error
GSM-40070: GSM is not able to establish connection to GDS catalog

GSM alert log, /u01/app/oracle/diag/gsm/shard1/sharddgdirector/trace/
alert_gds.log
GSM-40112: OCI error. Code (-1). See GSMOCI trace for details.
GSM-40122: OCI Catalog Error. Code: 12514. Message: ORA-12514:
TNS:listener does not
currently know of service requested in connect descriptor
GSM-40112: OCI error. Code (-1). See GSMOCI trace for details.
2017-04-20T22:50:22.496362+05:30
Process 1 in GSM instance is down
GSM shutdown is successful
GSM shutdown is in progress
NOTE : if not message displayed in the GSM log then enable GSM trace level
to 16
while adding GSM itself.
```

1. Remove the newly created shard director (GSM) that failed to start.

   ```
   GDSCTL> remove gsm -gsm shardDGdirector
   ```

2. Add the shard director using trace level 16.

   ```
   GDSCTL> add gsm -gsm shardDGdirector -listener port_num -pwd
   gsmcatuser_password
    -catalog hostname:port_num:shard_catalog_name
    -region region1 -trace_level 16
   ```

3. If the shard catalog database is running on a non-default port (other than 1521), set the remote listener.

   ```
   SQL> alter system set
   local_listener='(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)
   (HOST=hostname)(PORT=port_num)))';
   ```

Errors From Shards Created with CREATE SHARD

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

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

Issues Using Create Shard

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

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

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

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

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

Privilege issues

```
GDSCTL>create shard -shardgroup primary_shardgroup -destination blr -
credential cred
GSM-45029: SQL error
ORA-02610: Remote job failed with error:
EXTERNAL_LOG_ID="job_79126_3",
USERNAME="oracle",
STANDARD_ERROR="Launching external job failed: Login executable not setuid-
root"
```

```
ORA-06512: at "GSMADMIN_INTERNAL.DBMS_GSM_POOLADMIN", line 6920
ORA-06512: at "SYS.DBMS_SYS_ERROR", line 86
ORA-06512: at "GSMADMIN_INTERNAL.DBMS_GSM_POOLADMIN", line 4596
ORA-06512: at line 1
```

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

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

Error on create shard

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

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

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

Issues Using Deploy Command

```
GDSCTL> deploy
GSM-45029: SQL error
ORA-29273: HTTP request failed
ORA-06512: at "SYS.DBMS_ISCHED", line 3715
ORA-06512: at "SYS.UTL_HTTP", line 1267
ORA-29276: transfer timeout
ORA-06512: at "SYS.UTL_HTTP", line 651
ORA-06512: at "SYS.UTL_HTTP", line 1257
ORA-06512: at "SYS.DBMS_ISCHED", line 3708
ORA-06512: at "SYS.DBMS_SCHEDULER", line 2609
ORA-06512: at "GSMADMIN_INTERNAL.DBMS_GSM_POOLADMIN", line 14284
ORA-06512: at line 1
```

Solution : Check the $ORACLE_HOME/data/pendingjobs for the exact error. ORA-1017 is thrown if any issues on wallet.

1. On problematic Shard host stop the remote scheduler agent.

```
schagent -stop
```

2. rename wallet direcotry on Database home

```
mv $ORACLE_HOME/data/wallet $ORACLE_HOME/data/wallet.old
```

3. start the remote scheduler agent and it will create new wallet directory

```
schagent -start
schagent -status
echo welcome | schagent -registerdatabase 10.10.10.10 8080
```

Issues Moving Chunks

If you encounter issues with `MOVE CHUNK`, try the following:

Issue: Initialization parameter `remote_depenencies_mode` has a default value of `timestamp`; therefore, because `prvtgwmut.plb` is run and `DBMS_GSM_UTILITY` recompiled durning upgrade, `GDSCTL MOVE CHUNK` runs into ORA-04062 errors similar to the following.

```
GSM Errors:
server:ORA-03749: Chunk move cannot be performed at this time.
ORA-06512: at "SYS.DBMS_SYS_ERROR", line 79
ORA-06512: at "GSMADMIN_INTERNAL.DBMS_GSM_DBADMIN", line 5497
ORA-04062: timestamp of package "GSMADMIN_INTERNAL.DBMS_GSM_UTILITY" has
been
changed
ORA-06512: at line 1
ORA-06512: at "GSMADMIN_INTERNAL.DBMS_GSM_DBADMIN", line 5366
ORA-06512: at line 1 (ngsmoci_execute)
```

Workaround 1: Restart the source and target shards after upgrade.

Workaround 2: `ALTER SYSTEM SET remote_depenencies_mode=signature` on both source and target.

11

Oracle Sharding Solutions

The following solutions show you how to use Oracle Sharding to solve a business problem.

- Combine Exisiting Non-Sharded Databases into a Federated Sharded Database
  If you have several database installations in different locations that run the same application, and you want to to include the data from all of them, to run data analytics queries for example, you can combine the independent databases into a sharded database to take advantage of Oracle Sharding multi-shard queries.
- Creating Affinity Between Middle-Tier and Shards
  Middle-tier routing allows smart routers to route to the middle tier associated with a sharding key.

Combine Exisiting Non-Sharded Databases into a Federated Sharded Database

If you have several database installations in different locations that run the same application, and you want to to include the data from all of them, to run data analytics queries for example, you can combine the independent databases into a sharded database to take advantage of Oracle Sharding multi-shard queries.

- Overview
- Creating and Deploying a Federated Sharding Configuration
  To deploy a federated sharding environment using existing databases, you define the database layout just as you would for user-defined sharding, using `GDSCTL` commands.
- Federated Sharding Reference

Overview

- About Federated Sharding
  Learn what a federated sharding configuration is, why you need it, and how it works.
- Federated Sharding Schema Requirements
  You can convert existing databases running the same application into a federated sharding configuration, without modifying the database schemas or the application.
- Sharded and Duplicated Tables in a Federated Sharding Configuration
  Tables that have *different* sets of data on each of the federated databases are equivalent to the sharded tables in a traditional sharded database. Tables with the *same* content on all of the federated databases are equivalent to the duplicated tables in a traditional sharded database.

About Federated Sharding

Learn what a federated sharding configuration is, why you need it, and how it works.

Federated sharding is an Oracle Sharding configuration where the shards consist of independent databases with similar schemas.

Creating a sharded database from independent databases reduces the need to import tons of data into a single location for data analytics.

Consider the following benefits to this approach.

- Create a sharding environment using existing, geographically distributed databases--there is no need to provision new systems
- Run multi-shard queries--access data from many locations in a single query

Oracle Sharding, in a federated sharding configuration, treats each independent database as a shard, and as such can issue multi-shard queries on those shards.

You can create a federated sharding configuration with minor version mismatches between the shards. For example, one region could be on Oracle Database 20.2 and another could be on Oracle Database 20.3. All database shards and the shard catalog must be on Oracle Database 20c or later.

Federated Sharding Schema Requirements

You can convert existing databases running the same application into a federated sharding configuration, without modifying the database schemas or the application.

However, the databases must have the same schema or minor differences. For example, a table can have an extra column in one of the databases.

An application upgrade can trigger changes in the schema, such as when you add a new table, new column, new check constraint, or/and modify a column data type. When part of an overall federated sharding configuration, Oracle Sharding handles the schema differences caused by an application upgrade, as long as the overall schema structure stays the same.

Sharded and Duplicated Tables in a Federated Sharding Configuration

Tables that have *different* sets of data on each of the federated databases are equivalent to the sharded tables in a traditional sharded database. Tables with the *same* content on all of the federated databases are equivalent to the duplicated tables in a traditional sharded database.

When you create the federated sharding configuration, the system assumes that all of the tables are sharded, so you must explicitly mark the tables that must be considered duplicated by the multi-shard query coordinator.

Limitations to Federated Sharding

There are some limitations to creating a federated sharding configuration.

- There is no concept of chunk in a federated sharding configuration, so the `GDSCTL MOVE CHUNK` command is not supported.
- Application sharding key-based routing is not supported.
- The existing databases, before being added to a federated sharding configuration, must be upgraded to Oracle Database 20c or later.

Federated Sharding Security

The database users do not need to exist on all of the federated databases, but the schema owners should exist on all of the databases. The privileges and the passwords of these schema owners can be different. Only common privileges are imported for security.

Creating and Deploying a Federated Sharding Configuration

To deploy a federated sharding environment using existing databases, you define the database layout just as you would for user-defined sharding, using `GDSCTL` commands.

The following is a high level description of the process for creating and deploying a federated sharding configuration.

1. Run the `GDSCTL CREATE SHARDCATALOG` command with the `FOR_FEDERATED_DATABASE` option to create the federated sharding configuration
2. Add shard directors to the configuration.
3. Add a shardspace to the configuration. A shardspace is defined as an existing database and its replica.
4. Add a shard by adding the existing database to the shardspace, then run `DEPLOY`.
5. Run `GDSCTL SYNC SCHEMA` to compare the schemas in the federated sharding configuration and retrieve the common shared schemas. Use `SYNC SCHEMA` to inspect and apply the DDLs.
6. Use `SQL ALTER TABLE` on the shard catalog to convert tables containing the same data across the federated shards to duplicated tables.
7. Prepare the shards in the federated sharding configuration for multi-shard queries.

The following topics describe the federated sharding-specific tasks in detail.

- Create the Federated Sharding Configuration
  The `GDSCTL` command `CREATE SHARDCATALOG` is used to create the federated sharding configuration, with the `FOR_FEDERATED_DATABASE` option used instead of selecting a sharding method in the `SHARDING` parameter.
- Retrieve, Inspect, and Apply the DDLs
  Run the `GDSCTL SYNC SCHEMA` command in phases to create the schema objects common to the existing databases in the shard catalog.
- Convert Tables to Duplicated Tables
  Use `ALTER TABLE` *`table_name`* `externally duplicated` to mark tables as duplicated in a federated sharding configuration.
- Prepare the Shards For Multi-Shard Queries
  Create **all shard** users and use the `ORA_SHARDSPACE_NAME` pseudo-column to perform queries on specific shards.

Create the Federated Sharding Configuration

The GDSCTL command CREATE SHARDCATALOG is used to create the federated sharding configuration, with the FOR_FEDERATED_DATABASE option used instead of selecting a sharding method in the SHARDING parameter.

The usage for the GDSCTL command CREATE SHARDCATALOG in creating a federated sharding configuration is similar to how it is used to create the shard catalog in user-defined sharding, except that instead of specifying a sharding method in the SHARDING parameter, you use the FOR_FEDERATED_DATABASE option. That is, the FOR_FEDERATED_DATABASE option is mutually exclusive with the SHARDING option.

```
CREATE SHARDCATALOG -DATABASE connect_identifier
  [-USER username[/password]]
  [-REGION region_name_list]
  [-CONFIGNAME config_name]
  [-AUTOVNCR ON/OFF]
  [-FORCE]
  [-SDB sdb_name]
  [-SHARDSPACE shardspace_name_list]
  -FOR_FEDERATED_DATABASE
```

The CREATE SHARDCATALOG syntax statement above shows which parameters are supported. The parameters not shown are not supported when used with the FOR_FEDERATED_DATABASE sharding method, for example, –AGENT_PASSWORD, REPFACTOR, and the Oracle Data Guard protection mode PROTECTMODE.

> **Note:**
>
> Only Oracle Data Guard replication is supported for federated sharding configurations. You don't create or manage the Data Guard configuration, but you can use Data Guard parameters with the ADD SHARD command so that you can add the primary and standbys to see the status in GDSCTL.

> **See Also:**
>
> The GDSCTL create shardcatalog topic in *Oracle Database Global Data Services Concepts and Administration Guide* for usage notes and command options.

Retrieve, Inspect, and Apply the DDLs

Run the GDSCTL SYNC SCHEMA command in phases to create the schema objects common to the existing databases in the shard catalog.

The `GDSCTL SYNC SCHEMA` syntax shown here illustrates the three phases of the opertion.

```
sync[hronize] schema
  [-schema [schemalist | all] [-retrieve_only] [-restart [-force]]
 | -apply [-skip_first]
 | -show [[-ddl ddlnum] [-count n] | [-failed_only]]]
```

`SYNC DDL` should be run in phases, as described here.

1. **Retrieve Phase**

   Run `SYNC SCHEMA` with the `-retrieve_only` option to inspect and verify the DDLs before they are run on the shard catalog.

   ```
   sync schema -schema schemalist –retrieve_only
   ```

   When `SYNC SCHEMA` is run without `-retrieve_only`, the DDL is retrieved and applied at the same time.

2. **Inspection Phase**

   You can examine the DDL statements and their execution status with the `-show` option. The `-ddl ddlnum` option shows the specified DDL, and the `-count n` option specifies the maximum number of entries to show.

   ```
   sync schema –show -ddl ddlnum -count n
   ```

   Or you can use the `-failed_only` option to examine only the errored out statements.

   ```
   sync schema –show -failed_only
   ```

3. **Apply Phase**

   In the final phase, you run the DDLs on the shard catalog to create the schemas and their objects.

   ```
   sync schema –apply
   ```

   If you get an error in the apply phase, there are a couple of ways to work around it:

   - If you can fix the cause of the error, fix and then retry `SYNC SCHEMA -apply`, which retries the failed DDL.
   - If the DDL cannot be fixed or it is not required, you can run `SYNC SCHEMA –apply -skip_first`, which resumes the apply phase from the point of the DDL failure.

   For security reasons, Oracle Sharding doesn't offer a way to edit the DDLs.

4. **Import Incremental Changes**

   If there are changes in the schema at a later point, the previous phases can be run again to import incremental changes. For example, when new objects are added, or a new column is added to a table, which will generate an `ALTER TABLE ADD` statement.

> **See Also:**
>
> The sync schema (synchronize schema) topic in *Oracle Database Global Data Services Concepts and Administration Guide* for more `SYNC SCHEMA` usage notes and option details.
>
> SYNC SCHEMA Operations for information about the tasks performed by `SYNC SCHEMA`

Convert Tables to Duplicated Tables

Use `ALTER TABLE table_name externally duplicated` to mark tables as duplicated in a federated sharding configuration.

Any table created by `SYNC SCHEMA` is considered by the multi-shard query layer as an **externally sharded** table. If the table contains the same data on all of the shards, you can alter the table to **externally duplicated**, so that the multi-shard query retrieves the data from one shard only, even if it is a query on a table with no filter predicates on `ORA_SHARDSPACE_NAME`.

```
ALTER TABLE table_name [externally duplicated | externally sharded]
```

Prepare the Shards For Multi-Shard Queries

Create **all shard** users and use the `ORA_SHARDSPACE_NAME` pseudo-column to perform queries on specific shards.

All Shard Users

Before running multi-shard queries from the shard catalog, you must create **all shard** users and grant them access to the sharded and duplicated tables. These users and their privileges should be created in the shard catalog under `shard DDL enabled`.

Create Shardspace-Specific Queries

A **shardspace** in federated sharding is a set consisting of a primary shard and zero or more standby shards. To filter query results for a particular shard[space], a pseudo-column called `ORA_SHARDSPACE_NAME` is added to every `externally sharded` table. The value of this pseudo column in the tables is the name of the shardspace.

Depending on the value of `MULTISHARD_QUERY_DATA_CONSISTENCY`, the rows can be fetched from the primary or from any of the standbys in the shardspace. To run a multi-shard query on a given shard, you can filter the query with the predicate `ORA_SHARDSPACE_NAME = shardspace_name_shard_belongs_to`.

A query like `SELECT CUST_NAME, CUST_ID FROM CUSTOMER`, where the table CUSTOMER is marked as `externally sharded`, runs on all of the shards.

A query like `SELECT CUST_NAME, CUST_ID FROM CUSTOMER WHERE ora_shardspace_name = 'EUROPE'` runs on the shards belonging to the `shardspace_name` Europe. Depending on the `MULTISHARD_QUERY_DATA_CONSISTENCY` parameter value, the query is run on either the primary shard of the shardspace Europe or on its standbys.

You can join sharded tables from different shardspaces. For example, to find the customers from shardspace Europe with orders in shardspace NA, write a query similar to the following.

```
SELECT order_id, customer_name FROM customers c , orders o WHERE c.cust_id
= o.cust_id and
c.ora_shardspace_name = 'Europe' and o.ora_shardspace_name = 'NA'
```

Querying an `externally duplicated` table, with or without the `ORA_SHARDSPACE_NAME` predicate, should go to only one of the shardspaces. The `MULTISHARD_QUERY_DATA_CONSISTENCY` parameter value determines whether to query a primary shard in the shardspace or its replicas.

Federated Sharding Reference

- SYNC SCHEMA Operations
- Troubleshooting Federated Sharding
  Solve common federated sharding issues with these troubleshooting tips.

SYNC SCHEMA Operations

- DDL Synchronization
  DDL synchronization is an operation that `SYNC SCHEMA` runs just after the deployment of the shards in a federated sharding configuration.
- Import Users
  A user or schema is a candidate for import by `SYNC SCHEMA` if it exists on all of the shards and owns importable schema objects.
- Grant User Roles and Priviledges
  For the imported users, `SYNC SCHEMA` compares users' privileges.
- Import Object Definitions
  The objects compared and imported by `SYNC SCHEMA` to the shard catalog are the objects that will be referenced in multi-shard queries or used by multi-shard query processing.
- Schema Object Comparison
  The objects, from one shard to another, can have different definitions. `SYNC SCHEMA` compares the different definitions and creates a common definition to enable multi-shard queries against imported objects.

DDL Synchronization

DDL synchronization is an operation that `SYNC SCHEMA` runs just after the deployment of the shards in a federated sharding configuration.

The goal of this operation is to import the object definitions from all of the shards, compare the definitions across the shards, and generate DDLs for the objects that exist on all of the shards (common objects). Once the DDLs are run and the objects are created, you can reference these objects in multi-shard queries.

Import Users

A user or schema is a candidate for import by `SYNC SCHEMA` if it exists on all of the shards and owns importable schema objects.

You can narrow the list of users to be imported by passing a list of users in the `-SCHEMA` parameter. For example,

```
gdsctl> sync schema -schema scott


gdsctl> sync schema -schema scott,myschema
```

For case-sensitive schemas use quoted identifiers.

```
gdsctl> sync schema -schema "O'Brien",scott
```

To include all non-Oracle schemas, use the value `ALL` in the `SCHEMA` parameter.

```
gdsctl> sync schema -schema all
```

Before importing the users, `SYNC SCHEMA` verifies that any discovered users exist on all shards, and no user already exists on the shard catalog with the same name. The users are then created on the shard catalog as local users and they are locked. Because these are local users, they only share the same name with shards and are essentially the same as any other user that may have the same name across different databases. Note that these users are not able to login and issue queries because they are not **all shard** users. To issue multi-shard queries, an all shard user must be created.

> **Note:**
>
> Only users local to a PDB are imported. Common CDB users are not imported.

Grant User Roles and Priviledges

For the imported users, `SYNC SCHEMA` compares users' privileges.

`SYNC SCHEMA` grants only the privileges that are granted on all of the shards (common grants). A user **A** who has a DBA role on shard1, but does not have DBA role on shard2, is not granted the DBA role in the shard catalog.

Import Object Definitions

The objects compared and imported by `SYNC SCHEMA` to the shard catalog are the objects that will be referenced in multi-shard queries or used by multi-shard query processing.

These objects are:

- Tables
- Views and Materialized Views (exported as tables)
- Check Constraints
- Object Types
- Synonyms

Running `SYNC SCHEMA` does not import objects related to storage, or objects that have no impact on multi-shard query processing, such as tablespaces, indexes, indextypes, directories, or zone maps.

Schema Object Comparison

The objects, from one shard to another, can have different definitions. `SYNC SCHEMA` compares the different definitions and creates a common definition to enable multi-shard queries against imported objects.

`SYNC SCHEMA` detects the objects' differences at two levels: number of objects, and object definitions.

First, `SYNC SCHEMA` considers the number of objects. It is likely that, during an application upgrade, some objects are added to the schemas. Only objects that are on all of the shards will be imported into the shard catalog.

Second, the object definitions from one shard to another can have different attributes. For the objects that `SYNC SCHEMA` imports, the following differences are noted:

- Differences in Tables
- Differences in Views
- Differences in Constraints
- Differences in Object Types

Differences in Tables

When comparing objects in a federated sharding configuration, some differences in tables have an impact on multi-shard queries and some do not.

Column Differences

Only column differences have an impact on multi-shard queries. `SYNC SCHEMA` addresses only this difference.

- The number of columns can be different.
- The data type of a given column can be different.
- The default value of a given column can be different.
- The expression of a virtual column can be different

When a table has a different numbers of columns, `SYNC SCHEMA` will opt for the creation of a table that contains the union of all of the columns. Taking the union of all of the columns, compared to just taking the intersection, will spare you from re-writing multi-shard queries in case of an incremental deploy, when the added shard has fewer columns than indicated in the shard catalog.

When a column has different data types, `SYNC SCHEMA` defines it as the highest (largest) datatype.

When a column has different data types, and one of the columns is a user-defined object type, then that column is not imported into the shard catalog.

When a column has different default values, `SYNC SCHEMA` sets `NULL` as the default value.

Nested table columns are not imported into the shard catalog.

Example: a Customer table is defined on shard1 and shard2 as shown here.

On shard1:

```
Customer( Cust_id number, Name varchar(30),
    Address varchar(50),Zip_code number)
```

On shard2:

```
Customer( Cust_id varchar(20), Name varchar(30),
    Address varchar(50),Zip_code number,
    Country_code number)
```

Note that the column `Cust_id` is a number on shard1 and a varchar(20) on shard2. Also, note that `Country_code` exists on shard2 but does not exist on shard1.

The Customer table created by `SYNC SCHEMA` in the shard catalog has all of the columns, including `Country_code`, and the `Cust_id` type is varchar(20).

```
Customer( Cust_id varchar(20), Name varchar(30),
    Address  varchar(50),Zip_code number,
    Country_code number)
```

`SYNC SCHEMA` keeps track of these differences between schemas in the shard catalog. A query issued on the catalog database that accesses these heterogeneous columns is rewritten to address the differences before it is sent to the shards. On the shard, if there is a data type mismatch, the data is CAST into the "superior" data type as created on the catalog. If the column is missing on the shard, the default value is returned as set on the catalog.

Partition Scheme Differences

Note that this difference has no impact on multi-shard queries, and is ignored.

- Partitioning column can be different.
- Partition type can be different.
- Number of partitions can be different.

Storage Attribute Differences

Note that this difference has no impact on multi-shard queries, and is ignored.

- Tablespaces, on which the table is created, are different.
- The encryption can be different.

- The `INMEMORY` attribute can be different.

Differences in Views

Views on shards are created and handled as tables in the shard catalog. The same restrictions that apply to tables also apply to views.

Differences in Constraints

Only CHECK constraints are created in the shard catalog. The CHECK constraint condition should be same on all of the shards.

Differences in Object Types

Object types and type bodies are only created if they have the same definition on all of the shards.

Troubleshooting Federated Sharding

Solve common federated sharding issues with these troubleshooting tips.

ORA-03851: Operation not supported in federated database

ORA-03701: Invalid parameter combination: federated database and ...

Some of the operations and command options that apply to a traditional sharded database are not applicable to a federated database. This is because:

- There is no concept of a chunk in a federated database. Any chunk-related operation is invalid, for example `SPLIT CHUNK` and `MOVE CHUNK`.
- The Data Guard broker configuration is not set up or managed by the system in federated database, because the existing shards may already have been set up with their own high availability configurations. Operations such as `SET DATAGUARD_PROPERTY` or `MODIFY SHARDSPACE` are not supported.
- Oracle Golden Gate configuration is not supported.
- The `CREATE SHARD` command is not supported.

ORA-03885: Some primary shards are undeployed or unavailable

The `SYNC SCHEMA` operation requires that all primary shards be available. Check the output of the `CONFIG SHARD` command, and check the status of all primary shards. Fix any issues and retry the operations when the shards become available.

ORA-03871: Some DDL statements are not applied to the catalog

The `SYNC SCHEMA` operation cannot import object definitions from the shards when some statements from the previous execution are still not applied on the shard catalog. Run `SYNC SCHEMA` with the `-apply` option to run these statements.

If a multi-shard query fails with this error due to a mismatch of the object definition on the shard and the catalog, make sure that the shard catalog has the latest schema changes imported. Any time there are schema changes in the federated database, you must run `SYNC SCHEMA` to import any changes in the schemas on the shards.

Note that subsequent runs of SYNC SCHEMA will not drop and recreate the object, but will generated ALTER statements to incorporate the definition changes. This ensures that if there are queries already running during the SYNC SCHEMA operation, they won't fail with invalid object errors.

Handling Errors During DDL Execution Phase

If DDL execution fails on the shard catalog, the status of each DDL can be examined with the SYNC SCHEMA -show option.

```
gdsctl> sync schema -show
```

Note: The SYNC SCHEMA -show command is different from the command SHOW DDL. SHOW DDL lists DDL statements run by an all-shard user that are first run on the catalog and then propagated to the shards, whereas SYNC SCHEMA -show DDL statements are generated from the objects imported from shards.

By default, SYNC SCHEMA -show lists a fixed number of the latest DDLs. The -count and -ddl options can be used to inspect specific range of DDLs. For example,

```
gdsctl> sync schema -show -count 20
gdsctl> sync schema -show -count 20 -ddl 5
```

To check the complete DDL text and error message, if any, use the -ddl option.

```
gdsctl> sync schema -show -ddl 5
```

To list only the failed DDL statements, use the -failed_only option.

```
gdsctl> sync schema –failed_only
```

Based on the error message of the failed DDL, fix the cause of the error and perform the apply phase.

```
gdsctl> sync schema -apply
```

The SYNC SCHEMA command also has a -restart option to perform the complete operation from the beginning as if it were run for the first time. This option will DROP all existing schemas imported during all previous executions of SYNC SCHEMA and any related metadata. Be aware that this will cause any running queries in these objects to fail.

```
gdsctl> sync schema -restart
```

Creating Affinity Between Middle-Tier and Shards

Middle-tier routing allows smart routers to route to the middle tier associated with a sharding key.

You can use the middle-tier routing API to publish the sharded database topology to the router tier so that requests based on specific sharding keys are routed to the

appropriate application middle tier, which in turn establishes connections on the given subset of shards.

In a typical 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 11-1 Middle-Tier Routing Using UCP API

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

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

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

```
 *
 */
public class MidtierShardingExample {

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

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

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

  static void testMidTierRouting() throws UniversalConnectionPoolException,
      SQLException {

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

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

    OracleShardRoutingCache routingCache = new OracleShardRoutingCache(
        dbConnectProperties);

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

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

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

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

  }
}
```

See Also:

Middle-Tier Routing Using UCP in *Oracle Universal Connection Pool Developer's Guide*

12

Oracle Sharding Reference

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

- Using GDSCTL with Oracle Sharding
- DDL Syntax Extensions for Oracle Sharding
  Oracle Sharding includes SQL DDL statements with syntax that can only be run against a sharded database.
- Using the Sharding Advisor Command Line Tool
  Use Sharding Advisor command line tool before migration to a sharded database to analyze your non-sharded database and get appropriate sharded database configuration recmmendations based on the current non-sharded database workload and schema.

Using GDSCTL with Oracle Sharding

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

GDSCTL Operation

Learn how to start GDSCTL, run commands, and get command help text.

- Starting GDSCTL
  To start GDSCTL, enter `gdsctl` at the operating system prompt.
- Running GDSCTL Commands Interactively
  You can run GDSCTL commands interactively at either the operating system prompt or the GDSCTL command prompt.
- Running GDSCTL Batch Operations
  You can gather all the GDSCTL commands in one file and run them as a batch with GDSCTL.
- GDSCTL Help Text
  You can display help for GDSCTL and GDSCTL commands.

Starting GDSCTL

To start GDSCTL, enter `gdsctl` at the operating system prompt.

```
$ gdsctl
```

GDSCTL starts and displays the GDSCTL command prompt.

```
GDSCTL>
```

Running GDSCTL Commands Interactively

You can run GDSCTL commands interactively at either the operating system prompt or the GDSCTL command prompt.

Run a GDSCTL command at the system prompt.

```
$ gdsctl add gsm -gsm gsm1 -catalog 127.0.0.1:1521:db1
```

Run a GDSCTL command at the GDSCTL command prompt.

```
GDSCTL> add gsm -gsm gsm1 -catalog 127.0.0.1:1521:db1
```

Both of these methods achieve the same result. The command syntax examples in this document use the GDSCTL command prompt.

Running GDSCTL Batch Operations

You can gather all the GDSCTL commands in one file and run them as a batch with GDSCTL.

The following command starts GDSCTL and runs the commands contained in the specified script file.

```
$ gdsctl @script_file_name
```

GDSCTL Help Text

You can display help for GDSCTL and GDSCTL commands.

The GDSCTL `HELP` command displays a summary of all GDSCTL commands.

```
GDSCTL> help
```

If you specify a command name after `HELP`, then the help text for that command is shown.

```
GDSCTL> help start gsm
```

You can also use the `-h` option with any GDSCTL command to show the help text for the specified command.

```
GDSCTL> start gsm -h
```

GDSCTL Connections

Some GDSCTL commands require a connection to the shard catalog, and for ceratin operations, GDSCTL must connect to a shard director.

- GDSCTL Shard Catalog Connections
  If you run GDSCTL commands that require a connection to the shard catalog, then you must run the GDSCTL `CONNECT` command before the first command that requires the connection.
- GDSCTL Shard Director Connections
  For certain operations, GDSCTL must connect to a shard director, also known as global service manager.

GDSCTL Shard Catalog Connections

If you run GDSCTL commands that require a connection to the shard catalog, then you must run the GDSCTL `CONNECT` command before the first command that requires the connection.

The `CONNECT` command only needs to be run once in a GDSCTL session.

GDSCTL uses Oracle Net Services to connect to the shard catalog database or another database in the Oracle Sharding configuration. For these connections you can run GDSCTL from any client or host that has the necessary network configuration.

Unless specified, GDSCTL resolves connect strings with the current name resolution methods (such as TNSNAMES).

The GDSCTL operations that require a connection to a shard catalog are noted in the usage notes for each command.

GDSCTL Shard Director Connections

For certain operations, GDSCTL must connect to a shard director, also known as global service manager.

Unless specified, GDSCTL resolves connect strings with the current name resolution methods (such as TNSNAMES). However, to resolve the shard director name, GDSCTL queries the `gsm.ora` file.

To connect to a shard director, GDSCTL must be running on the same host as the shard director. When connecting to a shard director, GDSCTL looks for the `gsm.ora` file associated with the local shard director.

The following are the GDSCTL operations that require a connection to a shard director.

- `ADD GSM` adds a shard director.
- `START GSM` starts the shard director.
- `STOP GSM` stops the shard director.
- `MODIFY GSM` modifies the configuration parameters of the shard director.
- `STATUS GSM` returns the status of a shard director.

- `SET INBOUND_CONNECT_LEVEL` sets the `INBOUND_CONNECT_LEVEL` listener parameter.
- `SET TRACE_LEVEL` sets the trace level for the listener associated with the specified shard director.
- `SET OUTBOUND_CONNECT_LEVEL` sets the timeout value for the outbound connections for the listener associated with a specific shard director.
- `SET LOG_LEVEL` sets the log level for the listener associated with a specific shard director.

GDSCTL Commands Used with Oracle Sharding

A subset of GDSCTL commands is used with Oracle Sharding.

- add cdb
- add credential
- add file
- add gsm
- add invitednode (add invitedsubnet)
- add region
- add service
- add shard
- add shardgroup
- add shardspace
- config
- config cdb
- config chunks
- config credential
- config file
- config gsm
- config region
- config sdb
- config service
- config shard
- config shardgroup
- config shardspace
- config table family
- config vncr
- configure
- connect
- create shard
- create shardcatalog

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

- status gsm
- status service
- stop gsm
- stop service
- sync schema (synchronize schema)
- validate catalog

DDL Syntax Extensions for Oracle Sharding

Oracle Sharding includes SQL DDL statements with syntax that can only be run against a sharded database.

Changes to query and DML statements are not required to support Oracle Sharding, and the changes to the DDL statements are very limited. Most existing DDL statements will work the same way on a sharded database, with the same syntax and semantics, as they do on a non-sharded database.

- CREATE TABLESPACE SET
  This statement creates a tablespace set that can be used as a logical storage unit for one or more sharded tables and indexes. A tablespace set consists of multiple Oracle tablespaces distributed across shards in a shardspace.
- ALTER TABLESPACE SET
  This statement alters a tablespace set that can be used as a logical storage unit for one or more sharded tables and indexes.
- DROP TABLESPACE SET and PURGE TABLESPACE SET
  These statements drop or purge a tablespace set, which can be used as a logical storage unit for one or more sharded tables and indexes.
- CREATE TABLE
  The `CREATE TABLE` statement has been extended to create sharded and duplicated tables, and specify a table family.
- ALTER TABLE
  The `ALTER TABLE` statement is extended to modify sharded and duplicated tables.
- ALTER SESSION
  The `ALTER SESSION` statement is extended to support sharded databases.

CREATE TABLESPACE SET

This statement creates a tablespace set that can be used as a logical storage unit for one or more sharded tables and indexes. A tablespace set consists of multiple Oracle tablespaces distributed across shards in a shardspace.

The `CREATE TABLESPACE SET` statement is intended specifically for Oracle Sharding. Its syntax is similar to `CREATE TABLESPACE`.

```
CREATE TABLESPACE SET tablespace_set
       [IN SHARDSPACE shardspace]
              [USING TEMPLATE (
   { MINIMUM EXTENT size_clause
   | BLOCKSIZE integer [ K ]
```

```
  | logging_clause
  | FORCE LOGGING
  | ENCRYPTION tablespace_encryption_spec
  | DEFAULT [ table_compression ] storage_clause
  | { ONLINE | OFFLINE }
  | extent_management_clause
  | segment_management_clause
  | flashback_mode_clause
 }...
)];
```

Note that in system-managed sharding there is only one default shardspace in the sharded database. The number of tablespaces in a tablespace set is determined automatically and is equal to the number of chunks in the corresponding shardspace.

All tablespaces in a tablespace set are bigfile 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

This statement alters a tablespace set that can be used as a logical storage unit for one or more sharded tables and indexes.

The `SHARDSPACE` property of a tablespace set cannot be modified. All other attributes of a tablespace set can be altered just as for a regular permanent tablespace. Because tablespaces in a tablespace set are bigfile, the `ADD DATAFILE` and `DROP DATAFILE` clauses are not supported.

DROP TABLESPACE SET and PURGE TABLESPACE SET

These statements drop or purge a tablespace set, which can be used as a logical storage unit for one or more sharded tables and indexes.

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

CREATE TABLE

The `CREATE TABLE` statement has been extended to create sharded and duplicated tables, and specify a table family.

Syntax

```
CREATE [ { GLOBAL TEMPORARY | SHARDED | DUPLICATED} ]
         TABLE [ schema. ] table
      { relational_table | object_table | XMLType_table }
          [ PARENT [ schema. ] table ] ;
```

The following parts of the `CREATE TABLE` statement are intended to support Oracle Sharding:

- The `SHARDED` and `DUPLICATED` keywords indicate that the table content is either partitioned across shards or duplicated on all shards respectively. The `DUPLICATED` keyword is the only syntax change to create duplicated tables. All other changes described below apply only to sharded tables.
- The `PARENT` clause links a sharded table to the root table of its table family.
- To create a sharded table, `TABLESPACE SET` is used instead of `TABLESPACE`. All clauses that contain `TABLESPACE` are extended to contain `TABLESPACE SET`.
- Three clauses: `consistent_hash_partitions`, `consistent_hash_with_subpartitions`, and `partition_set_clause` in the `table_partitioning_clauses`.

```
table_partitioning_clauses ::=
{range_partitions
| hash_partitions
| list_partitions
| composite_range_partitions
| composite_hash_partitions
| composite_list_partitions
| reference_partitioning
| system_partitioning
| consistent_hash_partitions
| consistent_hash_with_subpartitions
| partition_set_clause
}
```

Example

```
CREATE SHARDED TABLE customers
( cust_id     NUMBER NOT NULL
, name        VARCHAR2(50)
, address     VARCHAR2(250)
, location_id VARCHAR2(20)
, class       VARCHAR2(3)
, signup_date DATE
,
CONSTRAINT cust_pk PRIMARY KEY(cust_id, class)
```

```
)
PARTITIONSET BY LIST (class)
PARTITION BY CONSISTENT HASH (cust_id)
PARTITIONS AUTO
(PARTITIONSET gold   VALUES ('gld') TABLESPACE SET ts2,
 PARTITIONSET silver VALUES ('slv') TABLESPACE SET ts1)
;
```

Limitations

Limitations for sharded tables in the current release:

- There is no default tablespace set for sharded tables.
- A temporary table cannot be sharded or duplicated.
- Index-organized sharded tables are not supported.
- A sharded table cannot contain a nested table column or an identity column.
- A primary key constraint defined on a sharded table must contain the sharding column(s). A foreign key constraint on a column of a sharded table referencing a duplicated table column is not supported.
- System partitioning and interval range partitioning are not supported for sharded tables. Specification of individual hash partitions is not supported for partitioning by consistent hash.
- A column in a sharded table used in `PARTITION BY` or `PARTITIONSET BY` clauses cannot be a virtual column.

Duplicated tables in the current release are not supported with the following:

- System and reference partitioned tables
- LONG, abstract (MDSYS 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

ALTER TABLE

The `ALTER TABLE` statement is extended to modify sharded and duplicated tables.

There are limitations on using `ALTER TABLE` with a sharded database.

The following options are not supported for a sharded table in a system-managed or composite sharded database:

- Rename
- Add foreign key constraint
- All operations on individual partitions and subpartitions
- All partition-related operations on the shard, except `TRUNCATE` partition, `UNUSABLE LOCAL INDEXES`, and `REBUILD UNUSABLE LOCAL INDEXES`

The following are not supported for duplicated tables:

- Data types: long, abstract (`MDSYS` datatypes are supported), `REF`

- Column options: vector encode, invisible column, nested tables
- Object types
- Clustered table
- External table
- ILM policy
- `PARENT` clause
- Flashback table operation
- System and Reference partitioning
- Enable `NOLOGGING` option
- 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 `ALTER SESSION` statement is extended to support sharded databases.

The session-level `SHARD DDL` parameter sets the scope for DDLs issued against the shard catalog database.

```
ALTER SESSION { ENABLE | DISABLE } SHARD DDL;
```

When `SHARD DDL` is enabled, all DDLs issued in the session are executed on the shard catalog and all shards. When `SHARD DDL` is disabled, a DDL is executed only against the shard catalog database. `SHARD DDL` is enabled by default for a sharded database user (the user that exists on all shards and the catalog). To create a sharded database user, the `SHARD DDL` parameter must be enabled before running `CREATE USER`.

Using the Sharding Advisor Command Line Tool

Use Sharding Advisor command line tool before migration to a sharded database to analyze your non-sharded database and get appropriate sharded database configuration recmmendations based on the current non-sharded database workload and schema.

Refer to the following reference topics for details about Sharding Advisor syntax, usage, output tables, example queries on the output tables, and security inforamtion.

- Sharding Advisor Usage and Options
- Sharding Advisor Output Tables
  To review the sharding configurations and related information, you can query the following output database tables, which are stored in the same schema as your source database.

- Sharding Advisor Output Review SQL Examples
  Because the Sharding Advisor output is contained in regular database tables, you can run many kinds of SQL queries against them to look at the output from different perspectives.
- Sharding Advisor Security
  Sharding Advisor is a client-side utility that connects to the non-sharded database using authenticated OCI connections.

Sharding Advisor Usage and Options

Sharding Advisor is a client command-line tool that connects to an existing non-sharded database and provides sharding configuration recommendations.

Syntax

```
gwsadv
 [-n nodeName[:portnum]]
 [-s serviceName]
  -u username
  -p password
 [-c]
 [-awr_snap_begin timestamp]
 [-awr_snap_end timestamp]
  –w
 [sch=(schema1, schema2, …)]
 [-tab importantTabsFile]
```

Options

Each option must be prefixed with a minus sign (-).

Option	Description	Required (Y/N)
`-u username`	Oracle user name	Y
`-p password`	Oracle password	Y
`-w`	Directs Sharding Advisor to use the query workload for sharding configuration generation and ranking.	Y
`-n nodeName[:portnum]`	Node name and port number, if connecting to a database on another host	N
`-s serviceName`	Service name, if connecting to a database on another host	N

Option	Description	Required (Y/N)
`-c`	Capture a new or changed workload. Required on first run of Sharding Advisor on a new or changed workload. Not required on subsequent runs on the same workload. By default, the workload is captured from the `V$SQL_PLAN_STATISTICS_ALL` table. Alternatively, the workload can be captured from Automatic Workload Repository (AWR) snapshots by using the `-awr_snap_begin` and `-awr_snap_end` options with the `-c` option to specify the beginning and ending time stamps of the AWR snapshots.	N
`-awr_snap_begin` *`timestamp`*	Specify the beginning timestamp, in the format 'YYYY-MM-DD HH24:MI:SS', to specify the AWR snapshots to capture the workload from.	N
`-awr_snap_end` *`timestamp`*	Specify the end timestamp, in the format 'YYYY-MM-DD HH24:MI:SS', to specify the AWR snapshots to capture the workload from.	N
`sch`	The `sch` option specifies the list of schemas to run Sharding Advisor against, if you want to run as a different user.	N
`-tab` *`importantTabsFile`*	Name of file that consists of table names, one per line, in the format *schemaname.tablename*, to restrict the number of tables that the Sharding Advisor needs to analyze.	N

Usage Notes

For procedures describing how to run the Sharding Advisor with example commands see Run Sharding Advisor and Run Sharding Advisor on a Non-Production System.

Sharding Advisor Output Tables

To review the sharding configurations and related information, you can query the following output database tables, which are stored in the same schema as your source database.

- SHARDINGADVISOR_CONFIGURATIONS Table
- SHARDINGADVISOR_CONFIGDETAILS Table
- SHARDINGADVISOR_QUERYTYPES Table

SHARDINGADVISOR_CONFIGURATIONS Table

Each row of the SHARDINGADVISOR_CONFIGURATIONS table represents a table in a ranked sharded configuration, and provides information about whether to shard or duplicate it, and if sharded, its level in a table family hierarchy, its parent table, root table sharding key, foreign key reference constraints, and table size per shard.

SHARDINGADVISOR_CONFIGURATIONS Schema

Column	Description
RANK	The rank of the sharding configuration based on the ranking algorithm
TABLENAME	Name of the table in the sharding configuration
TABLETYPE	'S' (Sharded), 'D' (Duplicated), or 'L' (Local)
TABLELEVEL	Level of the table in the table family hierarchy, NULL for duplicated tables
PARENT	Parent of the table in the table family hierarchy, NULL for duplicated tables
SHARDBY	Sharding method. REFERENCE for sharding by reference, or PARENT for sharding by PARENT clause, for child tables.
SHARDINGORREFERENCECOLS	Sharding key for the root table, partition by REFERENCE or PARENT for the child tables in a table family, and NULL for duplicated tables
UNENFORCEABLECONSTRAINTS	Foreign key constraints other than the reference columns, which cannot be enforced
SIZEOFTABLE	Size of the table per shard

SHARDINGADVISOR_CONFIGDETAILS Table

Each row of the SHARDINGADVISOR_CONFIGDETAILS table represents a ranked sharding configuration, and provides the number and collective size, per shard, of

each type of table, the number of each type of query, and based on your source database's current workload, an estimated cost.

SHARDINGADVISOR_CONFIGDETAILS Schema

Column	Description
RANK	The rank of the sharding configuration based on the ranking algorithm
CHOSENBYUSER	'Y' if the sharding configuration is chosen by the user, NULL for other sharding configurations
NUMSHARDEDTABLES	Number of sharded tables in this sharding configuration
SIZEOFSHARDEDTABLES	Cumulative size of sharded tables (per shard) in this sharding configuration
NUMDUPLICATEDTABLES	Number of duplicated tables in this sharding configuration
SIZEOFDUPLICATEDTABLES	Cumulative size of duplicated tables (per shard) in this sharding configuration
NUMSINGLESHARDQUERIES	Number of single shard queries in the query workload for this sharding configuration
NUMMULTISHARDQUERIES	Number of multi-shard queries in the query workload for this sharding configuration
NUMCROSSSHARDQUERIES	Number of multi-shard queries that require an external join in the query workload for this sharding configuration
COST	Cost of the sharding configuration based on the costing algorithm

SHARDINGADVISOR_QUERYTYPES Table

Each row of the SHARDINGADVISOR_QUERYTYPES table represents a query in the workload, and lists the query type and SQL ID. Note that the same query can be of a different query type depending on the sharding configuration.

SHARDINGADVISOR_QUERYTYPES Schema

Column	Description
RANK	The rank of the sharding configuration based on the ranking algorithm
SQLID	The query SQL ID
QUERYTYPE	The type of the query in this sharding configuration: `SINGLE SHARD QUERY`, `MULTI SHARD QUERY`, or `CROSS SHARD QUERY`

Sharding Advisor Output Review SQL Examples

Because the Sharding Advisor output is contained in regular database tables, you can run many kinds of SQL queries against them to look at the output from different perspectives.

Example 12-1 Display the sharding configurations in ranking order

```
SELECT rank, tableName as tname, tabletype as type,
           tablelevel as tlevel, parent, shardby as shardBy,
           shardingorreferencecols as cols, unenforceableconstraints,
           sizeoftable
FROM SHARDINGADVISOR_CONFIGURATIONS
ORDER BY rank, tlevel, tname, parent;
```

Example 12-2 Display the table family of the top ranked sharding configuration

```
SELECT rank, tableName as tname, tabletype as type,
        tablelevel as tlevel, parent, shardby as shardBy,
        shardingorreferencecols as cols, unenforceableconstraints,
        sizeoftable
FROM SHARDINGADVISOR_CONFIGURATIONS
WHERE rank = 1 AND tabletype = 'S'
ORDER BY tlevel, tname, parent;
```

Example 12-3 Display the table families in ranking order

```
SELECT rank, tableName as tname, tabletype as type,
        tablelevel as tlevel, parent, shardby as shardBy,
        shardingorreferencecols as cols, unenforceableconstraints,
        sizeoftable
FROM SHARDINGADVISOR_CONFIGURATIONS
WHERE tabletype = 'S'
ORDER BY rank, tlevel, tname, parent;
```

Example 12-4 Display the duplicated tables of the top ranked sharding configuration

```
SELECT rank, tableName as tname, tabletype as type,
           tablelevel as tlevel, parent, shardby as shardBy,
           shardingorreferencecols as cols, unenforceableconstraints,
           sizeoftable
FROM SHARDINGADVISOR_CONFIGURATIONS
WHERE rank = 1 AND tabletype = 'D'
ORDER BY tlevel, tname, parent;
```

Example 12-5 Display the number of sharding configurations with *table_name* as the root table

```
SELECT COUNT(*)
FROM SHARDINGADVISOR_CONFIGURATIONS
WHERE tablename = 'TABLE_NAME' AND tablelevel = 0;
```

Example 12-6 Display the table families of the sharding configurations with root table *table_name*

```
SELECT rank, tableName as tname, tabletype as type,
        tablelevel as tlevel, parent, shardby as shardBy,
```

```
        shardingorreferencecols as cols
FROM SHARDINGADVISOR_CONFIGURATIONS
WHERE tabletype = 'S'
    AND rank IN
        (SELECT rank
        FROM SHARDINGADVISOR_CONFIGURATIONS
        WHERE tablename = 'TABLE_NAME' and tablelevel = 0)
ORDER BY rank, tlevel, tname, parent;
```

Example 12-7 Display the details of the sharding configurations in ranking order

```
SELECT rank, chosenbyuser,
        numshardedtables as stabs, sizeofshardedtables as sizestabs,
        numduplicatedtables as dtabs,
        sizeofduplicatedtables as sizedtabs,
        numsingleshardqueries as numssq,
        nummultishardqueries as nummsq,
        numcrossshardqueries as numcsq, cost
FROM SHARDINGADVISOR_CONFIGDETAILS
ORDER BY rank;
```

Example 12-8 Display the details of your chosen sharding configuration

```
SELECT rank,
        numshardedtables as stabs, sizeofshardedtables as sizestabs,
        numduplicatedtables as dtabs,
        sizeofduplicatedtables as sizedtabs,
        numsingleshardqueries as numssq,
        nummultishardqueries as nummsq,
        numcrossshardqueries as numcsq, cost
FROM SHARDINGADVISOR_CONFIGDETAILS
WHERE CHOSENBYUSER = 'Y'
ORDER BY RANK;
```

Sharding Advisor Security

Sharding Advisor is a client-side utility that connects to the non-sharded database using authenticated OCI connections.

- The Sharding Advisor requires the appropriate credentials (user name and password) to connect to the non-sharded source database. Sharding Advisor can be run as a different user than the user that owns the source database schema that the Sharding Advisor analyzes. This user must have `SELECT` privilges on the tables in the non-sharded schema.
- The user needs `SELECT` privileges on the `GV$SQL_PLAN` and `GV$SQL_PLAN_STATISTICS_ALL` views, and on the `DBA_HIST_SQL_PLAN`, `DBA_HIST_SQLSTAT`, and `DBA_HIST_SNAPHSOT` tables. The user does not need any other special privileges.
- Sharding Advisor is not vulnerable to privilege escalation and denial of service.

- Sharding Advisor does not store or expose any sensitive data such as passwords, database service names, or user names.
- Sharding Advisor does not expose sensitive details about the inner workings of the product.
- Sharding Advisor does not include any interfaces or APIs which are not externally documented.
- Sharding Advisor does not require any insecure protocols to be enabled.
- Sharding Advisor does not use any insecure modes of operation.
- Sharding Advisor does not store any data or other information in any files.
- All connections to the database are through authenticated OCI connections.
- There are no SETUID executables created.
- No new grants to PUBLIC are done.
- No new default schemas are created, but Sharding Advisor internal tables are created under the user that is used to run Sharding Advisor.

Index

O

P

R

S

T

U