System Managed Sharding with Active Data Guard using ADD SHARD Method

Cookbook

SEPTEMBER 2017

Safe Harbor Statement

THE INFORMATION CONTAINED IN THIS DOCUMENT IS FOR INFORMATIONAL SHARING PURPOSES ONLY, AND SHOULD BE CONSIDERED IN YOUR CAPACITY AS A CUSTOMER ADVISORY BOARD MEMBER OR PURSUANT TO YOUR BETA TRIAL AGREEMENT ONLY. IT IS NOT A COMMITMENT TO DELIVER ANY MATERIAL, CODE, OR FUNCTIONALITY, AND SHOULD NOT BE RELIED UPON IN MAKING PURCHASING DECISIONS. THE DEVELOPMENT, RELEASE, AND TIMING OF ANY FEATURES OR FUNCTIONALITY DESCRIBED IN THIS DOCUMENT REMAINS AT THE SOLE DISCRETION OF ORACLE.

THIS DOCUMENT IN ANY FORM, SOFTWARE OR PRINTED MATTER, CONTAINS PROPRIETARY INFORMATION THAT IS THE EXCLUSIVE PROPERTY OF ORACLE. YOUR ACCESS TO AND USE OF THIS CONFIDENTIAL MATERIAL IS SUBJECT TO THE TERMS AND CONDITIONS OF YOUR ORACLE SOFTWARE LICENSE AND SERVICE AGREEMENT, WHICH HAS BEEN EXECUTED AND WITH WHICH YOU AGREE TO COMPLY. THIS DOCUMENT AND INFORMATION CONTAINED HEREIN MAY NOT BE DISCLOSED, COPIED, REPRODUCED OR DISTRIBUTED TO ANYONE OUTSIDE ORACLE WITHOUT PRIOR WRITTEN CONSENT OF ORACLE. THIS DOCUMENT IS NOT PART OF YOUR LICENSE AGREEMENT NOR CAN IT BE INCORPORATED INTO ANY CONTRACTUAL AGREEMENT WITH ORACLE OR ITS SUBSIDIARIES OR AFFILIATES

What is Oracle Sharding?

Oracle Sharding is a scalability and availability feature for custom-designed OLTP applications that enables distribution and replication of data across a pool of discrete Oracle databases that share no hardware or software. The pool of databases is presented to the application as a single logical database. Applications elastically scale (data, transactions and users) to any level, on any platform, simply by adding additional databases (shards) to the pool. Scaling up to 1000 shards is supported in the first release.

Oracle Sharding provides superior run-time performance and simpler life-cycle management compared to home-grown deployments that use a similar approach to scalability. It also provides the advantages of an enterprise DBMS, including: relational schema, SQL, and other programmatic interfaces, support for complex data types, online schema changes, multi-core scalability, advanced security, compression, high-availability, ACID properties, consistent reads, developer agility with JSON, and much more.

Examples of target customers and applications for sharding

The target customer for Oracle sharding can come from any industry vertical. Examples include:

- Mass Media and Financial Information Services providers who need massive scalability with high availability for online storage and retrieval of information.
- Airline ticketing systems whose main driver for sharding is fault isolation. They want to shard across tens of independent databases. Failure of a database only makes 1/N of the data momentarily unavailable.
- Social Media companies who may wish to allocate different shards for different classes of users/customer profiles, at different price levels.
- Online Payment Systems that shard for linear scalability and fault isolation, and who may need to satisfy regulatory requirements for storing user data in the country of citizenship.
- Financial and Tax preparation companies who shard by customer id to scale users, workload and transactions. Sharding provides these companies with elasticity required when demand for service peaks during tax filing season.
- Large billing systems where each customer can be identified by a customer ID, phone number, or user ID

Benefits of Oracle Sharding

Sharding with Oracle Database 12c Release 2 provides a number of benefits:

- Linear scalability with complete fault isolation. OLTP applications designed for Oracle sharding can elastically scale (data, transactions and users) to any level, on any platform, simply by deploying new shards on additional stand-alone servers. The unavailability or slowdown of a shard due to either an unplanned outage or planned maintenance affects only the users of that shard, it does not affect the availability or performance of the application for users of other shards. Each shard may run a different release of the Oracle Database as long as the application is backward compatible with the oldest running version – making it simple to maintain availability of an application while performing database maintenance.
- Simplicity via automation of many life-cycle management tasks including: automatic creation of

shards and replication, system managed partitioning, single command deployment, and fine-grained rebalancing.

» Superior run-time performance using intelligent, data-dependent routing.

» All of the advantages of sharding without sacrificing the capabilities of an enterprise RDBMS, including: relational schema, SQL, and other programmatic interfaces, complex data types, online schema changes, multi-core scalability, advanced security, compression, high-availability, ACID properties, consistent reads, developer agility with JSON, and much more.

Sharding Methods

Oracle Sharding supports two methods of sharding: system-managed and composite sharding.

» System-managed sharding does not require the user to specify mapping of data to shards. Data is automatically distributed across shards using partitioning by consistent hash. The partitioning algorithm evenly and randomly distributes data across shards. Such distribution is intended to eliminate hot spots and provide uniform performance across shards. Oracle Sharding automatically maintains balanced distribution of data when shards are added to an SDB. System-managed sharding uses a consistent-hash partitioning strategy that is optimized for Oracle Sharding. System-managed sharding is the most used form of sharding.

» With composite sharding, data is first partitioned by list or range and then further partitioned by consistent hash. The two levels of sharding make it possible to map data to a set of shards, and then automatically maintain balanced distribution of data across that set of shards.

System Managed Sharding - Introduction

Oracle Sharding is a scalability and availability feature that supports distribution and replication of data across hundreds of discrete Oracle databases.

Oracle Sharding uses the Global Data Services (GDS) framework for automatic deployment and management of sharding and replication topologies. GDS also provides load balancing and location-based routing capabilities in an SDB. Global Service Manager – a central component of the GDS framework, acts as the shard director which provides direct routing of requests from the application tier to shards. Shard catalog is a special database that is used to store SDB configuration data and provide other functionality, such as cross-shard queries, centralized schema maintenance and as a source for duplicated tables.

Oracle Sharding is tightly integrated with replication, which provides high availability, and additional scalability for reads. Replication is supported using Oracle Active Data Guard or Oracle Data Guard.

Oracle Sharding provides the capability to automatically deploy the sharded database (SDB), which includes both the shards and the replicas. The SDB administrator defines the topology (regions, shard hosts, replication technology etc.) and invokes the DEPLOY command with declarative specification using the GDSCTL command-line interface.

The high-level steps of the deployment of a sharded database include the following:

a) Prerequisites:

Note: In 12.2.0.1 release, all shards and shard catalog must be Non-CDB databases.

- Create a Non-CDB database that hosts the shard catalog

- Install Oracle Database software on shard nodes
- Install shard director (GSM) software on shard director nodes

Note: For production deployments, it is highly recommended to configure Data Guard for the shard catalog database.

b) Specify the topology layout using:

- CREATE SHARDCATALOG
- ADD GSM
- START GSM
- ADD CREDENTIAL (IF USING 'CREATE SHARD')
- ADD SHARDGROUP
- ADD INVITEDNOTE
- CREATE SHARD (OR ADD SHARD) (for each shard)

c) Run the "Deploy" command and add the global service to access any shard in the SDB:

- DEPLOY
- ADD SERVICE

Oracle Sharding supports two deployment methods. The first method is with the "CREATE SHARD" command, where the creation of shards and the configuration of the replication setup are automatically done by the Oracle Sharding management tier.

The "DEPLOY" command creates the shards. This is done via the DBMS_SCHEDULER package (executed on the shard catalog), which communicates with the scheduler agents on the remote shard hosts. Agents then invoke DBCA and NETCA to create the shards and the local listeners. Once the primary shards are created, the corresponding standby shards are built using the RMAN 'duplicate' command. Once the primary and standby shards are built, the "DEPLOY" command configures the Data Guard Broker with Fast-Start Failover (FSFO) enabled. The FSFO observers are automatically started on the regional shard director.

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

The second method is with the "ADD SHARD" command. Many customers have their own database creation standards and they may opt to deploy the SDB using their own pre-created databases. The ADD SHARD based deployment method supports this requirement by simply adding the shards, which are pre-built by the user.

If the "ADD SHARD" command is used for deployment, the "DEPLOY" command handles the configuration of the Data Guard, Broker and Fast-start Failover. It also handles the scenario where the user has pre-configured Data Guard for the shard that is being added. This cookbook walks you through the SDB deployment using the "ADD SHARD" method.

In the 12.2.0.1 release, two replication schemes are supported. The SDB administrator can select either Data Guard or Active Data Guard while specifying the topology.

In Oracle Sharding, there are two types of deployment:

1) Initial deployment – Initial creation of the sharded database (shards and standbys) is termed as initial deployment
2) Incremental deployment - This allows you to expand or shrink your pool by adding or removing shards.
    a. Scale-out: In System Managed sharding, the addition of new shards to the pool will automatically trigger resharding wherein the chunks are automatically moved in order to attain balanced distribution of data.

    b. Scale-in: In order to shrink the pool, you can use the "REMOVE SHARD" command. In 12.2.0.1, the chunks must be explicitly moved to the other shards before removing the shard.

       Note: Do not use REMOVE SHARD –FORCE unless all the chunks on the given shard (that is being removed) are relocated manually using the "MOVE CHUNK" command.

Oracle Sharding is implemented based on the Oracle partitioning feature. Partitioning decomposes a large table into smaller and more manageable pieces called partitions. Oracle Sharding is essentially *distributed partitioning* since it extends the partitioning feature by supporting distribution of table partitions across shards.
Oracle Sharding uses the familiar SQL syntax for table partitioning to specify how table rows are partitioned across shards. A partitioning key for a sharded table is also the *sharding key*. For example, the following SQL statement can be used to create a sharded table:

```
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;
```

This table is horizontally partitioned across shards based on the value of custId. It is partitioned by *consistent hash* - a special type of hash partitioning commonly used in scalable distributed systems. It provides more flexibility and efficiency in migrating data between shards which is important for elastic scalability.

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

Distribution of partitions across shards is done at the tablespace level. Each partition of a sharded table resides in a separate tablespace and each tablespace is associated with a certain shard. Depending on the sharding method, the association can be established automatically or by the user.

When sharding by consistent hash, tablespaces are automatically spread across shards to provide even distribution of data and workload. All tablespaces are created and managed as a unit called *tablespace set*. PARTITIONS AUTO means that the number of partitions is determined automatically by the system.

In addition to consistent hash, Oracle Sharding also supports sharding by *range* and *list*, as well as composite two-level sharding by *range-consistent hash* and *list-consistent hash*.

Oracle Sharding is tightly integrated with replication which provides high availability and additional scalability for reads. Replication is supported using Oracle Data Guard and Oracle GoldenGate. A unit of replication can be a shard, a part of a shard, or a group of shards.

The variety of sharding and replication methods provided by Oracle Sharding allows customers to customize the topology of the SDB to satisfy specific scalability and availability requirements.

In addition to GDS, Data Guard, GoldenGate and partitioning, Oracle Sharding is integrated with many other Oracle features and products, including JDBC/OCI/ODP.NET connection pools, DBCA, OEM etc.

Cookbook Overview

The objective of this cookbook is to walk through the deployment of a sharded database using system managed sharding method. You will learn the following:

- Creation of shard catalog and shard directors for system managed sharding
- Specify the metadata and deploy the sharded database
- Creation of table family using system managed sharding
- Specify Sharding_Key for session based routing (using SQL*Plus)
- Observe uniform data distribution by executing Read Write and Read Only workloads on the sharded database using a demo application (using UCP)
- Execute cross-shard queries
- Elastically scale the sharded database

Environment Overview

In this cookbook, we will use the following components:

- Two Shard Directors (GSMs)
- One Shard Catalog (chunks=12)
- One Shardspace
    - Primary Shardgroup with 2 Shards
    - Standby Shardgroup with 2 Shards
    - Data Guard for redundancy
- During the last part of the cookbook we will add 2 additional shards (sh5 and sh6) through the elastic sharding functionality

Here is the cookbook topology used for System Managed Sharding:

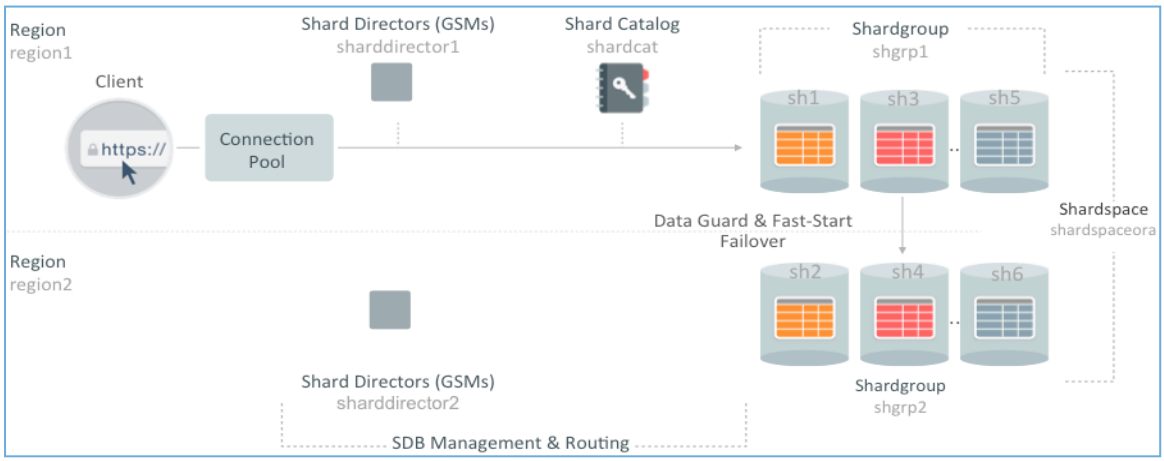


Figure 1. System-Managed Sharding (with Active Data Guard) Demo Topology

Note: For production deployments, it is highly recommended to configure Data Guard for the shard catalog database and three shard directors per region.

Node	Component	Oracle_Home
shard0	shardcat	/u01/app/oracle/product/12.2.0/dbhome_1
shard0	sharddirector1	/u01/app/oracle/product/12.2.0/gsmhome_1
shard1	sh1	/u01/app/oracle/product/12.2.0/dbhome_1
shard2	sh2	/u01/app/oracle/product/12.2.0/dbhome_1
shard3	sh3	/u01/app/oracle/product/12.2.0/dbhome_1
shard4	sh4	/u01/app/oracle/product/12.2.0/dbhome_1
shard5	sh5	/u01/app/oracle/product/12.2.0/dbhome_1
shard6	sh6	/u01/app/oracle/product/12.2.0/dbhome_1
shard6	sharddirector2	/u01/app/oracle/product/12.2.0/gsmhome_1

Figure 2. Environment for the cookbook

Note: Shard1, Shard2, Shard3 and Shard4 will be used in the initial deployment of the Sharded Database (SDB). Shard5 and Shard6 will be used for the incremental deployment.

Environment Prerequisites

Before the sharded database is deployed, perform the following few pre-requisite steps:

A. Hardware and Software sizing for the labs
B. OS user setup
C. Configure the hosts files (for easier navigation) and setup aliases for VMs
D. Setup environment scripts
E. Download the database and gsm media
F. Install Oracle Database software on shard0 to shard6 VMs
G. Create a Non-CDB database on shard0 (which will be used to host the shard catalog)
H. Install shard directors (GSMs) (on shard0 and shard6)

Here are the detailed steps that cover the prerequisites:

A. Hardware and software sizing

You must acquire brand new 7 VMs which do not have any pre-existing Oracle database or listeners running on them.

Here is the minimum hardware/software configuration that we recommend for the cookbook VMs:

CPUs - 2 Cores
Memory - 4G
Disk Space (/u01) > 200G (based on the load that is planned to be executed)
Network - Low Latency GigE
OS – Oracle Enterprise Linux (OEL 64Bit - OS System kernel version – 2.6.39-400.211.1)

B. OS user setup

Create an “oracle” OS user on all VMs – assigned to “dba” group and the password set to “oracle”

Allow the ability for "oracle" OS user to run “su”. (Please check with your Systems Administrator.)

Note: Some of the commands executed as part of this cookbook will take more than 60 seconds to complete. Please ensure that any terminals or remote sessions that are opened have the appropriate keep-alive values set so that the session is not terminated.

C. Configure hosts file and setup aliases for VMs

Update the /etc/hosts file on all the VMs as shown below. Update the ip-addresses based on your environment and replace shardcatvm.bogus.com with the name of the server acting as shard0, which will also be the shard catalog and first shard director (see figure 1).

```
127.0.0.1  localhost.localdomain  localhost

156.151.97.40    shard0 shardcatvm.bogus.com
156.151.97.41         shard1
156.151.97.44         shard2
156.151.97.45         shard3
156.151.97.46         shard4
156.151.97.53         shard5
156.151.97.54         shard6
```

D. Setup environment scripts

Under $HOME of your dedicated VMs, create various environment shell scripts to set your shard env, shard catalog env and shard director (GSM) env.

- shard1.sh , shard2.sh, shard3.sh, shard4.sh, shard4.sh, shard5.sh, shard6.sh (for Shards – on shard1 to shard6 respectively)
- shardcat.sh (for Shard Catalog – on shard0 VM)
- shard-director1.sh and shard-director2.sh (for Shard Directors – on shard0 and shard6 VMs)

On shard0

Save your current path: export SAVEPATH=$PATH

Note: In this example, oracle OS user is using bash shell.

```
$ cd $HOME
$ more shardcat.sh
export ORACLE_SID=shardcat
export ORACLE_BASE=/u01/app/oracle
export ORACLE_HOME=/u01/app/oracle/product/12.2.0/dbhome_1
export LD_LIBRARY_PATH=$ORACLE_HOME/lib
export PATH=$SAVEPATH:$ORACLE_HOME/bin

$ more shard-director1.sh
export ORACLE_BASE=/u01/app/oracle
export ORACLE_HOME=/u01/app/oracle/product/12.2.0/gsmhome_1
export LD_LIBRARY_PATH=$ORACLE_HOME/lib
export PATH=$SAVEPATH:$ORACLE_HOME/bin
```

On shard1:
Save your current path: export SAVEPATH=$PATH

```
$ more shard1.sh
export ORACLE_SID=sh1
export ORACLE_BASE=/u01/app/oracle
export ORACLE_HOME=/u01/app/oracle/product/12.2.0/dbhome_1
export LD_LIBRARY_PATH=$ORACLE_HOME/lib
export PATH=$SAVEPATH:$ORACLE_HOME/bin
```

On shard2:
Save your current path: export SAVEPATH=$PATH

```
$ more shard2.sh
export ORACLE_SID=sh2
export ORACLE_BASE=/u01/app/oracle
export ORACLE_HOME=/u01/app/oracle/product/12.2.0/dbhome_1
export LD_LIBRARY_PATH=$ORACLE_HOME/lib
export PATH=$SAVEPATH:$ORACLE_HOME/bin
```

Similarly, configure the shell scripts - shard3.sh, shard4.sh, shard5.sh and shard6.sh on each of the shard3, shard4, shard5, shard6 VMs.

Additionally, since we are planning to host the 2nd shard director on VM6, create a shell script for shard-director2.sh on shard6

On shard6:

Save your current path: export SAVEPATH=$PATH

```
$ more shard-director2.sh
export ORACLE_BASE=/u01/app/oracle
export ORACLE_HOME=/u01/app/oracle/product/12.2.0/gsmhome_1
export LD_LIBRARY_PATH=$ORACLE_HOME/lib
export PATH=$SAVEPATH:$ORACLE_HOME/bin
```

To make it easier to follow the steps below, open 4 terminal server sessions and set the title for each of the terminal server sessions accordingly.

- Go to TERMINAL -> SET TITLE -> SHARDS
- Go to TERMINAL -> SET TITLE -> SHARDCAT
- Go to TERMINAL -> SET TITLE -> SHARDDIRECTOR1
- Go to TERMINAL -> SET TITLE -> SHARDDIRECTOR2

To connect to a given shard (e.g., shard1), execute the shell script as shown below in the Terminal window SHARDS:

```
$ ssh shard1
$ . ./shard1.sh
$ env |grep ORA
ORACLE_SID=sh1
ORACLE_HOME=/u01/app/oracle/product/12.2.0/dbhome_1
```

E. Download the Oracle Database and Oracle GSM 12.2 software

Download the Oracle Database 12.2 and Oracle Database Global Service Manager 12.2 from
https://edelivery.oracle.com
or
http://www.oracle.com/technetwork/database/enterprise-edition/downloads/index.html

Copy the database software to all the VMs and unzip it to /u01/stage/database.
Copy the gsm software to the shard0 and shard6 VMs and unzip it to /u01/stage/gsm.

F. Install database software on all VMs

Refer to Appendix A for the installation of Oracle 12.2.0.1 database software.

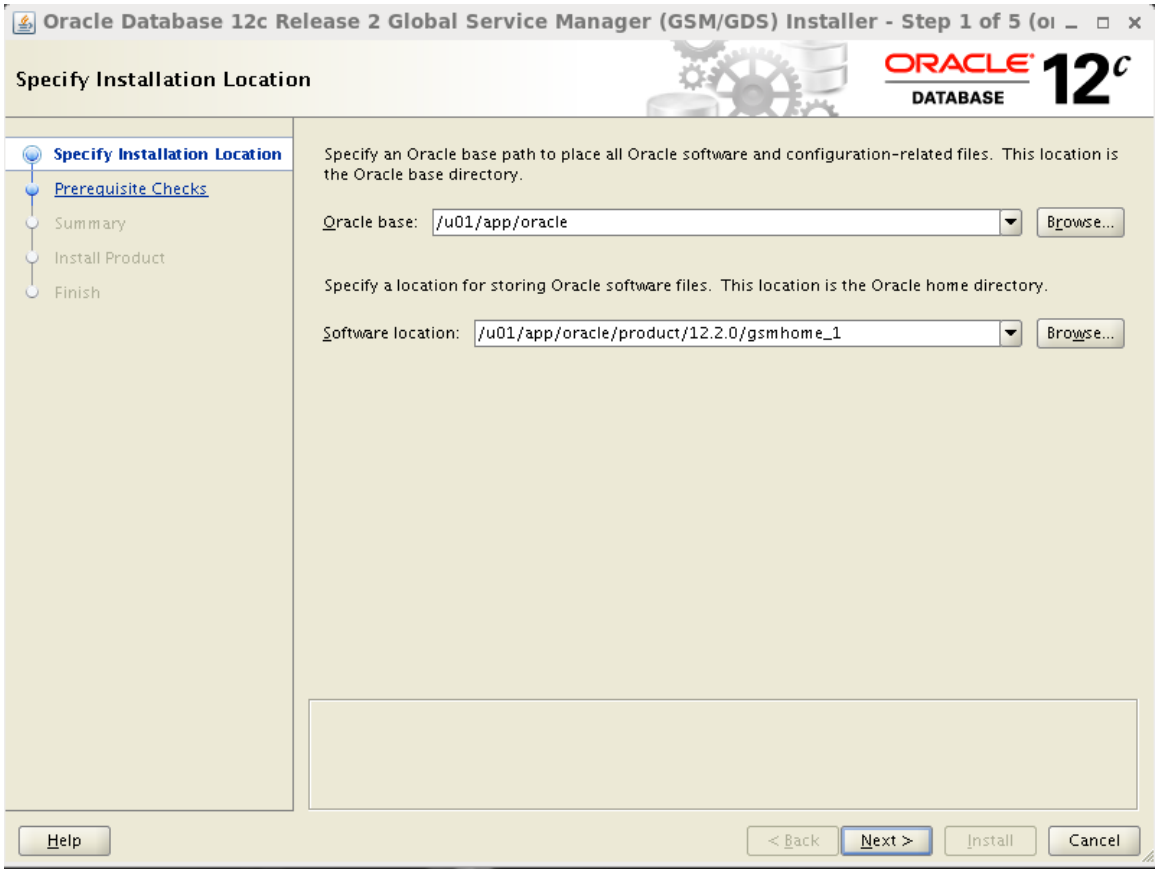
In this cookbook, you will be installing the database software on shard0, shard1, shard2, shard3, shard4, shard5 and shard6

Note: You may opt to create a response file and perform silent database installs on all the shards (listed above)

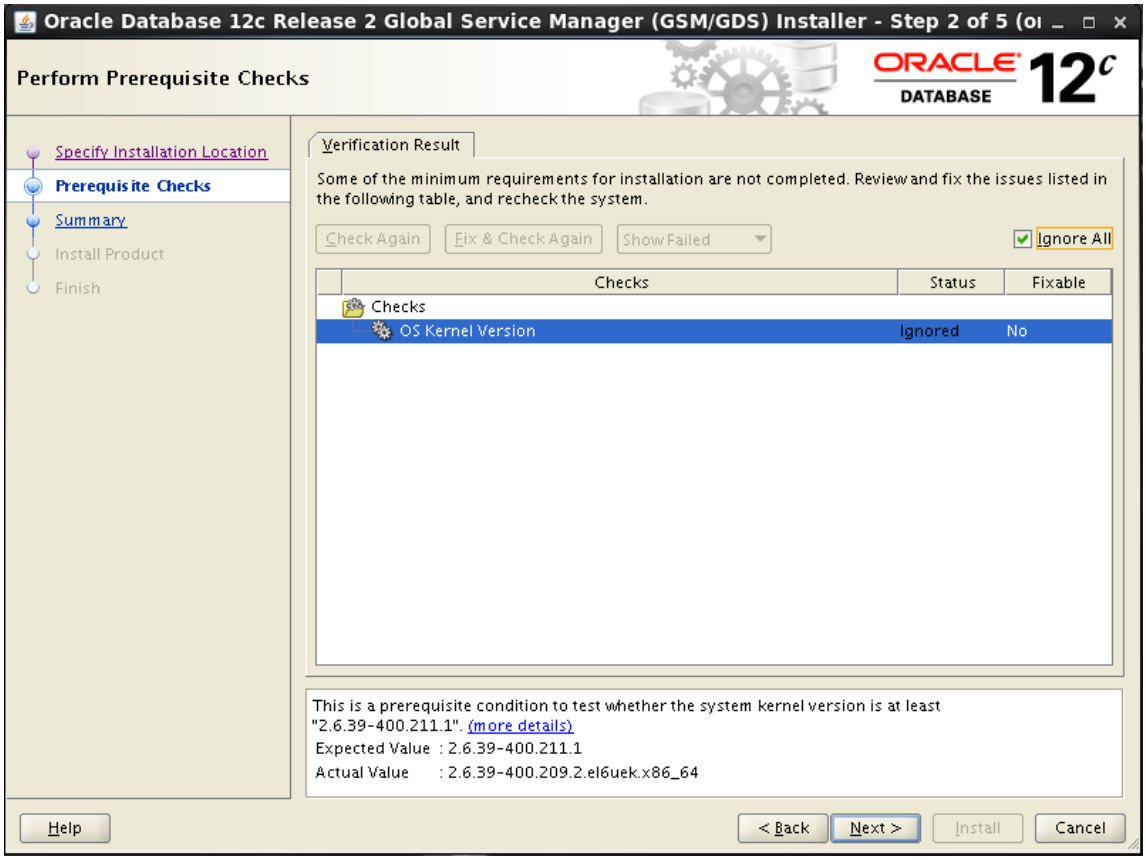
G. Install GSM software on shard0 and shard6

Note: In the lab, we are hosting the two shard directors on shard0 and shard6 respectively.

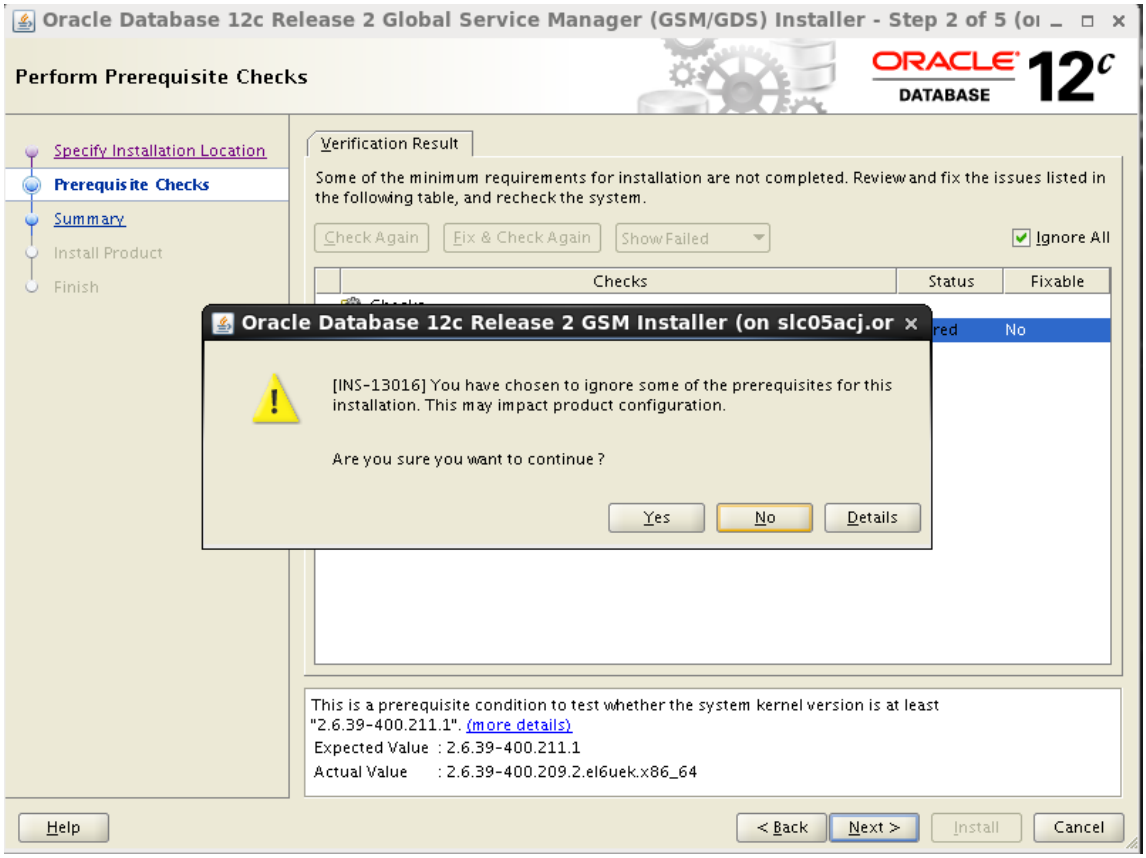
```
$ pwd
/u01/stage/gsm

$ ls
install  response  runInstaller  stage  welcome.html
$ ./runInstaller
```



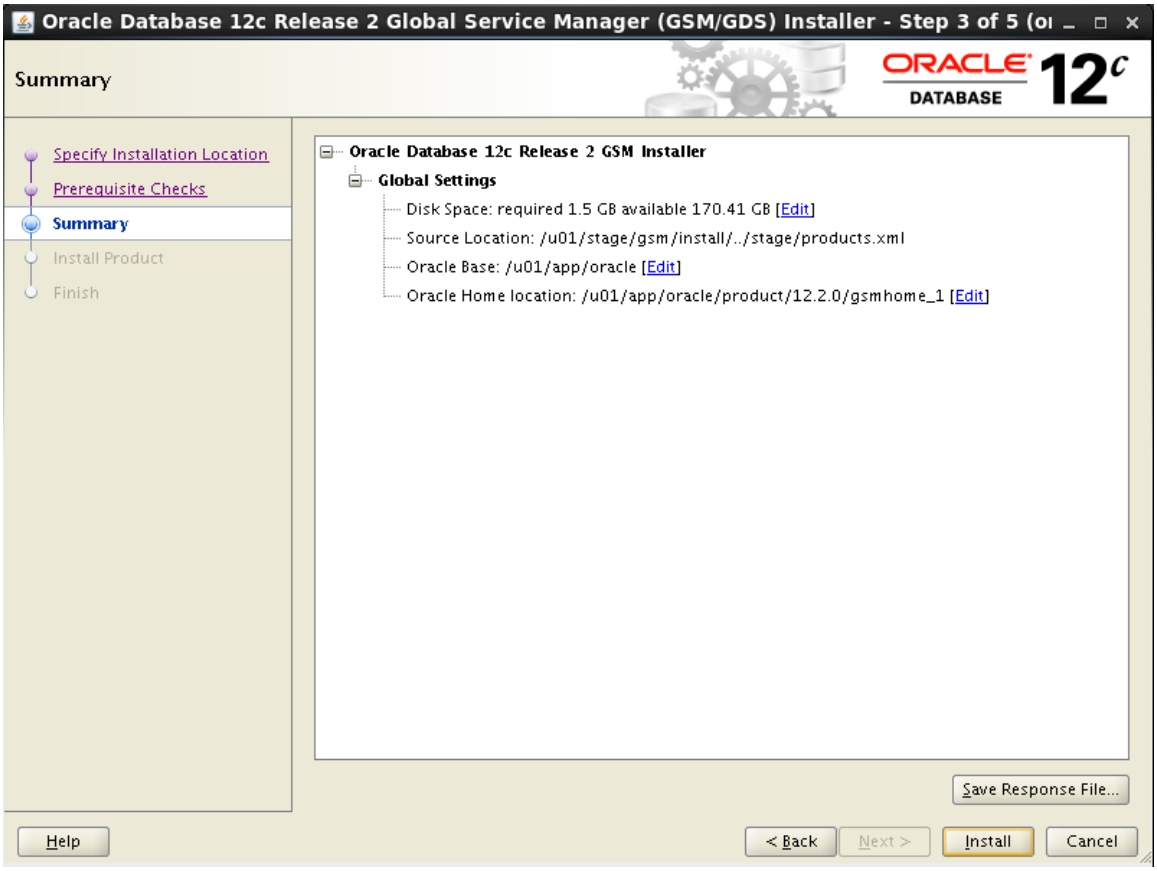
Set the “Oracle base” and “software location” or the GSM binaries and hit “Next”.



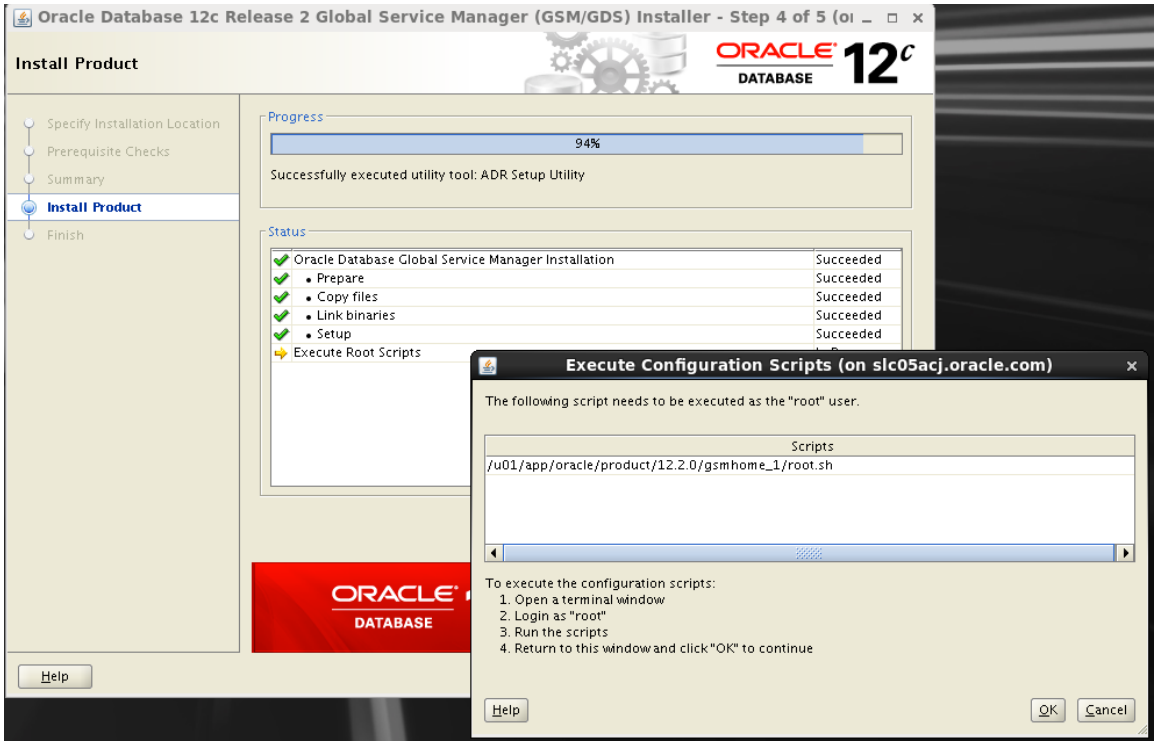
Check the box on “Ignore All” and click “Next”.



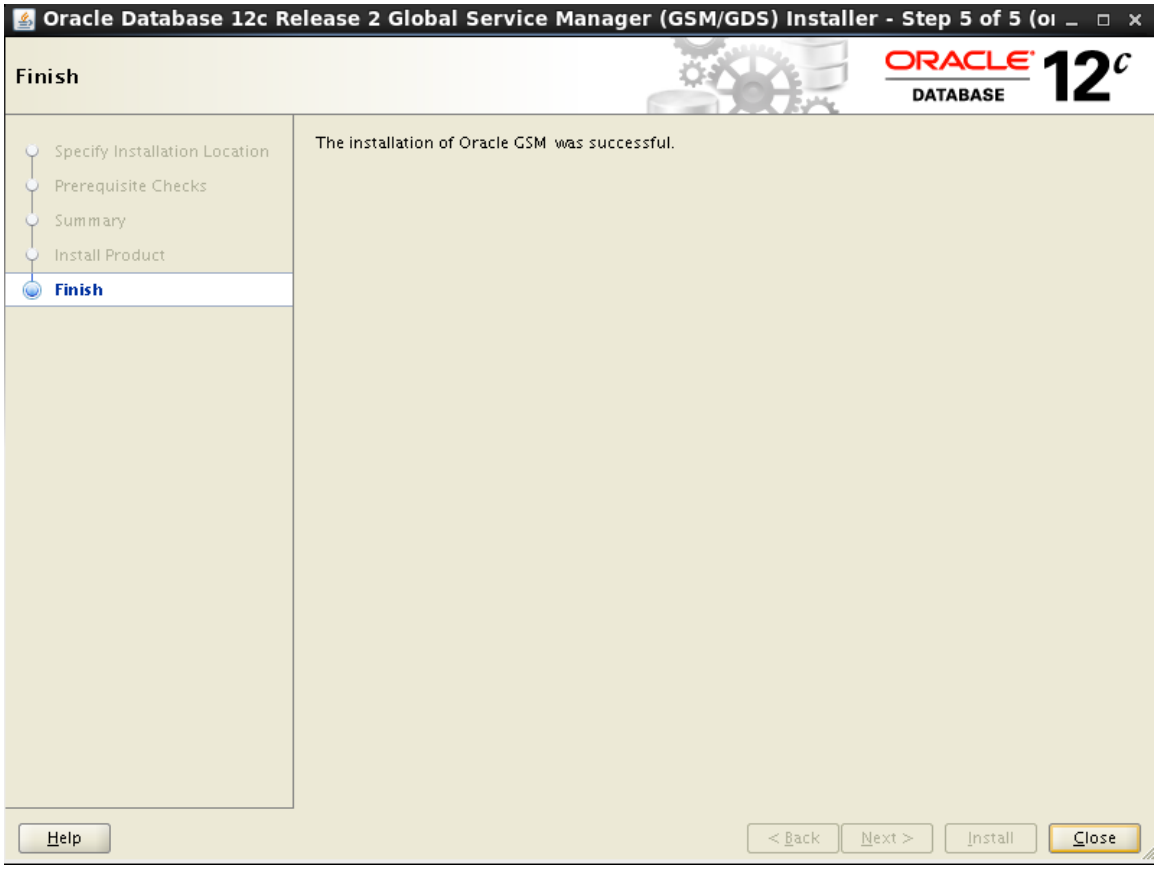
Click on “Yes” and hit “Next”.



Click on “Install”.



Execute the root.sh as root in a separate terminal and click “OK”



Click on “Close” and complete the installation.

Note: Check if you have installed the GSM media on shard0 and shard6.

H. Create a Non-CDB database that hosts the shard catalog

Refer to Appendix B to create the database that hosts the shard catalog.

Note: In 12.2.0.1 release, only the Non-CDB databases are supported for shard catalog and shards.
In this cookbook, you will be creating the shardcat on shard0.

Lab1: Setup the Sharding Management and Routing Tier

Important Pre-Requisites (MUST READ):

1. Each and every shard must be able to reach each and every Shard Director's Listener and ONS ports (Default Listener Port of the Shard Director is 1522 and the default ONS ports are: 6123 for the localONS and 6234 for remoteONS - on most platforms). These Shard Director Listener ports and the ONS ports must also be opened to the Application/Client tier, all the Shards, the shard catalog and all other Shard Directors.
   - *Notes: Shard director listener ports are used by the clients, which make the connections to global services. Each shard also uses these ports to register themselves to the listener, and the Shard Catalog also uses the ports to register the catalog service, so these ports should be accessible to all clients, all shards, and the Shard Catalog.*
2. Each and every shard must be able to reach the TNS Listener port (Default: 1521) of the Shard Catalog (both Primary and Standby)
   - *Notes: All shards do need to connect to the Shard Catalog database via a database link. Therefore, the port being used by the TNS Listener on the catalog host for incoming connections must be made available for connections from all the shard hosts.*
3. The TNS Listener port (Default: 1521) of each shard must be opened to Shard Directors and Shard Catalog

In this lab, you will perform few pre-requisite steps:

- Set the db_create_file_dest parameter on Shard Catalog
- Execute grants and privileges used by the Shard Directors and SDB administrator
- Create shard catalog and shard directors

1. Bring up the Shardcat Env:

From the SHARDCAT terminal, connect to the shardcat database using SQL*Plus :

```
$ cd $HOME
$ . ./shardcat.sh

$ env |grep ORA
ORACLE_SID=shardcat
ORACLE_BASE=/u01/app/oracle
ORACLE_HOME=/u01/app/oracle/product/12.2.0/dbhome_1
```

2. Start the listener of the shardcat:

```
$ lsnrctl start
```

3. Set the db_create_file_dest parameter on shardcat:

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

$ sqlplus / as sysdba

SQL> alter system set db_create_file_dest='/u01/app/oracle/oradata' scope=both;

SQL> alter system set open_links=16 scope=spfile;
SQL> alter system set open_links_per_instance=16 scope=spfile;
```

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

SQL> startup
ORACLE instance started.

Total System Global Area 4798283776 bytes
Fixed Size                  4430760 bytes
Variable Size            1006634072 bytes
Database Buffers         3774873600 bytes
Redo Buffers               12345344 bytes
Database mounted.
Database opened.
SQL>
```

4. Grant Roles/Privileges (on accounts used by Shard Directors) on shardcat:

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

Unlock and set the password for the gsmcatuser schema. This schema is used by the shard director while connecting to the shard catalog database.

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

Create the administrator schema (mygdsadmin) and give the privileges

```
SQL> create user mygdsadmin identified by passwd_mygdsadmin;
SQL> grant connect, create session, gsmadmin_role to mygdsadmin;
```

Enable tracing on the shard catalog

```
SQL> alter system set events 'immediate trace name GWM_TRACE level 7';
SQL> alter system set event='10798 trace name context forever, level 7'
SCOPE=spfile;

SQL> spool off
```

5. Launch GDSCTL and create the shard catalog:

On a new terminal and name it SHARDDIRECTOR1 Terminal:

```
$ cd $HOME
$ . ./shard-director1.sh
$ env |grep ORA
ORACLE_BASE=/u01/app/oracle
```

ORACLE_HOME=/u01/app/oracle/product/12.2.0/gsmhome_1

Note: For all GDSCTL commands that span more than one line, they should not be copy pasted to terminal directly. Please copy to any other text editor, make them one line instead of multiple lines as given in doc and then paste in terminal.
The other option is to type the command by hand.

```
$ gdsctl
```

Create the shardcatalog

```
GDSCTL> create shardcatalog -database shard0:1521:shardcat -chunks 12 -user
mygdsadmin/passwd_mygdsadmin -sdb cust_sdb -region region1,region2
```

Create and start the first shard director

```
GDSCTL> add gsm -gsm sharddirector1 -listener 1571 -pwd passwd_gsmcatuser -
catalog shard0:1521:shardcat -region region1 -trace_level 16

GDSCTL> start gsm -gsm sharddirector1
```

Enable debugging on the shard director

```
GDSCTL> set _event 17 -config_only
```

6. Add the Shard Director2 :

Note: In this lab, we are hosting Shard Director2 on the shard6 node.

```
$ ssh oracle@shard6

$ cd $HOME
$ . ./shard-director2.sh

$ env |grep ORA
ORACLE_BASE=/u01/app/oracle
ORACLE_HOME=/u01/app/oracle/product/12.2.0/gsmhome_1


$ gdsctl

GDSCTL> add gsm -gsm sharddirector2 -listener 1572 -pwd passwd_gsmcatuser -
catalog shard0:1521:shardcat -trace_level 16 -region region2

GDSCTL> start gsm -gsm sharddirector2
```

Enable debugging on the shard director

```
GDSCTL> set _event 17 -config_only


Type "exit" to exit from GDSCTL
```

LAB2: Sharded Database Deployment:

In this lab, you will perform the following steps:

- Compile the metadata for the sharded database
  - Create two shardgroups - one for Primary and another for Standby regions
  - Create and configure databases to be used as shards.
- Execute the "DEPLOY" command to create the SDB as specified in the metadata
- Create role-based global services

First, create 4 databases to be used as shards in whatever manner you choose (manually, DBCA, etc.). A database needs to be created on shard1, shard2, shard3, and shard4.

These databases created must have the following characteristics:

- They must not be container databases (CDBs)
- They must have an associated TNS Listener on port 1521 in each VM
- The GSMUSER account must be unlocked with a known password
- Grant SYSDG and SYSBACKUP privileges to GSMUSER
- The databases on shard1 and shard2 must be primary databases
- The databases on shard3 and shard4 must be physical standby databases
- Redo apply should be setup between the corresponding primary and standby databases prior to adding them the sharding configuration.
- Database flashback and force logging should be enabled.
- The 'compatible' parameter must be at least 12.2.0
- A server parameter file (SPFILE) must be in use.
- A 'DATA_PUMP_DIR' directory object must be created in each database and must point to a valid directory.

To validate that a database is correctly set up for sharding, execute the following against each database before adding it to the configuration:

```
$ cd $HOME
$ . ./shard1.sh

$ sqlplus / as sysdba

set serveroutput on
execute DBMS_GSM_FIX.validateShard
```

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

On SHARDDIRECTOR1 Terminal:

```
$ ssh oracle@shard1

$ cd $HOME
$ . ./shard-director1.sh
```

7. Compile the metadata for all the shards that are being added to the sharded database (Define shardgroups and Shards):

```
$ gdsctl

GDSCTL> set gsm -gsm sharddirector1

GDSCTL> connect mygdsadmin/passwd_mygdsadmin
```

Add a shardgroup shgrp1 for primary shards

```
GDSCTL> add shardgroup -shardgroup shgrp1 -deploy_as primary -region region1
```

Add a shardgroup shgrp2 for Active Data Guard standby shards

```
GDSCTL> add shardgroup -shardgroup shgrp2 -deploy_as active_standby -region region2
```

Execute the “add invitednode”

Notes: "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 (GDS 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. If the host has multiple public IP addresses, then the one on which the database registers may not be the same as the one which was added using auto-VNCR, as a result, registration many be rejected. If the target database host has multiple public IPs, it is advisable to 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 the user should simply check with “config vncr” and use “add invitednode” as necessary. There is no harm in doing this, if the node is added already, auto-VNCR will ignore it, and if the user tries to add it after auto-VNCR already added it, they will simply get a warning stating that it already exists.

```
GDSCTL> add invitednode shard1
```

Specify the shardgroup, connect string and GSMUSER password for each shard.

```
GDSCTL> add shard -shardgroup shgrp1 -connect shard1:1521/sh1 -pwd <GSMUSER_password>

GDSCTL> add invitednode shard2

GDSCTL> add shard -shardgroup shgrp1 -connect shard2:1521/sh2 -pwd <GSMUSER_password>

GDSCTL> add invitednode shard3

GDSCTL> add shard -shardgroup shgrp1 -connect shard3:1521/sh3 -pwd <GSMUSER_password>
```

```
GDSCTL> add invitednode shard4

GDSCTL> add shard -shardgroup shgrp1 -connect shard4:1521/sh4 -pwd <GSMUSER_password>
```

8. Check the configuration from any Shard Director :

```
GDSCTL> config

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

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

Sharded Database
------------------------
cust_sdb

Databases
------------------------
sh1
sh2
sh3
sh4

Shard Groups
------------------------
shgrp1
shgrp2

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

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

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

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

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

GDSCTL> config shardgroup
Shard Group          Chunks Region              SHARDSPACE
-----------          ------ ------              ----------
shgrp1               12     region1             shardspaceora
shgrp2               12     region2             shardspaceora

GDSCTL> config vncr

Name                          Group ID
----                          --------
shard1
shard2
shard3
shard4
10.xxx.yy.zz1
10.xxx.yy.zz2
10.xxx.yy.zz3
10.xxx.yy.zz4

GDSCTL> config shard
Name                Shard Group         Status    State       Region    Availability
----                -----------         ------    -----       ------    ------------
sh1                 shgrp1              U         none        region1   -
sh2                 shgrp2              U         none        region2   -
sh3                 shgrp1              U         none        region1   -
sh4                 shgrp2              U         none        region2   -
```

9. Run the DEPLOY command to create the shards and the replicas :

(NOTE – The “deploy” command will take some time to run - anywhere from 15 to 30 minutes.)

```
GDSCTL> deploy
```

Once the primary and standby shards are built, the “DEPLOY” command configures the Data Guard Broker with Fast-Start Failover (FSFO) enabled. The FSFO observers are automatically started on the region2’s (which is assigned to the standby’s shardgroup) shard director.

10. Check the configuration of all shards and observe that the state is “Deployed” :

```
GDSCTL> config shard
Name                Shard Group         Status    State       Region    Availability
----                -----------         ------    -----       ------    ------------
sh1                 shgrp1              Ok        Deployed    region1   ONLINE
sh2                 shgrp2              Ok        Deployed    region2   READ_ONLY
sh3                 shgrp1              Ok        Deployed    region1   ONLINE
sh4                 shgrp2              Ok        Deployed    region2   READ_ONLY
```

Verify that shard1.sh (on shard1), shard2.sh (on shard2), shard3.sh (on shard3) and shard4.sh (on shard4) reflect the correct names of the shards that have been created specific to your deployment. If not, update the scripts to reflect the names of the shards accordingly.

11. Observe that all shards are “Registered”:

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

12. Check the configuration of a given shard (e.g., sh1) :

```
GDSCTL> config shard -shard sh1
Name: sh1
Shard Group: 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: 12.2.0.0
Last Failed DDL:
DDL Error: ---
Failed DDL id:
Availability: ONLINE


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

13. Add a global service that runs on all the Primary shards :

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

GDSCTL> config service

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

14. Start the Shard Director :

```
GDSCTL> start service -service oltp_rw_srvc
GDSCTL> status service
Service "oltp_rw_srvc.cust_sdb.oradbcloud" has 2 instance(s). Affinity:
ANYWHERE
   Instance "cust_sdb%1", name: "sh1", db: "sh1", region: "region1",
status: ready.
   Instance "cust_sdb%21", name: "sh3", db: "sh3", region: "region1",
status: ready.
```

15. Add the global service for Read Only workload to run on the shards in standby mode

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

GDSCTL> config service
Name           Network name                  Pool          Started Preferred all
----           ------------                  ----          ------- -------------
oltp_rw_srvc   oltp_rw_srvc.cust_sdb.oradbcl cust_sdb      Yes     Yes
               oud
oltp_ro_srvc   oltp_ro_srvc.cust_sdb.oradbcl cust_sdb      Yes     Yes
               oud
```

16. Verify the status of the global services:

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

LAB 3: Creation of Schema for System Managed Sharding

In this lab, you will perform the following steps:

- Create the schema user, tablespace set, sharded tables and duplicated tables
- Verify that the DDLs have been propagated to all the shards
- While connected to the shards, verify the automatic Data Guard broker configuration with fast_start failover

17. Create the sharded and duplicated tables as shown below from the shard catalog database.

On the shard catalog serve, ensure that your ORACLE_SID = shardcat.:

```
$ cd $HOME
$ . ./shardcat.sh

$ sqlplus / as sysdba
```

Execute the following SQL commands to create the customer table family using system managed sharding method.

```
set echo on
set termout on
set time on
spool /u01/stage/labs/create_app_schema.lst
REM
REM Connect to the Shard Catalog and Create Schema
REM
connect / as sysdba
alter session enable shard ddl;
create user app_schema identified by app_schema;
grant connect, resource, alter session to app_schema;
grant execute on dbms_crypto to app_schema;
grant create table, create procedure, create tablespace, create
materialized view to app_schema;
grant unlimited tablespace to app_schema;
grant select_catalog_role to app_schema;

grant all privileges to app_schema;
grant gsmadmin_role to app_schema;
grant dba to app_schema;


REM
REM Create a tablespace set for SHARDED tables
REM
CREATE TABLESPACE SET  TSP_SET_1 using template (datafile size 100m
autoextend on next 10M maxsize unlimited extent management  local
segment space management auto );

REM
REM Create a tablespace for DUPLICATED tables
```

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

REM
REM Create Sharded and Duplicated tables
REM
connect app_schema/app_schema
alter session enable shard ddl;
REM
REM Create a Sharded table for Customers  (Root table)
REM
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;

REM
REM Create a Sharded table for Orders
REM
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);

REM
REM Create the sequence used for the OrderId column
REM
CREATE SEQUENCE Orders_Seq;

REM
REM Create a Sharded table for LineItems
REM
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);

REM
REM Create Duplicated table for Products
REM
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;

REM
REM Create functions for Password creation and checking - used by the
REM demo loader application
REM

CREATE OR REPLACE FUNCTION PasswCreate(PASSW IN RAW)
  RETURN RAW
IS
  Salt RAW(8);
BEGIN
  Salt := DBMS_CRYPTO.RANDOMBYTES(8);
  RETURN UTL_RAW.CONCAT(Salt, DBMS_CRYPTO.HASH(UTL_RAW.CONCAT(Salt,
PASSW), DBMS_CRYPTO.HASH_SH256));
END;
/

CREATE OR REPLACE FUNCTION PasswCheck(PASSW IN RAW, PHASH IN RAW)
  RETURN INTEGER IS
BEGIN
  RETURN UTL_RAW.COMPARE(
      DBMS_CRYPTO.HASH(UTL_RAW.CONCAT(UTL_RAW.SUBSTR(PHASH, 1, 8),
PASSW), DBMS_CRYPTO.HASH_SH256),
      UTL_RAW.SUBSTR(PHASH, 9));
END;
/

REM
REM
select table_name from user_tables;
REM
REM
spool off
```

18. From the SHARDDIRECTOR1 Terminal, run the following commands to observe that there are no failures during the creation of tablespaces

```
GDSCTL>show ddl
id      DDL Text                                 Failed shards
--      --------                                 -------------
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 database link "PRODUCTSDBLINK@...
15      CREATE MATERIALIZED VIEW "PRODUCTS"  ...
16      CREATE OR REPLACE FUNCTION PasswCreat...
17      CREATE OR REPLACE FUNCTION PasswCheck...
```

19. Run the config commands as shown below for each of the shards and verify if there are any DDL errors

```
GDSCTL>config shard -shard sh1
Name: sh1
Shard Group: 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: 12.2.0.0
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
```

20. Observe that the tablespaces of the tablespace set are created on all shards based on the number of chunks specified in the “create shardcatalog” command. Also, check on all shards to verify that the products_tsp tablespace has been created.

On Shard1:

```
$ . ./shard1.sh
$ 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.
```

On Shard3:

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

TABLESPACE_NAME                MB
------------------------------ ----------
C007TSP_SET_1                         100
C008TSP_SET_1                         100
C009TSP_SET_1                         100
```

```
C00ATSP_SET_1                          100
C00BTSP_SET_1                          100
C00CTSP_SET_1                          100
PRODUCTS_TSP                           100
SYSAUX                                 650
SYSTEM                                 890
SYS_SHARD_TS                           100
TSP_SET_1                              100

TABLESPACE_NAME                     MB
------------------------------ ----------
UNDOTBS1                               100
USERS                                    5

13 rows selected.
```

21. Login into each shard (sh1 and sh3) and verify that the chunks and chunk tablespaces are created

```
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

SQL> select table_name, partition_name, tablespace_name from dba_tab_partitions
where tablespace_name like 'C%TSP_SET_1' order by tablespace_name;


NAME                                 TYPE      VALUE
------------------------------------ ----------- ------------------------------
db_unique_name                       string    sh3

TABLE_NAME          PARTITION_NAME     TABLESPACE_NAME
------------------- ------------------- --------------------
ORDERS              CUSTOMERS_P7       C007TSP_SET_1
CUSTOMERS           CUSTOMERS_P7       C007TSP_SET_1
LINEITEMS           CUSTOMERS_P7       C007TSP_SET_1
CUSTOMERS           CUSTOMERS_P8       C008TSP_SET_1
LINEITEMS           CUSTOMERS_P8       C008TSP_SET_1
ORDERS              CUSTOMERS_P8       C008TSP_SET_1
CUSTOMERS           CUSTOMERS_P9       C009TSP_SET_1
ORDERS              CUSTOMERS_P9       C009TSP_SET_1
LINEITEMS           CUSTOMERS_P9       C009TSP_SET_1
ORDERS              CUSTOMERS_P10      C00ATSP_SET_1
CUSTOMERS           CUSTOMERS_P10      C00ATSP_SET_1

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

22. Login into the system on the shardcatalog and query the gsmadmin_internal.chunk_loc table to observe that the chunks are uniformly distributed.

```
$ . ./shardcat.sh

SQL> 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
```

23. Login into the app_schema/app_schema on the shardcatalog1, shard1, shard2, shard3, shard4 databases and verify that the sharded and duplicated tables are created.

```
$ . ./shard1.sh
$ sqlplus app_schema/app_schema
Connected.
SQL> select table_name from user_tables;

TABLE_NAME
--------------------------------------------------------------------------------
CUSTOMERS
ORDERS
LINEITEMS
PRODUCTS

4 rows selected.
```

24. Verify the automatic configuration of Data Guard broker and fast_start failover

Note: You may perform the following steps on sh3 and sh1 shards to verify the data guard broker configuration.

```
$ ssh oracle@shard3
$ . ./shard3.sh
$ dgmgrl


DGMGRL for Linux: Release 12.2.0.0.2 - Beta on Wed Jan 20 02:49:58 2016

Copyright (c) 1982, 2015, Oracle and/or its affiliates.  All rights
reserved.
```

```
Welcome to DGMGRL, type "help" for information.
DGMGRL> connect sys/oracle
Connected to "sh3"
Connected as SYSDG.
DGMGRL> show configuration

Configuration - sh3

  Protection Mode: MaxPerformance
  Members:
  sh3 - Primary database
    sh4 - (*) Physical standby database

Fast-Start Failover: ENABLED

Configuration Status:
SUCCESS   (status updated 15 seconds ago)

DGMGRL> show database sh3

Database - sh3

  Role:               PRIMARY
  Intended State:     TRANSPORT-ON
  Instance(s):
    sh3

Database Status:
SUCCESS

DGMGRL> show database sh4

Database - sh4

  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):
    sh4

Database Status:
SUCCESS
```

```
DGMGRL> show fast_start failover

Fast-Start Failover: ENABLED

  Threshold:          30 seconds
  Target:             sh4
  Observer:           10.xxx.yy.zz6
  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)
```

25. Locate the FSFO observers by connecting to SHARDCAT and SHARD6 nodes and execute the following:

```
$ 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/12.2.0/gsmhome_1/network/admin/gsm_observer_1.cfg
oracle   20193     1  0 02:57 ?        00:02:43 dgmgrl -delete_script
@/u01/app/oracle/product/12.2.0/gsmhome_1/network/admin/gsm_observer_2.cfg
```

Lab4: Data-dependent Routing

In this lab, you will perform the following steps:

- Connect to a shard by specifying a sharding_key – via the shard directors
- Connect to the shardcatalog via GDS$CATALOG service

Note: This lab is just to understand how the routing works when a sharding_key is specified using SQL*Plus. For production application scenario, you would be using Oracle Integrated Connection pools – UCP, OCI, ODP.NET, JDBC etc which will allow direct routing based on the sharding_key. The Demo loader application (in Lab# 5) uses UCP.

26. Verify Data-dependent routing via SHARDING_KEY

For single-shard queries, connect to a shard with a given sharding_key using GSMs:

```
sqlplus
app_schema/app_schema@'(description=(address=(protocol=tcp)(host=shard0
)(port=1571))(connect_data=(service_name=oltp_rw_srvc.cust_sdb.oradbclo
ud)(region=region1)(SHARDING_KEY=james.parker@x.bogus)))'

SQL> INSERT INTO Customers (CustId, FirstName, LastName, CustProfile,
Class, Geo, Passwd) VALUES ('james.parker@x.bogus', 'James', 'Parker',
NULL, 'Gold', 'east', hextoraw('8d1c00e'));

SQL> commit;

SQL> select db_unique_name from v$database;

DB_UNIQUE_NAME
------------------------------
sh1

SQL>  exit;

$ sqlplus
app_schema/app_schema@'(description=(address=(protocol=tcp)(host=shard0
)(port=1571))(connect_data=(service_name=oltp_rw_srvc.cust_sdb.oradbclo
ud)(region=region1)(SHARDING_KEY=james.parker@x.bogus)))'

SQL> column custid format a20
SQL> column firstname format a15
SQL> column lastname format a15
SQL> select custid, FirstName, LastName, class, geo from customers
where custid = 'james.parker@x.bogus';

CUSTID               FIRSTNAME       LASTNAME        CLASS      GEO
-------------------  --------------  --------------  ---------- --------
james.parker@x.bogus  James           Parker          Gold       east

SQL> SELECT sys_context('USERENV', 'INSTANCE_NAME') FROM DUAL;

SYS_CONTEXT('USERENV','INSTANCE_NAME')
--------------------------------------------------------------------------------
cust_sdb%1
```

```
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
```

Modify the sharding_key while connecting via SQL*Plus and observe that your connection is routed to the shard which maps to the consistent-hash value of the sharding_key.

To perform cross-shard queries, connect to the shardcatalog (coordinator database) using the GDS$CATALOG service (from any shard):

```
sqlplus app_schema/app_schema@shard0:1521/GDS\$CATALOG.oradbcloud
```

Examples of Cross Shard queries are covered in Lab#6

Lab5: Custom Data Loading Application

27. **Download the SDB Demo App zip file**

To learn more about Oracle Sharded Databases, download and deploy the system-managed SDB demo application. The demo application uses the SDB environment and schema you have just created to simulate the workload of an online retail store. You can download the latest version of the demo application, along with a README file that describes how to run and monitor it, from Master Note for Handling Oracle Sharding - Oracle Database 12.2 Technology (Doc ID 2226341.1)

Download the sdb_demo_app.zip from

https://support.oracle.com/epmos/faces/DocumentDisplay?id=2226341.1

Copy the zip file to the HOME directory of oracle on shard0. (In this lab, we are running the demo app on shard0.)

```
$ cd $HOME
$ . ./shardcat.sh
$ unzip sdb_demo_app.zip
```

This will create sdb_demo_app directory under the $HOME.

28. **Setup and configure the Sharding Demo Application**

Refer to the README_SDB_Demo_Application document (in the sdb_demo_app directory) for the information on the setup and configuration of the demo application and monitoring tool.

Lab6: Cross-shard Querying

The objective of this hands-on lab is to walk through the execution of various cross shard queries on a sharded database.

Note: All the exercises are performed on the shard catalog database. Also, the data sample used in the workbook is different from your environment and hence the output that you will observe will be different (due to data load randomization).

In a sharded database, a database client connects to the shard using a connection string with a sharding key value. If a sharding key is specified then all requests submitted in that session will be routed to the shard corresponding to the key value. They are referred to as Single Shard Queries (SSQ).

If the sharding key cannot be provided as part of database connection string, then a session will have to be established on the coordinator database (shardcat). All the queries submitted from such sessions can in principle touch data on any set of shard databases. They are referred to as Cross Shard Queries (CSQ).

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

$$Q \Rightarrow CQ\,(\,Shard\_Iterator(\,SQ\,)\,)$$

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

Q1 : SELECT COUNT(*) FROM customers

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

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

During the compilation of a query on the coordinator database, the query optimizer analyzes the predicates on the sharding key and extracts the ones that can be used to identify the participating shards, i.e. shards that will contribute rows for the sharded tables referenced in the query. The other shards are referred to as pruned shards. In the case only one participating shard was identified then the full query is routed to that shard for a full execution otherwise the query is rewritten. The rewriting process takes into account the expressions computed by the query as well as the query shape. The examples provided in this Lab will help illustrate both the rewrite process as well as the identification of the participating shards.

```
$ . ./shardcat.sh
```

```
$ sqlplus  app_schema/app_schema
```

From the SHARDCAT terminal, connect to the shardcat database using SQL*Plus :

```
$ cd /home/oracle
$ . ./shardcat.sh

$ env |grep ORA
ORACLE_SID=shardcat
ORACLE_BASE=/u01/app/oracle
ORACLE_HOME=/u01/app/oracle/product/12.2.0/dbhome_1
```

Run the CSQ queries to load some sample rows into the tables.

```
SQL> set termout on
SQL> set linesize 120
SQL> set echo on
SQL> REM
SQL> REM Conventional Insert
SQL> REM
SQL>
SQL> INSERT INTO Customers (CustId, FirstName, LastName, CustProfile, Class, Geo, Passwd)
VALUES ('Scott.Tiger@x.bogus', 'Scott', 'Tiger', NULL, 'free', 'west',
hextoraw('7d1b00f'));

1 row created.

SQL>
SQL> INSERT INTO Customers (CustId, FirstName, LastName, CustProfile, Class, Geo, Passwd)
VALUES ('Mary.Parker@x.bogus', 'Mary', 'Parker', NULL, 'Gold', 'east',
hextoraw('8d1c00e'));

1 row created.

SQL> commit;

Commit complete.
```

Now, let's run a CSQ query which does a SELECT with ORDER BY query accessing multiple shards but not all shards

```
SQL> set termout on
SQL> set linesize 120
SQL> set echo on
SQL> column firstname format a20
SQL> column lastname format a20
SQL> REM  SELECT with ORDER BY query accessing multiple shards but not all shards
SQL> explain plan for SELECT FirstName,LastName, geo, class FROM Customers
WHERE CustId in ('Scott.Tiger@x.bogus', 'Mary.Parker@x.bogus') AND class != 'free' ORDER
BY geo, class;

Explained.

SQL> set echo off

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------------------------
-
Plan hash value: 1622328711

-----------------------------------------------------------------------------------------------
| Id  | Operation     | Name        | Rows  | Bytes | Cost (%CPU)| Time     | Inst   |IN-OUT|
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |                  |   100 |  7700 |     1 (100)| 00:00:01 |        |      |
|   1 |  SORT ORDER BY          |                  |   100 |  7700 |     1 (100)| 00:00:01 |        |      |
|   2 |   VIEW                | VW_SHARD_5B3ACD5D |   100 |  7700 |     5 (100)| 00:00:01 |        |      |
|   3 |    SHARD ITERATOR |                  |       |       |            |          |        |      |
|   4 |     REMOTE    |                  |       |       |            |          | ORA_S~ | R->S |
-----------------------------------------------------------------------------------------------
```

```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------
--------------------------------

Remote SQL Information (identified by operation id):
----------------------------------------------------

   4 - EXPLAIN PLAN INTO PLAN_TABLE@! FOR SELECT
       "A1"."FIRSTNAME","A1"."LASTNAME","A1"."GEO","A1"."CLASS" FROM "CUSTOMERS" "A1"
WHERE
       ("A1"."CUSTID"='Mary.Parker@x.bogus' OR "A1"."CUSTID"='Scott.Tiger@x.bogus') AND
       "A1"."CLASS"<>'free' /* coord_sql_id=gq42axzj3ns5t */  (accessing
       'ORA_SHARD_POOL@ORA_MULTI_TARGET' )


21 rows selected.

SQL> REM  SELECT with ORDER BY query accessing multiple shards but not all shards
SQL> REM
SQL> SELECT FirstName,LastName, geo, class FROM Customers
WHERE CustId in ('Scott.Tiger@x.bogus', 'Mary.Parker@x.bogus') AND class != 'free' ORDER
BY geo, class;

FIRSTNAME            LASTNAME             GEO      CLASS
-------------------- -------------------- -------- ----------
Mary                 Parker               east     Gold
```

Let's run a CSQ query which joins sharded and duplicated table (join on non sharding key) to get the fast moving products (qty sold > 10)

```
SQL> set echo on
SQL> column name format a40
SQL> REM Join sharded and duplicated table (join on non sharding key) to get the fast
moving products (qty sold > 10)
SQL> explain plan for SELECT name, SUM(qty) qtysold FROM lineitems l, products p
WHERE l.productid = p.productid
GROUP BY name HAVING sum(qty) > 500 ORDER BY qtysold desc;

Explained.

SQL> set echo off

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------------------------
-
Plan hash value: 2127005259

-------------------------------------------------------------------------------------------------------
| Id  | Operation       | Name                    | Rows    | Bytes | Cost (%CPU)| Time   | Inst     |IN-OUT|
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |                     |   100 |  7900 |     5 (100)| 00:00:01 |          |        |
|   1 |  SORT ORDER BY           |                     |   100 |  7900 |     5 (100)| 00:00:01 |          |
          |
|*  2 |   FILTER        |                         |       |       |            |          |          |
          |
|   3 |    HASH GROUP BY    |                     |   100 |  7900 |     5 (100)| 00:00:01 |          |        |
|   4 |     VIEW        | VW_SHARD_372F2D25 |   100 |  7900 |     5 (100)| 00:00:01 |      |        |
|   5 |      SHARD ITERATOR |                     |       |       |            |          |          |
        |

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------------------------
-
|   6 |       REMOTE             |                     |       |       |            |          | ORA_S~
| R->S |
-------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
```

```
   2 - filter(SUM("ITEM_1")>500)

Remote SQL Information (identified by operation id):
----------------------------------------------------


PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------
--------------------------------
   6 - EXPLAIN PLAN INTO PLAN_TABLE@! FOR SELECT SUM("A2"."QTY"),"A1"."NAME" FROM
"LINEITEMS"
       "A2","PRODUCTS" "A1" WHERE "A2"."PRODUCTID"="A1"."PRODUCTID" GROUP BY "A1"."NAME"
/*
       coord_sql_id=c0v333jfqvd4s */  (accessing 'ORA_SHARD_POOL@ORA_MULTI_TARGET' )


26 rows selected.

SQL> REM Join sharded and duplicated table (join on non sharding key) to get the fast
moving products (qty sold > 10)
SQL> SELECT name, SUM(qty) qtysold FROM lineitems l, products p
WHERE l.productid = p.productid
GROUP BY name HAVING sum(qty) > 10 ORDER BY qtysold desc;

NAME                                       QTYSOLD
---------------------------------------- ----------
Fuel cell                                        14
Fuel tank cover                                  11
Tire pressure gauge                              11
```

Let's run a CSQ query which runs an IN subquery to get # orders that includes product with price > 999499.

```
SQL> set echo on
SQL> column name format a20
SQL> REM  IN subquery to get # orders that includes product with price > 999499.
SQL> explain plan for SELECT COUNT(orderid) FROM orders o
WHERE orderid IN (SELECT orderid FROM lineitems l, products p
   WHERE l.productid = p.productid AND o.custid = l.custid AND p.lastprice > 999499);

Explained.

SQL> set echo off

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------------------------
-
Plan hash value: 2403723386

-------------------------------------------------------------------------------------------------
| Id  | Operation     | Name        | Rows  | Bytes | Cost (%CPU)| Time     | Inst   |IN-OUT|
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |                  |     1 |    13 |     5 (100)| 00:00:01 |        |      |
|   1 |  SORT AGGREGATE   |                  |     1 |    13 |            |          |        |      |
|   2 |   VIEW                | VW_SHARD_72AE2D8F |   100 |  1300 |     5 (100)| 00:00:01 |        |      |
|   3 |    SHARD ITERATOR |                  |       |       |            |          |        |      |
|   4 |     REMOTE    |                  |       |       |            | ORA_S~ | R->S |
-------------------------------------------------------------------------------------------------

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------------------------
-

Remote SQL Information (identified by operation id):
----------------------------------------------------

   4 - EXPLAIN PLAN INTO PLAN_TABLE@! FOR SELECT COUNT(*) FROM "ORDERS" "A1" WHERE
       "A1"."ORDERID"=ANY (SELECT "A3"."ORDERID" FROM "LINEITEMS" "A3","PRODUCTS" "A2"
WHERE
```

```
       "A3"."PRODUCTID"="A2"."PRODUCTID" AND "A1"."CUSTID"="A3"."CUSTID" AND
"A2"."LASTPRICE">999499)
       /* coord_sql_id=d5u75rbhjb3vd */  (accessing 'ORA_SHARD_POOL@ORA_MULTI_TARGET' )


20 rows selected.

SQL> REM  IN subquery to get # orders that includes product with price > 999499.
SQL> SELECT COUNT(orderid) FROM orders o
    WHERE orderid IN (SELECT orderid FROM lineitems l, products p
    WHERE l.productid = p.productid AND o.custid = l.custid AND p.lastprice > 999499);

COUNT(ORDERID)
--------------
             5
```

Let's run a CSQ query that calculates customer distribution based on the number of orders placed.

```
SQL> set echo on
SQL> REM Customer Distribution Query
SQL> explain plan for SELECT ordercount, COUNT(*) as custdist
    FROM (SELECT c.custid, COUNT(orderid) ordercount
          FROM customers c LEFT OUTER JOIN orders o
          ON c.custid = o.custid AND
          orderdate BETWEEN sysdate-4 AND sysdate GROUP BY c.custid)
    GROUP BY ordercount
    ORDER BY custdist desc, ordercount desc;

Explained.

SQL> set echo off

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------------------------
-
Plan hash value: 1140215033

-----------------------------------------------------------------------------------------------
| Id  | Operation      | Name        | Rows  | Bytes | Cost (%CPU)| Time     | Inst       |IN-OUT|
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |                  |   100 |  2600 |     5 (100)| 00:00:01 |  |      |
|   1 |  SORT ORDER BY          |                  |   100 |  2600 |     5 (100)| 00:00:01 |  |      |
|   2 |   HASH GROUP BY    |                  |   100 |  2600 |     5 (100)| 00:00:01 |  |      |
|   3 |    VIEW        | VW_SHARD_DB5A5BE0 |   100 |  2600 |     5 (100)| 00:00:01 |  |      |
|   4 |     SHARD ITERATOR |                  |       |       |            |          |  |      |
|   5 |      REMOTE    |                  |       |       |            |          | ORA_S~ | R->S |

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------------------------
-
-----------------------------------------------------------------------------------------------

Remote SQL Information (identified by operation id):
----------------------------------------------------

   5 - EXPLAIN PLAN INTO PLAN_TABLE@! FOR SELECT COUNT(*),"A1"."ORDERCOUNT" FROM  (SELECT
       "A3"."CUSTID" "CUSTID",COUNT("A2"."ORDERID") "ORDERCOUNT" FROM "CUSTOMERS"
"A3","ORDERS" "A2"
       WHERE "A3"."CUSTID"="A2"."CUSTID"(+) AND "A2"."ORDERDATE"(+)>=CAST(SYSDATE@!-4 AS
TIMESTAMP)
       AND "A2"."ORDERDATE"(+)<=CAST(SYSDATE@! AS TIMESTAMP) GROUP BY "A3"."CUSTID") "A1"
GROUP BY
       "A1"."ORDERCOUNT" /* coord_sql_id=45wf75sj0dpu7 */  (accessing
       'ORA_SHARD_POOL@ORA_MULTI_TARGET' )

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------
```

```
23 rows selected.

SQL> REM Customer Distribution Query
SQL> SELECT ordercount, COUNT(*) as custdist
    FROM (SELECT c.custid, COUNT(orderid) ordercount
          FROM customers c LEFT OUTER JOIN orders o
          ON c.custid = o.custid AND
          orderdate BETWEEN sysdate-4 AND sysdate GROUP BY c.custid)
    GROUP BY ordercount
    ORDER BY custdist desc, ordercount desc;

ORDERCOUNT   CUSTDIST
---------- ----------
         1        180
         2         59
         3         19
         4          3
         0          3
```

Lab7: Elastic Scaling

In this lab, we will add the shards (on shard5 and shard6) to the Shard Database and thus elastically scale the SDB. We will also observe that chunks are automatically rebalanced after the new shards are added.

29. **Incremental Deployment of a Shards**

Following the same procedure as in Lab 2, create two new databases. One should have a db_unique_name and SID of 'sh5' on shard5 and one should be 'sh6' on shard6. Run validateShard() as in Lab 2 to confirm the configuration of the databases prior to adding them to the sharding configuration. Both databases should have an associated TNS Listener on port 1521.

On SHARDDIRECTOR1 Terminal:
Launch gdsctl

```
$ cd /home/oracle
$ . ./shard-director1.sh
$ env |grep ORA
ORACLE_BASE=/u01/app/oracle
ORACLE_HOME=/u01/app/oracle/product/12.2.0/gsmhome_1

$ cd /u01/stage/labs
```

Launch gdsctl

```
$ gdsctl

GDSCTL> set gsm -gsm sharddirector1
GDSCTL> connect mygdsadmin/passwd_mygdsadmin

GDSCTL> config shard
Name                Shard Group         Status    State       Region    Availability
----                -----------         ------    -----       ------    ------------
sh1                 shgrp1              Ok        Deployed    region1   ONLINE
sh2                 shgrp2              Ok        Deployed    region2   READ_ONLY
sh3                 shgrp1              Ok        Deployed    region1   ONLINE
sh4                 shgrp2              Ok        Deployed    region2   READ_ONLY
```

Specify the shardgroup, destination and the credentials for each shard (shard5 and shard6). In this lab we are using the default templates for NETCA and DBCA

```
GDSCTL> add invitednode shard5

GDSCTL> add shard -shardgroup shgrp1 -connect shard5:1521/sh5 -pwd <GSMUSER_password>

GDSCTL> add invitednode shard6

GDSCTL> add shard -shardgroup shgrp2 -connect shard6:1521/sh6 -pwd <GSMUSER_password>
```

Note: Make sure the demo app is running.

30. Run the deploy command to create the shards and the replicas :

```
GDSCTL>config shard
Name                Shard Group         Status    State       Region    Availability
----                -----------         ------    -----       ------    ------------
sh5                 shgrp1              U         none        region1   -
sh6                 shgrp2              U         none        region2   -
sh1                 shgrp1              Ok        Deployed    region1   ONLINE
sh2                 shgrp2              Ok        Deployed    region2   READ_ONLY
sh3                 shgrp1              Ok        Deployed    region1   ONLINE
sh4                 shgrp2              Ok        Deployed    region2   READ_ONLY

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


GDSCTL> config vncr

Name                            Group ID
----                            --------
shard1
shard2
shard3
shard4
shard5
shard6
10.xxx.yy.zz1
10.xxx.yy.zz2
10.xxx.yy.zz3
10.xxx.yy.zz4
10.xxx.yy.zz5
10.xxx.yy.zz6
```

Run the deploy command to create the new shards and their replicas. This automatically rebalances the chunks:

```
GDSCTL> deploy

$ gdsctl config shard
Name                Shard Group         Status    State       Region    Availability
----                -----------         ------    -----       ------    ------------
sh1                 shgrp1              Ok        Deployed    region1   ONLINE
sh2                 shgrp2              Ok        Deployed    region2   READ_ONLY
sh3                 shgrp1              Ok        Deployed    region1   ONLINE
sh4                 shgrp2              Ok        Deployed    region2   READ_ONLY
sh5                 shgrp1              Ok        Deployed    region1   ONLINE
sh6                 shgrp2              Ok        Deployed    region2   READ_ONLY
```

Verify that shard5.sh (on shard5) and shard6.sh (on shard6) reflect the correct names of the shards that have been created specific to your deployment. If not, update the scripts to reflect the names of the shards accordingly.

Run the following command 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
-----     ------                       ------                         -
-----
```

Observe that the chunks are automatically rebalanced upon the addition of new shards.

```
$ 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
```

Observe that the “databases” are automatically registered.

```
$ 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.
```

Observe that the “services” are automatically brought up on the newly added shards.

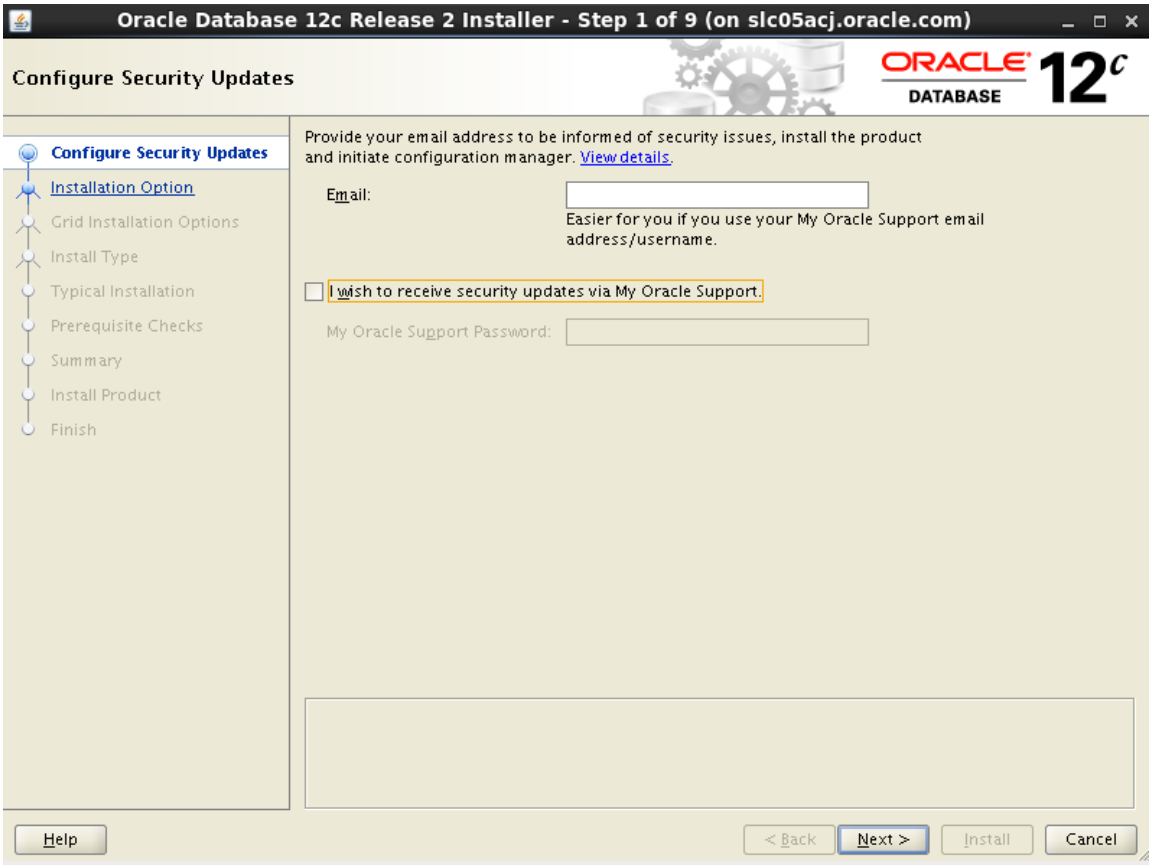
Conclusion

In this lab, you have deployed a sharded database using system managed sharding method. As part of the test cases you have:

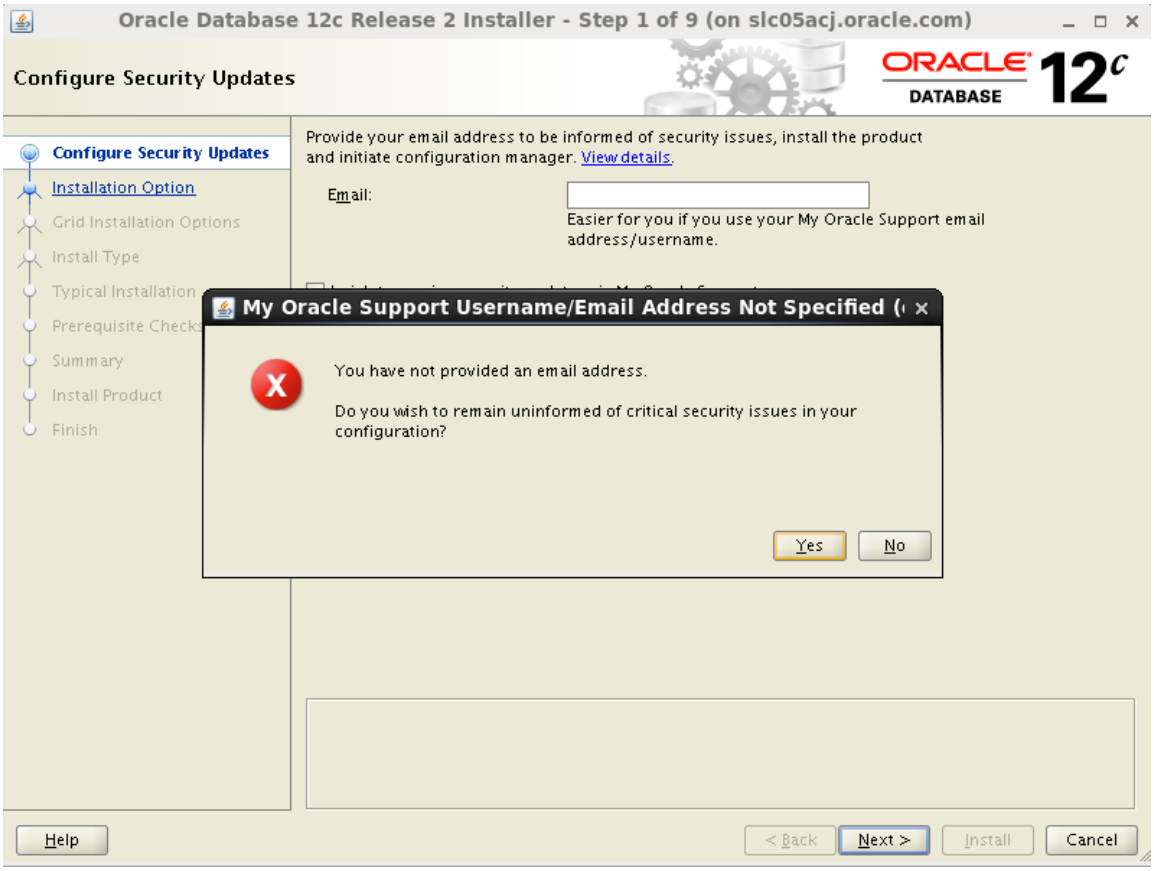
- Created a shard catalog and shard directors for system managed sharding
- Specified the metadata and deployed the sharded database
- Created table family using system managed sharding method
- Specified sharding_Key for session based routing (using SQL*Plus)
- Observed uniform data distribution by executing Read Write and Read Only workloads on the sharded database using a demo application (using UCP)
- Executed cross-shard queries
- Elastically scaled the sharded database

Appendix A - Installation of Oracle 12.2.0.1 database software

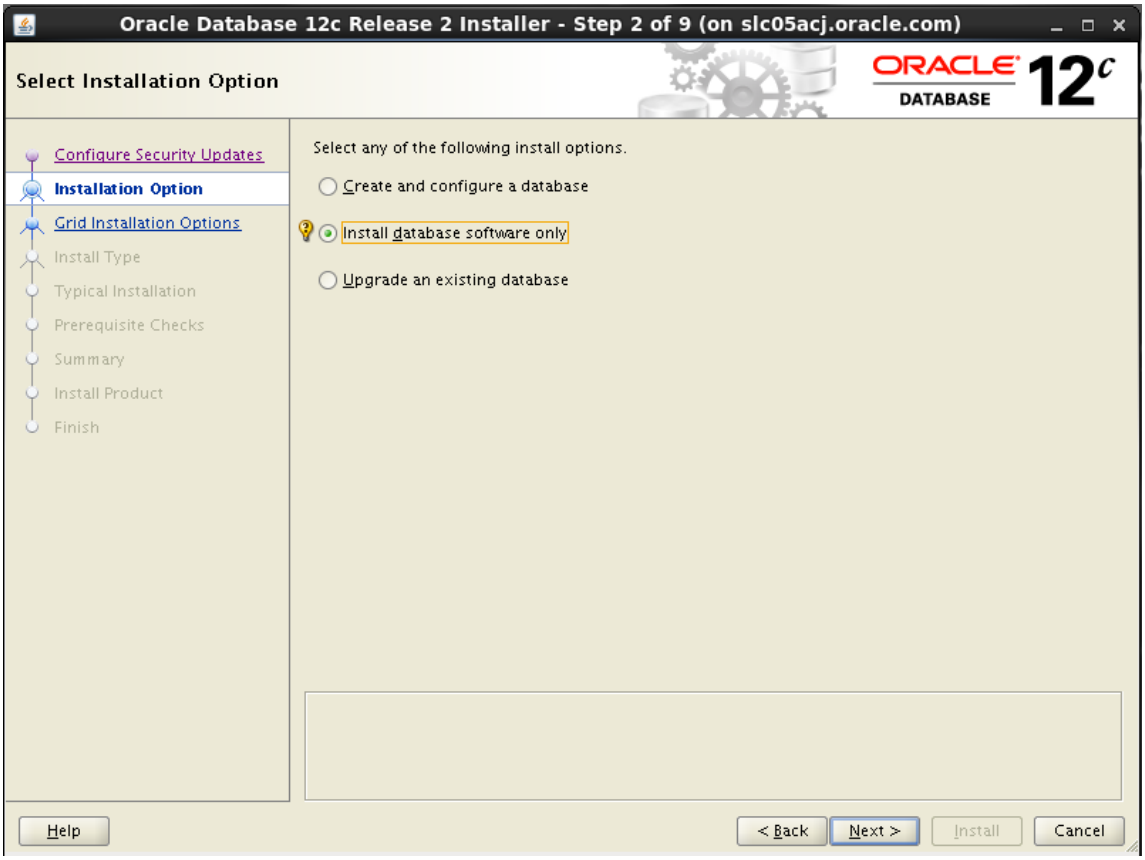
```
$ cd /u01/stage/database
$ ./runInstaller
```



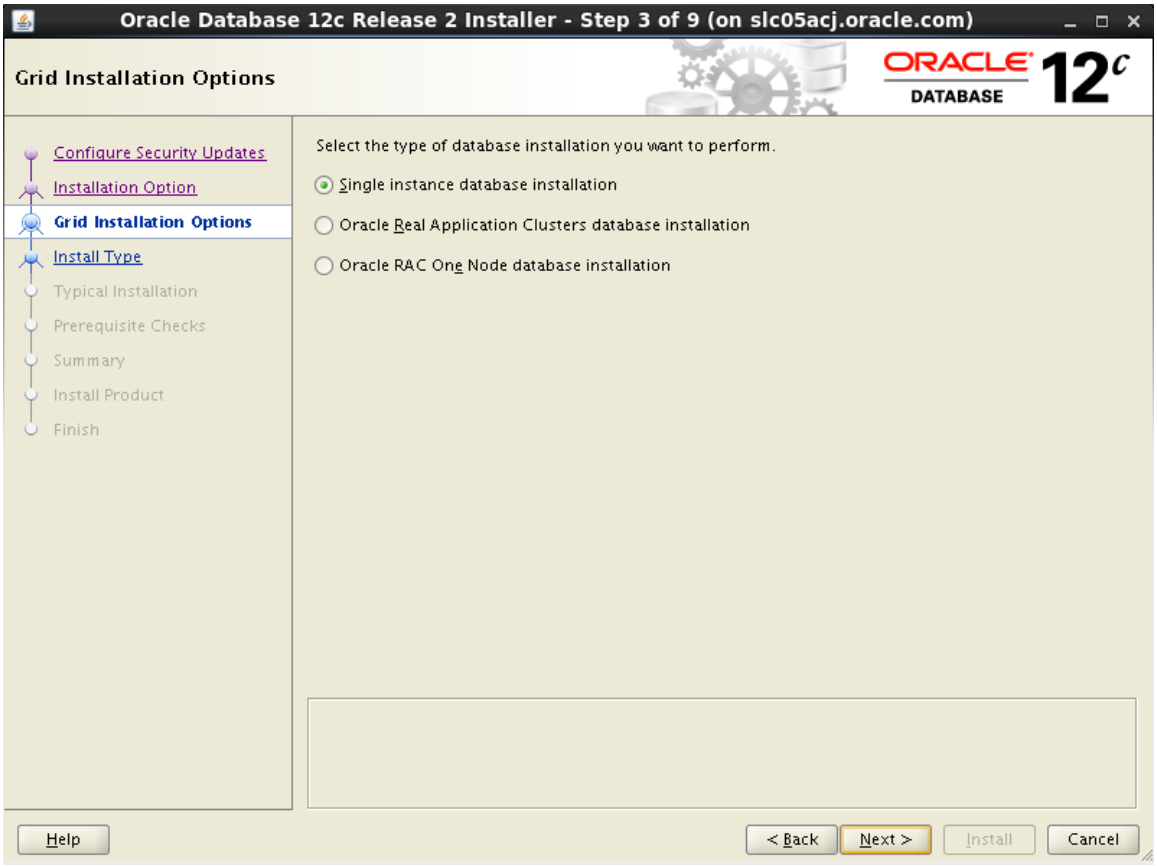
Uncheck the box “I wish to receive security updates via My Oracle Support”.



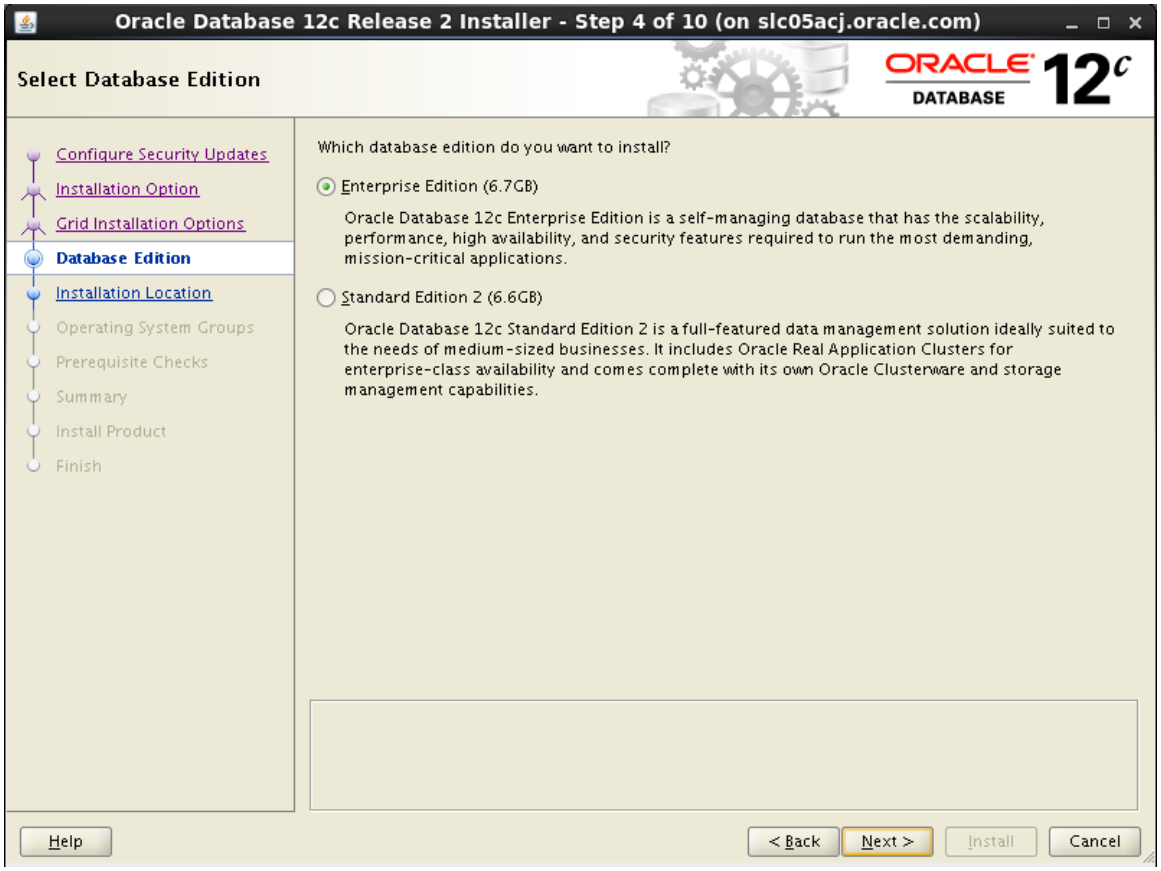
Click on “Yes” and hit “Next”.



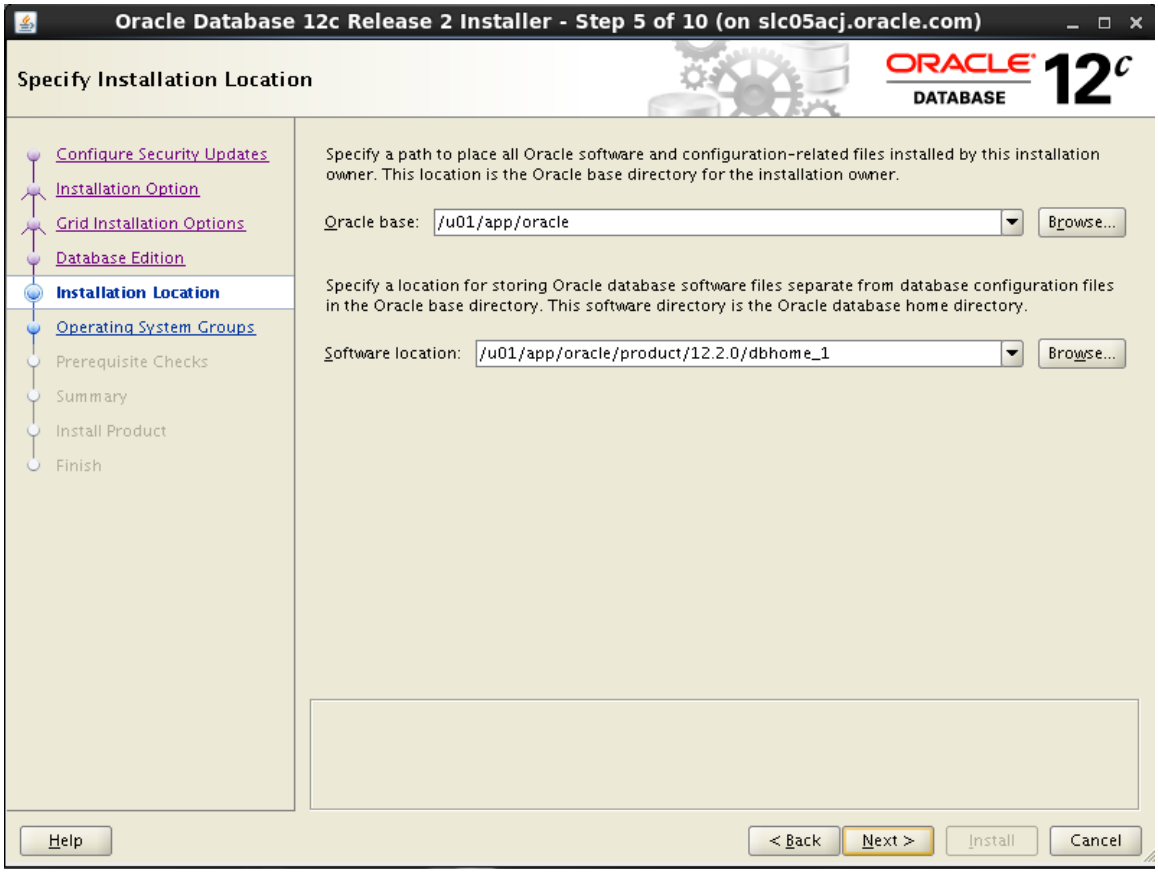
Select “Install database software only” and hit “Next”.



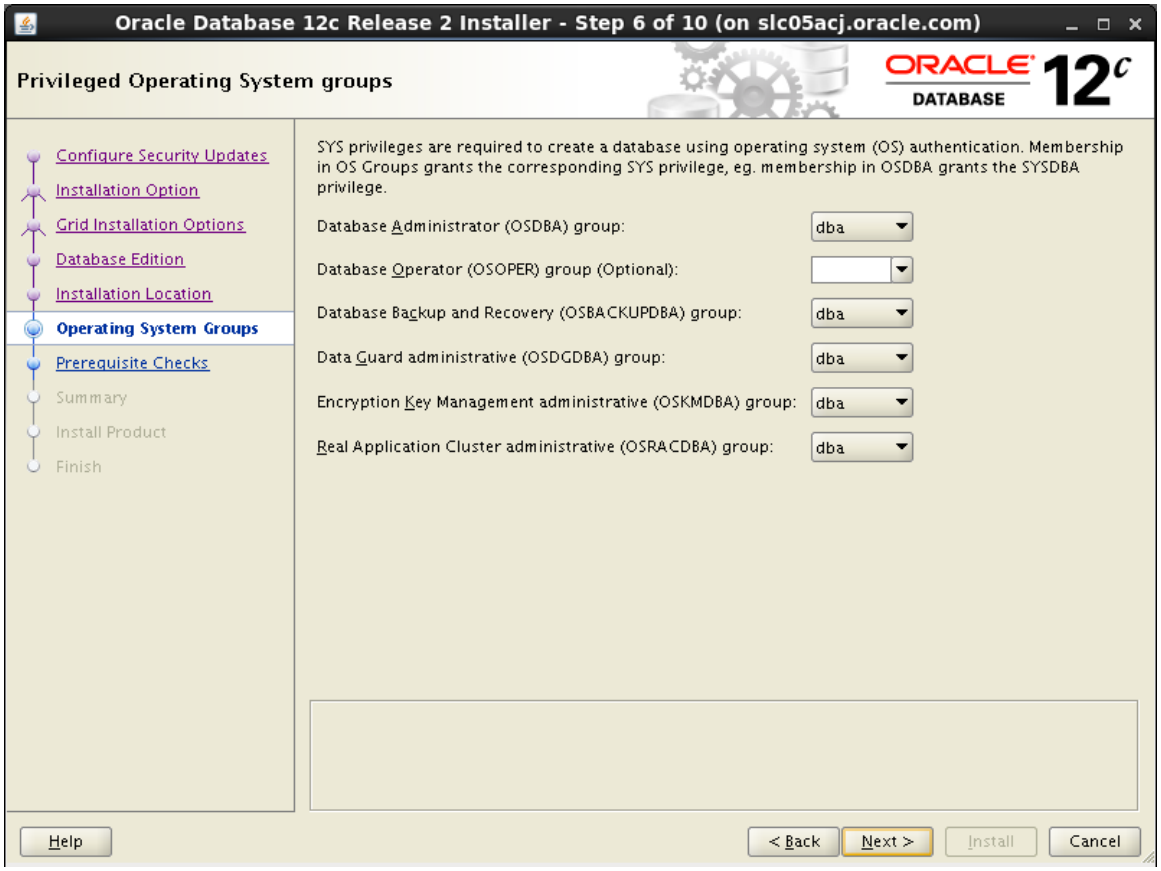
Select “Single instance database installation” and hit “Next”.



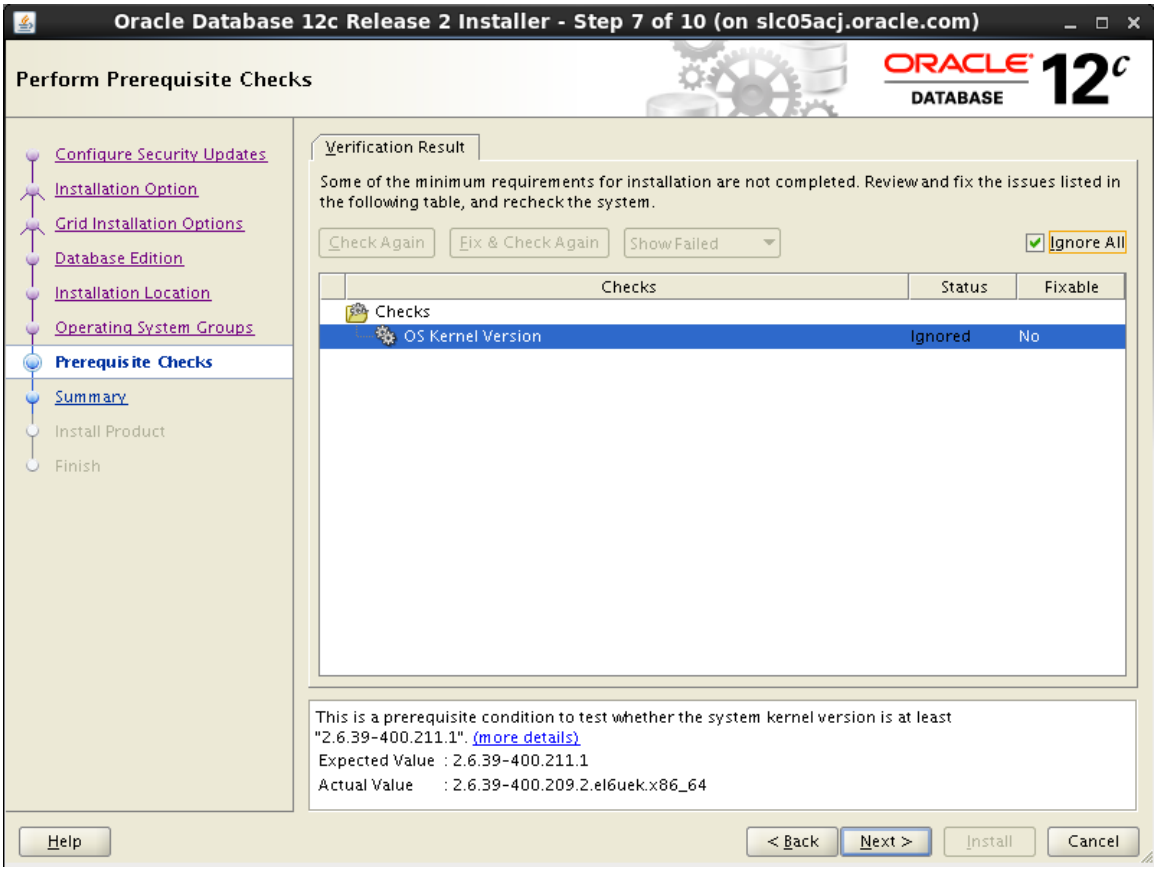
Select “Enterprise Edition” and hit “Next”.



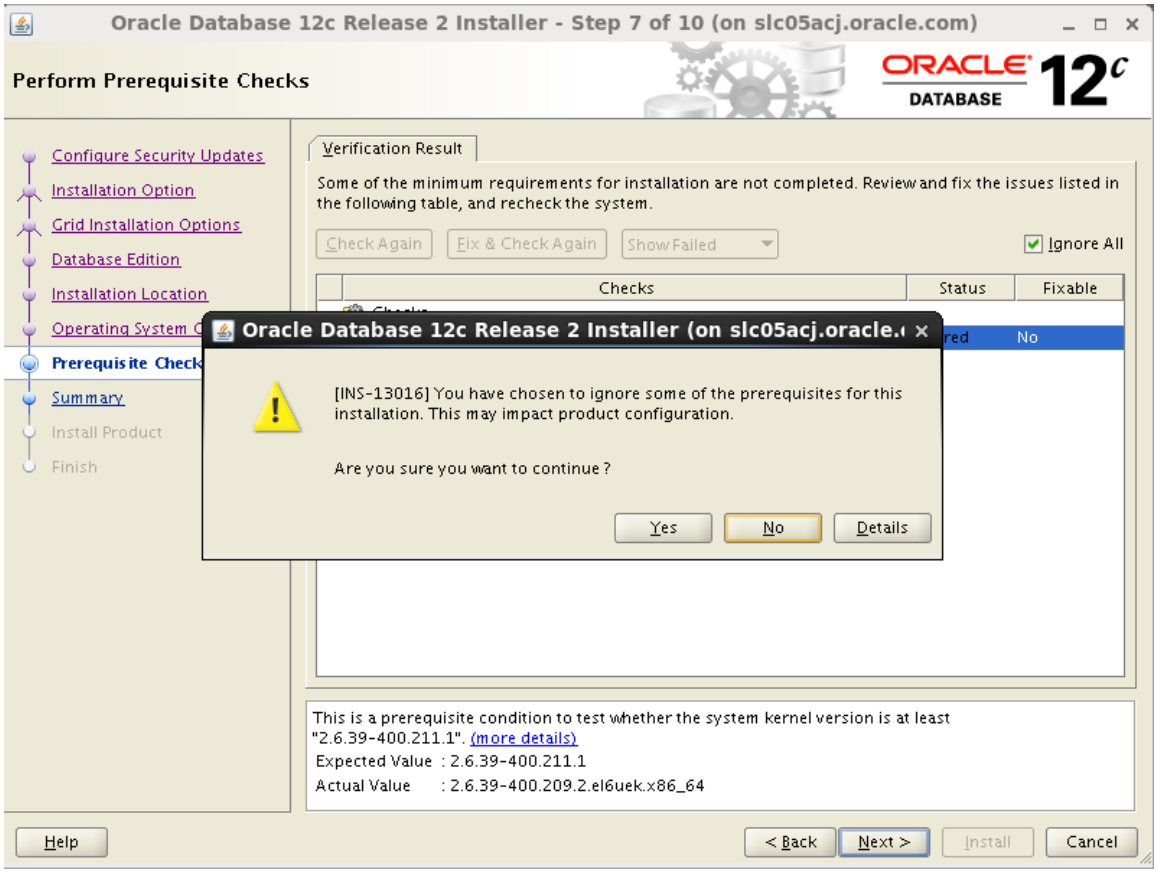
Set the “Oracle base” and “software location” for the database binaries and hit “Next”. The location of the software (ORACLE_HOME) is important as that directory is used in numerous locations within this document.


Oracle Database 12c Release 2 Installer - Step 6 of 10 (on slc05acj.oracle.com)
Privileged Operating System groups
ORACLE DATABASE 12c
Configure Security Updates
Installation Option
Grid Installation Options
Database Edition
Installation Location
Operating System Groups
Prerequisite Checks
Summary
Install Product
Finish
SYS privileges are required to create a database using operating system (OS) authentication. Membership in OS Groups grants the corresponding SYS privilege, eg. membership in OSDBA grants the SYSDBA privilege.
Database Administrator (OSDBA) group: dba
Database Operator (OSOPER) group (Optional):
Database Backup and Recovery (OSBACKUPDBA) group: dba
Data Guard administrative (OSDGDBA) group: dba
Encryption Key Management administrative (OSKMDBA) group: dba
Real Application Cluster administrative (OSRACDBA) group: dba
Help
< Back
Next >
Install
Cancel

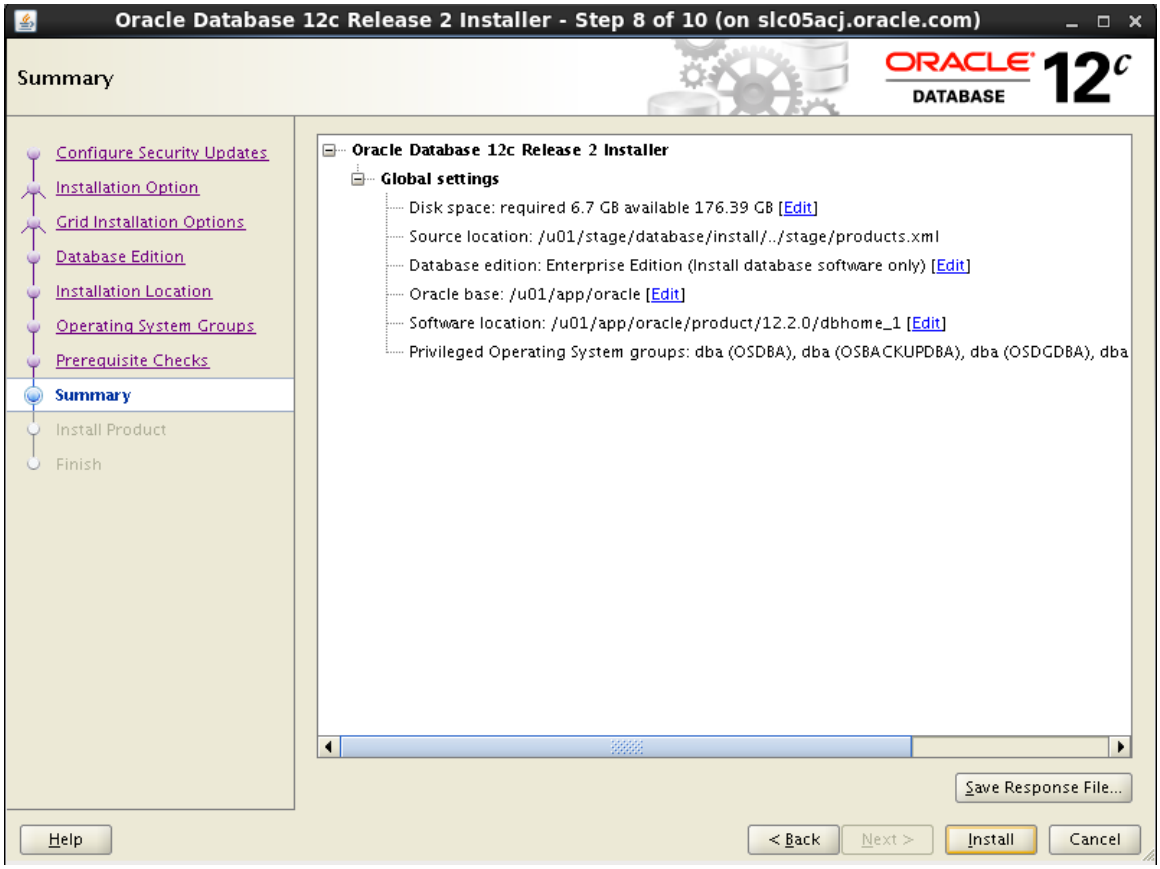
Hit “Next”.



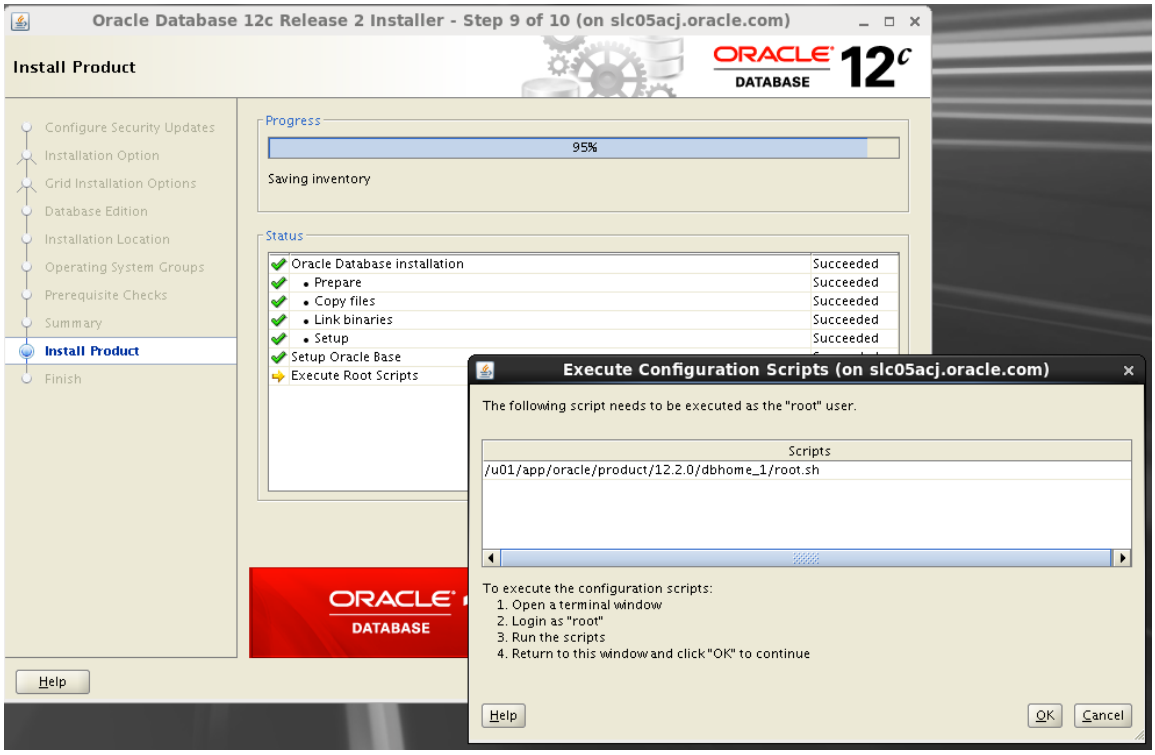
Check the “Ignore All” for the OS Kernel Version and hit “Next”.



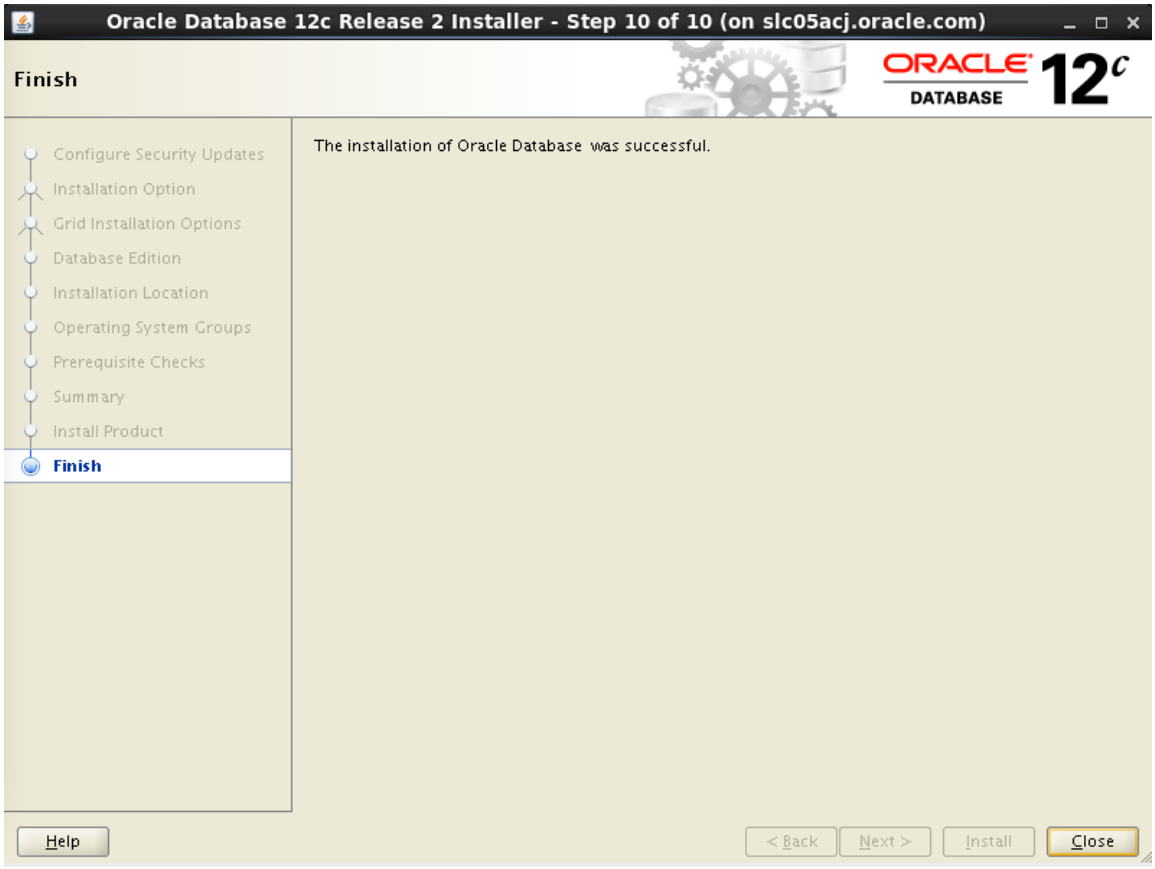
Select “Yes” and hit “Next”.



Click on “Install”.



Execute the root.sh as root in a separate terminal and click “OK”



Click on “Close” to complete the installation.

Appendix B - Creation of database for shard catalog

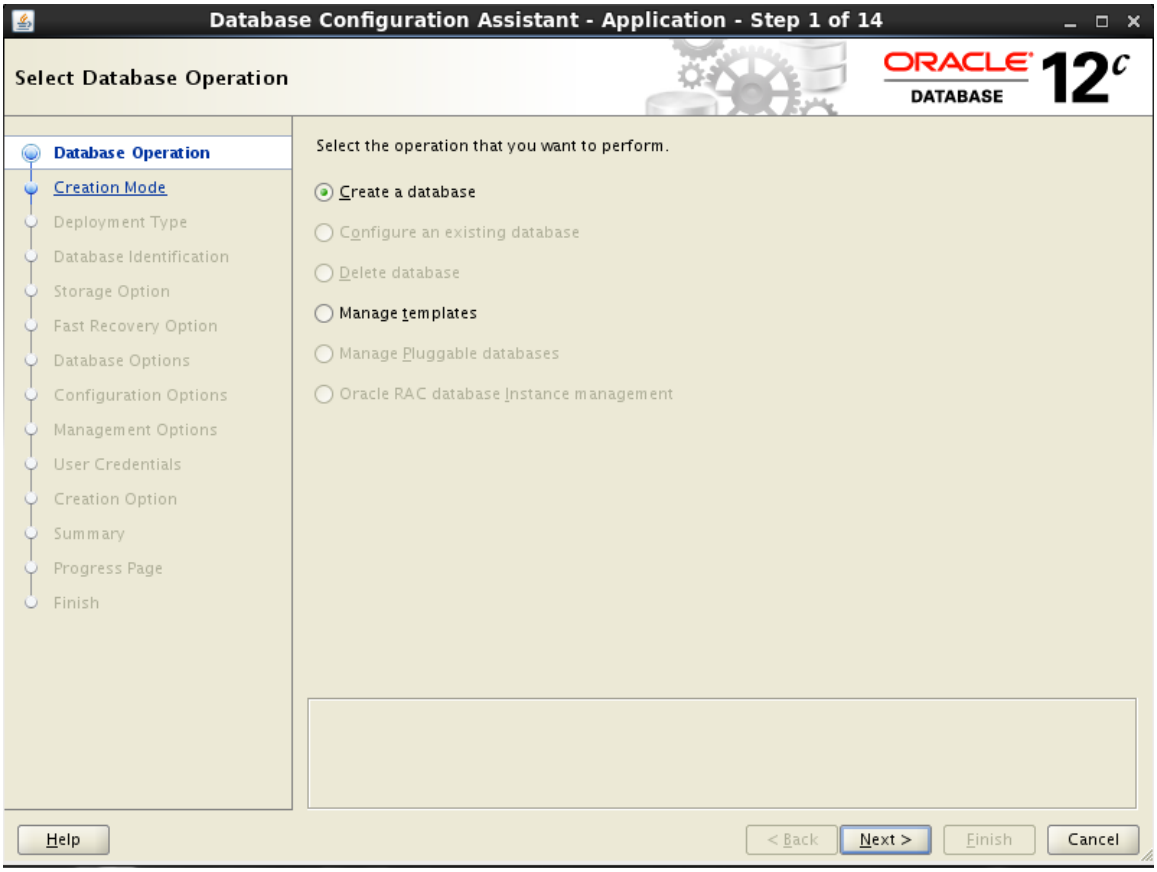
Note: Currently, only the Non-CDB databases are supported for shardcatalog and shards.

On Shard0:

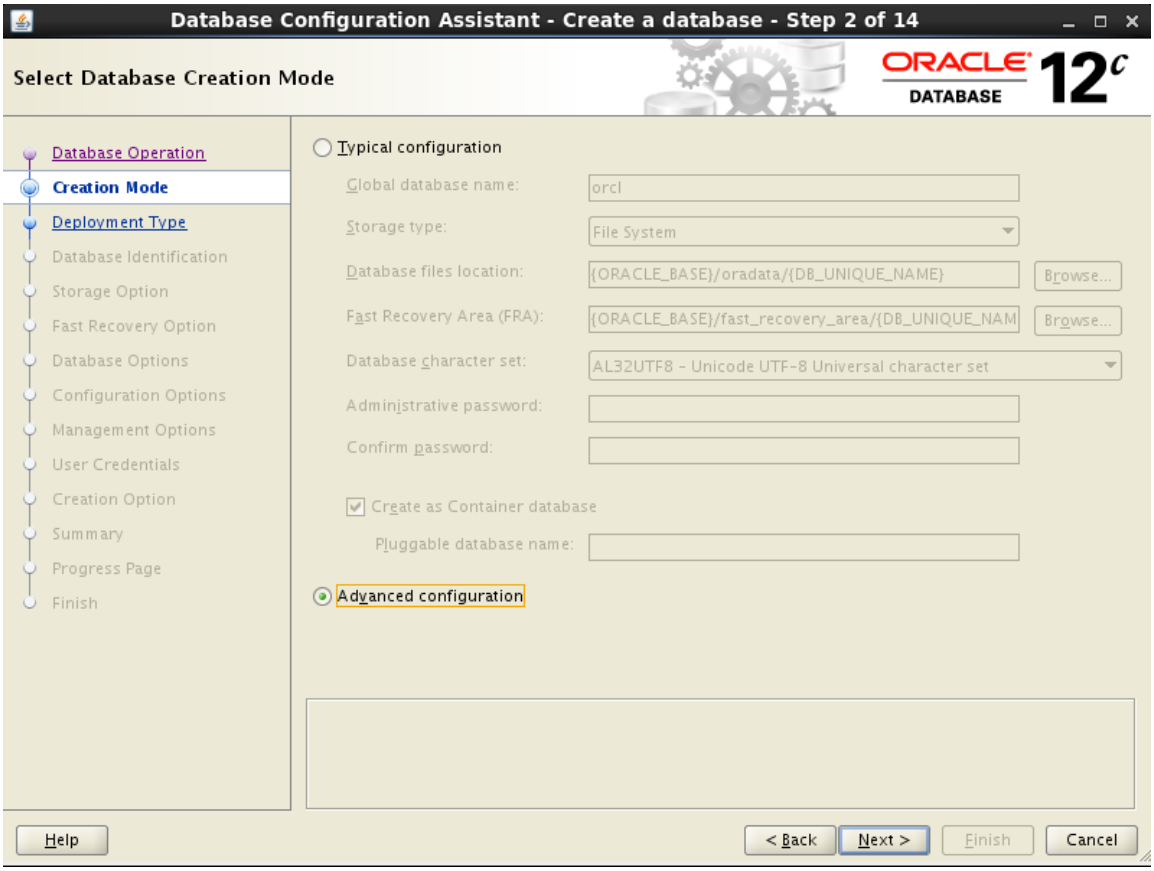
```
$ . ./shardcat.sh
$ env |grep ORA
ORACLE_SID=shardcat
ORACLE_BASE=/u01/app/oracle
ORACLE_HOME=/u01/app/oracle/product/12.2.0/dbhome_1
```

Run the Database Configuration Assistant (dbca) to create the database for shard catalog.

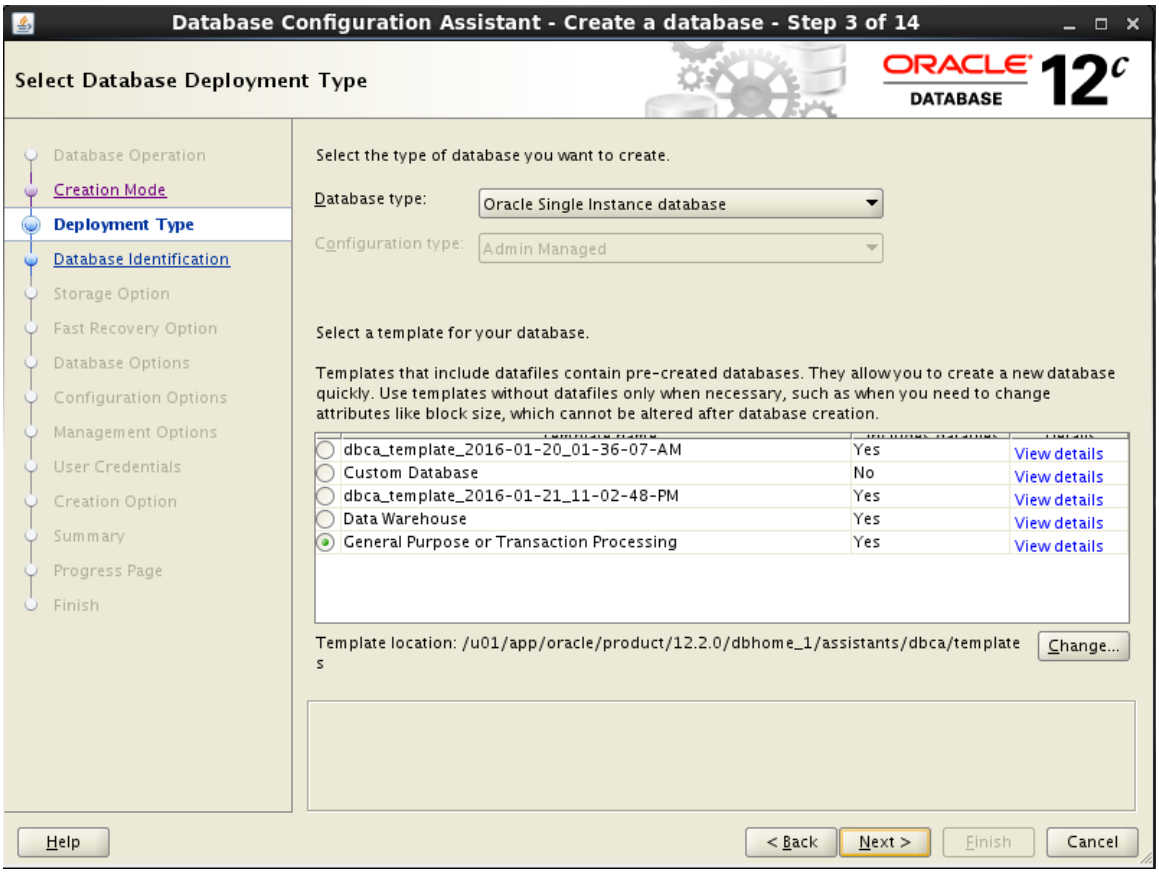
```
$ cd /u01/stage/database
$ dbca
```



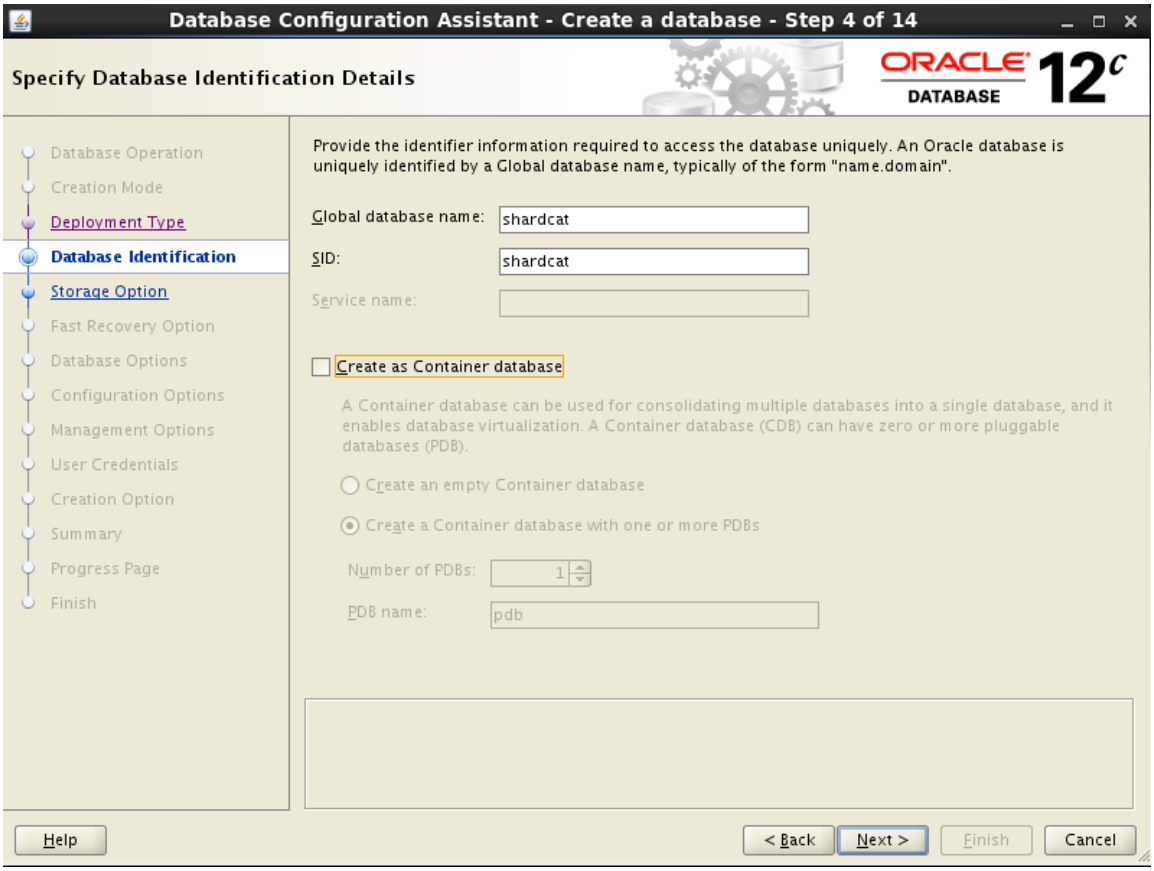
Select “Create a database” and hit “Next”.



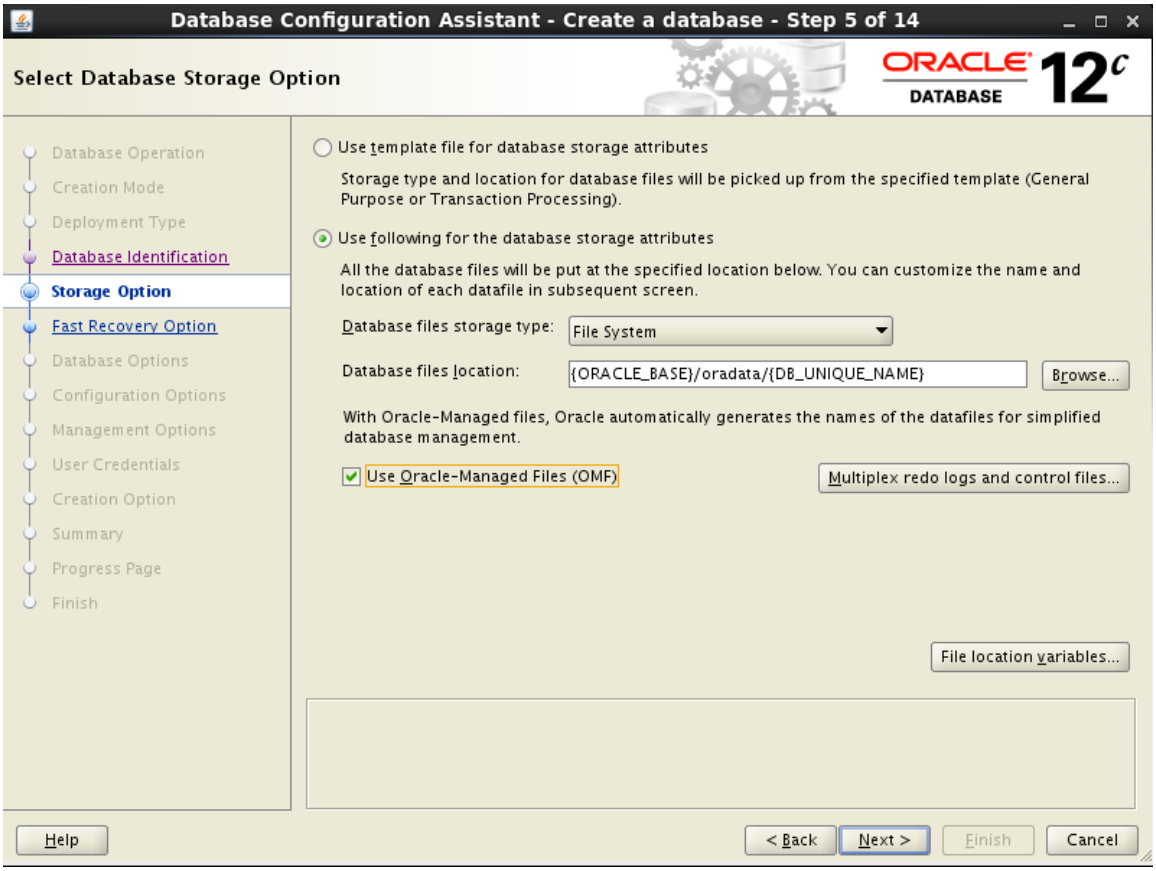
Select “Advanced configuration” and hit “Next”.



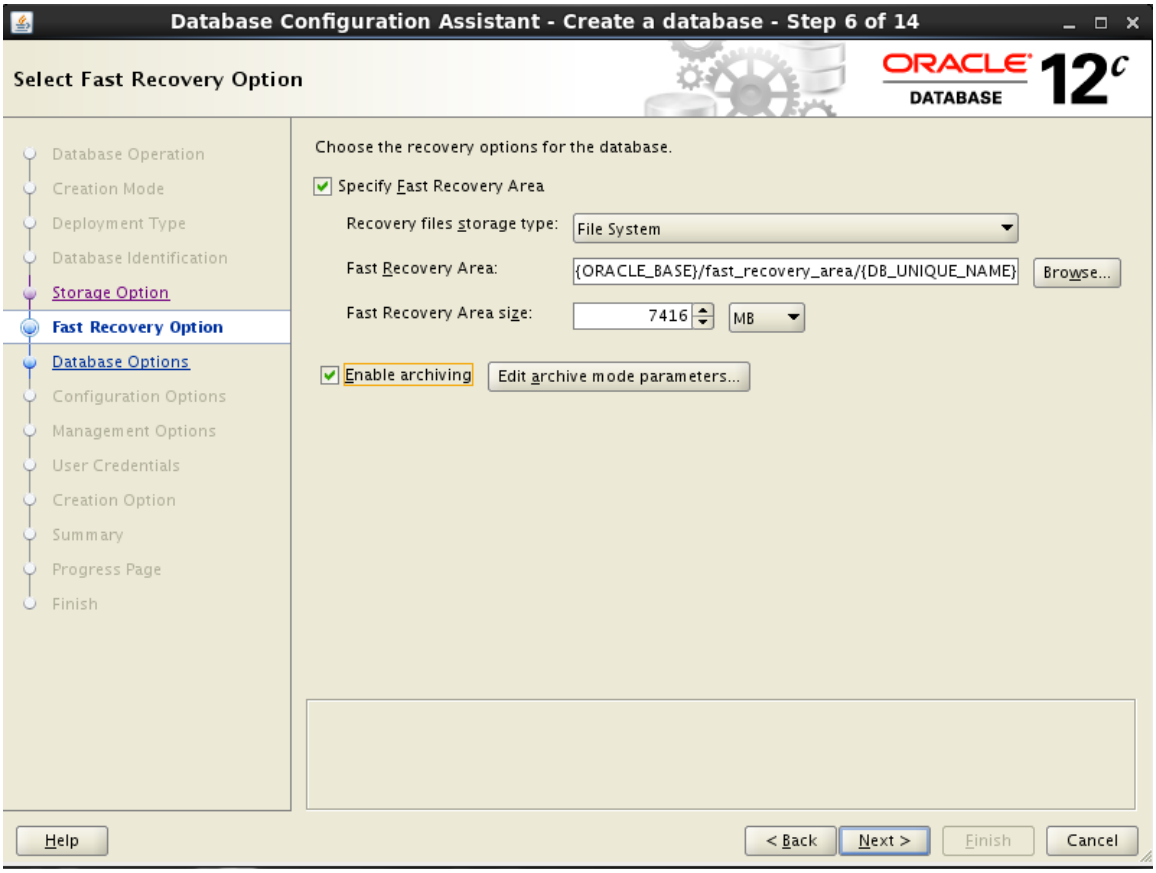
For the Database type, select “Oracle Single Instance database” and for the template for the database, select “General Purpose or Transaction Processing” and hit “Next”.



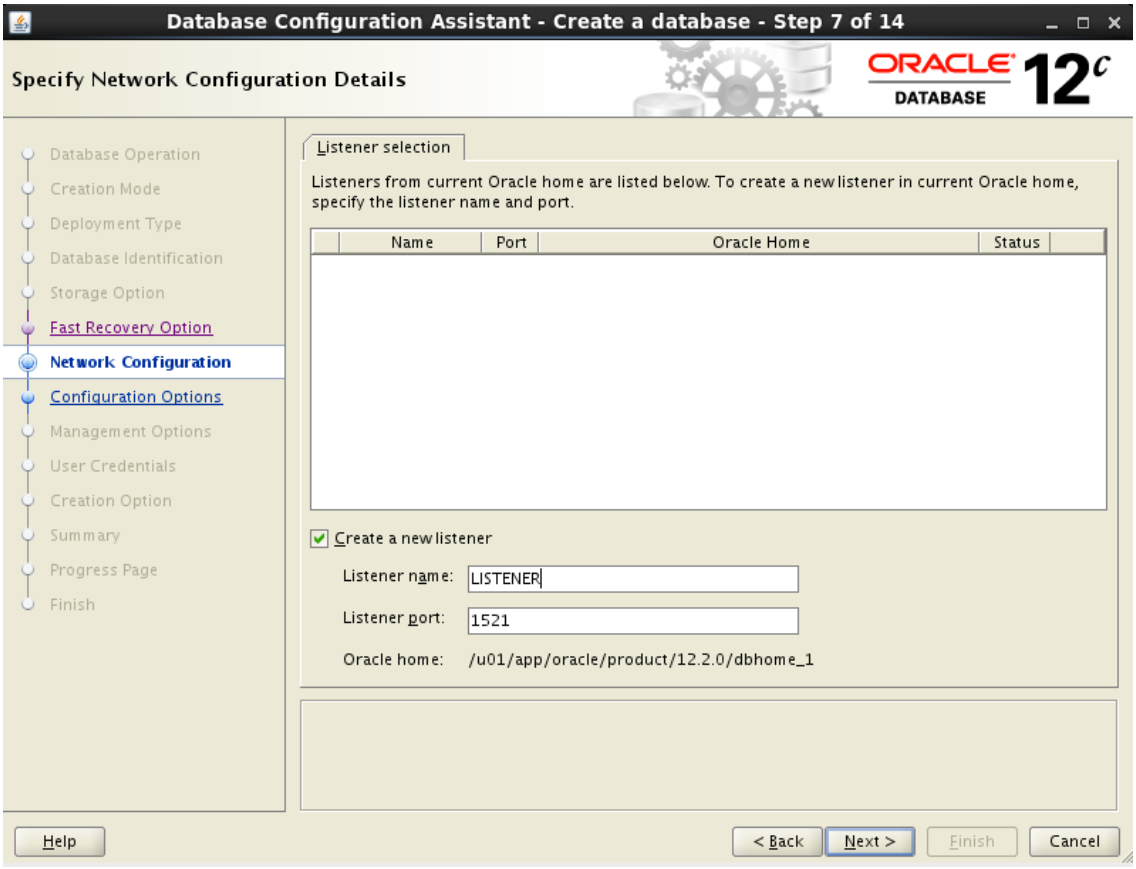
Set the Global database name and SID to “shardcat”
Uncheck the “Create as Container database" and hit “Next”.



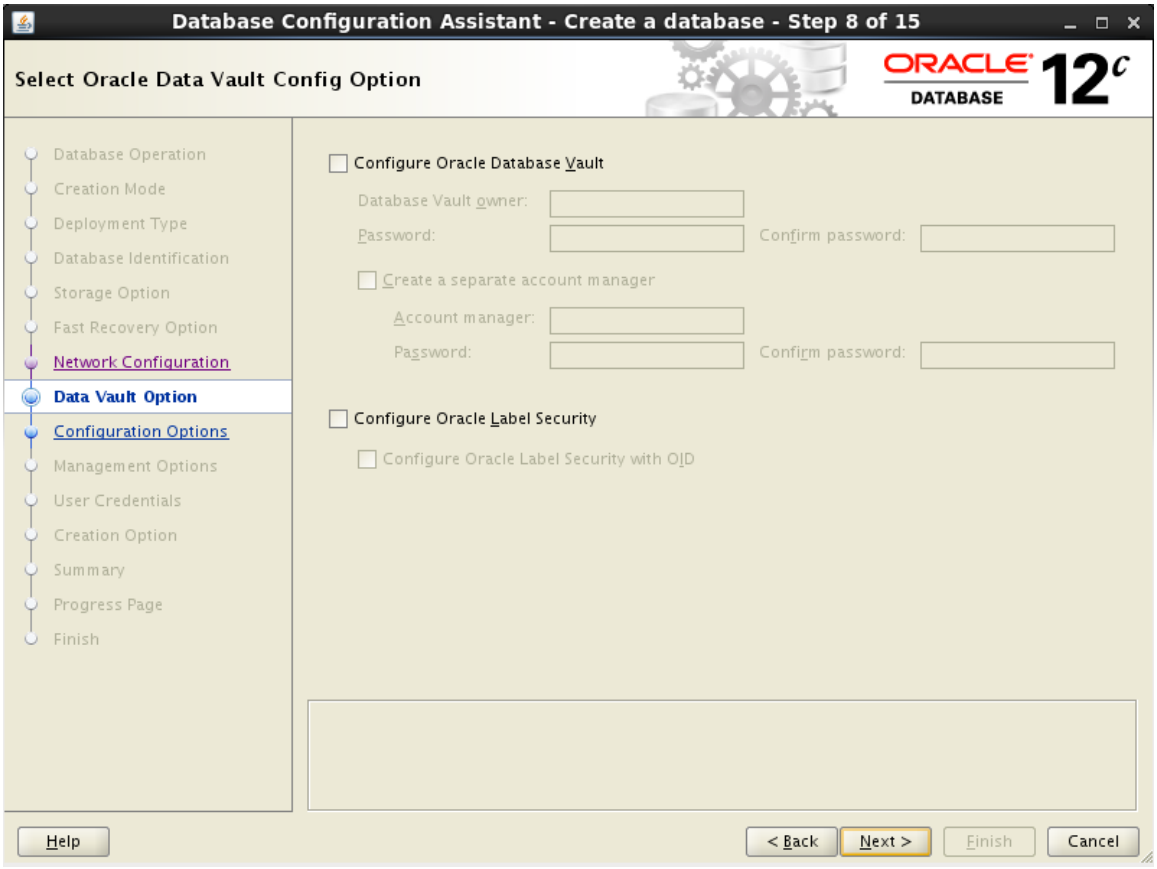
Select “Use following for the database storage attributes”.
Select “Use Oracle-Managed Files (OMF)” and hit “Next”.



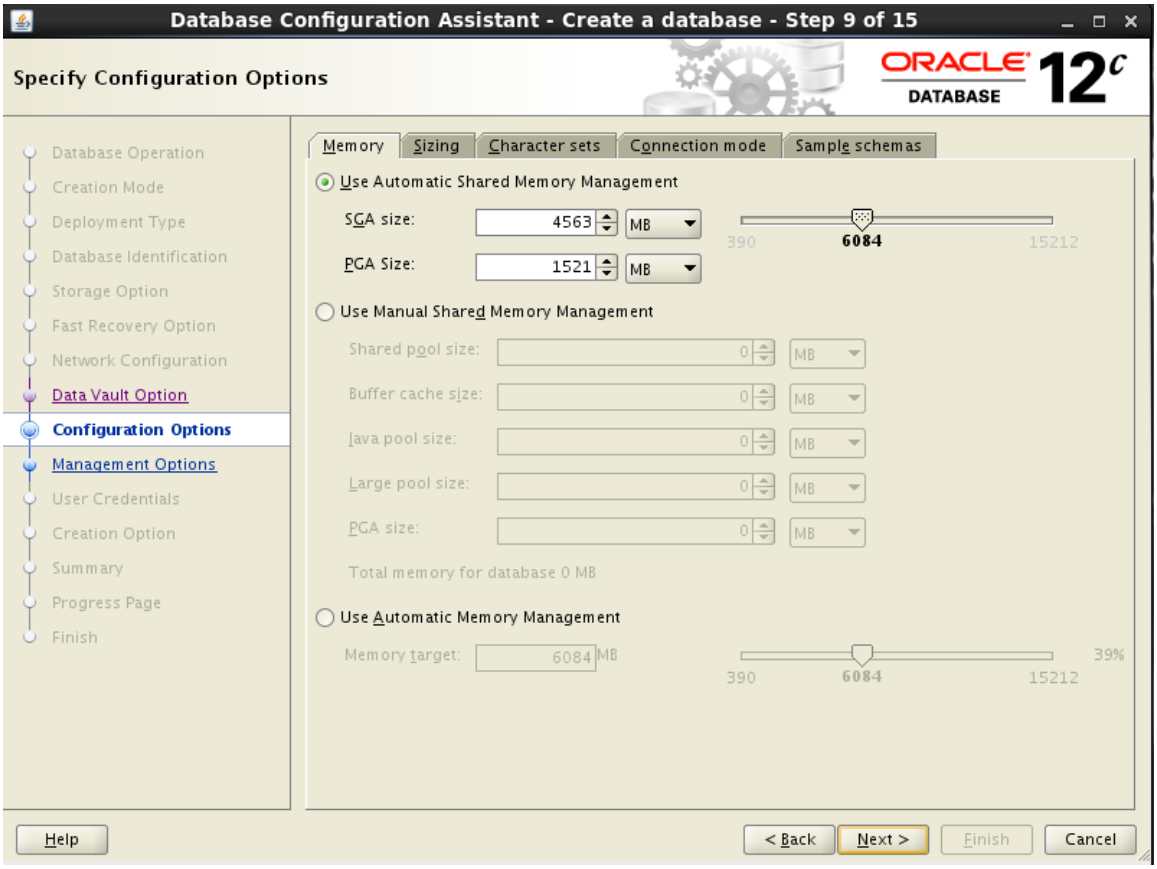
Select “Specify Fast Recovery Area” and “Enable archiving” and hit “Next”.



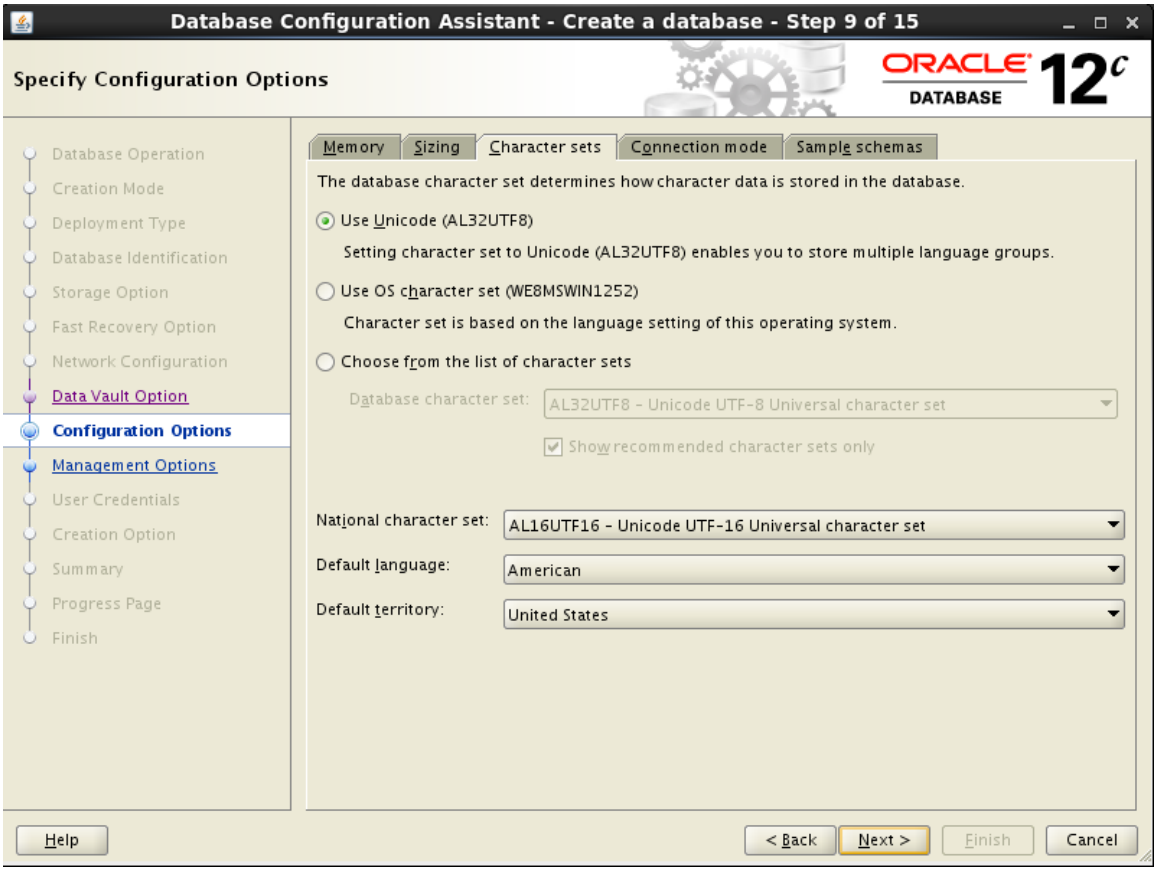
Select “Create a new listener” and set the “Listener name” and “Listener port” as shown above.


Database Configuration Assistant - Create a database - Step 8 of 15
Select Oracle Data Vault Config Option
ORACLE DATABASE 12c
Database Operation
Creation Mode
Deployment Type
Database Identification
Storage Option
Fast Recovery Option
Network Configuration
Data Vault Option
Configuration Options
Management Options
User Credentials
Creation Option
Summary
Progress Page
Finish
Configure Oracle Database Vault
Database Vault owner:
Password:
Confirm password:
Create a separate account manager
Account manager:
Password:
Confirm password:
Configure Oracle Label Security
Configure Oracle Label Security with OID
Help
< Back
Next >
Finish
Cancel

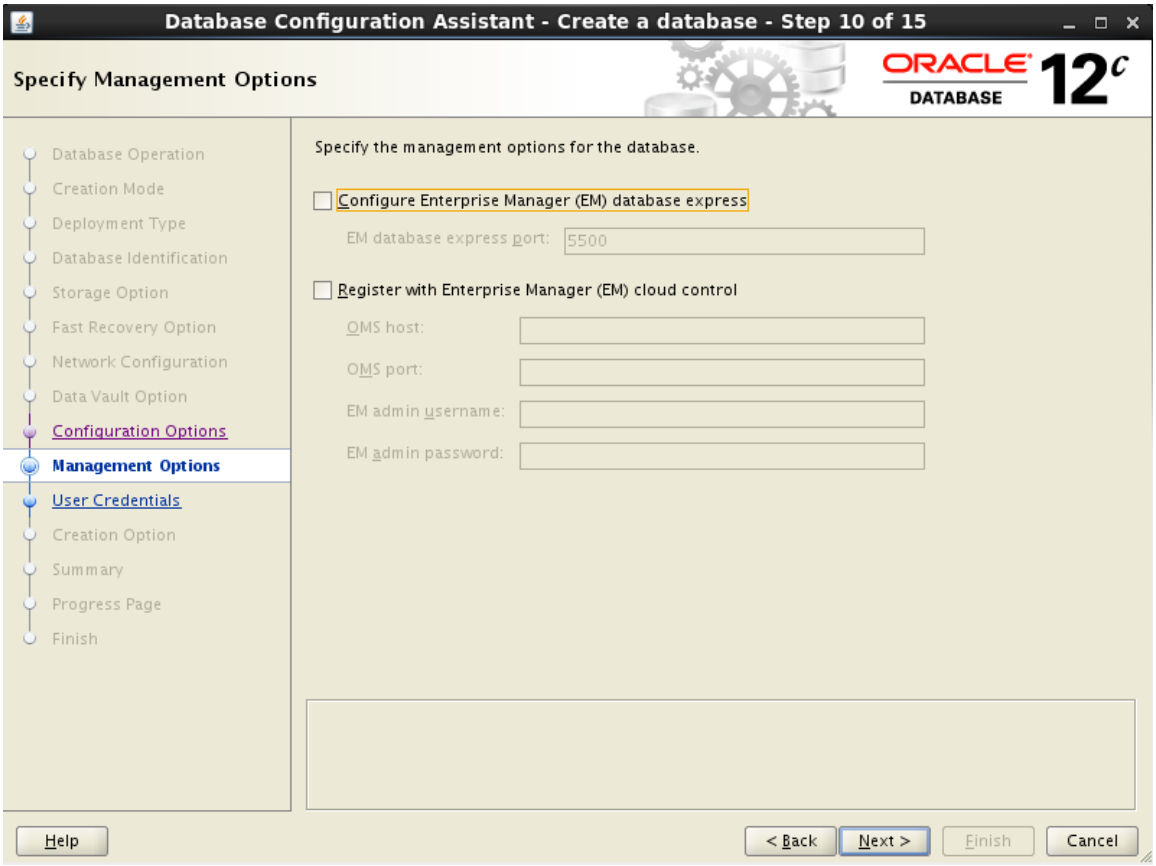
Hit “Next”.



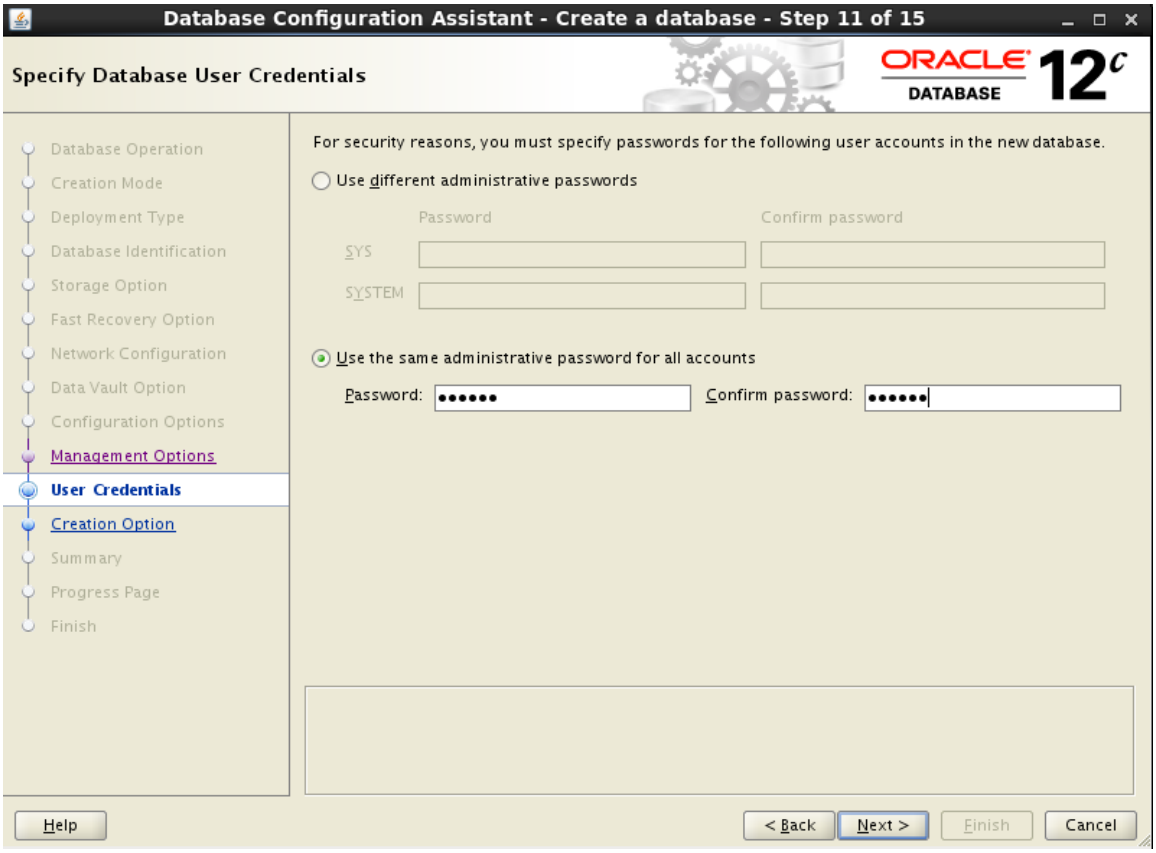
Select “Use Automatic Shared Memory Management” and allocate SGA and PGA Sizes as shown above . Hit “Next”.



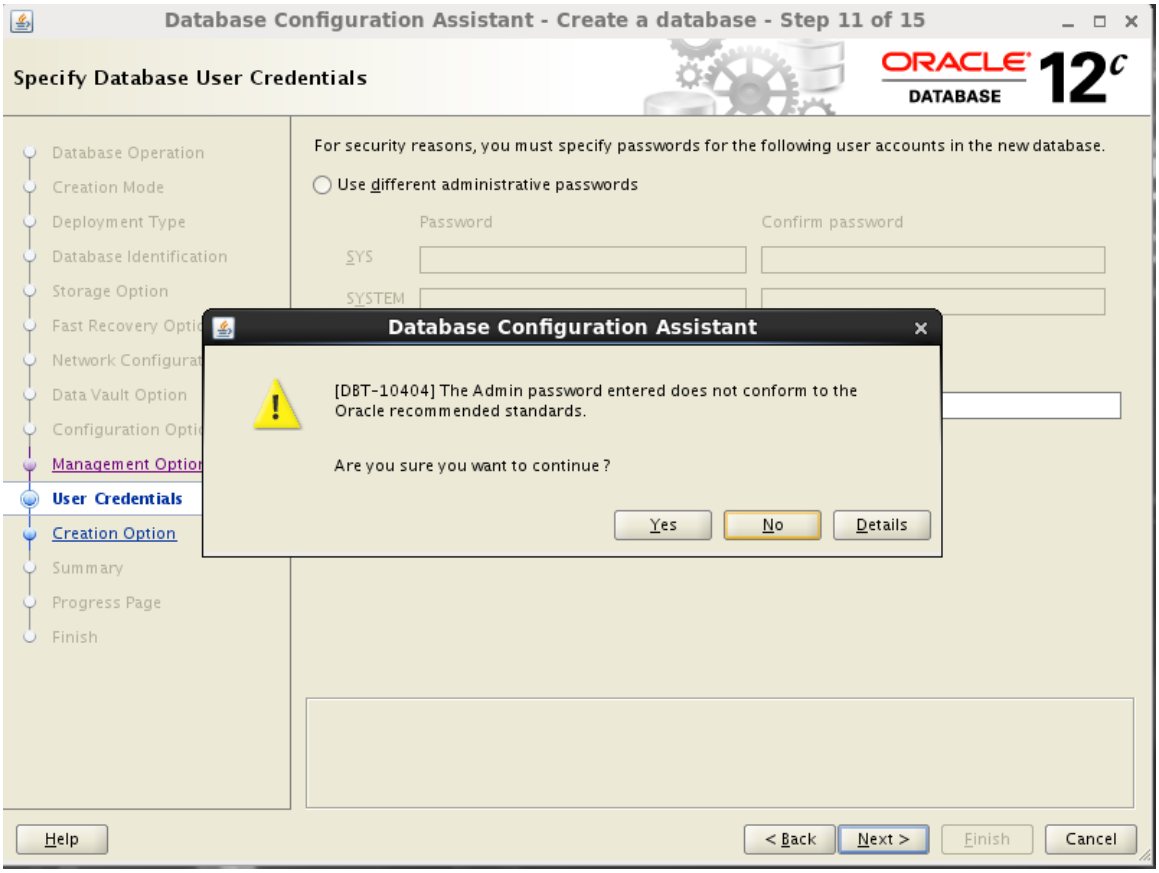
For the database character set, select “Use Unicode (AL32UTF8)” and hit “Next”



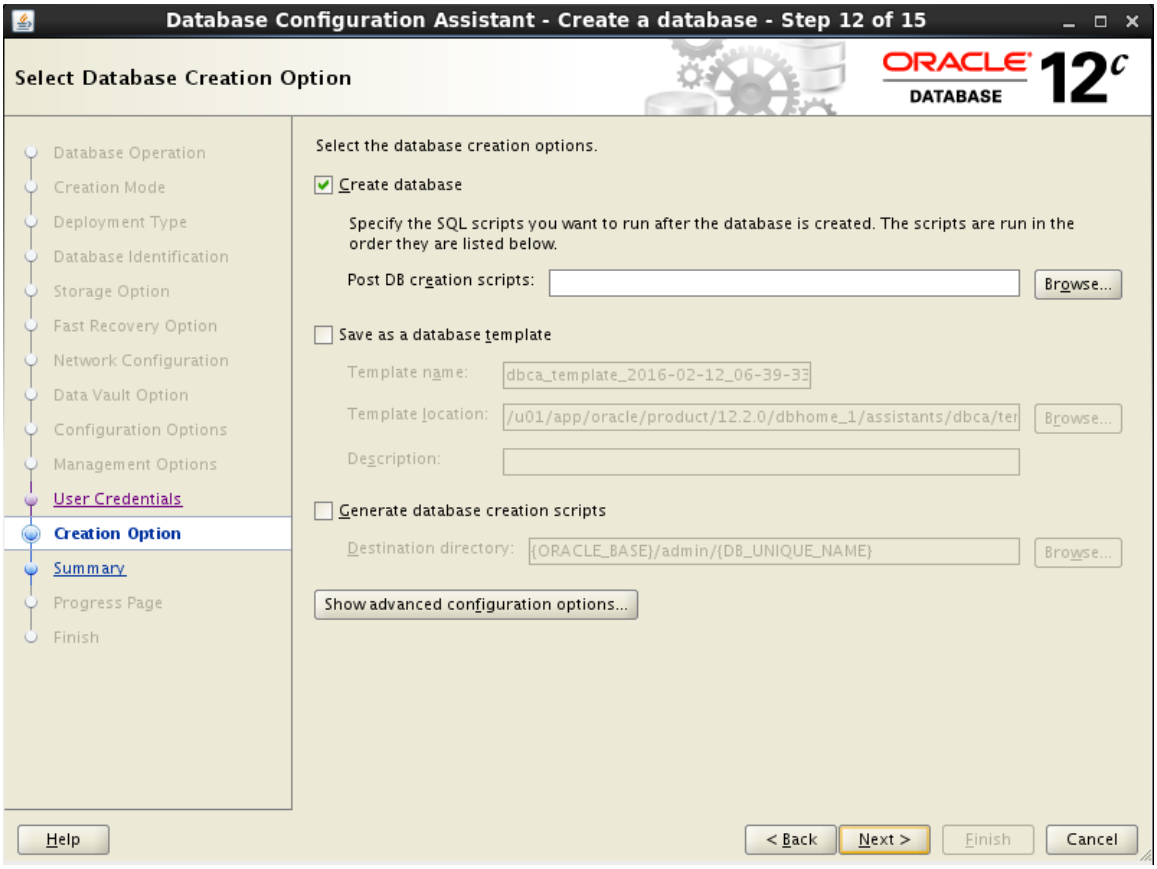
Uncheck the “Configure EM database express” and hit “Next”.



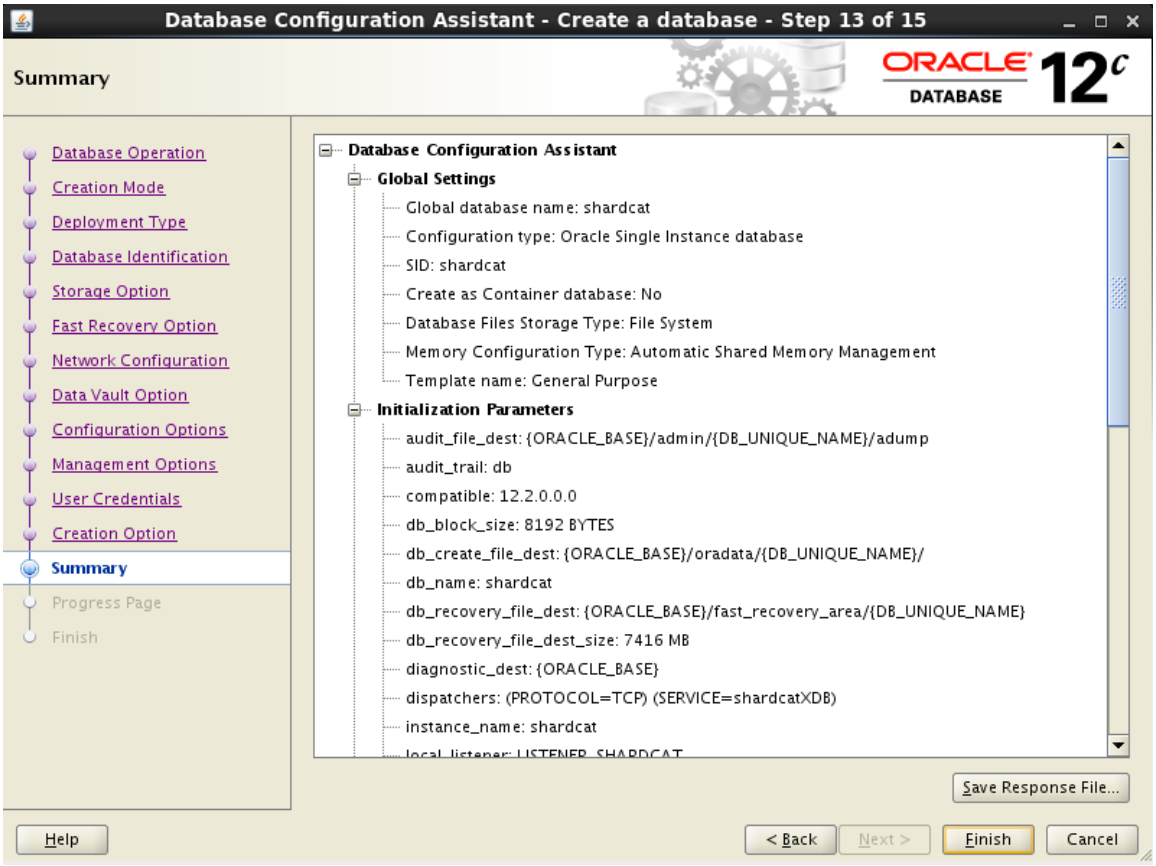
For the lab environment, select “Use the same administrative password for all accounts” and set the password as “oracle”. Hit “Next”.



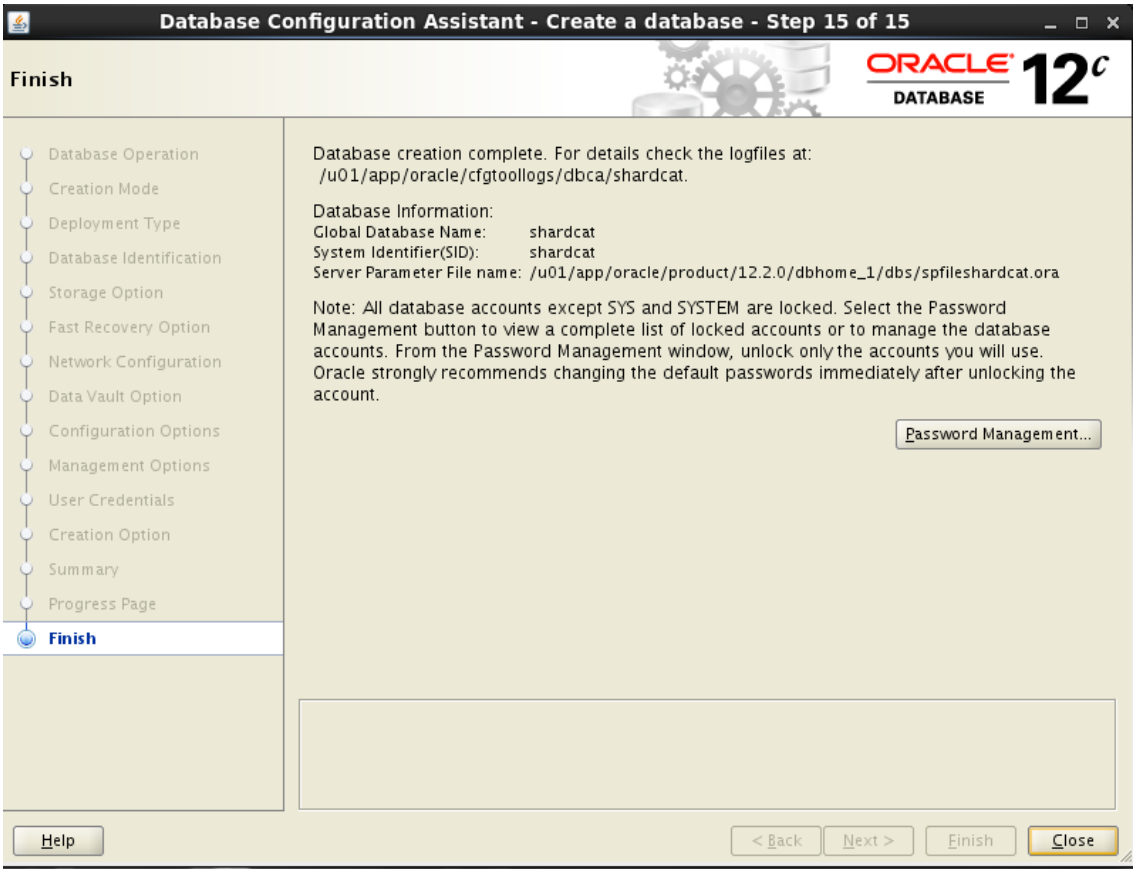
Click “Yes” and hit “Next”.



Select “Create database” and hit “Next”.



Click on Finish to complete the installation.



Make a note of the database information, Global Database Name, SID and SPfile of the shardcat database and hit “Close”.

Appendix C- Troubleshooting Tips

Alert Logs/Trace File Location:

Shard Director Alert Log Location: `/u01/app/oracle/diag/gsm/<shard-director-node>/sharddirector1/trace/alert*.log`

Shard Director Listener Trace: (See highlighted)

```
GDSCTL>status
Alias                     SHARDDIRECTOR1
Version                   12.2.0.0.2
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
```

GWM tracing on shardcatalog:

To get full tracing in the RDBMS, set GWM_TRACE level as shown below:

Note: Typically shared servers are used for many of the connections to the catalog/shards and therefore the tracing is typically in a shared server trace file named `<sid>_s00*.trc` (for example).

The following command(s) activate GDS tracing for RDBMS:

For immediate tracing (Activates all GDS tracing immediately, but will be lost after an RDBMS re-start)

```
alter system set events 'immediate trace name GWM_TRACE level 7';
```

To continue tracing after re-start of RDBMS (Activates all GDS tracing permanently, but does not take effect until next RDBMS re-start)

```
ALTER SYSTEM SET EVENT='10798 trace name context forever, level 7'
SCOPE=spfile;
```

To be safe, just set both traces. To trace everything, you will need to set this on both the shard catalog and all 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 (GSM) (on the shards).

For DDL propagation issues on Sharded, Duplicated tables and Materialized views setup (for Duplicated tables), execute the following:

```
$gdsctl show ddl
```

Observe the "`Failed shards`" column for the DDL of interest.

```
$gdsctl config shard -shard <shard>
```

Observe these fields in the output for the error signature.

```
"Last Failed DDL"
"DDL Error:" ---
"Failed DDL id:"
```

Deploy

Initial deploy and incremental deploy are (for the most part) in the RDBMS trace files as far as the PL/SQL that is run in the catalog. These would be files to be checked if there is every any issue during 'deploy'.

Move Chunk

Move chunk tracing is in the GSM alert log and the RDBMS trace files. Also, doing a '`gdsctl config chunks -show_reshard`' can also provide information on the current 'move chunk' status.

Appendix E - SampleShardedApp

Here is an example of simple app that uses the enhanced JDBC APIs to set the sharding_key and retrieves the pertinent rows. Leverage this code sample to build a sharded app on the sharded database that you have created as part of the labs covered in the cookbook.

MyJdbcShardingTest.java:

```
// Source Author: Oracle Development
// Compile:
//  export PATH=$JDKHOME_18/bin:$PATH
//  export
CLASSPATH=$ORACLE_HOME/jdbc/lib/ojdbc8.jar:$ORACLE_HOME/ons/lib/ons.jar:$ORACLE_HOME/opmn/
lib/ons.jar:.
//  export PATH=$JAVA_HOME/bin:$PATH
//  javac MyJdbcShardingTest.java
//  java MyJdbcShardingTest 1
//
// Run:
// java MyJdbcShardingTest <shard_key>
// e.g.
// java MyJdbcShardingTest 122

import java.sql.*;
import java.sql.PreparedStatement;
import oracle.jdbc.pool.OracleDataSource;
import oracle.jdbc.*;
 public class MyJdbcShardingTest {

  public static void main(String[] args)
  {
    String url = "jdbc:oracle:thin:@(description=(address=(protocol=tcp)"
        + "(host=hosta)(port=1540))"
        + "(connect_data=(service_name=rsvc.orasdb.shards)(region=dc1)))";
    try {
      int sk, CustID;
      String CustName;

      sk = Integer.parseInt(args[0]);
      System.out.println("Shard_key: "+ sk);

      OracleDataSource ods = new OracleDataSource();
      ods.setURL(url);
      ods.setUser("appuser");
      ods.setPassword("appuser");
       OracleShardingKey key = ods.createShardingKeyBuilder()
                            .subkey(sk, JDBCType.NUMERIC)
                            .build();
      Connection conn = ods.createConnectionBuilder()
                        .shardingKey(key)
                        .build();
       // Create a Statement
      Statement stmt1 = conn.createStatement ();
      Statement stmt2 = conn.createStatement ();

      // Select the SHARD DB NAME
      ResultSet rs1 = stmt1.executeQuery ("select name from v$database");
      while (rs1.next ())
      System.out.println ("Shard db name: " +rs1.getString (1));

      // Select the NAME column from the mycustomer table
```

```
      PreparedStatement pstmt = conn.prepareStatement("select cust_id, name from
mycustomer where cust_id=?");
      pstmt.setInt(1, sk );
      ResultSet rs2 = pstmt.executeQuery();

      // Iterate through the results and print the names
      while (rs2.next ()) {
        CustID = rs2.getInt("CUST_ID");
        CustName = rs2.getString("NAME");
        String outputString = String.format("%10d %s", CustID, CustName);
        System.out.println(outputString);
      }

    // Close the ResultSet
    rs1.close();
    rs2.close();

    // Close the Statement
    stmt1.close();
    stmt2.close();

    // Close the connection
    }
    catch (SQLException e) {
      e.printStackTrace();
    }
  }
}
```

Appendix F – SampleShardedApp2

```
/* Copyright (c) 2015, Oracle and/or its affiliates. All rights reserved.*/

/*
 DESCRIPTION
 The code sample demonstrates Universal Connection Pool (UCP) as a client
 side connection pool and does the following.
 (a) Picks up customer id from a list of customers.
 (b) Connects to a sharded database that contains a customer id.
 (c) Shows the details for that customer.

 Step 1:
 USER, PASSWORD, and URL are required.
 Step 2: Run the sample with "ant UCPSample"

 NOTES
 Use JDK 1.7 and above
 Make sure to have ojdbc8.jar, ucp.jar and ons.jar in the classpath


 MODIFIED    (MM/DD/YY)
 nbsundar    10/31/16 - Creation (Contributor - kmensah)
 */
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.PreparedStatement;

import oracle.ucp.jdbc.PoolDataSourceFactory;
import oracle.ucp.jdbc.PoolDataSource;

import oracle.jdbc.OracleShardingKey;
import oracle.jdbc.OracleType;

public class UCPShardingSample {
 //final static String DB_URL=
"jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(HOST=localhost)(PORT=5521)(PROTOCOL=tcp))(CONNEC
T_DATA=(SERVICE_NAME=jvma.regress.rdbms.dev.us.oracle.com)))";
   final static String DB_URL =
  "jdbc:oracle:thin:@ (DESCRIPTION="
  + "(FAILOVER=on)"
  + "(CONNECT_TIMEOUT=5)(TRANSPORT_CONNECT_TIMEOUT=3)(RETRY_COUNT=3)"
  + "(ADDRESS_LIST="
    +"(LOAD_BALANCE=ON)"
       +"(ADDRESS=(HOST=orclgsm1.oracle.com)(PORT=1571)(PROTOCOL=tcp))"
          +"(ADDRESS=(HOST=orclgsm2.oracle.com)(PORT=1571)(PROTOCOL=tcp)))"
  +"(ADDRESS_LIST="
    +"(LOAD_BALANCE=ON)"
       +"(ADDRESS=(HOST=orclgsm3.oracle.com)(PORT=1572)(PROTOCOL=tcp))"
          +"(ADDRESS=(HOST=orclgsm4.oracle.com)(PORT=1572)(PROTOCOL=tcp)))"
  +"(CONNECT_DATA="
    +"(SERVICE_NAME=oltp_rw_srvc.cust_sdb.oradbcloud) (REGION=region1)))";

  final static String DB_USER                = "sharduser";
  final static String DB_PASSWORD            = "test1";
  final static String CONN_FACTORY_CLASS_NAME = "oracle.jdbc.pool.OracleDataSource";
  final static String[] emailIds = new String[]{ "abc@abc.net", "xyz@google.com",
"test1@yahoo.com", "adam.test@x.bogus", "john.adams@test.com" };

  /*
```

```
   * The sample demonstrates UCP as client side connection pool.
   */
  public static void main(String args[]) throws Exception {
    // Get the PoolDataSource for UCP
    PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
        Connection connection = null;

    // Set the connection factory first before all other properties
    pds.setConnectionFactoryClassName(CONN_FACTORY_CLASS_NAME);
    pds.setURL(DB_URL);
    pds.setUser(DB_USER);
    pds.setPassword(DB_PASSWORD);
    pds.setConnectionPoolName("UCP_SHARDING_POOL");

    // Default is 0. Set the initial number of connections to be created
    // when UCP is started.
    pds.setInitialPoolSize(5);

    // Default is 0. Set the minimum number of connections
    // that is maintained by UCP at runtime.
    pds.setMinPoolSize(5);

    // Default is Integer.MAX_VALUE (2147483647). Set the maximum number of
    // connections allowed on the connection pool.
    pds.setMaxPoolSize(20);

        try {
                // Loop through this for 5 customers
                for (int i=0 ; i < 5 ; i++ ) {
                        String email = emailIds[i];
                        // Build the Sharding key by passing the email id, which is the
sharding key
                        OracleShardingKey
                           shardKey =  pds.createShardingKeyBuilder()
                                                   .subkey(email, OracleType.VARCHAR2)
                                                   .build();
                        // Get the connection by passing the sharding key
                        connection = pds.createConnectionBuilder()
                                                .shardingKey(shardKey)
                                                .build();
                    // Perform a database operation
            doSQLWork(connection, email);
                }
        }  catch (SQLException e) {
       System.out.println("UCPShardingSample - " + "SQLException occurred : "
           + e.getMessage());
    }
  }

  /*
   * This method shows the customer id, first name and last name
   */
  public static void doSQLWork(Connection conn, String email) {
    try {
      // Prepare a statement to execute the SQL Queries.
      PreparedStatement prepStatement =
          conn.prepareStatement("SELECT CUSTID, FIRSTNAME, LASTNAME FROM CUSTOMERS WHERE
CUSTID=?");
          prepStatement.setString(1, email);
      ResultSet customers = prepStatement.executeQuery();
      while (customers.next()) {
        //System.out.println("- order : " + rsOrders.getNString(1));
        System.out.print("Customer Id" + customers.getString("CUSTID"));
                System.out.print("First Name:" + customers.getString("FIRSTNAME"));
                System.out.println("Last Name:" + customers.getString("LASTNAME"));
```

```
      }
    } catch (SQLException e) {
      System.out.println("UCPShardingSample - "
          + "doSQLWork()- SQLException occurred : " + e.getMessage());
    }
  }
}
```

ORACLE

Oracle Corporation, World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065, USA

Worldwide Inquiries
Phone: +1.650.506.7000
Fax: +1.650.506.7200