2024

Oracle 19c Database Administration

INSTALL | CONFIGURE | MANAGE | OPTIMIZE

Contents

Oracle Database Architecture

Installation for Windows (Windows 8,10,11, Server2012, Server2016, Server2019, Server2022)

a. Download and install: https://www.oracle.com/database/technologies/oracle19c-windows-downloads.html

Installation on RHEL

a. Set static ip
b. Set hostname

```
$ sudo hostnamectl set-hostname <hostname>
$ hostname
```

c. Add ip and hostname to hosts file

```
$ sudo vim /etc/hosts
```

```
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
```

d. Set SELinux to permissive

```
$ getenforce                    //check selinux status
$ sudo setenforce 0
```

e. Update and install prereqs

```
$ sudo dnf update
$ sudo dnf install oracle-database-preinstall-19c
$ reboot
```

f. Download the rpm package
https://www.oracle.com/database/technologies/oracle19c-linux-downloads.html

g. Navigate to the file and install

```
$ sudo dnf -y localinstall <filename>.rpm
$ sudo rpm -i <filename>.rpm
```

h. Check Configuration file

```
$ cat /etc/sysconfig/oracledb_ORCLCDB-19c.conf
$ sudo /etc/init.d/oracledb_ORCLCDB-19c configure       //run configuration script
```

i. After installing oracle database 19c, you will get a new user “oracle”

```
$ cat /etc/passwd               //check if oracle user exists
$ passwd oracle                 //reset oracle user password
$ su oracle                     //switch user to oracle
$ ps | grep ora                 //check if there are running processes or database instance
```

j. Configure oracle user account profile

```
$ su oracle
$ sudo vim .bash_profile
```

```
# add to the end
umask 022
export ORACLE_SID=ORCLCDB
export ORACLE_BASE=/opt/oracle
export ORACLE_HOME=/opt/oracle/product/19c/dbhome_1
export PATH=$PATH:$ORACLE_HOME/bin
```

```
$ source .bash_profile
```

k. Access database using sqlplus

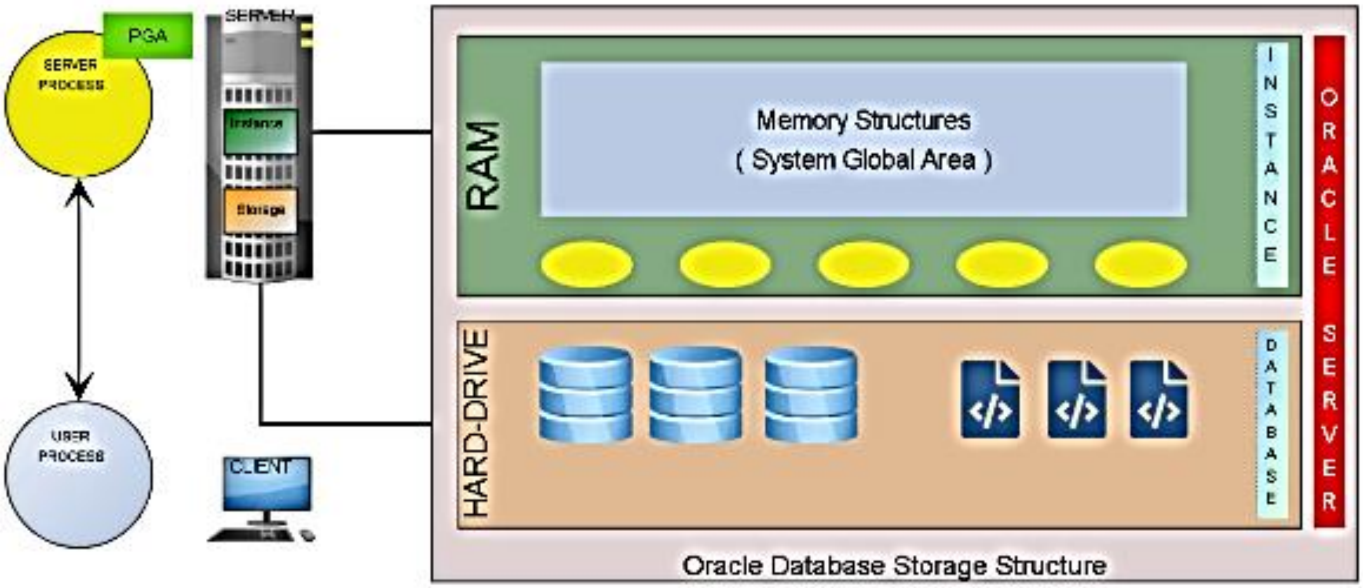
```
$ sqlplus / as sysdba
```

l. Commonly used sqlplus commands

```
sql> startup                    // start the instance
sql> shutdown immediate         // stop the instance
sql> exit                       // exit sqlplus
sql> clear screen               // clear sqlplus console
sql> set hist on                // enable command history temporarily for the session
sql> hist 2 run                 // run the 2nd command in history
sql> connect username/password@database        // connect to another database
sql> disconnect
sql> show user                  // currently connected user
sql> select * from employees    // select query to a table
sql> describe employees         // show table structure
sql> spool output.txt           // output to file
sql> spool off
sql> variable var1 number       // create and set data type
sql> print var1                 // show variable value
sql> define var1=24             // set value to variable
sql> undefine var1              // remove value to variable
sql> define var1=24             // set value to variable
sql> define var1=24             // set value to variable
sql> define var1=24             // set value to variable
sql> CONNECT sys/password@CDB_NAME AS SYSDBA        // Connect to CDB
sql> CONNECT sys/password@PDB_NAME AS SYSDBA        // Connect to PDB

// Create a PDB
sql>    CREATE PLUGGABLE DATABASE pdb_name
        ADMIN USER pdb_admin IDENTIFIED BY password
        FILE_NAME_CONVERT = ('/path/to/cdb/datafile/', '/path/to/pdb/datafile/');

sql> show pdbs                  // show all pdbs in the instance
sql> ALTER PLUGGABLE DATABASE pdb_name OPEN;                    // Open a PDB
sql> ALTER PLUGGABLE DATABASE pdb_name CLOSE IMMEDIATE;         // Close a PDB
sql> ALTER PLUGGABLE DATABASE ALL OPEN;                         // Open All PDBs
sql> ALTER PLUGGABLE DATABASE ALL CLOSE IMMEDIATE;              // Close All PDBs
sql> ALTER SESSION SET CONTAINER = pdb_name;                    // Switch to a PDB
sql> ALTER SESSION SET CONTAINER = CDB$ROOT;               // Switch to the Root Container
sql> SELECT con_id, name, open_mode FROM v$pdbs;           // List All PDBs
sql> SHOW CON_NAME;                                        // Show Current Container
```

```
// Unplug a PDB
sql> ALTER PLUGGABLE DATABASE pdb_name CLOSE IMMEDIATE;
sql> ALTER PLUGGABLE DATABASE pdb_name UNPLUG INTO '/path/to/pdb_name.xml';
sql> DROP PLUGGABLE DATABASE pdb_name KEEP DATAFILES;       // Drop an Unplugged PDB

// Plug an Existing PDB
sql>    CREATE PLUGGABLE DATABASE pdb_name USING '/path/to/pdb_name.xml'
        FILE_NAME_CONVERT = ('/path/to/source/', '/path/to/destination/');

sql> BACKUP PLUGGABLE DATABASE pdb_name;                    // Backup a PDB
sql> RECOVER PLUGGABLE DATABASE pdb_name;                   // Recover a PDB
```

Oracle Database Server Architecture



Database Server 3 Main Components

- ✓ Database Instance in memory (the SGA and the background processes)
- ✓ Files that store the data and database information
- ✓ DB software (binary/executable files for functioning of Oracle)

After starting a database instance, the Oracle software associates the instance with a specific database. This is called *mounting the database*. The database is then ready to be opened, which makes it accessible to authorized users.

Oracle Database Instance Configurations

Configuring an Oracle Database instance involves several steps, including setting up the initialization parameters, configuring memory management, and tuning performance. Here’s a comprehensive guide to Oracle Database instance configurations:

Initialization Parameters
Initialization parameters control the behavior of an Oracle Database instance. These parameters can be set in the init.ora or spfile.ora files.

File Locations:

- PFILE (init.ora): Plain text file located at $ORACLE_HOME/dbs/init<SID>.ora.
- SPFILE (spfile.ora): Binary file located at $ORACLE_HOME/dbs/spfile<SID>.ora.

Basic Initialization Parameters:

DB_NAME: Name of the database.

```
DB_NAME = mydb
```

DB_DOMAIN: Domain name of the database.

```
DB_DOMAIN = example.com
```

INSTANCE_NAME: Name of the instance.

```
INSTANCE_NAME = mydb1
```

CONTROL_FILES: Locations of control files.

```
CONTROL_FILES = ('/path/to/control01.ctl', '/path/to/control02.ctl')
```

Memory Management
Oracle provides two types of memory management: Automatic Memory Management (AMM) and Manual Memory Management.

Automatic Memory Management (AMM):

MEMORY_TARGET: Total amount of memory allocated to the database.

```
MEMORY_TARGET = 2G
```

MEMORY_MAX_TARGET: Maximum memory target.

```
MEMORY_MAX_TARGET = 2G
```

Manual Memory Management:

SGA_TARGET: Size of the System Global Area.

```
SGA_TARGET = 1G
```

PGA_AGGREGATE_TARGET: Size of the Program Global Area.

```
PGA_AGGREGATE_TARGET = 500M
```

Configuring Shared Pool and Buffer Cache

Shared Pool Size:

```
SHARED_POOL_SIZE = 256M
```

Database Buffer Cache Size:

```
DB_CACHE_SIZE = 512M
```

Redo Log and Archive Log

Redo Log File Size:

```
LOG_FILE_SIZE = 100M
```

Archive Log Mode:

Enable Archive Log Mode:

```
ALTER SYSTEM SET log_archive_dest_1='LOCATION=/path/to/archive/';
ALTER SYSTEM SET log_archive_format='%t_%s_%r.dbf';
ALTER DATABASE ARCHIVELOG;
ALTER DATABASE OPEN;
```

Diagnostic and Tuning

Diagnostic Destination:

```
DIAGNOSTIC_DEST = /path/to/diag
```

Automatic Workload Repository (AWR):

Configure AWR snapshot interval and retention:

```
EXEC DBMS_WORKLOAD_REPOSITORY.modify_snapshot_settings(interval => 60, retention => 7*24*60);
```

Performance Tuning

Optimizer Mode:

```
OPTIMIZER_MODE = ALL_ROWS
```

Cursor Sharing:

```
CURSOR_SHARING = FORCE
```

Parallel Execution:

```
PARALLEL_MAX_SERVERS = 16
PARALLEL_MIN_SERVERS = 4
```

Undo Management

Automatic Undo Management:

```
UNDO_MANAGEMENT = AUTO
UNDO_TABLESPACE = undotbs1
```

Network Configuration

Listener Configuration (listener.ora):

```
SID_LIST_LISTENER =
 (SID_LIST =
  (SID_DESC =
   (SID_NAME = mydb)
   (ORACLE_HOME = /path/to/oracle_home)
   (GLOBAL_DBNAME = mydb.example.com)
  )
 )

LISTENER =
 (DESCRIPTION_LIST =
  (DESCRIPTION =
   (ADDRESS = (PROTOCOL = TCP)(HOST = myhost)(PORT = 1521))
  )
 )
```

TNS Configuration (tnsnames.ora):
(text)

```
mydb =
 (DESCRIPTION =
  (ADDRESS_LIST =
   (ADDRESS = (PROTOCOL = TCP)(HOST = myhost)(PORT = 1521))
  )
  (CONNECT_DATA =
   (SERVICE_NAME = mydb.example.com)
  )
 )
```

Backup and Recovery Configuration

Enable Flashback Database:

```
ALTER DATABASE FLASHBACK ON;
```

Configure RMAN (Recovery Manager):

```
RMAN> CONFIGURE CONTROLFILE AUTOBACKUP ON;
RMAN> CONFIGURE RETENTION POLICY TO REDUNDANCY 2;
```

User and Security Management

Create Users and Assign Roles:

```
CREATE USER myuser IDENTIFIED BY password;
GRANT CONNECT, RESOURCE TO myuser;
```

Profile Management:

```
CREATE PROFILE myprofile LIMIT
 SESSIONS_PER_USER 5
 CPU_PER_SESSION 10000
 CONNECT_TIME 60
 IDLE_TIME 10;

ALTER USER myuser PROFILE myprofile;
```

Comprehensive Example: Initialization Parameter File (init.ora)

Here's an example of a complete initialization parameter file:

```
DB_NAME = mydb
DB_DOMAIN = example.com
INSTANCE_NAME = mydb1
CONTROL_FILES = ('/path/to/control01.ctl', '/path/to/control02.ctl')
MEMORY_TARGET = 2G
MEMORY_MAX_TARGET = 2G
SHARED_POOL_SIZE = 256M
DB_CACHE_SIZE = 512M
LOG_FILE_SIZE = 100M
DIAGNOSTIC_DEST = /path/to/diag
OPTIMIZER_MODE = ALL_ROWS
CURSOR_SHARING = FORCE
PARALLEL_MAX_SERVERS = 16
PARALLEL_MIN_SERVERS = 4
UNDO_MANAGEMENT = AUTO
UNDO_TABLESPACE = undotbs1
```

Connecting to the Database Instance

Connecting to an Oracle Database instance involves understanding various methods and tools available for establishing a connection. Here's a comprehensive guide on how to connect to an Oracle Database instance using different methods:

Using SQL*Plus

SQL*Plus is a command-line tool provided by Oracle to interact with the database.

1. Open SQL*Plus:

```
 sqlplus
```

2. Connect using a username and password:

```
  CONNECT username/password@host:port/service_name
```

Example:

```
  CONNECT scott/tiger@localhost:1521/orclpdb1
```

3. Connect using Easy Connect Naming Method:

```
CONNECT username/password@//host:port/service_name
```

Example:

```
CONNECT scott/tiger@//localhost:1521/orclpdb1
```

4. Connect using TNS alias:
Ensure your `tnsnames.ora` file is configured properly.

```
CONNECT username/password@tns_alias
```

Example:

```
CONNECT scott/tiger@orcl
```

5. Connect as SYSDBA or SYSOPER:

```
CONNECT sys/password@host:port/service_name AS SYSDBA
```

Example:

```
CONNECT sys/password@localhost:1521/orclpdb1 AS SYSDBA
```

Using Oracle SQL Developer

Oracle SQL Developer is a graphical tool for database development.

1. Open Oracle SQL Developer.

2. Create a new connection:
  - Click on the New Connection button.
  - Enter the connection details:
    - Connection Name: Any name for your connection.
    - Username: Your database username.
    - Password: Your database password.
    - Hostname: The database host (e.g., `localhost`).
    - Port: The database port (default is `1521`).
    - SID or Service Name: The database SID or service name.
  - Click Test to verify the connection.
  - Click Connect to establish the connection.

Using JDBC (Java Database Connectivity)

JDBC allows Java applications to connect to an Oracle Database.

1. Include the Oracle JDBC Driver in your project.
  - Download the Oracle JDBC driver from the Oracle website and add it to your project's classpath.

2. Write Java code to connect to the database:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class OracleJDBCExample {
    public static void main(String[] args) {
```

```
    String jdbcUrl = "jdbc:oracle:thin:@localhost:1521:orclpdb1";
    String username = "scott";
    String password = "tiger";

    try {
      Connection connection = DriverManager.getConnection(jdbcUrl, username, password);
      System.out.println("Connected to the database!");
      connection.close();
    } catch (SQLException e) {
      e.printStackTrace();
    }
  }
}
```

Using Python with cx_Oracle
cx_Oracle is a Python library that allows interaction with Oracle databases.

1. Install cx_Oracle:

```
pip install cx_Oracle
```

2. Write Python code to connect to the database:

```
import cx_Oracle

dsn = cx_Oracle.makedsn("localhost", 1521, service_name="orclpdb1")
connection = cx_Oracle.connect(user="scott", password="tiger", dsn=dsn)

print("Connected to the database!")
connection.close()
```

Using Oracle Net Services (NetCA and Net Manager)
Oracle Net Services configuration involves setting up the listener and configuring client-side connections.

1. Configure the Listener (listener.ora):

```
SID_LIST_LISTENER =
 (SID_LIST =
  (SID_DESC =
   (SID_NAME = orcl)
   (ORACLE_HOME = /u01/app/oracle/product/19.0.0/dbhome_1)
  )
 )

LISTENER =
 (DESCRIPTION_LIST =
  (DESCRIPTION =
   (ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1521))
  )
 )
```

2. Configure Client-Side Connections (tnsnames.ora):

```
ORCL =
 (DESCRIPTION =
  (ADDRESS_LIST =
   (ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1521))
  )
  (CONNECT_DATA =
   (SERVICE_NAME = orcl)
  )
 )
```

3. Start the Listener:

```
lsnrctl start
```

4. Connect using SQL*Plus with TNS alias:

```
CONNECT scott/tiger@ORCL
```

Using Oracle Data Pump for Remote Connections

Oracle Data Pump allows importing and exporting data remotely.

1. Export Data:

```
  expdp username/password@//localhost:1521/orclpdb1 DIRECTORY=exp_dir DUMPFILE=expdp.dmp
LOGFILE=expdp.log SCHEMAS=schema_name
```

2. Import Data:

```
  impdp username/password@//localhost:1521/orclpdb1 DIRECTORY=imp_dir DUMPFILE=expdp.dmp
LOGFILE=impdp.log SCHEMAS=schema_name
```

Oracle Database Memory Structures

Oracle 19c, like its predecessors, utilizes a sophisticated memory architecture designed to manage and process data efficiently. Understanding these memory structures is crucial for database administrators to optimize performance and ensure stability. Here’s a comprehensive guide to the memory structures in Oracle 19c:

Overview of Oracle Memory Structures

Oracle Database uses two main memory structures:

- ✓ System Global Area (SGA)
- ✓ Program Global Area (PGA)

System Global Area (SGA)

The SGA is a shared memory region that contains data and control information for one Oracle Database instance. It is allocated when the database instance starts and deallocated when the instance shuts down. The SGA consists of several components:

Key Components of the SGA:

- Database Buffer Cache
- Shared Pool
- Redo Log Buffer
- Large Pool
- Java Pool
- Streams Pool
- Data Dictionary Cache
- Result Cache
- In-Memory Column Store (optional)

1. Database Buffer Cache
   - Purpose: Stores copies of data blocks read from data files. Frequently accessed data blocks are cached here to improve performance.
   - Tuning Parameter: `DB_CACHE_SIZE`

2. Shared Pool
   - Purpose: Caches various types of information that can be shared among users, such as SQL statements, PL/SQL code, and data dictionary information.
   - Components:
     - Library Cache: Stores executable forms of SQL and PL/SQL.
     - Data Dictionary Cache: Stores metadata about database objects.
   - Tuning Parameter: `SHARED_POOL_SIZE`

3. Redo Log Buffer
   - Purpose: Temporarily stores redo entries (change vectors) before they are written to the redo log files. Ensures recovery of committed transactions.
   - Tuning Parameter: `LOG_BUFFER`

4. Large Pool
   - Purpose: Provides memory for large memory allocations, such as Oracle backup and recovery operations, and I/O server processes.
   - Tuning Parameter: `LARGE_POOL_SIZE`

5. Java Pool
   - Purpose: Used for all session-specific Java code and data within the JVM.
   - Tuning Parameter: `JAVA_POOL_SIZE`

6. Streams Pool
   - Purpose: Used by Oracle Streams to store information related to capture and apply processes.
   - Tuning Parameter: `STREAMS_POOL_SIZE`

7. Data Dictionary Cache
   - Purpose: Part of the shared pool; stores metadata about database objects.
   - Tuning Parameter: Managed within `SHARED_POOL_SIZE`

8. Result Cache
    - Purpose: Caches the results of SQL queries and PL/SQL functions for faster retrieval.
    - Tuning Parameter: `RESULT_CACHE_MAX_SIZE`

9. In-Memory Column Store (optional)
    - Purpose: Stores a copy of table data in a columnar format for analytic query performance improvements.
    - Tuning Parameter: `INMEMORY_SIZE`

Program Global Area (PGA)
The PGA is a memory region that contains data and control information for a single server process or background process. Unlike the SGA, the PGA is not shared and is allocated when a process is started.

Key Components of the PGA:

- Session Memory
- Private SQL Area
- Sort Area
- Hash Area
- Bitmap Merge Area

1. Session Memory
    - Purpose: Contains session variables, logon information, and other session-specific information.

2. Private SQL Area
    - Purpose: Contains data such as bind variable values and runtime buffers. Each session that issues a SQL statement has a private SQL area.

3. Sort Area
    - Purpose: Used for sort operations such as ORDER BY, GROUP BY, and creating indexes.

4. Hash Area
    - Purpose: Used for hash join operations.

5. Bitmap Merge Area
    - Purpose: Used for bitmap index merge operations.

Automatic Memory Management (AMM)
Oracle can automatically manage the sizes of the SGA and PGA memory areas using Automatic Memory Management (AMM).

Parameters for AMM:
1. MEMORY_TARGET: Total memory allocated to Oracle for both SGA and PGA.
2. MEMORY_MAX_TARGET: Maximum size to which MEMORY_TARGET can be set dynamically.

Example Configuration:

```
ALTER SYSTEM SET MEMORY_TARGET=4G SCOPE=BOTH;
ALTER SYSTEM SET MEMORY_MAX_TARGET=4G SCOPE=BOTH;
```

Automatic Shared Memory Management (ASMM)

If you prefer more control over the SGA, you can use Automatic Shared Memory Management (ASMM). This allows Oracle to automatically adjust the sizes of the various SGA components based on workload.

Parameters for ASMM:
1. SGA_TARGET: Total size of all SGA components.
2. SGA_MAX_SIZE: Maximum size to which SGA_TARGET can be set dynamically.

Example Configuration:

```
ALTER SYSTEM SET SGA_TARGET=2G SCOPE=BOTH;
ALTER SYSTEM SET SGA_MAX_SIZE=2G SCOPE=BOTH;
```

Manual Memory Management

If you opt for manual memory management, you will need to set the sizes for each individual component of the SGA and PGA.

Example Configuration:

```
ALTER SYSTEM SET DB_CACHE_SIZE=1G SCOPE=BOTH;
ALTER SYSTEM SET SHARED_POOL_SIZE=500M SCOPE=BOTH;
ALTER SYSTEM SET LARGE_POOL_SIZE=128M SCOPE=BOTH;
ALTER SYSTEM SET JAVA_POOL_SIZE=64M SCOPE=BOTH;
ALTER SYSTEM SET STREAMS_POOL_SIZE=64M SCOPE=BOTH;
ALTER SYSTEM SET PGA_AGGREGATE_TARGET=1G SCOPE=BOTH;
```

Memory Advisors

Oracle provides several memory advisors to help tune the memory components:
1. SGA Target Advisor
2. PGA Target Advisor
3. Buffer Cache Advisor
4. Shared Pool Advisor

These advisors provide recommendations based on the workload and performance metrics.

Example: Using Memory Advisors

```
EXEC DBMS_ADVISOR.create_task('ADDM', :task_name, :task_desc);
EXEC DBMS_ADVISOR.set_task_parameter(:task_name, 'START_SNAPSHOT', :start_snap);
EXEC DBMS_ADVISOR.set_task_parameter(:task_name, 'END_SNAPSHOT', :end_snap);
EXEC DBMS_ADVISOR.execute_task(:task_name);
```

Creating DBCS Database Deployments

Automated Database Provisioning and Creating a Database Deployment

In Oracle 19c, automated database provisioning can be performed using Oracle's Database Creation Assistant (DBCA) in silent mode or using Oracle Enterprise Manager Cloud Control. Here are the steps to perform automated database provisioning using both methods:

Using DBCA in Silent Mode
Prepare the Response File: Create a response file with all the necessary parameters for the database creation. This file is a plain text file that contains key-value pairs.

Example of a response file (dbca.rsp):

```
[GENERAL]
RESPONSEFILE_VERSION = "19.0.0"
OPERATION_TYPE = "createDatabase"

[CREATEDATABASE]
GDBNAME = "orcl"
SID = "orcl"
TEMPLATE_NAME = "General_Purpose.dbc"
CHARACTERSET = "AL32UTF8"
NATIONALCHARACTERSET = "AL16UTF16"
TOTALMEMORY = "2048"
SYSPASSWORD = "sys_password"
SYSTEMPASSWORD = "system_password"
```

Run DBCA in Silent Mode: Use the dbca command with the -silent flag and the response file to create the database.

```
dbca -silent -responseFile /path/to/dbca.rsp
```

Using Oracle Enterprise Manager Cloud Control
Oracle Enterprise Manager (OEM) Cloud Control provides a more graphical and centralized approach for database provisioning. Here's how you can automate provisioning using OEM:

1. Create a Provisioning Profile:
   - Navigate to the Enterprise menu, select Provisioning and Patching, then Database Provisioning.
   - Create a new profile by capturing an existing database configuration or using a template.

2. Create a Deployment Procedure:
   - Go to the Deployments tab and select Database Provisioning.
   - Create a new deployment procedure using the profile you created.

3. Schedule the Provisioning Job:
   - Schedule the job to run immediately or at a specified time.
   - You can also set up recurring schedules for regular provisioning tasks.

4. Monitor and Manage:
    - Use the OEM interface to monitor the status of your provisioning jobs.
    - Manage and adjust the procedures as needed based on your requirements.

Example Response File for Automated Database Creation
Here's a more detailed example of a response file (dbca_response.rsp) for DBCA:

```
[GENERAL]
RESPONSEFILE_VERSION = "19.0.0"
OPERATION_TYPE = "createDatabase"

[CREATEDATABASE]
GDBNAME = "orcl.example.com"
SID = "orcl"
TEMPLATENAME = "General_Purpose.dbc"
SYSPASSWORD = "Welcome1"
SYSTEMPASSWORD = "Welcome1"
DATAFILEDESTINATION = "/u01/app/oracle/oradata"
RECOVERYAREADESTINATION = "/u01/app/oracle/flash_recovery_area"
STORAGETYPE = "FS"
CHARACTERSET = "AL32UTF8"
NATIONALCHARACTERSET= "AL16UTF16"
TOTALMEMORY = "4096"
```

Running the DBCA Command
Once you have your response file prepared, you can run the following command to initiate the automated provisioning:

```
dbca -silent -responseFile /path/to/dbca_response.rsp
```

This approach allows for full automation of the database creation process, making it suitable for environments where rapid provisioning is required.

Accessing an Oracle Database

Connecting to an Oracle Database Instance

In Oracle 19.3c, there are several ways to connect to an Oracle database instance. These methods can be used based on the specific requirements and environments. Here are the common methods:

1. SQL*Plus
SQL*Plus is a command-line tool that comes with Oracle Database for running SQL and PL/SQL commands.

Connecting using SQL*Plus

```
sqlplus username/password@hostname:port/SID
```

Example:

```
sqlplus system/Welcome1@localhost:1521/orcl
```

2. Oracle SQL Developer
Oracle SQL Developer is a free integrated development environment that simplifies the management of Oracle databases in both traditional and Cloud deployments.

Connecting using Oracle SQL Developer

1) Open SQL Developer.
2) Click on the "New Connection" button.
3) Enter the connection details such as:
   - Connection Name
   - Username
   - Password
   - Hostname
   - Port
   - SID or Service Name
4) Click "Connect."

3. JDBC
Java applications connect to Oracle Database using JDBC (Java Database Connectivity).

Example JDBC Connection

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class OracleJDBCExample {
   public static void main(String[] argv) {
      try {
         Class.forName("oracle.jdbc.driver.OracleDriver");

         Connection connection = DriverManager.getConnection(
            "jdbc:oracle:thin:@localhost:1521:orcl", "username", "password");

         System.out.println("Connected to Oracle database!");

      } catch (ClassNotFoundException | SQLException e) {
         e.printStackTrace();
      }
   }
}
```

4. ODBC
ODBC (Open Database Connectivity) allows C and C++ applications to connect to Oracle Database.

ODBC Connection String

```
Driver={Oracle in OraHome19};Dbq=localhost:1521/orcl;Uid=username;Pwd=password;
```

5. Oracle Net Services (TNS)
Oracle Net Services provide enterprise-wide connectivity solutions in distributed, heterogeneous computing environments.

TNS Names Configuration
In the tnsnames.ora file:

```
ORCL =
 (DESCRIPTION =
  (ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1521))
  (CONNECT_DATA =
   (SERVER = DEDICATED)
   (SERVICE_NAME = orcl)
  )
 )
```

Connecting using TNS

```
sqlplus username/password@ORCL
```

6. Python with cx_Oracle
cx_Oracle is a Python extension module that enables access to Oracle Database.

Example Python Connection using cx_Oracle

```
import cx_Oracle
connection = cx_Oracle.connect("username/password@localhost:1521/orcl")

print("Connected to Oracle Database")
```

Oracle Database Tools

Oracle 19.3c offers a wide range of tools for managing, developing, and administering Oracle databases. Here are some of the primary tools available:

1. SQL*Plus

- Description: A command-line tool used to run SQL and PL/SQL commands, manage database objects, and execute scripts.
- Usage: Ideal for quick, manual interactions with the database.

2. Oracle SQL Developer

- Description: A free graphical integrated development environment (IDE) for database development and management.
- Features: Supports running SQL queries, designing database schemas, managing database objects, and developing PL/SQL code.

3. Oracle Enterprise Manager (OEM) Cloud Control

- Description: A comprehensive web-based management tool for monitoring and managing Oracle databases and other Oracle products.
- Features: Performance monitoring, alerting, automated maintenance, and provisioning.

4. Database Configuration Assistant (DBCA)

- Description: A graphical tool to create, configure, and delete Oracle databases.
- Features: Supports silent mode operations for automated database creation.

5. Oracle Data Pump

- Description: A utility for fast data movement between Oracle databases.
- Components: expdp for export, impdp for import.
- Features: High-speed data transfer, parallel execution, and advanced filtering.

6. Oracle Recovery Manager (RMAN)

- Description: A command-line tool for backup and recovery operations.
- Features: Full and incremental backups, point-in-time recovery, and integration with Oracle Enterprise Manager.

7. Oracle Universal Installer (OUI)

- Description: A tool for installing Oracle software.
- Features: GUI-based and silent mode installation options.

8. Oracle Net Manager

- Description: A tool to configure Oracle Net Services.
- Features: Manage listener configurations, naming methods, and directory usage.

9. Oracle Enterprise Manager Database Express (EM Express)

- Description: A web-based management tool included with Oracle Database 19c.
- Features: Lightweight, for database performance management, and simple administrative tasks.

10. Oracle SQL*Loader

- Description: A tool to load data from external files into Oracle Database tables.
- Features: Supports various data formats, direct path loads, and efficient data processing.

11. Oracle APEX (Application Express)

- Description: A low-code development platform for building web applications.
- Features: Integrated development environment, SQL Workshop, and user interface design tools.

12. Oracle Listener Control Utility (LSNRCTL)

- Description: A command-line utility to manage Oracle Listeners.
- Features: Start, stop, and manage database listener processes.

13. Oracle Wallet Manager

- Description: A tool to manage security credentials on Oracle databases.
- Features: Create, manage, and store encryption keys and certificates.

14. Oracle Clusterware and Oracle Grid Infrastructure

- Description: Tools and utilities for managing Oracle Real Application Clusters (RAC) and Oracle Automatic Storage Management (ASM).
- Features: High availability, resource management, and cluster configuration.

15. Oracle Data Guard Broker
    - Description: A management and automation tool for Oracle Data Guard configurations.
    - Features: Simplified setup, management, and monitoring of Data Guard environments.

16. Oracle GoldenGate
    - Description: A comprehensive software package for real-time data integration and replication.
    - Features: Data capture, transformation, and delivery with high availability.

17. Oracle OLAP (Online Analytical Processing)
    - Description: Tools for multidimensional analysis within Oracle databases.
    - Features: Cube building, advanced analytics, and data warehousing capabilities.

SQL*Plus

Connecting to SQL*Plus

1. Open a terminal or command prompt.
2. Run the sqlplus command with your username and password.

```
sqlplus username/password@hostname:port/SID
```

Example:

```
sqlplus system/Welcome1@localhost:1521/orcl
```

Basic Commands

- Connect to Database:
  ```
  CONNECT username/password@hostname:port/SID;
  ```

- Run SQL Commands:
  ```
  SELECT * FROM employees;
  ```

- Execute Scripts:
  ```
  @path/to/script.sql
  ```

- Exit:
  ```
  EXIT;
  ```

Oracle SQL Developer

Download and Installation

1) Download SQL Developer from the Oracle website.
2) Extract the downloaded file and run sqldeveloper.exe (on Windows) or the corresponding executable on other OS.

Connecting to a Database

1) Open SQL Developer.
2) Click the "New Connection" button.
3) Enter the connection details:
    a. Connection Name: A name for your connection.
    b. Username: Your database username.

   c. Password: Your database password.
   d. Hostname: The hostname of your database server.
   e. Port: The port number (default is 1521).
   f. SID/Service Name: The SID or Service Name of your database.
4) Click "Connect".

Running Queries

1) Open a new SQL Worksheet.
2) Type your SQL query.
3) Click the "Run" button (or press F5).

SQL Developer Command Line (SQLcl)

Download and Installation

1) Download SQLcl from the Oracle website https://www.oracle.com/database/sqldeveloper/technologies/sqlcl/download/
2) Unzip the downloaded file.
3) Add the SQLcl directory to your system's PATH.

Connecting to a Database

1) Open a terminal or command prompt.
2) Run the sql command with your connection details:

```
sql username/password@hostname:port/SID
```

Example:

```
sql system/Welcome1@localhost:1521/orcl
```

Basic Commands

Run SQL Commands:

```
SELECT * FROM employees;
```

Execute Scripts:

```
@path/to/script.sql
```

Exit:

```
EXIT;
```

Database Configuration Assistant (DBCA)

Running DBCA

1) Open a terminal or command prompt.
2) Run the dbca command.

```
dbca
```

Creating a Database

1) Choose "Create a database".
2) Follow the prompts to specify your database settings:
    - ✓ Database Name: Enter the Global Database Name.
    - ✓ Database Configuration: Choose a template or customize the database settings.
    - ✓ Management Options: Configure Enterprise Manager settings.
    - ✓ Database Credentials: Set passwords for administrative accounts.
    - ✓ Storage Options: Specify storage locations and options.
    - ✓ Initialization Parameters: Set memory, character set, and other parameters.
    - ✓ Database Creation: Review and create the database.

Oracle Enterprise Manager Database Express

Accessing OEM Database Express

1) Ensure the database is running.
2) Open a web browser and go to the URL:

```
https://hostname:5500/em
```

Logging In

1) Enter the administrative username (e.g., SYS) and password.
2) Select the "Login" button.

Basic Operations

- ✓ Monitor Performance: Navigate to the "Performance" tab to monitor database performance.
- ✓ Manage Storage: Use the "Storage" tab to manage tablespaces, data files, and redo logs.
- ✓ User Management: Create and manage database users under the "Security" tab.
- ✓ Running SQL Queries: Use the "SQL" tab to run SQL statements.

Enterprise Manager Cloud Control

Installation and Setup

1) Download OEM Cloud Control:
2) Obtain the latest version of OEM Cloud Control from the Oracle website.
3) Follow the installation guide provided by Oracle to install OEM on your server. The installation typically involves setting up the Oracle Management Service (OMS) and Oracle Management Repository (OMR).

Post-Installation Configuration:

1) Access the OEM console via a web browser. The URL is typically in the format:

```
https://<hostname>:<port>/em
```

2) Log in with the SYSMAN account or another administrative account.

Adding Targets

To manage databases and other targets, you need to add them to OEM Cloud Control.

1) Navigate to the Setup Menu:
    a. Click on Setup in the top right corner.
    b. Select Add Target > Auto Discovery Results.

2) Configure Auto Discovery:
    a. Run the auto discovery process to find databases and other targets on the network.
    b. Alternatively, you can manually add targets by selecting Add Target Manually.

3) Add a Database:
    a. Choose the option to add a database.
    b. Enter the connection details for the database (hostname, port, SID/Service Name, and credentials).
    c. Follow the prompts to complete the registration.

Managing Databases

1) Navigate to the Database Home Page:
    a. From the main dashboard, click on the Targets menu.
    b. Select Databases to view a list of managed databases.
    c. Click on the database you want to manage to open its home page.

2) Monitor Database Performance:
    a. Use the Performance menu to view real-time and historical performance metrics.
    b. Monitor CPU usage, memory usage, I/O activity, and more.

3) Manage Storage:
    a. Navigate to the Storage menu to manage tablespaces, data files, and ASM (if applicable).
    b. Perform tasks such as adding or resizing data files.

4) User and Security Management:
    a. Use the Security menu to manage users, roles, and privileges.
    b. Implement security policies and audit settings.

5) Backup and Recovery:
    a. Navigate to the Availability menu.
    b. Configure and manage RMAN backups, perform restore operations, and manage Data Guard configurations.

Automated Tasks and Alerts

1) Create and Schedule Jobs:
    a. From the main menu, navigate to Enterprise > Job > Library.
    b. Create new jobs or use predefined job templates.
    c. Schedule jobs to run at specified intervals.

2) Set Up Alerts:
    a. Navigate to Setup > Incidents > Incident Rules.
    b. Define rules to generate alerts based on specific conditions (e.g., performance thresholds, availability issues).
    c. Configure notifications to be sent via email or other channels.

Reporting

1) Generate Reports:
    a. Navigate to Reports > Reports Library.
    b. Use predefined reports or create custom reports based on your needs.

   c. Schedule reports to be generated and sent via email.

Patching and Provisioning

1) Patching:
   a. Navigate to Enterprise > Provisioning and Patching > Database Provisioning.
   b. Use OEM to automate the patching process for databases and other targets.

2) Provisioning:
   a. Use the Provisioning menu to clone databases, set up test environments, and deploy new database instances.
   b. Using the Command Line Interface (EMCLI)

OEM Cloud Control also provides a command-line interface (EMCLI) for scripting and automation.

1) Install EMCLI:
   a. Download and install the EMCLI client from the OEM console.
   b. Follow the setup instructions provided by Oracle.

2) Using EMCLI:
   a. Authenticate using your OEM credentials:

```
emcli login -username=SYSMAN
```

   b. Run EMCLI commands to manage targets, jobs, and other aspects of your environment. For example:

```
emcli add_target -name="mydb" -type="oracle_database" -
properties="host=myhost,port=1521,sid=mydb"
```

Managing DBCS Database Deployments

Managing the Compute Node

Connecting to the Compute Node

SSH Access:

- Obtain the SSH private key that matches the public key specified when the database deployment was created.
- Use an SSH client to connect to the compute node. The default username is opc.

```
ssh -i /path/to/private-key opc@compute-node-ip
```

Switch to the Oracle User:

- After connecting as opc, switch to the oracle user to manage the database.

```
sudo su - oracle
```

Monitoring and Managing Performance

1) Check System Resources:
   - Use standard Linux commands to monitor CPU, memory, and disk usage.

```
top
vmstat
iostat
df -h
```

2) Oracle Database Performance:

- Use Oracle tools like SQL*Plus or SQL Developer to monitor database performance.
- Connect to the database using sqlplus and run performance queries.

```
sqlplus / as sysdba
SELECT * FROM v$instance;
SELECT * FROM v$session;
```

3) Enterprise Manager:

- If Enterprise Manager is set up, use it to monitor database performance and system metrics.

Managing Storage

1) Check Disk Usage:

- Monitor the disk usage of the file systems.

```
df -h
```

2) Manage ASM Storage:

- If using Oracle ASM, use asmcmd or SQL commands to manage storage.

```
asmcmd lsdg
sqlplus / as sysasm
SELECT name, total_mb, free_mb FROM v$asm_diskgroup;
```

3) Add Storage:

- Use the Oracle Cloud Infrastructure console to add storage to your compute node if needed.

Patching and Updates

1) Apply Patches:

- Use the Oracle Cloud Infrastructure console to apply patches to your compute node and database.

2) Update Operating System Packages:

- Regularly update the OS packages to ensure security and stability.

```
sudo yum update -y
```

Backup and Recovery

1) Configure Backups:

- Ensure that backups are configured and running correctly. Use the Oracle Cloud Infrastructure console or RMAN.

2) Perform Manual Backups:

- Run RMAN commands to perform manual backups if needed.

```
rman target /
BACKUP DATABASE;
```

3) Restore and Recover:

- Use RMAN to restore and recover the database in case of failure.

```
rman target /
RESTORE DATABASE;
RECOVER DATABASE;
```

Security Management

1) Manage SSH Keys:

- Add or remove SSH keys as needed for secure access to the compute node.

```
vi ~/.ssh/authorized_keys
```

2) Firewall and Network Security:

- Use the Oracle Cloud Infrastructure console to manage security lists and firewall rules.

Automation and Scripting

1) Create Scripts for Routine Tasks:

- Write shell scripts or use cron jobs to automate routine maintenance tasks.

  Example of a simple backup script:

```
#!/bin/bash
rman target / <<EOF
BACKUP DATABASE;
EXIT;
EOF
```

2) Schedule Jobs:

- Use cron to schedule scripts.

```
crontab -e
```

3) Add a cron job to run the backup script daily at 2 AM:

```
0 2 * * * /path/to/backup-script.sh
```

Scaling a Database Deployment

Scaling an Oracle Database deployment in the Oracle Cloud Infrastructure (OCI) involves adjusting the resources allocated to the database instance, such as increasing CPU, memory, or storage.

1. Scaling Compute Resources
Using the Oracle Cloud Infrastructure Console

a) Log in to the Oracle Cloud Infrastructure Console:

- Navigate to the Oracle Cloud Infrastructure console at https://cloud.oracle.com.

b) Go to the Database Section:

- In the navigation menu, select Oracle Database > Bare Metal, VM, and Exadata.

c) Select Your Database Deployment:

- Click on the database deployment you want to scale.

d) Edit Compute Shape:

- On the database details page, click on the Actions menu (three dots) in the upper right corner.
- Select Update from the drop-down menu.
- Under the Compute section, choose a new shape with more OCPUs and memory.
- Click Save Changes.

e) Confirm Changes:

- Review the changes and confirm. The database service will handle the scaling operation. This may involve a brief downtime depending on the type of scaling operation.

2. Scaling Storage
Using the Oracle Cloud Infrastructure Console

a) Navigate to the Database Section:

- In the OCI console, go to Oracle Database > Bare Metal, VM, and Exadata.

b) Select Your Database Deployment:

- Click on the database deployment you want to scale.

c) Add Storage:

- On the database details page, click on the Actions menu.
- Select Update Storage from the drop-down menu.
- Enter the additional storage amount you need and click Update Storage.

3. Scaling Database with Oracle Autonomous Database
If you're using Oracle Autonomous Database, the scaling process is simplified and can be done without downtime.

Using the Oracle Cloud Infrastructure Console

a) Navigate to Autonomous Database:

- In the OCI console, go to Oracle Database > Autonomous Database.

b) Select Your Autonomous Database:

- Click on the autonomous database you want to scale.

c) Scale Up/Down Resources:

- On the autonomous database details page, click Scale Up/Down.
- Adjust the number of OCPUs and the storage as needed.
- Click Save Changes.

4. Using OCI Command Line Interface (CLI)
You can also use the OCI CLI to scale your database deployment.

a) Install and Configure OCI CLI:

- If you haven't already, install and configure the OCI CLI following the instructions in the OCI CLI documentation.

b) Scale Compute Resources:

- Use the following command to change the shape of your database deployment.

```
oci db system update --db-system-id <DB_SYSTEM_OCID> --shape <NEW_SHAPE>
```

c) Scale Storage:

- Use the following command to increase the storage.

```
oci db system update --db-system-id <DB_SYSTEM_OCID> --data-storage-size-in-gbs <NEW_SIZE_IN_GB>
```

Patching DBCS

Pre-requisites

- ✓ Backup: Always take a full backup of your database before applying any patches.
- ✓ Review: Check the patch documentation for any prerequisites and the impact of the patch.

Using Oracle Cloud Infrastructure (OCI) Console

1) Log in to the Oracle Cloud Infrastructure Console:
- Navigate to the Oracle Cloud Infrastructure console at https://cloud.oracle.com.

2) Go to the Database Section:
- In the navigation menu, select Oracle Database > Bare Metal, VM, and Exadata.

3) Select Your Database Deployment:
- Click on the database deployment you want to patch.

4) Manage Patches:
- On the database details page, click on the Actions menu (three dots) in the upper right corner.
- Select Manage Patches.

5) Check for Available Patches:
- The system will list available patches for your database deployment.
- Review the list of available patches and select the appropriate one.

6) Apply Patch:
- Click Apply to start the patching process.
- Confirm the action and wait for the patching process to complete. This may involve downtime depending on the type of patch being applied.

Using Oracle Cloud Infrastructure (OCI) Command Line Interface (CLI)

1) Install and Configure OCI CLI:
- If you haven't already, install and configure the OCI CLI by following the instructions in the OCI CLI documentation.

2) List Available Patches:
- Use the following command to list available patches for your database system:

```
oci db patch list --db-system-id <DB_SYSTEM_OCID>
```

3) Apply Patch:
- Use the following command to apply a patch to your database system:

```
oci db patch apply --db-system-id <DB_SYSTEM_OCID> --patch-id <PATCH_ID>
```

- Replace <DB_SYSTEM_OCID> with the OCID of your database system and <PATCH_ID> with the ID of the patch you want to apply.

Post-Patching Steps

1) Verify the Patch:

- After the patching process is complete, verify that the patch has been successfully applied.

```
sqlplus / as sysdba
SELECT * FROM dba_registry_history ORDER BY action_time;
```

2) Test the Database:

- Perform any necessary testing to ensure that the database is functioning correctly after the patch.

3) Update Documentation:

- Document the patching process, including any issues encountered and how they were resolved.

Managing Database Instances

Working with Initialization Parameters

Oracle uses a parameter file to configure instance settings. There are two types of parameter files:

- SPFILE (Server Parameter File): Binary file that can be modified dynamically.
- PFILE (Parameter File): Text file that needs to be edited manually.

Viewing and Modifying Parameters

To view parameters:

```
SHOW PARAMETER <parameter_name>;
```

To modify parameters using SPFILE:

```
ALTER SYSTEM SET <parameter_name>=<value> SCOPE=SPFILE;
```

To modify parameters using PFILE, you need to edit the file manually and restart the database.

Examples:

Modifying Parameters Using SPFILE

The SPFILE allows you to dynamically change parameters without restarting the database. Here’s how you can modify parameters using the SPFILE:

Check Current Value of a Parameter:

```
SHOW PARAMETER db_cache_size;
```

Modify the Parameter Value:

```
ALTER SYSTEM SET db_cache_size=500M SCOPE=SPFILE;
```

Verify the Change:

```
SHOW PARAMETER db_cache_size;
```

Example:

```
-- Connect to SQL*Plus as SYSDBA
sqlplus / as sysdba

-- Check the current value of db_cache_size
SHOW PARAMETER db_cache_size;

-- Modify the parameter value
ALTER SYSTEM SET db_cache_size=500M SCOPE=SPFILE;

-- Verify the change
SHOW PARAMETER db_cache_size;
```

Modifying Parameters Using PFILE

The PFILE is a text file, and changes to it require a restart of the database. Here's how to modify parameters using the PFILE:

1) Locate the PFILE:
   The PFILE is usually located in the $ORACLE_HOME/dbs directory and named init<DB_NAME>.ora.

2) Edit the PFILE:
   Open the PFILE in a text editor and modify the desired parameter.
   ```
   db_cache_size=500M
   ```

3) Restart the Database:

   a. Shutdown the Database:
   sqlplus / as sysdba
   ```
   SHUTDOWN IMMEDIATE;
   ```

   b. Start the Database with the PFILE:
   ```
   STARTUP PFILE='$ORACLE_HOME/dbs/init<DB_NAME>.ora';
   ```

Example:

Edit the PFILE:

```
-- Open the PFILE (initORCL.ora) and modify the parameter
db_cache_size=500M
```

Restart the Database:

```
-- Connect to SQL*Plus as SYSDBA
sqlplus / as sysdba

-- Shutdown the database
SHUTDOWN IMMEDIATE;

-- Start the database with the modified PFILE
STARTUP PFILE='$ORACLE_HOME/dbs/initORCL.ora';
```

Starting the Oracle Database Instance

To start an Oracle instance, you typically use SQL*Plus:

Connect to SQL*Plus as SYSDBA:

```
sqlplus / as sysdba
```

Start the instance:

```
STARTUP;
```

Shutting Down an Oracle Database Instance

Shutdown the instance:

```
SHUTDOWN IMMEDIATE;
```

Other shutdown options include:

```
SHUTDOWN NORMAL;              – waits for users to disconnect.
SHUTDOWN TRANSACTIONAL;       – waits for transactions to complete.
SHUTDOWN ABORT;               – immediate shutdown, no waiting.
```

Opening and Closing PDBs

Opening a PDB

Connect to the Container Database (CDB) as a privileged user:

```
sqlplus / as sysdba
```

Open a PDB:

To open a specific PDB, you need to first alter the session to the root container and then open the PDB.

```
-- Alter the session to the root container
ALTER SESSION SET CONTAINER = CDB$ROOT;

-- Open the PDB
ALTER PLUGGABLE DATABASE pdb_name OPEN;
```

To open all PDBs:

```
ALTER PLUGGABLE DATABASE ALL OPEN;
```

Closing a PDB

Connect to the Container Database (CDB) as a privileged user:

```
sqlplus / as sysdba
```

Close a PDB:

To close a specific PDB, you need to first alter the session to the root container and then close the PDB.

```
-- Alter the session to the root container
ALTER SESSION SET CONTAINER = CDB$ROOT;

-- Close the PDB
ALTER PLUGGABLE DATABASE pdb_name CLOSE IMMEDIATE;
```

To close all PDBs:

```
ALTER PLUGGABLE DATABASE ALL CLOSE IMMEDIATE;
```

```
-- Query the status of all PDBs
SELECT pdb_name, open_mode FROM cdb_pdbs;
```

Alert Log

The alert log is a special trace file that records significant events for the Oracle database, including:

- ✓ Startup and shutdown information.
- ✓ Errors (e.g., ORA-00600).
- ✓ Warnings.
- ✓ Administrative operations (e.g., CREATE, ALTER, DROP).

Location of the Alert Log

The alert log is typically located in the alert directory of the Oracle Diagnostic Destination (DIAGNOSTIC_DEST). By default, it’s in the trace subdirectory:

```
$ORACLE_BASE/diag/rdbms/<DB_NAME>/<SID>/trace/alert_<SID>.log
```

Location on Windows

The alert log on Windows is located in the Oracle Diagnostic Destination directory, typically under the trace subdirectory. The path follows this structure:

```
%ORACLE_BASE%\diag\rdbms\<DB_NAME>\<SID>\trace\alert_<SID>.log
```

Example Path

If your Oracle base is C:\app\oracle and your database SID is ORCL, the path would be:

```
C:\app\oracle\diag\rdbms\orcl\orcl\trace\alert_orcl.log
```

Viewing the Alert Log

To view the alert log, you can use the following methods:

- Use the tail command in Unix/Linux:

  ```
  tail -f $ORACLE_BASE/diag/rdbms/<DB_NAME>/<SID>/trace/alert_<SID>.log
  ```

- Open the file directly in a text editor.

Trace Files

Trace files are generated by Oracle processes to provide detailed diagnostic information. They are used for debugging and diagnosing problems. Each Oracle process (e.g., DBWR, LGWR) can generate trace files.

Location of Trace Files

Trace files are located in the same directory as the alert log:

```
$ORACLE_BASE/diag/rdbms/<DB_NAME>/<SID>/trace/
```

Location on Windows

Trace files are also located in the Oracle Diagnostic Destination directory, under the trace subdirectory. The path follows this structure:

```
%ORACLE_BASE%\diag\rdbms\<DB_NAME>\<SID>\trace\<trace_filename>.trc
```

Example Path

For a database with SID ORCL:

```
C:\app\oracle\diag\rdbms\orcl\orcl\trace\<trace_filename>.trc
```

Types of Trace Files

- User Trace Files: Generated by user sessions.
- Background Trace Files: Generated by background processes like DBWR, LGWR, etc.

Viewing Trace Files

Similar to the alert log, you can view trace files using the tail command or by opening them in a text editor:

```
tail -f $ORACLE_BASE/diag/rdbms/<DB_NAME>/<SID>/trace/<trace_filename>.trc
```

DDL Log File

DDL log files record DDL statements executed in the database. This can be useful for auditing and monitoring schema changes.

Enabling DDL Logging

DDL logging is not enabled by default. To enable it, you need to set the following parameters in the sqlnet.ora file or dynamically:

Add the following line to the sqlnet.ora file:

```
DIAG_ADR_ENABLED = OFF
```

Enable DDL Logging Dynamically:

```
ALTER SYSTEM SET ENABLE_DDL_LOGGING=TRUE SCOPE=BOTH;
```

Location of DDL Log Files

The DDL log file is located in the log directory of the Oracle Diagnostic Destination:

```
$ORACLE_BASE/diag/rdbms/<DB_NAME>/<SID>/log/ddl.log
```

Location on Windows

DDL log files, if enabled, are located in the log directory of the Oracle Diagnostic Destination:

```
%ORACLE_BASE%\diag\rdbms\<DB_NAME>\<SID>\log\ddl.log
```

Example Path

For a database with SID ORCL:

```
C:\app\oracle\diag\rdbms\orcl\orcl\log\ddl.log
```

Viewing the DDL Log File
You can view the DDL log file using the tail command or by opening it in a text editor:
tail -f $ORACLE_BASE/diag/rdbms/<DB_NAME>/<SID>/log/ddl.log

Oracle Net Services

Oracle Net Services

Oracle Net Services is a suite of networking components that enable Oracle database clients and servers to communicate with each other. It supports the Transparent Network Substrate (TNS) protocol, which allows for secure and efficient data transport.

Key Components of Oracle Net Services

- ✓ Listener
- ✓ Oracle Net Configuration Files
- ✓ Oracle Net Manager
- ✓ Oracle Net Configuration Assistant

Oracle Net Listener

The Oracle Net Listener is a process that runs on the database server and handles incoming client connection requests. It listens for connection requests on a specified port (default is 1521) and forwards them to the appropriate database instance.

Configuring the Listener
The listener configuration file, listener.ora, is located in the network/admin directory under the Oracle Home directory.

Example listener.ora file:

```
LISTENER =
 (DESCRIPTION_LIST =
  (DESCRIPTION =
   (ADDRESS = (PROTOCOL = TCP)(HOST = myserver)(PORT = 1521))
  )
 )

SID_LIST_LISTENER =
 (SID_LIST =
  (SID_DESC =
   (GLOBAL_DBNAME = orcl)
   (ORACLE_HOME = /u01/app/oracle/product/19.3.0/dbhome_1)
   (SID_NAME = orcl)
  )
 )
```

Starting and Stopping the Listener

You can start and stop the listener using the lsnrctl command:

```
lsnrctl start
lsnrctl stop
```

Establishing Oracle Network Connections

Oracle Net Configuration Files

- **tnsnames.ora**
  The tnsnames.ora file contains network service names mapped to connect descriptors. This file is used by clients to identify database servers.

Example tnsnames.ora file:

```
ORCL =
 (DESCRIPTION =
  (ADDRESS_LIST =
   (ADDRESS = (PROTOCOL = TCP)(HOST = myserver)(PORT = 1521))
  )
  (CONNECT_DATA =
   (SERVICE_NAME = orcl)
  )
 )
```

- **sqlnet.ora**
  The sqlnet.ora file contains client-side network configuration parameters. It can specify the default domain, logging options, and other settings.

Example sqlnet.ora file:

```
SQLNET.AUTHENTICATION_SERVICES = (NONE)
NAMES.DIRECTORY_PATH = (TNSNAMES, EZCONNECT)
```

Oracle Net Manager

Oracle Net Manager is a GUI tool that helps you configure and manage Oracle Net Services. You can use it to create and edit listener.ora, tnsnames.ora, and sqlnet.ora files.

To start Oracle Net Manager:

```
netmgr
```

Oracle Net Configuration Assistant (NETCA)

NETCA is a wizard-based tool that simplifies the configuration of network components. It can create and configure listeners, naming methods, net service names, and directory usage.

To start NETCA:

```
netca
```

Connecting to an Oracle Database

Example Workflow

Configure the Listener (listener.ora):

```
LISTENER =
 (DESCRIPTION_LIST =
  (DESCRIPTION =
   (ADDRESS = (PROTOCOL = TCP)(HOST = myserver)(PORT = 1521))
  )
 )

SID_LIST_LISTENER =
 (SID_LIST =
  (SID_DESC =
   (GLOBAL_DBNAME = orcl)
   (ORACLE_HOME = /u01/app/oracle/product/19.3.0/dbhome_1)
   (SID_NAME = orcl)
  )
 )
```

Start the Listener:

```
lsnrctl start
```

Configure the Client (tnsnames.ora):

```
ORCL =
 (DESCRIPTION =
  (ADDRESS_LIST =
   (ADDRESS = (PROTOCOL = TCP)(HOST = myserver)(PORT = 1521))
  )
  (CONNECT_DATA =
   (SERVICE_NAME = orcl)
  )
 )
```

Configure the Client (sqlnet.ora):

```
SQLNET.AUTHENTICATION_SERVICES = (NONE)
NAMES.DIRECTORY_PATH = (TNSNAMES, EZCONNECT)
```

Connect to the Database:

```
sqlplus user/password@ORCL
```

Troubleshooting Tips

Check Listener Status:

```
lsnrctl status
```

Check Connectivity:

```
tnsping ORCL
```

Enable Logging:
Add the following to sqlnet.ora:

```
TRACE_LEVEL_CLIENT = 16
```

Review Log Files:
Listener logs can be found in the log directory under the Oracle Home directory:

```
$ORACLE_BASE/diag/tnslsnr/<hostname>/listener/trace/listener.log
```

User Security

Database User Accounts

1. Creating and Managing User Accounts

Creating a User
To create a new user account, you use the CREATE USER statement. You need to specify the username and password, and optionally other settings like default tablespace, temporary tablespace, and profile.

```
CREATE USER username IDENTIFIED BY password
  DEFAULT TABLESPACE tablespace_name
  TEMPORARY TABLESPACE temp_tablespace_name
  PROFILE profile_name;
```

Example:

```
CREATE USER john IDENTIFIED BY securepassword
  DEFAULT TABLESPACE users
  TEMPORARY TABLESPACE temp
  PROFILE default;
```

Viewing All Users
The DBA_USERS view contains information about all users in the database. To query this view, you need DBA privileges.

```
SELECT username, account_status, default_tablespace, created
FROM dba_users;
```

Viewing Users Accessible to the Current User
The ALL_USERS view shows information about all users that the current user has access to.

```
SELECT username, created
FROM all_users;
```

Viewing the Current User's Information
The USER_USERS view provides information about the current user.

```
SELECT username, default_tablespace, created
FROM user_users;
```

Detailed Examples

Example 1: Viewing All Users (Using DBA_USERS)

```
-- Connect as a user with DBA privileges
sqlplus / as sysdba

-- Query the DBA_USERS view
SELECT username, account_status, default_tablespace, created
FROM dba_users
ORDER BY username;
```

Example 2: Viewing Users Accessible to the Current User (Using ALL_USERS)

```
-- Connect as any user
sqlplus username/password@database

-- Query the ALL_USERS view
SELECT username, created
FROM all_users
ORDER BY username;
```

Example 3: Viewing Current User's Information (Using USER_USERS)

```
-- Connect as the current user
sqlplus username/password@database

-- Query the USER_USERS view
SELECT username, default_tablespace, created
FROM user_users;
```

Example Outputs

Output for DBA_USERS Query:

```
USERNAME                      ACCOUNT_STATUS      DEFAULT_TABLESPACE    CREATED
----------------------------- ------------------- -------------------- -------------------
ANONYMOUS                     EXPIRED & LOCKED    USERS                 01-JAN-2020 12:00:00
APEX_040000                   EXPIRED & LOCKED    SYSAUX                01-JAN-2020 12:00:00
HR                            OPEN                USERS                 01-JAN-2020 12:00:00
SCOTT                         OPEN                USERS                 01-JAN-2020 12:00:00
SYS                           OPEN                SYSTEM                01-JAN-2020 12:00:00
SYSTEM                        OPEN                SYSTEM                01-JAN-2020 12:00:00
```

Output for ALL_USERS Query:

```
USERNAME                      CREATED
----------------------------- -------------------
HR                            01-JAN-2020 12:00:00
SCOTT                         01-JAN-2020 12:00:00
SYS                           01-JAN-2020 12:00:00
SYSTEM                        01-JAN-2020 12:00:00
```

Output for USER_USERS Query:

```
USERNAME                DEFAULT_TABLESPACE          CREATED
------------------------------ -------------------- -------------------
HR                      USERS                       01-JAN-2020 12:00:00
```

Assigning Roles and Privileges
After creating a user, you need to grant appropriate roles and privileges to allow the user to perform required tasks.

Granting Roles
Roles are collections of privileges that can be granted to users.

```
GRANT role_name TO username;
```

Common Predefined Roles

CONNECT
Grants basic privileges to connect to the database.
Privileges Included:
CREATE SESSION

RESOURCE
Grants privileges to create and manage schema objects.
Privileges Included:
CREATE TABLE
CREATE VIEW
CREATE SEQUENCE
CREATE PROCEDURE
CREATE TRIGGER

DBA
Grants all system privileges, including the ability to administer the database.
Privileges Included:
All system privileges, such as CREATE USER, DROP USER, CREATE ANY TABLE, DROP ANY TABLE, etc.

IMP_FULL_DATABASE
Grants privileges to perform full database import operations.
Privileges Included:
EXECUTE_CATALOG_ROLE
SELECT ANY TABLE
INSERT ANY TABLE
UPDATE ANY TABLE
DELETE ANY TABLE

EXP_FULL_DATABASE
Grants privileges to perform full database export operations.
Privileges Included:
EXECUTE_CATALOG_ROLE
SELECT ANY TABLE

SELECT_CATALOG_ROLE
Grants privileges to select data from any data dictionary view.
Privileges Included:
SELECT ANY DICTIONARY

EXECUTE_CATALOG_ROLE
Grants execute privileges on various PL/SQL packages.
Privileges Included:
EXECUTE privileges on packages such as DBMS_RLS, DBMS_CRYPTO, DBMS_AQ, DBMS_FGA, etc.

Example:

```
GRANT CONNECT, RESOURCE TO john;
```

Granting System Privileges
System privileges allow users to perform specific actions on the database.

```
GRANT privilege TO username;
```

Example:

```
GRANT CREATE SESSION TO john;
GRANT CREATE TABLE TO john;
```

Granting Object Privileges
Object privileges allow users to perform actions on specific schema objects like tables, views, and procedures.

```
GRANT privilege ON object TO username;
```

Example:

```
GRANT SELECT, INSERT ON employees TO john;
```

User Profiles
Profiles help manage and control database resource usage for users. You can set limits on various resources like CPU time, logical reads, and the number of sessions.

Creating a Profile

```
CREATE PROFILE profile_name LIMIT
 SESSIONS_PER_USER 3
 CPU_PER_SESSION 10000
 CONNECT_TIME 60;
```

Example:

```
CREATE PROFILE limited_user LIMIT
 SESSIONS_PER_USER 2
 CONNECT_TIME 30;
```

Assigning a Profile to a User

```
ALTER USER username PROFILE profile_name;
```

Example:

```
ALTER USER john PROFILE limited_user;
```

Password Management

Oracle provides features for enforcing password complexity and expiration policies.

Enforcing Password Policies

Password policies are defined within profiles. You can enforce password complexity, expiration, and reuse.

```
ALTER PROFILE DEFAULT LIMIT
  PASSWORD_LIFE_TIME 30
  PASSWORD_REUSE_TIME 180
  PASSWORD_REUSE_MAX 10
  PASSWORD_LOCK_TIME 1
  PASSWORD_GRACE_TIME 7
  FAILED_LOGIN_ATTEMPTS 3;
```

Auditing

Auditing helps track and log user activities for security and compliance purposes.

Enabling Auditing

```
AUDIT CREATE SESSION;
AUDIT SELECT TABLE, INSERT TABLE, UPDATE TABLE, DELETE TABLE BY john BY ACCESS;
```

Example Workflow

```
--Create a User:
CREATE USER alice IDENTIFIED BY strongpassword;

--Grant Roles and Privileges:
GRANT CONNECT, RESOURCE TO alice;
GRANT CREATE SESSION TO alice;
GRANT SELECT, INSERT ON employees TO alice;

--Create a Profile and Assign It:
CREATE PROFILE developer_profile LIMIT
  SESSIONS_PER_USER 5
  CPU_PER_SESSION 10000
  CONNECT_TIME 120;

ALTER USER alice PROFILE developer_profile;

--Set Password Policies:
ALTER PROFILE developer_profile LIMIT
  PASSWORD_LIFE_TIME 45
  PASSWORD_REUSE_TIME 180
  PASSWORD_REUSE_MAX 5
  PASSWORD_LOCK_TIME 1
  PASSWORD_GRACE_TIME 10
  FAILED_LOGIN_ATTEMPTS 5;
```

```
--Enable Auditing:
AUDIT CREATE SESSION BY alice BY ACCESS;
AUDIT SELECT TABLE, INSERT TABLE, UPDATE TABLE, DELETE TABLE BY alice BY ACCESS;
```

Useful SQL Commands

Changing a User's Password:

```
ALTER USER alice IDENTIFIED BY newpassword;
```

Locking and Unlocking a User Account:

```
ALTER USER alice ACCOUNT LOCK;
ALTER USER alice ACCOUNT UNLOCK;
```

Dropping a User:

```
DROP USER alice CASCADE;
```

Oracle-Supplied Administrator Accounts

1. SYS

Description:

- The SYS user is the most powerful account in an Oracle database. It owns all the base tables and views for the database's data dictionary.
- The SYS account is created automatically when a database is created.

Privileges:

- The SYS user has the SYSDBA privilege, which grants all system privileges with administrative options.
- SYS owns the data dictionary tables, which store metadata about the database.

Usage:

- This account is used for all administrative tasks, including startup, shutdown, and recovery of the database.
- The SYS user should not be used for routine tasks; it's meant for high-level administrative functions.

Example:

```
sqlplus / as sysdba
```

2. SYSTEM

Description:

- The SYSTEM user is also created automatically when the database is created.
- It is used for creating and managing tables, views, and other schema objects.

Privileges:

- The SYSTEM user has a set of administrative privileges but does not have the SYSDBA privilege by default.
- It can create additional database users, roles, and profiles.

Usage:

- This account is used for routine administrative tasks and should be used instead of SYS for most administrative work.

Example:

```
sqlplus system/password
```

3. SYSBACKUP

Description:

- The SYSBACKUP user is designed specifically for performing backup and recovery operations.
- Introduced in Oracle Database 12c.

Privileges:

- The SYSBACKUP user has the SYSBACKUP privilege, which allows it to perform backup and recovery tasks without having full SYSDBA privileges.

Usage:

- This account is used for RMAN (Recovery Manager) operations and other backup tasks.

Example:

```
sqlplus / as sysbackup
```

4. SYSDG

Description:

- The SYSDG user is used for Data Guard operations.
- Introduced in Oracle Database 12c.

Privileges:

- The SYSDG user has the SYSDG privilege, which allows it to perform Data Guard-related tasks.

Usage:

- This account is used for managing Data Guard configurations.

Example:

```
sqlplus / as sysdg
```

5. SYSKM

Description:

- The SYSKM user is used for encryption key management.
- Introduced in Oracle Database 12c.

Privileges:

- The SYSKM user has the SYSKM privilege, which allows it to perform encryption key management tasks.

Usage:

- This account is used for managing Transparent Data Encryption (TDE) and other encryption-related tasks.

Example:

```
sqlplus / as syskm
```

6. SYSMAN

Description:

- The SYSMAN user is used by Oracle Enterprise Manager (OEM) for database management.

Privileges:

- The SYSMAN user has the necessary privileges to manage the database using Oracle Enterprise Manager.

Usage:

- This account is used internally by Oracle Enterprise Manager and should not be used for manual database administration tasks.

Example:

```
sqlplus sysman/password
```

Creating Oracle Database Users in a Multitenant Environment

- Common Users: Created in the root container and have access to all PDBs.
- Local Users: Created in specific PDBs and have access only to that PDB.
- Granting and Revoking Privileges: Use GRANT and REVOKE statements to assign and remove system and object privileges.
- Switching Containers: Use ALTER SESSION SET CONTAINER=pdb_name to switch between containers.

Creating Common Users

Common users are created in the root container (CDB$ROOT) and have access to all PDBs. They have the same identity across the entire multitenant environment.

Example:

```
-- Connect to the root container as SYSDBA
sqlplus / as sysdba

-- Create a common user
CREATE USER c##admin_user IDENTIFIED BY AdminPassword1 CONTAINER=ALL;

-- Grant system privileges to the common user
GRANT CONNECT, RESOURCE TO c##admin_user CONTAINER=ALL;
```

Creating Local Users

Local users are specific to individual PDBs and do not have access to other PDBs.

Example:

```
-- Connect to the root container as SYSDBA
sqlplus / as sysdba

-- Switch to the PDB
ALTER SESSION SET CONTAINER=pdb1;

-- Create a local user in the PDB
CREATE USER pdb_user IDENTIFIED BY PdbPassword1;
```

```
-- Grant system privileges to the local user
GRANT CONNECT, RESOURCE TO pdb_user;
```

Granting and Revoking Privileges

System Privileges
System privileges allow users to perform specific actions on the database.

Granting System Privileges

```
GRANT system_privilege TO username CONTAINER=ALL; -- For common users
GRANT system_privilege TO username; -- For local users
```

Example:

```
GRANT CREATE SESSION TO c##common_user CONTAINER=ALL;
GRANT CREATE TABLE TO pdb_user;
```

Revoking System Privileges

```
REVOKE system_privilege FROM username CONTAINER=ALL; -- For common users
REVOKE system_privilege FROM username; -- For local users
```

Example:

```
REVOKE CREATE SESSION FROM c##common_user CONTAINER=ALL;
REVOKE CREATE TABLE FROM pdb_user;
```

Object Privileges
Object privileges allow users to perform actions on specific schema objects like tables, views, and procedures.

Granting Object Privileges

```
GRANT object_privilege ON object TO username;
```

Example:

```
GRANT SELECT, INSERT ON employees TO pdb_user;
```

Revoking Object Privileges

```
REVOKE object_privilege ON object FROM username;
```

Example:

```
REVOKE SELECT, INSERT ON employees FROM pdb_user;
```

Managing Users in a Multitenant Environment
Listing Common Users:

```
SELECT username, common FROM cdb_users WHERE common = 'YES';
```

Listing Local Users in a PDB:

```
SELECT username FROM dba_users;
```

*Ensure you are connected to the specific PDB when running this query.

Switching Between Containers:

```
ALTER SESSION SET CONTAINER=cdb$root;
ALTER SESSION SET CONTAINER=pdb_name;
```

Example Workflow

```
--Connect to the CDB as SYSDBA:
sqlplus / as sysdba

--Create a Common User:
CREATE USER c##common_user IDENTIFIED BY commonPass CONTAINER=ALL;
GRANT CONNECT, RESOURCE TO c##common_user CONTAINER=ALL;

--Create a Local User in a PDB:
ALTER SESSION SET CONTAINER=pdb1;
CREATE USER pdb_user IDENTIFIED BY pdbPass;
GRANT CONNECT, RESOURCE TO pdb_user;

--Grant and Revoke Privileges:
-- Grant system privileges
GRANT CREATE SESSION TO c##common_user CONTAINER=ALL;
GRANT CREATE TABLE TO pdb_user;

-- Grant object privileges
GRANT SELECT, INSERT ON employees TO pdb_user;

-- Revoke system privileges
REVOKE CREATE SESSION FROM c##common_user CONTAINER=ALL;
REVOKE CREATE TABLE FROM pdb_user;

-- Revoke object privileges
REVOKE SELECT, INSERT ON employees FROM pdb_user;

--Verify Common and Local Users:
-- List common users
SELECT username, common FROM cdb_users WHERE common = 'YES';

-- List local users in the PDB
ALTER SESSION SET CONTAINER=pdb1;
SELECT username FROM dba_users;
```

Creating PDBs

Pluggable Databases (PDBs) are a key feature of Oracle's Multitenant architecture, introduced in Oracle Database 12c. They provide a way to consolidate multiple databases into a single Container Database (CDB), allowing for easier management and more efficient use of resources.

- Container Database (CDB): The root container that holds zero, one, or more PDBs. It includes the root container (CDB$ROOT), the seed PDB (PDB$SEED), and any user-created PDBs.
- Pluggable Database (PDB): A portable collection of schemas, schema objects, and non-schema objects that appears to an application as a separate database. Each PDB can have its own set of users, data, and configuration.
- Seed PDB (PDB$SEED): A template PDB that is used for creating new PDBs. It is read-only and provided by Oracle.

Benefits of PDBs

- Consolidation: Multiple PDBs can be consolidated into a single CDB, reducing the overhead of managing multiple database instances.
- Resource Management: Resources such as CPU and memory can be managed more efficiently across multiple PDBs within a single CDB.
- Security and Isolation: Each PDB is isolated from the others, ensuring that operations in one PDB do not affect the others.
- Portability: PDBs can be easily moved or cloned from one CDB to another, facilitating easier database deployment and management.
- Simplified Patching and Upgrades: Applying patches and upgrades to a single CDB can cover all PDBs, simplifying maintenance tasks.

Creating a New PDB from PDB$SEED

A new PDB can be created using the CREATE PLUGGABLE DATABASE statement. The PDB can be created from the seed PDB, from an existing PDB, or by cloning another PDB.

Showing PDB list

Using SHOW PDBS

```
-- Connect as a privileged user
sqlplus / as sysdba
```

```
-- Show all PDBs
SHOW PDBS;
```

Output:

```
  CON_ID     CON_NAME                  OPEN MODE  RESTRICTED
---------- ----------------------------- ---------- ----------
     2       PDB$SEED                  READ ONLY  NO
     3       PDB1                      READ WRITE NO
     4       PDB2                      MOUNTED    NO
```

Querying CDB_PDBS

```
-- Connect as a privileged user
sqlplus / as sysdba
```

```
-- Query to list all PDBs
SELECT pdb_id, pdb_name, status, open_mode FROM cdb_pdbs;
```

Output:

```
 PDB_ID       PDB_NAME              STATUS        OPEN_MODE
-------- ----------------------------- ----------- ----------
    2         PDB$SEED              NORMAL        READ ONLY
    3         PDB1                  NORMAL        READ WRITE
    4         PDB2                  NORMAL        MOUNTED
```

Additional Information from CDB_PDBS

The CDB_PDBS view provides more detailed information that can be customized in queries:

```
SELECT pdb_id, pdb_name, status, open_mode, creation_scn, con_id FROM cdb_pdbs;
```

Creating a PDB from the Seed PDB

```
CREATE PLUGGABLE DATABASE pdb1 ADMIN USER pdb_admin IDENTIFIED BY password
  FILE_NAME_CONVERT = ('/u01/app/oracle/oradata/cdb1/pdbseed', '/u01/app/oracle/oradata/cdb1/pdb1');
```

Opening a PDB

```
ALTER PLUGGABLE DATABASE pdb1 OPEN;
```

Closing a PDB

```
ALTER PLUGGABLE DATABASE pdb1 CLOSE IMMEDIATE;
```

Opening All PDBs

```
ALTER PLUGGABLE DATABASE ALL OPEN;
```

Closing All PDBs

```
ALTER PLUGGABLE DATABASE ALL CLOSE IMMEDIATE;
```

Switching Between Containers

To perform operations on a specific PDB, you need to switch your session to that PDB.

Switching to a PDB

```
ALTER SESSION SET CONTAINER = pdb1;
```

Switching Back to the Root Container

```
ALTER SESSION SET CONTAINER = CDB$ROOT;
```

Cloning PDBs

Prerequisites

- ✓ Ensure the source PDB is in read-only mode if cloning within the same CDB.
- ✓ Adequate storage must be available for the new PDB.
- ✓ Appropriate privileges are required (e.g., SYSDBA).

Steps to Clone a PDB

1. Cloning Within the Same CDB

Example for Cloning Within the Same CDB

```
-- Connect to the CDB as a privileged user
sqlplus / as sysdba

-- Set the source PDB to read-only mode
ALTER PLUGGABLE DATABASE source_pdb CLOSE IMMEDIATE;
ALTER PLUGGABLE DATABASE source_pdb OPEN READ ONLY;

-- Create the new PDB by cloning the source PDB
CREATE PLUGGABLE DATABASE target_pdb FROM source_pdb
  FILE_NAME_CONVERT = ('/u01/app/oracle/oradata/cdb1/source_pdb/',
'/u01/app/oracle/oradata/cdb1/target_pdb/');

-- Open the new PDB
ALTER PLUGGABLE DATABASE target_pdb OPEN;

-- Optionally, set the source PDB back to read-write mode
ALTER PLUGGABLE DATABASE source_pdb CLOSE IMMEDIATE;
ALTER PLUGGABLE DATABASE source_pdb OPEN READ WRITE;
```

2. Cloning from a Different CDB

Example for Cloning from a Different CDB

```
-- Connect to the target CDB as a privileged user
sqlplus / as sysdba

-- Create a database link in the target CDB
CREATE DATABASE LINK source_cdb_link CONNECT TO source_user IDENTIFIED BY source_password USING
'source_cdb_tns';

-- Create the new PDB by cloning the source PDB from the source CDB
CREATE PLUGGABLE DATABASE target_pdb FROM source_pdb@source_cdb_link
  FILE_NAME_CONVERT = ('/u01/app/oracle/oradata/source_cdb/source_pdb/',
'/u01/app/oracle/oradata/target_cdb/target_pdb/');

-- Open the new PDB
ALTER PLUGGABLE DATABASE target_pdb OPEN;
```

Additional Options

Using Snapshot Copy

If storage supports it, you can use a snapshot copy to clone the PDB, which is faster and more storage-efficient:

```
CREATE PLUGGABLE DATABASE target_pdb FROM source_pdb SNAPSHOT COPY
  FILE_NAME_CONVERT = ('/path/to/source_pdb/', '/path/to/target_pdb/');
```

Cloning with the NO DATA Option

To create a new PDB structure without data:

```
CREATE PLUGGABLE DATABASE target_pdb FROM source_pdb
  NO DATA
  FILE_NAME_CONVERT = ('/path/to/source_pdb/', '/path/to/target_pdb/');
```

Unplugging and Plugging in PDBs

Unplugging a PDB

Unplugging a PDB involves generating an XML file that contains metadata about the PDB. This file is used to plug the PDB into a new CDB.

```
-- Connect to the CDB as SYSDBA
sqlplus / as sysdba

-- Close the PDB
ALTER PLUGGABLE DATABASE pdb1 CLOSE IMMEDIATE;

-- Unplug the PDB
ALTER PLUGGABLE DATABASE pdb1 UNPLUG INTO '/u01/app/oracle/oradata/cdb1/pdb1.xml';

-- Drop the PDB (optional)
DROP PLUGGABLE DATABASE pdb1 KEEP DATAFILES;
```

Plugging in a PDB

Plugging in a PDB involves using the XML file generated during the unplugging process to integrate the PDB into a new or existing CDB.

Example

```
-- Connect to the target CDB as SYSDBA
sqlplus / as sysdba

-- Create the new PDB from the XML file
CREATE PLUGGABLE DATABASE pdb1 USING '/u01/app/oracle/oradata/cdb1/pdb1.xml' COPY
FILE_NAME_CONVERT = ('/u01/app/oracle/oradata/old_cdb/', '/u01/app/oracle/oradata/new_cdb/');

-- Open the new PDB
ALTER PLUGGABLE DATABASE pdb1 OPEN;
```

Additional Options

1. Using NOCOPY Option:
If the file locations are the same and you do not want to copy the files:

```
CREATE PLUGGABLE DATABASE pdb1 USING '/u01/app/oracle/oradata/cdb1/pdb1.xml' NOCOPY
FILE_NAME_CONVERT = ('/u01/app/oracle/oradata/old_cdb/', '/u01/app/oracle/oradata/new_cdb/');
```

2. Opening PDBs in Read-Only Mode:
If you need to open the PDB in read-only mode for verification:

```
ALTER PLUGGABLE DATABASE pdb1 OPEN READ ONLY;
```

3. Checking PDB Compatibility:
Before plugging in a PDB, you can check compatibility:

```
DBMS_PDB.CHECK_PLUG_COMPATIBILITY(
  pdb_descr_file => '/u01/app/oracle/oradata/cdb1/pdb1.xml',
  pdb_name => 'PDB1');
```

Creating and Managing Tablespaces

Tablespaces in Oracle Database are logical storage units that group related logical structures together. They play a crucial role in database storage management by providing a way to manage and allocate space for database objects such as tables and indexes.

Key Concepts

- ✓ Logical Storage Unit: Tablespaces group logical storage structures like segments, which consist of extents. Extents are made up of data blocks.
- ✓ Data Files: Each tablespace consists of one or more physical data files on disk that store the actual data.
- ✓ Default Tablespace: When creating users or objects, if a specific tablespace is not specified, they are created in the default tablespace.

Types of Tablespaces

Permanent Tablespaces: Store persistent data like tables and indexes.
Example: USERS, SYSTEM, SYSAUX

Temporary Tablespaces: Store temporary data that is generated during SQL operations such as sorts.
Example: TEMP

Undo Tablespaces: Store undo information for uncommitted transactions.
Example: UNDOTBS1

Common Tablespaces

- SYSTEM: Contains the data dictionary and essential system data.
- SYSAUX: Auxiliary tablespace to offload data from the SYSTEM tablespace.
- USERS: Default tablespace for user data.
- TEMP: Temporary tablespace for sorting operations.
- UNDO: Stores undo information for transactions.

Creating Tablespaces

Creating a Tablespace

You can create a tablespace using the CREATE TABLESPACE statement.

Example:

```
CREATE TABLESPACE example_tbs
DATAFILE '/u01/app/oracle/oradata/example_tbs01.dbf' SIZE 50M
AUTOEXTEND ON NEXT 10M MAXSIZE UNLIMITED;
```

Creating a Temporary Tablespace

```
CREATE TEMPORARY TABLESPACE temp_tbs
TEMPFILE '/u01/app/oracle/oradata/temp_tbs01.dbf' SIZE 50M
AUTOEXTEND ON NEXT 10M MAXSIZE UNLIMITED;
```

Creating an Undo Tablespace

```
CREATE UNDO TABLESPACE undo_tbs
DATAFILE '/u01/app/oracle/oradata/undo_tbs01.dbf' SIZE 100M
AUTOEXTEND ON NEXT 10M MAXSIZE UNLIMITED;
```

Verify the Tablespace Creation:

```
SELECT tablespace_name, status, contents, extent_management, allocation_type
FROM dba_tablespaces
WHERE tablespace_name = 'EXAMPLE_TBS';
```

Output:

```
TABLESPACE_NAME   STATUS        CONTENTS     EXTENT_MANAGEMENT       ALLOCATION_TYPE
--------------- --------- -------- ----------------- ---------------
EXAMPLE_TBS         ONLINE        PERMANENT   LOCAL                   SYSTEM
```

Altering and Dropping Tablespaces

Add a Data File to a Tablespace:

```
ALTER TABLESPACE example_tbs
ADD DATAFILE '/u01/app/oracle/oradata/example_tbs02.dbf' SIZE 50M
AUTOEXTEND ON NEXT 10M MAXSIZE 500M;
```

Enable Autoextend for a Data File:

```
ALTER DATABASE DATAFILE '/u01/app/oracle/oradata/example_tbs01.dbf'
AUTOEXTEND ON NEXT 10M MAXSIZE 1G;
```

Resize a Data File:

```
ALTER DATABASE DATAFILE '/u01/app/oracle/oradata/example_tbs01.dbf'
RESIZE 200M;
```

Take a Tablespace Offline:

```
ALTER TABLESPACE example_tbs OFFLINE;
```

Bring a Tablespace Online:

```
ALTER TABLESPACE example_tbs ONLINE;
```

Drop a Tablespace (Including Data Files):

```
DROP TABLESPACE example_tbs INCLUDING CONTENTS AND DATAFILES;
```

Drop a Tablespace (Keep Data Files):

```
DROP TABLESPACE example_tbs KEEP DATAFILES;
```

Viewing Tablespace Information

Querying Tablespace Information:

```
SELECT tablespace_name, status, contents, extent_management, allocation_type
FROM dba_tablespaces;
```

Tablespace Encryption by Default in DBCS

Oracle Database Cloud Service (DBCS) offers enhanced security features, including tablespace encryption by default. This feature ensures that all data stored in the tablespaces is encrypted, providing an additional layer of security to protect sensitive information.

Key Concepts of Tablespace Encryption

- ✓ Encryption Algorithms: Oracle supports various encryption algorithms such as AES128, AES192, and AES256. AES256 is the most secure and commonly used.
- ✓ Transparent Data Encryption (TDE): TDE encrypts data at rest, ensuring that data is automatically encrypted when written to the disk and decrypted when read from the disk.

When you create a new tablespace in DBCS, it is automatically encrypted using the default encryption algorithm (AES256) unless specified otherwise.

Example: Creating a Default Encrypted Tablespace

```
CREATE TABLESPACE secure_tbs
DATAFILE '/u01/app/oracle/oradata/secure_tbs01.dbf' SIZE 100M;
```

*In the above example, secure_tbs is automatically encrypted because tablespace encryption is enabled by default.

Example: Specifying Encryption Algorithm

If you want to specify a different encryption algorithm, you can do so using the ENCRYPTION clause:

```
CREATE TABLESPACE secure_tbs
DATAFILE '/u01/app/oracle/oradata/secure_tbs01.dbf' SIZE 100M
ENCRYPTION USING 'AES128';
```

Managing Encrypted Tablespaces

Viewing Encrypted Tablespaces

```
SELECT tablespace_name, encryptedts
FROM v$encrypted_tablespaces;
```

Example Output:

```
TABLESPACE_NAME          ENCRYPTEDTS
------------------------- -----------
SECURE_TBS               YES
USERS                    NO
```

Encrypting Existing Tablespaces

To encrypt an existing tablespace, you can use the ALTER TABLESPACE command. However, note that this operation requires moving the data to an encrypted tablespace, as direct encryption of an existing tablespace is not supported.

Example: Encrypting an Existing Tablespace

Create an Encrypted Tablespace:

```
CREATE TABLESPACE new_secure_tbs
DATAFILE '/u01/app/oracle/oradata/new_secure_tbs01.dbf' SIZE 100M;
```

Move the Data to the Encrypted Tablespace:

```
ALTER TABLE your_table MOVE TABLESPACE new_secure_tbs;
```

Drop the Original Tablespace (if needed):

```
DROP TABLESPACE old_tbs INCLUDING CONTENTS AND DATAFILES;
```

Key Management

Oracle TDE uses the Oracle Wallet to store encryption keys. In DBCS, the Oracle Wallet is managed automatically, but you can perform operations on the wallet as needed.

Example: Opening the Wallet

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY wallet_password;
```

Example: Closing the Wallet

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE;
```

Managing Storage Space

Space Management Features

- ✓ ASSM: Automatic management of segment space using bitmaps.
- ✓ LMT: Extent management at the tablespace level using bitmaps.
- ✓ ASM: Simplified and high-performance storage management.
- ✓ Extent Management: Uniform size or autoallocate extents for efficient space usage.
- ✓ Block Space Management: Efficient management of free space within blocks.
- ✓ Online Segment Shrink: Reclaims wasted space in segments.
- ✓ Oracle Data Pump: High-performance data movement utility.
- ✓ Compression: Reduces storage requirements and improves performance.

Automatic Segment Space Management (ASSM)

ASSM simplifies and improves the management of free space within segments (e.g., tables and indexes). It uses bitmaps to track free space, which provides better performance and concurrency than traditional freelists.

Benefits:

- ✓ Reduced contention for free space.
- ✓ Improved performance for DML operations.
- ✓ Simplified space management.

Usage:
ASSM is enabled by default for tablespaces created with the SEGMENT SPACE MANAGEMENT AUTO clause.

Example:

```
CREATE TABLESPACE example_tbs
DATAFILE '/u01/app/oracle/oradata/example_tbs01.dbf' SIZE 100M
EXTENT MANAGEMENT LOCAL
SEGMENT SPACE MANAGEMENT AUTO;
```

Locally Managed Tablespaces (LMT)

LMTs manage extents locally within the tablespace using bitmaps, rather than in the data dictionary. This approach improves performance and scalability.

Benefits:

- ✓ Reduced contention on the data dictionary.
- ✓ Faster extent allocation and deallocation.
- ✓ Improved scalability for large databases.

Usage:
LMT is the default and recommended mode for tablespaces.

Example:

```
CREATE TABLESPACE example_tbs
DATAFILE '/u01/app/oracle/oradata/example_tbs01.dbf' SIZE 100M
EXTENT MANAGEMENT LOCAL;
```

Automatic Storage Management (ASM)

ASM is a volume manager and a file system for Oracle database files that provides high performance and reliability. It simplifies database storage management by automating file distribution and management.

Benefits:

- ✓ Simplified storage management.
- ✓ Improved performance and reliability.
- ✓ Automatic load balancing and redundancy.

Usage:
ASM is configured during database installation and managed through Oracle Grid Infrastructure.

Example: ASM Disk Group Creation:

```
CREATE DISKGROUP data_diskgroup NORMAL REDUNDANCY
DISK '/dev/sda1', '/dev/sdb1';
```

Extent Management

Extents are units of space allocation in a tablespace. Oracle manages extents automatically within LMTs.

Features:

- ✓ Uniform Extent Allocation: All extents are of a uniform size.
- ✓ Autoallocate: Extents are allocated in sizes optimized by Oracle.

Usage:
Uniform Extent Allocation Example:

```
CREATE TABLESPACE example_tbs
DATAFILE '/u01/app/oracle/oradata/example_tbs01.dbf' SIZE 100M
EXTENT MANAGEMENT LOCAL UNIFORM SIZE 1M;
```

Autoallocate Example:

```
CREATE TABLESPACE example_tbs
DATAFILE '/u01/app/oracle/oradata/example_tbs01.dbf' SIZE 100M
EXTENT MANAGEMENT LOCAL AUTOALLOCATE;
```

Block Space Management

Block space management deals with the management of free space within database blocks. Oracle uses bitmaps for managing free space within blocks when ASSM is enabled.

Features:

- ✓ Reduced block contention.
- ✓ Efficient space utilization.

Online Segment Shrink

Online segment shrink reclaims wasted space in segments (tables, indexes) and returns it to the tablespace, without causing significant downtime.

Benefits:

- ✓ Reclaims wasted space.
- ✓ Improves space utilization.
- ✓ Minimal impact on database operations.

Usage:

```
ALTER TABLE employees SHRINK SPACE;
ALTER INDEX emp_idx SHRINK SPACE;
```

Oracle Data Pump

Oracle Data Pump is a high-performance data movement utility used for fast data transfer between databases.

Benefits:

- ✓ Efficient data export and import.
- ✓ Supports compression, encryption, and parallelism.
- ✓ Can be used for space management tasks such as reorganizing tablespaces.

Usage:
Export Example:

```
expdp system/password DIRECTORY=dpump_dir1 DUMPFILE=expfull.dmp FULL=YES
```

Import Example:

```
impdp system/password DIRECTORY=dpump_dir1 DUMPFILE=expfull.dmp FULL=YES
```

Compression

Oracle offers several types of data compression to reduce storage requirements and improve performance.

Types:

- ✓ Advanced Row Compression: Compresses table data.
- ✓ Hybrid Columnar Compression (HCC): Provides high compression ratios for data warehousing.

Usage:
Advanced Row Compression Example:

```
CREATE TABLE employees COMPRESS FOR OLTP AS SELECT * FROM employees;
```

HCC Example:

```
CREATE TABLE employees COMPRESS FOR QUERY HIGH AS SELECT * FROM employees;
```

Example Workflow

```
--Create a Tablespace with ASSM and LMT:
CREATE TABLESPACE example_tbs
DATAFILE '/u01/app/oracle/oradata/example_tbs01.dbf' SIZE 100M
EXTENT MANAGEMENT LOCAL
SEGMENT SPACE MANAGEMENT AUTO;

--Add a Data File to the Tablespace:
ALTER TABLESPACE example_tbs
ADD DATAFILE '/u01/app/oracle/oradata/example_tbs02.dbf' SIZE 50M;

--Enable Autoextend for a Data File:
ALTER DATABASE DATAFILE '/u01/app/oracle/oradata/example_tbs01.dbf'
AUTOEXTEND ON NEXT 10M MAXSIZE UNLIMITED;

--Shrink a Table to Reclaim Space:
ALTER TABLE employees SHRINK SPACE;

--Export Data Using Data Pump:
expdp system/password DIRECTORY=dpump_dir1 DUMPFILE=expfull.dmp FULL=YES
```

Row Chaining and Migration

Row Chaining
Row chaining occurs when a row is too large to fit into a single database block. In this case, Oracle stores the row across multiple blocks. This situation is typical with rows that contain large objects (LOBs) such as CLOBs and BLOBs.

Causes:

- The row size exceeds the block size (e.g., the sum of column sizes in a row is larger than the block size).
- Impact:
- Increases I/O operations because retrieving a chained row requires reading multiple blocks.
- Can lead to performance degradation, especially for full table scans and other I/O-intensive operations.

Detection:

- You can detect row chaining by querying the DBA_TABLES or USER_TABLES view, which includes the CHAIN_CNT column indicating the number of chained rows in a table.

Example:

```
SELECT table_name, chain_cnt
FROM dba_tables
WHERE chain_cnt > 0;
```

Row Migration

Row migration occurs when an update to a row causes it to no longer fit in its original block, and Oracle moves the entire row to a new block. A pointer remains in the original block, pointing to the new block.

Causes:

- Rows that initially fit into a block but grow too large due to updates (e.g., adding data to variable-length columns).
- Impact:
- Increases I/O operations because retrieving a migrated row requires reading the original block and the block containing the migrated row.
- Can lead to performance issues, especially for index range scans and other read-intensive operations.

Detection:

- You can detect row migration by analyzing the ANALYZE TABLE ... LIST CHAINED ROWS command, which generates a report of chained and migrated rows.

Example:

Create a table to store the report:

```
CREATE TABLE chained_rows (
  owner_name    VARCHAR2(30),
  table_name    VARCHAR2(30),
  cluster_name  VARCHAR2(30),
  partition_name VARCHAR2(30),
  subpartition_name VARCHAR2(30),
  head_rowid    ROWID,
  analyze_timestamp DATE
);
```

Analyze a table and store the result in the chained_rows table:

```
ANALYZE TABLE employees LIST CHAINED ROWS INTO chained_rows;
```

Query the chained_rows table:

```
SELECT * FROM chained_rows;
```

Addressing Row Chaining and Migration

Row Chaining

Increase Block Size:

```
--Create tablespaces with larger block sizes to accommodate larger rows.
CREATE TABLESPACE large_block_tbs
DATAFILE '/u01/app/oracle/oradata/large_block_tbs01.dbf' SIZE 100M
BLOCKSIZE 16K;
```

Use LOB Storage Options:

```
--Store large objects (LOBs) in separate tablespaces or use LOB storage options to optimize storage.
CREATE TABLE large_table (
   id NUMBER,
   large_data CLOB
) LOB (large_data) STORE AS SECUREFILE (
   TABLESPACE large_lob_tbs
   ENABLE STORAGE IN ROW
);
```

Row Migration

Increase PCTFREE:

```
--Adjust the PCTFREE parameter to leave more free space in each block for future updates.
ALTER TABLE employees PCTFREE 20;
```

Reorganize Tables:

```
--Use the ALTER TABLE ... MOVE command to reorganize the table and eliminate migrated rows.
ALTER TABLE employees MOVE;
```

Rebuild Indexes:

```
--After reorganizing a table, rebuild indexes to ensure they point to the correct rows.
ALTER INDEX emp_idx REBUILD;
```

Example Workflow

```
--Detect Row Chaining and Migration:
SELECT table_name, chain_cnt
FROM dba_tables
WHERE chain_cnt > 0;

--Analyze a Table for Chained and Migrated Rows:
CREATE TABLE chained_rows (
   owner_name    VARCHAR2(30),
   table_name    VARCHAR2(30),
   cluster_name  VARCHAR2(30),
   partition_name VARCHAR2(30),
   subpartition_name VARCHAR2(30),
   head_rowid    ROWID,
   analyze_timestamp DATE
);
```

```
ANALYZE TABLE employees LIST CHAINED ROWS INTO chained_rows;

--Increase PCTFREE to Reduce Future Row Migration:
ALTER TABLE employees PCTFREE 20;

--Reorganize a Table to Eliminate Row Migration:
ALTER TABLE employees MOVE;
ALTER INDEX emp_idx REBUILD;
```

Managing Undo Data

Key Concepts

- Undo Data: Information stored in undo segments that allows the database to rollback transactions and maintain read consistency.
- Undo Tablespace: A special tablespace that stores undo segments.
- Automatic Undo Management (AUM): Oracle's method of managing undo data automatically.

Configuring Undo Management

1. Creating an Undo Tablespace

When creating a database, you typically create an undo tablespace. You can also create an additional undo tablespace if needed.

```
CREATE UNDO TABLESPACE undo_tbs
DATAFILE '/u01/app/oracle/oradata/undo_tbs01.dbf' SIZE 500M
AUTOEXTEND ON NEXT 50M MAXSIZE UNLIMITED;
```

2. Setting the Undo Tablespace

To use a specific undo tablespace, set the UNDO_TABLESPACE parameter.

```
ALTER SYSTEM SET UNDO_TABLESPACE = undo_tbs;
```

Monitoring Undo Space Usage

1. Viewing Undo Tablespace Information

You can query the DBA_TABLESPACES and DBA_DATA_FILES views to get information about the undo tablespace.

```
SELECT tablespace_name, status
FROM dba_tablespaces
WHERE contents = 'UNDO';

SELECT file_name, bytes, autoextensible
FROM dba_data_files
WHERE tablespace_name = 'UNDO_TBS';
```

2. Monitoring Undo Space Usage
Oracle provides several views to monitor undo space usage, such as V$UNDOSTAT, DBA_UNDO_EXTENTS, and DBA_HIST_UNDOSTAT.

```
SELECT begin_time, end_time, undoblks, maxquerylen, ssolderrcnt
FROM v$undostat
ORDER BY begin_time DESC;

SELECT tablespace_name, extents, blocks
FROM dba_undo_extents
ORDER BY tablespace_name;
```

Configuring Automatic Undo Retention

The UNDO_RETENTION parameter specifies the time in seconds that Oracle attempts to retain undo data to support consistent reads and flashback features.

```
ALTER SYSTEM SET UNDO_RETENTION = 900; -- Retain undo data for 15 minutes
```

Tuning Undo Tablespace

1. Sizing the Undo Tablespace
To determine the appropriate size for the undo tablespace, consider the undo retention period and the transaction workload. Oracle's DBMS_UNDO_ADV package can help with this.

```
DECLARE
 min_undo_retention NUMBER;
 undo_tablespace_size NUMBER;
BEGIN
 DBMS_UNDO_ADV.advisor(UNDO_RETENTION=>1800, -- Desired undo retention period in seconds
            RETENTION_SIZE=>undo_tablespace_size,
            MIN_RETENTION=>min_undo_retention);
 DBMS_OUTPUT.put_line('Recommended Undo Tablespace Size: ' || undo_tablespace_size || ' bytes');
 DBMS_OUTPUT.put_line('Minimum Undo Retention: ' || min_undo_retention || ' seconds');
END;
/
```

2. Extending the Undo Tablespace
If the undo tablespace needs more space, you can add data files or extend existing ones.

Example: Adding a Data File

```
ALTER TABLESPACE undo_tbs
ADD DATAFILE '/u01/app/oracle/oradata/undo_tbs02.dbf' SIZE 200M;
```

Example: Extending an Existing Data File

```
ALTER DATABASE DATAFILE '/u01/app/oracle/oradata/undo_tbs01.dbf' RESIZE 1G;
```

Best Practices for Managing Undo Data

- ✓ Use Automatic Undo Management (AUM): This simplifies undo management by automatically managing undo segments and retention.

- ✓ Monitor Undo Space Usage: Regularly monitor undo space to avoid running out of space and ensure adequate undo retention.
- ✓ Set Appropriate Undo Retention: Adjust the UNDO_RETENTION parameter based on the needs of your workload and flashback requirements.
- ✓ Size Undo Tablespace Appropriately: Ensure the undo tablespace is sized to handle peak workloads and retention requirements.
- ✓ Use Multiple Undo Tablespaces (Optional): Consider using multiple undo tablespaces for different workloads or to isolate undo data.

Example Workflow

```
--Create and Configure an Undo Tablespace:
CREATE UNDO TABLESPACE undo_tbs
DATAFILE '/u01/app/oracle/oradata/undo_tbs01.dbf' SIZE 500M
AUTOEXTEND ON NEXT 50M MAXSIZE UNLIMITED;

ALTER SYSTEM SET UNDO_TABLESPACE = undo_tbs;

--Set Undo Retention:
ALTER SYSTEM SET UNDO_RETENTION = 1800; -- Retain undo data for 30 minutes

--Monitor Undo Space Usage:
SELECT begin_time, end_time, undoblks, maxquerylen, ssolderrcnt
FROM v$undostat
ORDER BY begin_time DESC;

SELECT tablespace_name, extents, blocks
FROM dba_undo_extents
ORDER BY tablespace_name;

--Extend the Undo Tablespace:
ALTER TABLESPACE undo_tbs
ADD DATAFILE '/u01/app/oracle/oradata/undo_tbs02.dbf' SIZE 200M;

ALTER DATABASE DATAFILE '/u01/app/oracle/oradata/undo_tbs01.dbf' RESIZE 1G;

--Tune Undo Tablespace Size:
DECLARE
 min_undo_retention NUMBER;
 undo_tablespace_size NUMBER;
BEGIN
 DBMS_UNDO_ADV.advisor(UNDO_RETENTION=>1800, -- Desired undo retention period in seconds
            RETENTION_SIZE=>undo_tablespace_size,
            MIN_RETENTION=>min_undo_retention);
 DBMS_OUTPUT.put_line('Recommended Undo Tablespace Size: ' || undo_tablespace_size || ' bytes');
 DBMS_OUTPUT.put_line('Minimum Undo Retention: ' || min_undo_retention || ' seconds');
END;
/
```

Moving Data

Oracle Data Pump

Oracle Data Pump is a high-performance data movement utility used for fast data export and import. It provides several features for managing large data transfers.

Key Components

- ✓ Data Pump Export (expdp): Exports data from the database to a dump file.
- ✓ Data Pump Import (impdp): Imports data from a dump file into the database.

Example: Full Database Export

```
expdp system/password@db FULL=YES DIRECTORY=dpump_dir DUMPFILE=full_db.dmp LOGFILE=full_db.log
```

Example: Schema Export

```
expdp system/password@db SCHEMAS=schema_name DIRECTORY=dpump_dir DUMPFILE=schema.dmp
LOGFILE=schema.log
```

Example: Table Export

```
expdp system/password@db TABLES=table_name DIRECTORY=dpump_dir DUMPFILE=table.dmp
LOGFILE=table.log
```

Example: Using Parallelism

```
expdp system/password@db SCHEMAS=schema_name DIRECTORY=dpump_dir DUMPFILE=schema%U.dmp
LOGFILE=schema.log PARALLEL=4
```

Example: Full Database Import

```
impdp system/password@db FULL=YES DIRECTORY=dpump_dir DUMPFILE=full_db.dmp
LOGFILE=full_db_imp.log
```

Example: Schema Import

```
impdp system/password@db SCHEMAS=schema_name DIRECTORY=dpump_dir DUMPFILE=schema.dmp
LOGFILE=schema_imp.log
```

Example: Table Import

```
impdp system/password@db TABLES=table_name DIRECTORY=dpump_dir DUMPFILE=table.dmp
LOGFILE=table_imp.log
```

Example: Using Parallelism

```
impdp system/password@db SCHEMAS=schema_name DIRECTORY=dpump_dir DUMPFILE=schema%U.dmp
LOGFILE=schema_imp.log PARALLEL=4
```

Example Workflow: Using Data Pump

```
--Create Directory Object:
CREATE OR REPLACE DIRECTORY dpump_dir AS '/path/to/dpump_dir';
GRANT READ, WRITE ON DIRECTORY dpump_dir TO system;

--Export Schema:
expdp system/password@db SCHEMAS=hr DIRECTORY=dpump_dir DUMPFILE=hr.dmp LOGFILE=hr_exp.log

--Import Schema:
impdp system/password@db SCHEMAS=hr DIRECTORY=dpump_dir DUMPFILE=hr.dmp LOGFILE=hr_imp.log
```

SQL Loader

SQL*Loader is a utility for loading data from external files into Oracle tables. It is particularly useful for bulk loading data.

Key Components

- ✓ Control File: Defines how data is to be loaded.
- ✓ Data File: Contains the data to be loaded.
- ✓ Log File: Logs the loading process.
- ✓ Bad File: Records any rows that SQL*Loader cannot load.
- ✓ Discard File: Records any rows that do not meet selection criteria.

Preparing the Control File
The control file contains the SQL*Loader instructions. Here's a sample control file:

```
-- Example: load.ctl
LOAD DATA
INFILE 'data.csv'
INTO TABLE my_table
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
(
 column1,
 column2,
 column3
)
```

Loading Data Using SQL*Loader
Example Command:

```
sqlldr username/password@db control=load.ctl log=load.log bad=load.bad discard=load.dsc
```

Example Workflow: Using SQL*Loader
Prepare Data File (data.csv):

```
1,John Doe,5000
2,Jane Smith,6000
3,Bob Johnson,7000
```

Prepare Control File (load.ctl):

```
LOAD DATA
INFILE 'data.csv'
INTO TABLE employees
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
(
 employee_id,
 employee_name,
 salary
)
```

Load Data:

```
sqlldr system/password@db control=load.ctl log=load.log bad=load.bad discard=load.dsc
```

Backup and Recovery Concepts

Types of Failures

Types of Failures

1) User Errors
2) Statement Failures
3) Process Failures
4) Instance Failures
5) Media Failures
6) Network Failures

1. User Errors

User errors occur when a user performs an incorrect or unintended action, such as deleting or updating the wrong data. These errors do not result from system or hardware issues but from human mistakes.

Examples:

- Accidental deletion of critical data.
- Incorrect updates to records.
- Dropping a table or index by mistake.

Recovery Solutions:

- Flashback Technology: Use Flashback Query, Flashback Table, Flashback Transaction Query, and Flashback Database to revert the database to a previous state.
- Point-in-Time Recovery (PITR): Restore the database to a specific point in time before the error occurred.

2. Statement Failures

Statement failures occur when an SQL statement fails to execute successfully due to syntax errors, constraint violations, or other issues.

Examples:

- SQL syntax errors.
- Violations of integrity constraints (e.g., primary key or foreign key constraints).

- Runtime errors such as division by zero.

Recovery Solutions:

- Error Correction: Correct the SQL statement and re-execute it.
- Logging and Debugging: Review error logs and application logs to identify and correct the issues.

3. Process Failures

Process failures occur when a user process or background process terminates unexpectedly. This can be due to issues like hardware failures, software bugs, or resource exhaustion.

Examples:

- A user process gets terminated due to a session kill or network disconnection.
- A background process (e.g., DBWR, LGWR) crashes due to resource issues.

Recovery Solutions:

- Automatic Process Recovery: Oracle automatically detects and recovers from process failures by restarting the failed process.
- Monitoring Tools: Use Oracle Enterprise Manager (OEM) or other monitoring tools to detect and investigate process failures.

4. Instance Failures

Instance failures occur when the Oracle instance (i.e., the set of background processes and memory structures) crashes. This can be due to hardware failures, power outages, or software issues.

Examples:

- Power failure causing the server to shut down.
- Operating system crashes.
- Oracle internal errors causing instance termination.

Recovery Solutions:

- Automatic Instance Recovery: Oracle automatically performs instance recovery upon the next startup, using redo logs to apply committed changes and roll back uncommitted transactions.
- Manual Recovery: In rare cases, manual intervention may be required to complete the recovery process.

5. Media Failures

Media failures occur when a disk or storage device containing the database files fails. This can result in data corruption or loss.

Examples:

- Disk crash or hardware failure.
- Corruption of data files, control files, or redo logs.
- Logical corruption due to file system issues.

Recovery Solutions:

- Backup and Restore: Use RMAN (Recovery Manager) to restore the affected files from backups and apply redo logs to recover transactions.
- Data Guard: Use Oracle Data Guard to maintain standby databases that can be activated in case of primary database failures.
- Block Media Recovery: Use RMAN to recover specific corrupted blocks without restoring the entire file.

6. Network Failures
Network failures occur when there are issues with the network infrastructure that prevent communication between the database and its clients or between different database instances in a distributed environment.

Examples:

- Network connectivity loss or instability.
- Network hardware failures (e.g., routers, switches).
- Network configuration issues.

Recovery Solutions:

- Network Redundancy: Implement redundant network paths to ensure continuous connectivity.
- Network Monitoring: Use network monitoring tools to detect and resolve network issues promptly.
- Automatic Client Failover: Configure Transparent Application Failover (TAF) to redirect client connections to a surviving instance in a Real Application Clusters (RAC) environment.

Instance Recovery

Instance recovery is necessary to:

- Ensure the database is brought back to a consistent state after a failure.
- Apply any committed changes that were not yet written to the data files (redo logs).
- Roll back any uncommitted transactions to maintain data integrity.

How Instance Recovery Works
Instance recovery is automatically performed by Oracle when the database is restarted after a failure. The key steps involved in instance recovery are:

- Rolling Forward: Applying the redo log records to the data files to redo changes made by committed transactions.
- Rolling Back: Undoing changes made by uncommitted transactions to ensure data consistency.

Monitoring Instance Recovery
You can monitor the progress and details of instance recovery using various views and logs:

Alert Log:
The alert log file contains detailed information about the instance recovery process, including the time taken and the number of blocks processed.

```
tail -f alert_<SID>.log
```

V$INSTANCE_RECOVERY View:
This view provides information about the settings and progress of instance recovery.

```
SELECT * FROM v$instance_recovery;
V$LOG and V$LOGFILE Views:
```

These views provide information about the redo logs used during the recovery process.

```
SELECT * FROM v$log;
SELECT * FROM v$logfile;
```

Configuring Instance Recovery Parameters

You can configure instance recovery behavior by setting the following parameters:

```
FAST_START_MTTR_TARGET:
```

This parameter specifies the desired mean time to recover (MTTR) in seconds. Oracle uses this value to optimize the checkpointing frequency and minimize recovery time.

```
ALTER SYSTEM SET FAST_START_MTTR_TARGET = 300;
```

LOG_CHECKPOINT_INTERVAL:

Specifies the maximum number of redo log blocks that can be written before a checkpoint occurs.

```
ALTER SYSTEM SET LOG_CHECKPOINT_INTERVAL = 10000;
```

LOG_CHECKPOINT_TIMEOUT:

Specifies the maximum time in seconds between checkpoints.

```
ALTER SYSTEM SET LOG_CHECKPOINT_TIMEOUT = 300;
```

Example Workflow

```
--Configure Instance Recovery Parameters:
ALTER SYSTEM SET FAST_START_MTTR_TARGET = 300;
ALTER SYSTEM SET LOG_CHECKPOINT_INTERVAL = 10000;
ALTER SYSTEM SET LOG_CHECKPOINT_TIMEOUT = 300;

--Monitor Instance Recovery:
SELECT * FROM v$instance_recovery;

--Check the Alert Log for Recovery Details:
tail -f alert_<SID>.log
```

The Checkpoint (CKPT) Process

The Checkpoint (CKPT) process is responsible for signaling the Database Writer (DBWn) to write all dirty buffers (buffers that contain data that has been modified but not yet written to disk) to data files. This ensures that the database can be recovered to a consistent state after an instance failure.

Key Functions:

- ✓ Updates the control file with the checkpoint information.
- ✓ Signals the DBWn process to write dirty buffers to data files.
- ✓ Reduces the amount of recovery work needed by periodically flushing data to disk.

Configuration:

```
Checkpointing behavior can be influenced by parameters like FAST_START_MTTR_TARGET,
LOG_CHECKPOINT_INTERVAL, and LOG_CHECKPOINT_TIMEOUT.
```

Example:

```
ALTER SYSTEM SET FAST_START_MTTR_TARGET = 300; -- Target mean time to recover (MTTR) in seconds
```

Redo Log Files and the Log Writer (LGWR)

Redo Log Files:
Redo log files store all changes made to the database as redo entries. These files are essential for recovering the database to a consistent state after a failure.

Log Writer (LGWR):
The LGWR process writes redo entries from the redo log buffer in the SGA (System Global Area) to the redo log files on disk. LGWR ensures that committed transactions are safely stored in redo log files.

Key Functions:

- Writes redo entries to the online redo log files.
- Ensures that redo entries are written to disk before the user process is notified of a committed transaction.
- Writes occur during commit operations, when the redo log buffer is one-third full, or before DBWn writes dirty buffers to disk.

Example: Monitoring Redo Log Files:

```
SELECT * FROM v$log;
SELECT * FROM v$logfile;
```

Automatic Instance Recovery or Crash Recovery

Automatic instance recovery, or crash recovery, is performed by Oracle when an instance fails unexpectedly. This process is initiated automatically when the database is restarted.

Steps Involved:

- Rolling Forward: Apply all redo log entries generated after the last checkpoint to ensure all committed transactions are applied.
- Rolling Back: Undo any uncommitted transactions using undo segments to ensure data consistency.

Monitoring:
Check the alert log file for details on instance recovery.

```
tail -f alert_<SID>.log
```

Using the MTTR Advisor

The Mean Time to Recover (MTTR) Advisor helps DBAs determine the optimal settings for the FAST_START_MTTR_TARGET parameter, which specifies the target time for instance recovery.

Usage:
The MTTR Advisor provides recommendations based on the current workload and system performance.

Example: Enabling MTTR Advisor:

```
ALTER SYSTEM SET FAST_START_MTTR_TARGET = 300;
```

Example: Using the MTTR Advisor:

```
SELECT target_mttr, estimated_mttr
FROM v$instance_recovery;
```

Complete Recovery Process

Complete recovery involves restoring all data files from a backup and applying all redo log files to bring the database to the most current state.

Steps Involved:

- ✓ Restore Backup: Restore all data files from the most recent backup.
- ✓ Apply Redo Logs: Apply all archived and online redo logs.

Example:

```
-- Restore data files
rman target /
RESTORE DATABASE;

-- Recover database
RECOVER DATABASE;
```

The Point-in-Time Recovery Process

PITR restores the database to a specific point in time before an undesired event occurred, such as a user error.

Steps Involved:

- ✓ Restore Backup: Restore data files from a backup taken before the point in time.
- ✓ Apply Redo Logs: Apply redo logs up to the desired point in time.

Example:

```
-- Set the point in time for recovery
rman target /
RUN {
 SET UNTIL TIME '2023-06-01 10:00:00';
 RESTORE DATABASE;
 RECOVER DATABASE;
}
```

Flashback

Flashback technology allows you to quickly revert the database to a previous state without restoring from backups. This includes Flashback Database, Flashback Table, Flashback Query, Flashback Drop, and Flashback Transaction Query.

Flashback Database:
Allows you to rewind the entire database to a previous point in time.

```
-- Enable Flashback Database
ALTER DATABASE FLASHBACK ON;

-- Perform Flashback Database
FLASHBACK DATABASE TO TIMESTAMP (SYSDATE - 1/24); -- Rewind to one hour ago
```

Flashback Table:

Allows you to revert one or more tables to a previous point in time.

```
FLASHBACK TABLE employees TO TIMESTAMP (SYSDATE - 1/24); -- Rewind to one hour ago
```

Flashback Query:

Allows you to query the historical state of a table at a specific point in time.

```
SELECT * FROM employees AS OF TIMESTAMP (SYSDATE - 1/24); -- Query data as of one hour ago
```

Flashback Drop:

Allows you to undo a DROP TABLE operation.

```
FLASHBACK TABLE employees TO BEFORE DROP;
```

Flashback Transaction Query:

Allows you to view changes made by a specific transaction.

```
SELECT xid, operation, undo_sql
FROM flashback_transaction_query
WHERE xid = '0002000300000001';
```

Backup and Recovery Configuration

Configuring for Recoverability

1. Enabling Archivelog Mode

Enabling Archivelog mode is crucial for performing complete database recovery. In Archivelog mode, Oracle keeps a copy of each redo log file when it is filled, allowing you to recover the database to any point in time.

Steps to Enable Archivelog Mode:

Shut Down the Database:

```
SHUTDOWN IMMEDIATE;
```

Start the Database in Mount Mode:

```
STARTUP MOUNT;
```

Enable Archivelog Mode:

```
ALTER DATABASE ARCHIVELOG;
```

Open the Database:

```
ALTER DATABASE OPEN;
```

Verify Archivelog Mode:

```
ARCHIVE LOG LIST;
```

2. Configuring Archive Log Destinations
You need to specify the location where the archived redo log files will be stored.

```
ALTER SYSTEM SET log_archive_dest_1='LOCATION=/u01/app/oracle/archivelog' SCOPE=BOTH;
```

3. Setting Initialization Parameters
Several initialization parameters need to be configured to ensure proper recoverability.

Example:

```
ALTER SYSTEM SET db_recovery_file_dest='/u01/app/oracle/fast_recovery_area' SCOPE=BOTH;
ALTER SYSTEM SET db_recovery_file_dest_size=10G SCOPE=BOTH;
ALTER SYSTEM SET db_flashback_retention_target=1440 SCOPE=BOTH; -- Retention period in minutes
```

Redo Log Files

Redo log files are crucial for the recovery process, as they store all changes made to the database. Properly configuring and managing redo log files is essential for ensuring database recoverability.

1. Creating and Managing Redo Log Groups

Creating Redo Log Groups:
To create additional redo log groups, use the ALTER DATABASE statement.

```
ALTER DATABASE ADD LOGFILE GROUP 4 ('/u01/app/oracle/oradata/redo04.log') SIZE 50M;
ALTER DATABASE ADD LOGFILE GROUP 5 ('/u01/app/oracle/oradata/redo05.log') SIZE 50M;
```

Dropping Redo Log Groups:
To drop a redo log group, ensure it is not currently active.

```
ALTER DATABASE DROP LOGFILE GROUP 3;
```

2. Adding Redo Log Members
Adding multiple members to each redo log group provides redundancy. This ensures that if one redo log member is corrupted, the database can still be recovered using the other members.

```
ALTER DATABASE ADD LOGFILE MEMBER '/u01/app/oracle/oradata/redo04b.log' TO GROUP 4;
ALTER DATABASE ADD LOGFILE MEMBER '/u01/app/oracle/oradata/redo05b.log' TO GROUP 5;
```

3. Resizing Redo Log Files
If redo log files are too small, they may switch too frequently, leading to performance issues. Conversely, if they are too large, recovery time may increase.

Steps to Resize Redo Log Files:

Add New Redo Log Files with Desired Size:

```
ALTER DATABASE ADD LOGFILE GROUP 6 ('/u01/app/oracle/oradata/redo06.log') SIZE 100M;
ALTER DATABASE ADD LOGFILE GROUP 7 ('/u01/app/oracle/oradata/redo07.log') SIZE 100M;
```

Drop the Old Redo Log Files:

```
ALTER DATABASE DROP LOGFILE GROUP 1;
ALTER DATABASE DROP LOGFILE GROUP 2;
```

4. Monitoring Redo Log Activity
You can monitor redo log activity using dynamic performance views.

```
SELECT group#, members, bytes, status FROM v$log;
SELECT member, group#, type, is_recovery_dest_file FROM v$logfile;
```

Example Workflow for Configuring Recoverability and Managing Redo Log Files

```
--Enable Archivelog Mode:
SHUTDOWN IMMEDIATE;
STARTUP MOUNT;
ALTER DATABASE ARCHIVELOG;
ALTER DATABASE OPEN;
ARCHIVE LOG LIST;

--Configure Archive Log Destination:
ALTER SYSTEM SET log_archive_dest_1='LOCATION=/u01/app/oracle/archivelog' SCOPE=BOTH;

--Set Initialization Parameters:
ALTER SYSTEM SET db_recovery_file_dest='/u01/app/oracle/fast_recovery_area' SCOPE=BOTH;
ALTER SYSTEM SET db_recovery_file_dest_size=10G SCOPE=BOTH;
ALTER SYSTEM SET db_flashback_retention_target=1440 SCOPE=BOTH;

--Create Redo Log Groups and Members:
ALTER DATABASE ADD LOGFILE GROUP 4 ('/u01/app/oracle/oradata/redo04.log') SIZE 50M;
ALTER DATABASE ADD LOGFILE MEMBER '/u01/app/oracle/oradata/redo04b.log' TO GROUP 4;

--Monitor Redo Log Activity:
SELECT group#, members, bytes, status FROM v$log;
SELECT member, group#, type, is_recovery_dest_file FROM v$logfile;

--Perform Regular Backups:
rman target /
RUN {
 BACKUP DATABASE PLUS ARCHIVELOG;
 BACKUP CURRENT CONTROLFILE;
 BACKUP SPFILE;
}
```

Creating Database Backups

RMAN Backup

Types of RMAN Backups

- Full Backups: Capture the entire contents of the database.
- Incremental Backups: Capture only the changes made since the last backup.
- Differential Incremental Backup: Backs up all blocks changed since the last level 1 or level 0 backup.
- Cumulative Incremental Backup: Backs up all blocks changed since the last level 0 backup.
- Archivelog Backups: Back up archived redo logs to ensure all transactions are recoverable.

Key RMAN Commands

- BACKUP: Creates backups of the database, tablespaces, datafiles, control files, SPFILE, and archived redo logs.
- CONFIGURE: Sets persistent configurations for backup and recovery settings.
- SHOW: Displays the current RMAN configuration settings.
- RUN: Executes a series of RMAN commands within a block.

RMAN Backup Examples

1. Configuring RMAN for Backups

Before performing backups, configure RMAN with necessary settings like the default backup device type and retention policy.

Example: Configuring RMAN Settings

```
rman target /

-- Set the default device type to disk
CONFIGURE DEFAULT DEVICE TYPE TO DISK;

-- Set the retention policy to keep backups for 7 days
CONFIGURE RETENTION POLICY TO RECOVERY WINDOW OF 7 DAYS;

-- Enable backup optimization
CONFIGURE BACKUP OPTIMIZATION ON;

-- Configure the control file autobackup
CONFIGURE CONTROLFILE AUTOBACKUP ON;

-- Set the backup location
CONFIGURE CHANNEL DEVICE TYPE DISK FORMAT '/u01/app/oracle/backup/%U';
```

2. Performing a Full Database Backup

A full database backup captures the entire contents of the database.

Example: Full Database Backup

```
rman target /

BACKUP DATABASE PLUS ARCHIVELOG;
```

3. Performing Incremental Backups

Incremental backups can be used to reduce the amount of data backed up by only capturing changes.

Example: Level 0 Incremental Backup

```
rman target /

BACKUP INCREMENTAL LEVEL 0 DATABASE;
```

Example: Level 1 Incremental Backup

```
rman target /

BACKUP INCREMENTAL LEVEL 1 DATABASE;
```

4. Backing Up Archived Redo Logs
Backing up archived redo logs ensures that all transactions are recoverable.

Example: Archivelog Backup

```
rman target /

BACKUP ARCHIVELOG ALL;
```

Backing Up Databases on DBCS

On DBCS, Oracle provides automated tools and utilities for managing backups.

1. Automated Backups
DBCS offers automated daily backups that can be configured through the Oracle Cloud console.

Configuring Automated Backups:

1) Log in to the Oracle Cloud Console:
   - Navigate to the Oracle Cloud Infrastructure dashboard.
2) Select the Database Instance:
   - Go to the "Autonomous Database" or "DB Systems" section.
3) Configure Backup Settings:
   - Set the backup retention period and specify the backup window.
   - Enable automatic backups if not already enabled.

2. Manual Backups Using RMAN on DBCS
In addition to automated backups, you can perform manual backups using RMAN.

Example: Manual Full Database Backup on DBCS

1) SSH into the DBCS VM:
   Use SSH to connect to your DBCS VM.

2) Start RMAN and Connect to the Target Database:

```
ssh -i <private_key> opc@<DBCS_VM_IP>
sudo su - oracle
rman target /
```

3) Perform the Backup:

```
BACKUP DATABASE PLUS ARCHIVELOG;
```

Example Workflow for RMAN Backups on DBCS

```
--SSH into the DBCS VM:
ssh -i <private_key> opc@<DBCS_VM_IP>
sudo su - oracle

--Start RMAN and Configure Settings:
rman target /

CONFIGURE DEFAULT DEVICE TYPE TO DISK;
CONFIGURE RETENTION POLICY TO RECOVERY WINDOW OF 7 DAYS;
CONFIGURE BACKUP OPTIMIZATION ON;
CONFIGURE CONTROLFILE AUTOBACKUP ON;
CONFIGURE CHANNEL DEVICE TYPE DISK FORMAT '/u01/app/oracle/backup/%U';

--Perform a Full Database Backup:
BACKUP DATABASE PLUS ARCHIVELOG;

--Perform an Incremental Backup:
BACKUP INCREMENTAL LEVEL 1 DATABASE;

--Back Up Archived Redo Logs:
BACKUP ARCHIVELOG ALL;
```

Performing Database Recovery

Data Recovery Advisor

The Data Recovery Advisor is an Oracle tool that helps diagnose and repair data failures. It provides recommendations and automated scripts to fix the issues.

Usage:
Start RMAN and Connect to the Target Database:

```
rman target /
```

List Failures:

```
LIST FAILURE;
```

Advise on Failures:

```
ADVISE FAILURE;
```

Execute Repair Script:

```
REPAIR FAILURE;
```

Loss of a Control File

Control files are crucial for database operation, containing metadata about the database structure. Loss of all control files can prevent the database from starting.

Recovery Steps:
If the control file is lost, Oracle will indicate it during the startup process.

Use RMAN to restore the control file from a backup.

```
--Shutdown the Database:
SHUTDOWN IMMEDIATE;

--Start RMAN and Connect to the Target Database:
rman target /

--Restore the Control File:
RESTORE CONTROLFILE FROM '/backup/location/controlfile_backup.ctl';

--Mount the Database:
ALTER DATABASE MOUNT;

--Recover the Database:
RECOVER DATABASE;

--Open the Database:
ALTER DATABASE OPEN RESETLOGS;
```

Loss of a Redo Log File

Redo log files are critical for maintaining transaction integrity. Loss of a redo log file can impact the database's ability to recover from crashes.

Recovery Steps:
Oracle will indicate the loss of a redo log file during operation or startup.

Use SQL commands to drop and recreate the redo log file if it's missing or corrupted.

```
--Identify the Corrupted Redo Log Group:
SELECT group#, status FROM v$log;

--Drop the Corrupted Redo Log Group:
ALTER DATABASE DROP LOGFILE GROUP 3;

--Recreate the Redo Log Group:
ALTER DATABASE ADD LOGFILE GROUP 3 ('/path/to/redo03.log') SIZE 50M;

--Switch Log Files:
ALTER SYSTEM SWITCH LOGFILE;
```

DBCS: Performing Recovery by Using the Console

In Oracle Database Cloud Service (DBCS), the Oracle Cloud Console provides tools to perform database recovery tasks.

Steps:

1) Log in to the Oracle Cloud Console:
    - Navigate to the Oracle Cloud Infrastructure dashboard.

2) Select the Database Instance:
    - Go to the "Autonomous Database" or "DB Systems" section.

3) Perform Recovery:
    - Use the available options to restore from backups or recover specific files.

dbaascli Utility

The dbaascli utility is a command-line tool provided by Oracle Cloud for managing Oracle Database Cloud Service instances, including performing backup and recovery tasks.

Key Commands:

Backing Up the Database:

```
dbaascli backup db --target=DBAAS --type=FULL
```

Restoring the Database:

```
dbaascli restore db --bckid=backup_id
```

Listing Backups:

```
dbaascli list backups
```

Example: Using dbaascli for Recovery

SSH into the DBCS VM:

```
ssh -i <private_key> opc@<DBCS_VM_IP>
sudo su - oracle
```

List Available Backups:

```
dbaascli list backups
```

Restore the Database from a Specific Backup:

```
dbaascli restore db --bckid=backup_id
```

Monitoring and Tuning Database Performance

Automatic Workload Repository (AWR)

AWR is a built-in repository in Oracle Database that collects, processes, and maintains performance statistics. It provides a historical view of database performance and is the foundation for many Oracle performance tuning tools.

Key Features:

- ✓ Snapshots: AWR collects performance data snapshots at regular intervals (default is every hour).
- ✓ Reports: Generate AWR reports to analyze performance over specific periods.
- ✓ Baselines: Create baselines to compare current performance against historical performance.

Usage:
Generating AWR Reports:

Using SQL*Plus:

```
-- Connect to the database as SYSDBA
sqlplus / as sysdba

-- Generate an AWR report for a specific time period
@?/rdbms/admin/awrrpt.sql
```

Using Oracle Enterprise Manager (OEM):

1) Navigate to Performance > AWR > AWR Reports.
2) Select the time period and generate the report.

Managing AWR Snapshots:
List Snapshots:

```
SELECT snap_id, begin_interval_time, end_interval_time
FROM dba_hist_snapshot
ORDER BY snap_id;
```

Create a Manual Snapshot:

```
EXEC DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT();
```

Creating Baselines:
Create a Baseline:

```
EXEC DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE(
 start_snap_id => 100,
 end_snap_id => 110,
 baseline_name => 'MY_BASELINE');
```

Automatic Database Diagnostic Monitor (ADDM)

ADDM analyzes AWR data to identify performance issues and provide recommendations for improvement. It runs automatically after each AWR snapshot is taken.

Key Features:

- ✓ Findings: Provides detailed findings on performance issues.
- ✓ Recommendations: Suggests actionable recommendations to address identified issues.
- ✓ Impact Analysis: Shows the potential impact of recommendations.

Usage:
Running ADDM Analysis:

Using SQL*Plus:

```
-- Connect to the database as SYSDBA
sqlplus / as sysdba

-- Generate an ADDM report for a specific time period
@?/rdbms/admin/addmrpt.sql
```

Using Oracle Enterprise Manager (OEM):

1) Navigate to Performance > Advisors Home > ADDM.
2) View the findings and recommendations for the selected time period.

Viewing ADDM Findings and Recommendations:
Query Findings:

```
SELECT dbms_addm.get_report(
  dbid => 123456789,
  begin_snap_id => 100,
  end_snap_id => 110) AS report
FROM dual;
```

Advisory Framework

The advisory framework in Oracle Database provides a set of advisors that help you optimize various aspects of database performance, including memory, storage, and SQL execution.

Key Advisors:

- ✓ SQL Tuning Advisor: Analyzes SQL statements and provides recommendations for improving their performance.
- ✓ SQL Access Advisor: Recommends the creation of indexes and materialized views to optimize SQL queries.
- ✓ Memory Advisors: Includes SGA and PGA advisors to help allocate memory efficiently.

Usage:
SQL Tuning Advisor:

Using SQL*Plus:

```
-- Connect to the database as SYSDBA
sqlplus / as sysdba

-- Execute SQL Tuning Advisor for a specific SQL statement
EXEC DBMS_SQLTUNE.CREATE_TUNING_TASK(
  sql_text => 'SELECT * FROM employees',
  user_name => 'HR',
  task_name => 'sql_tuning_task');

EXEC DBMS_SQLTUNE.EXECUTE_TUNING_TASK(
  task_name => 'sql_tuning_task');

SELECT DBMS_SQLTUNE.REPORT_TUNING_TASK('sql_tuning_task') AS report
FROM dual;
```

Using Oracle Enterprise Manager (OEM):

1) Navigate to Performance > Advisors Home > SQL Tuning Advisor.
2) Select the SQL statement and generate recommendations.

SQL Access Advisor:
Using SQL*Plus:

```
-- Create a workload for SQL Access Advisor
EXEC DBMS_ADVISOR.CREATE_TASK(
  advisor_name => 'SQL Access Advisor',
  task_name => 'access_advisor_task');

EXEC DBMS_ADVISOR.ADD_SQLWKLD_SQLSET(
  task_name => 'access_advisor_task',
  sqlset_name => 'my_sqlset');

EXEC DBMS_ADVISOR.EXECUTE_TASK(
  task_name => 'access_advisor_task');

SELECT DBMS_ADVISOR.GET_TASK_REPORT('access_advisor_task') AS report
FROM dual;
```

Memory Advisors:

```
Using SQL*Plus:
-- Query SGA Target Advice
SELECT size_mb, size_factor, estd_db_time, estd_physical_reads
FROM v$sga_target_advice;

-- Query PGA Target Advice
SELECT pga_target_for_estimate, pga_target_factor, estd_extra_bytes_rw
FROM v$pga_target_advice;
```

Tuning SQL

Tuning SQL involves optimizing SQL statements to improve their performance. This includes analyzing execution plans, creating indexes, rewriting queries, and gathering statistics.

Steps for SQL Tuning:
Analyze Execution Plans:

Using SQL*Plus:

```
-- Display the execution plan for a SQL statement
EXPLAIN PLAN FOR
SELECT * FROM employees WHERE department_id = 10;

SELECT * FROM table(DBMS_XPLAN.DISPLAY);
```

Gather Optimizer Statistics:
Using SQL*Plus:

```
-- Gather statistics for a table
EXEC DBMS_STATS.GATHER_TABLE_STATS('HR', 'EMPLOYEES');
```

Create Indexes:
Using SQL*Plus:

```
-- Create an index to optimize a query
CREATE INDEX emp_dept_idx ON employees(department_id);
```

Rewrite Queries:
Example: Using Subquery Factoring:

```
-- Use WITH clause to simplify and optimize a query
WITH dept_emps AS (
 SELECT department_id, COUNT(*) AS emp_count
 FROM employees
 GROUP BY department_id
)
SELECT * FROM dept_emps WHERE emp_count > 5;
```

Example Workflow for Performance Monitoring and Tuning

```
--Generate an AWR Report:
sqlplus / as sysdba
@?/rdbms/admin/awrrpt.sql

--Run ADDM Analysis:
sqlplus / as sysdba
@?/rdbms/admin/addmrpt.sql

--Use SQL Tuning Advisor:
EXEC DBMS_SQLTUNE.CREATE_TUNING_TASK(
 sql_text => 'SELECT * FROM employees',
 user_name => 'HR',
 task_name => 'sql_tuning_task');
```

```
EXEC DBMS_SQLTUNE.EXECUTE_TUNING_TASK(
  task_name => 'sql_tuning_task');

SELECT DBMS_SQLTUNE.REPORT_TUNING_TASK('sql_tuning_task') AS report
FROM dual;

--Gather Optimizer Statistics:
EXEC DBMS_STATS.GATHER_TABLE_STATS('HR', 'EMPLOYEES');

--Analyze Execution Plan:
EXPLAIN PLAN FOR
SELECT * FROM employees WHERE department_id = 10;

SELECT * FROM table(DBMS_XPLAN.DISPLAY);
```

SQL Tuning Process

Oracle Optimizer

The Oracle Optimizer is a key component of the database that determines the most efficient way to execute a SQL query. It considers various factors, such as available indexes, data distribution, and query conditions, to generate an optimal execution plan.

Key Components:

- ✓ Cost-Based Optimizer (CBO): Uses statistics to estimate the cost of different execution plans and chooses the one with the lowest cost.
- ✓ Rule-Based Optimizer (RBO): An older method that uses a fixed set of rules to determine the execution plan. (Deprecated and not recommended for use).

Optimization Phases:

a) Parsing: The SQL statement is parsed, and syntax and semantic checks are performed.
b) Optimization: The optimizer generates multiple execution plans and chooses the one with the lowest cost.
c) Row Source Generation: The chosen execution plan is converted into a set of row sources that produce the result set.
d) Execution: The execution engine executes the plan and returns the results.

Optimizer Statistics

Optimizer statistics are metadata about database objects (tables, indexes, columns) that the optimizer uses to estimate the cost of various execution plans. Accurate statistics are crucial for the optimizer to make the best decisions.

Types of Statistics:

- Table Statistics: Information about the number of rows, number of blocks, average row length, etc.
- Column Statistics: Information about data distribution, such as histograms, distinct values, and nulls.
- Index Statistics: Information about index selectivity, clustering factor, and leaf blocks.

Gathering Statistics:
Statistics can be gathered manually or automatically using the DBMS_STATS package. Oracle automatically gathers statistics for database objects during maintenance windows.

Example:

```
-- Gather statistics for a specific table
EXEC DBMS_STATS.GATHER_TABLE_STATS('HR', 'EMPLOYEES');

-- Gather statistics for the entire schema
EXEC DBMS_STATS.GATHER_SCHEMA_STATS('HR');

-- Gather statistics for the entire database
EXEC DBMS_STATS.GATHER_DATABASE_STATS;
```

SQL Execution Plan

A SQL execution plan (also known as an execution plan or query plan) is a detailed roadmap of how the database will execute a SQL query. It shows the sequence of operations (such as table scans, index scans, joins) and the flow of data through these operations.

Viewing Execution Plans:
Execution plans can be viewed using the EXPLAIN PLAN statement, the DBMS_XPLAN package, or Oracle Enterprise Manager (OEM).

Example:

```
-- Explain the execution plan for a SQL statement
EXPLAIN PLAN FOR
SELECT * FROM employees WHERE department_id = 10;

-- Display the execution plan
SELECT * FROM table(DBMS_XPLAN.DISPLAY);
```

SQL Tuning Steps

1) Identify Problematic SQL Statements:
   - Use tools like AWR, ADDM, and SQL Trace to identify SQL statements that are consuming significant resources.

2) Generate and Analyze Execution Plans:
   - Use EXPLAIN PLAN and DBMS_XPLAN to generate and analyze the execution plans of problematic SQL statements.

3) Gather Optimizer Statistics:
   - Ensure that accurate statistics are gathered for all relevant database objects.

4) Use SQL Tuning Advisor:
   - Use the SQL Tuning Advisor to get recommendations on how to improve the performance of SQL statements.

5) Apply Recommended Changes:
    - Implement the recommendations provided by the SQL Tuning Advisor, such as creating indexes or rewriting SQL statements.

Example Workflow for SQL Tuning

Identify Problematic SQL Statements:

```
-- Use AWR to identify high-resource-consuming SQL statements
SELECT * FROM dba_hist_sqlstat WHERE executions > 1000 ORDER BY buffer_gets DESC;
```

Generate and Analyze Execution Plans:

```
-- Explain the execution plan for a SQL statement
EXPLAIN PLAN FOR
SELECT * FROM employees WHERE department_id = 10;

-- Display the execution plan
SELECT * FROM table(DBMS_XPLAN.DISPLAY);
```

Gather Optimizer Statistics:

```
-- Gather statistics for the EMPLOYEES table
EXEC DBMS_STATS.GATHER_TABLE_STATS('HR', 'EMPLOYEES');
```

Use SQL Tuning Advisor:

```
-- Create a tuning task for a specific SQL statement
EXEC DBMS_SQLTUNE.CREATE_TUNING_TASK(
  sql_text => 'SELECT * FROM employees WHERE department_id = 10',
  user_name => 'HR',
  task_name => 'sql_tuning_task');

-- Execute the tuning task
EXEC DBMS_SQLTUNE.EXECUTE_TUNING_TASK(task_name => 'sql_tuning_task');

-- Get the tuning recommendations
SELECT DBMS_SQLTUNE.REPORT_TUNING_TASK('sql_tuning_task') AS report
FROM dual;
```

Apply Recommended Changes:

```
-- Create an index based on the SQL Tuning Advisor recommendation
CREATE INDEX emp_dept_idx ON employees(department_id);
```