DBA(DATABASE ADMINISTRATION) CSE402B

QUESTION BANK

Unit-1

Q1. What is Oracle and Explain Oracle Architecture?
Q2. Explain installing and Managing Oracle.
Q3. Explain the process of creating a Database.
Q4. What is data dictionary? How it works?
Q5.What are Control and Redo log files? How its related to database?
Q6.How to manage table spaces and data files in a database?
Q7. How to manage tables in database?
Q8.What are indexes. Explain its types?
Q9. What are constraints. How many constraints are applicable on database?
Q10. How to manage users and security in Database?

Note:- These are the questions which we can expect in your university exams. you can do your preparation for your university exams based on these questions.

DATA BASE ADMINISTRATION

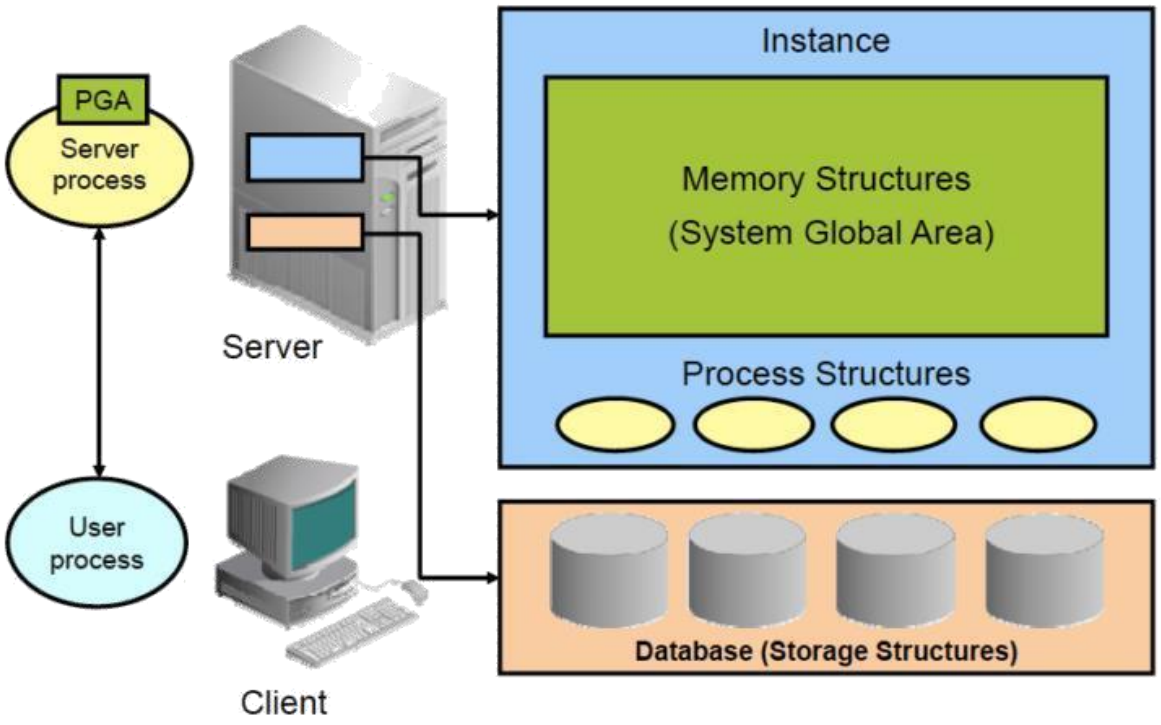
UNIT-1

Oracle overview and Architecture

ORACLE is a fourth generation relational database management system. In general, a database management system (DBMS) must be able to reliably manage a large amount of data in a multi-user environment so that many users can concurrently access the same data. All this must be accomplished while delivering high performance to the users of the database. A DBMS must also be secure from unauthorized access and provide efficient solutions for failure recovery. The ORACLE Server provides efficient and effective solutions for the major database features.

Architecture of Oracle:- There are three major structures in Oracle Database server architecture: memory structures, process structures, and storage structures. A basic Oracle database system consists of an Oracle database and a database instance.

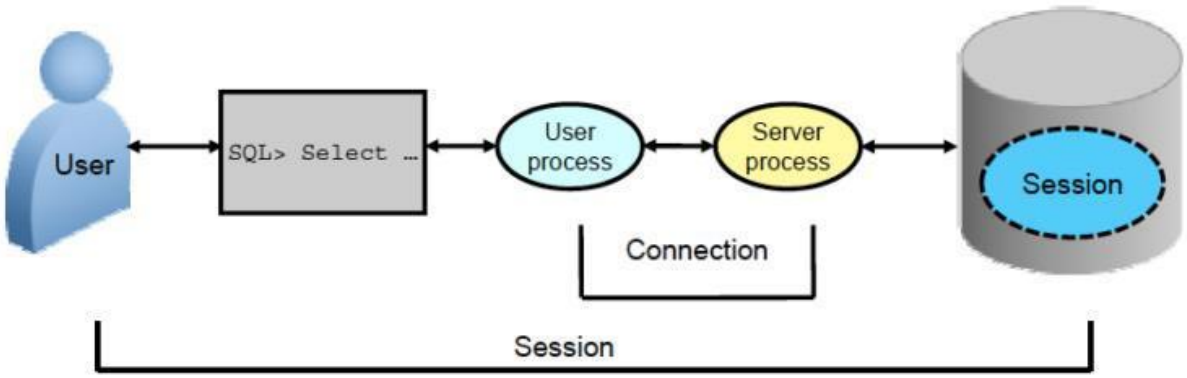
The database consists of both physical structures and logical structures. Because the physical and logical structures are separate, the physical storage of data can be managed without affecting access to logical storage structures.



The instance consists of memory structures and background processes associated with that instance. Every time an instance is started, a shared memory area called the System Global Area (SGA) is allocated and the background processes are started. Processes are jobs that work in the memory of computers. A process is defined as a "thread of control" or a mechanism in an operating system that can run a series of steps. After starting a database instance, the Oracle software associates the instance with a specific physical database. This is called mounting the database. The database is then ready to be opened, which makes it accessible to authorized users.

Connecting to the Database Instance

Connections and **sessions** are closely related to user processes but are very different in meaning.

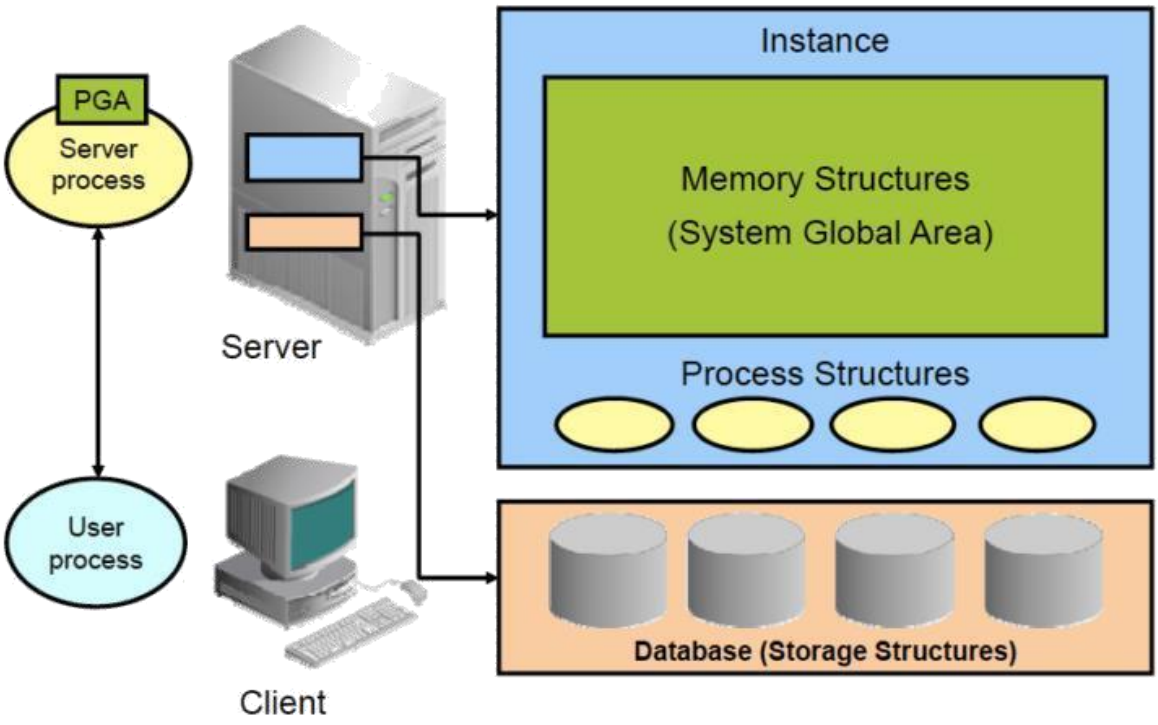


A **connection** is a communication pathway between a user process and an Oracle Database instance. A communication pathway is established using available interprocess communication mechanisms (on a computer that runs both the user process and Oracle Database) or network software (when different computers run the database application and Oracle Database and communicate through a network).

A **session** represents the state of a current user login to the database instance. For example, when a user starts SQL*Plus, the user must provide a valid username and password, and then a session is established for that user. A session lasts from the time a user connects until the user disconnects or exits the database application. Multiple sessions can be created and exist concurrently for a single Oracle database user using the same username. For example, a user with the username/password of HR/HR can connect to the same Oracle Database instance several times.

Oracle Database Memory Structures

Oracle Database creates and uses memory structures for various purposes. For example, memory stores program code being run, data that is shared among users, and private data areas for each connected user.



Two basic memory structures are associated with an instance:

- **System Global Area (SGA)**: Group of shared memory structures, known as SGA components, that contain data and control information for one Oracle Database instance. The SGA is shared by all server and background processes. Examples of data stored in the SGA include cached data blocks and shared SQL areas.
- **Program Global Areas (PGA)**: Memory regions that contain data and control information for a server or background process. A PGA is nonshared memory created by Oracle Database when a server or background process is started. Access to the PGA is exclusive to the server process. Each server process and background process has its own PGA.

The SGA is the memory area that contains data and control information for the instance.
The SGA includes the following data structures:

- **Shared pool**: Caches various constructs that can be shared among users
- **Database buffer cache**: Caches blocks of data retrieved from the database
- **Redo log buffer**: Caches redo information (used for instance recovery) until it can be written to the physical redo log files stored on the disk

- **Large pool**: Optional area that provides large memory allocations for certain large processes, such as Oracle backup and recovery operations, and I/O server processes
- **Java pool**: Used for all session-specific Java code and data in the Java Virtual Machine (JVM)
- **Streams pool**: Used by Oracle Streams to store information required by capture and apply
- **Fixed SGA**: An internal housekeeping area containing general information about the state of the database and the instance, and information communicated between processes When you start the instance, the amount of memory allocated for the SGA is displayed.

A **Program Global Area (PGA)** is a memory region that contains data and control information for each server process. An Oracle server process services a client's requests. Each server process has its own private PGA that is allocated when the server process is started. Access to the PGA is exclusive to that server process, and the PGA is read and written only by the Oracle code acting on its behalf. The PGA is divided into two major areas: stack space and the user global area (UGA).

With the dynamic SGA infrastructure, the sizes of the database buffer cache, the shared pool, the large pool, the Java pool, and the Streams pool can change without shutting down the instance.

Process Architecture

The processes in an Oracle Database system can be divided into three major groups:

- **User processes** that run the application or Oracle tool code
- **Oracle Database processes** that run the Oracle Database server code (including server processes and background processes)
- **Oracle daemons and application processes** not specific to a single database

When a user runs an application program or an Oracle tool such as SQL*Plus, the term user process is used to refer to the user's application. The user process may or may not be on the database server machine. Oracle Database also creates a server process to execute the commands issued by the user process. In addition, the Oracle server also has a set of background processes for an instance that interact with each other and with the operating system to manage the memory structures, asynchronously perform I/O to write data to disk, and perform other required tasks. The process structure varies for different Oracle Database configurations, depending on the operating system and the choice of Oracle Database options. The code for connected users can be configured as a dedicated server or a shared server.

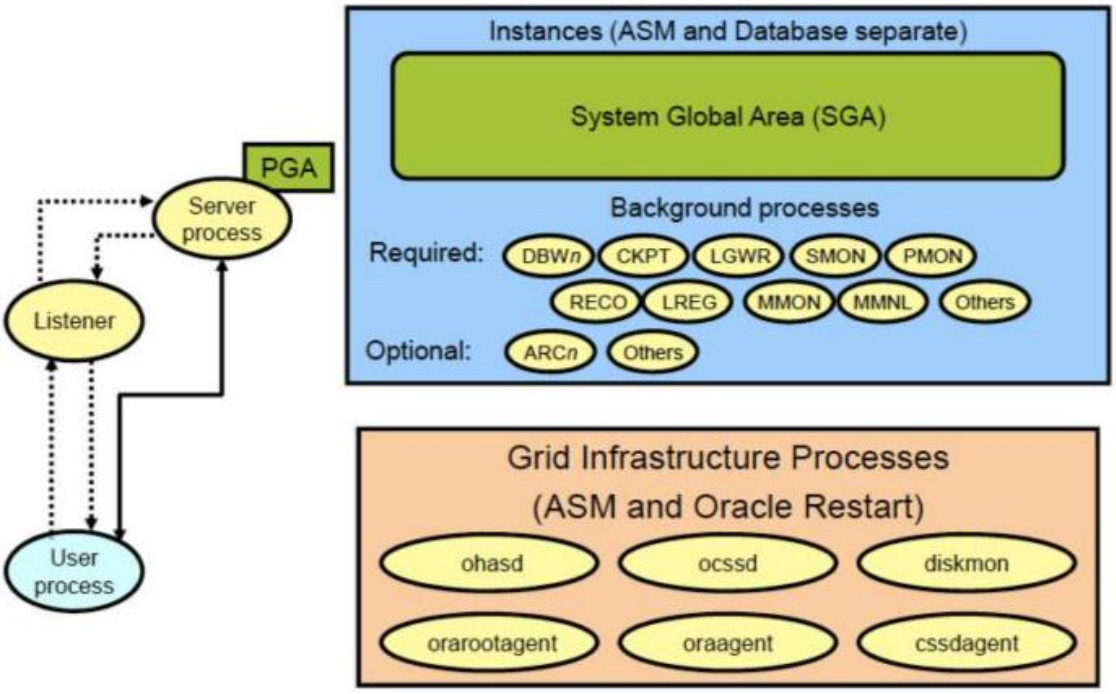
Dedicated server

For each session, the database application is run by a user process that is served by a dedicated server process that executes Oracle database server code.

Shared server

Eliminates the need for a dedicated server process for each connection. A dispatcher directs multiple incoming network session requests to a pool of shared server processes. A shared server process serves any client request.

Process Structures



Server Processes

Oracle Database creates server processes to handle the requests of user processes connected to the instance. The user process represents the application or tool that connects to the Oracle database. It may be on the same machine as the Oracle database, or it may exist on a remote client and use a network to reach the Oracle database. The user process first communicates with a listener process that creates a server process in a dedicated environment.

Server processes created on behalf of each user's application can perform one or more of the following:

- Parse and run SQL statements issued through the application.
- Read necessary data blocks from data files on disk into the shared database buffers of the SGA (if the blocks are not already present in the SGA).
- Return results in such a way that the application can process the information.

Background Processes

To maximize performance and accommodate many users, a multiprocess Oracle Database system uses some additional Oracle Database processes called background processes. An Oracle Database instance can have many background processes.

The background processes commonly seen in non-RAC, non-ASM environments can include the following:

- Database Writer process (DBWn)
- Log Writer process (LGWR)
- Checkpoint process (CKPT)
- System monitor process (SMON)
- Process monitor process (PMON)
- Recoverer process (RECO)
- Listener registration process (LREG)
- Manageability monitor process (MMON)
- Manageability monitor lite process (MMNL)
- Job queue coordinator (CJQ0)
- Job slave processes (Jnnn)
- Archiver processes (ARCn)
- Queue monitor processes (QMNn)
- The process spawner process (PSP0)

Other background processes may be found in more advanced configurations such as RAC. See the **V$BGPROCESS** view for more information about background processes. Some background processes are created automatically when an instance is started, whereas others are started as required. Other process structures are not specific to a single database, but rather can be shared among many databases on the same server. The Grid Infrastructure and networking processes fall into this category.

Oracle Grid Infrastructure processes on Linux and UNIX systems include the following:

- **ohasd (Oracle High Availability Service daemon)**: Is responsible for starting Oracle Clusterware processes
- **ocssd**: Cluster Synchronization Service daemon
- **diskmon (Disk Monitor daemon)**: Is responsible for input and output fencing for Oracle Exadata Storage
- **cssdagent**: Starts, stops, and check the status of the CSS daemon, ocssd
- **oraagent**: Extends clusterware to support Oracle-specific requirements and complex resources
- **orarootagent**: Is a specialized Oracle agent process that helps manage resources owned by root, such as the network.

INSTALLATION and MANAGING ORACLE

Installing method

The installation methods are divided into Desktop Class and Server Class:

- **Desktop Class**—This installation class is most appropriate for laptop or desktop computers. It includes a starter database and requires minimal configuration.
- **Server Class**—This installation class is for servers, such as you would find in a data center, or used to support enterprise-level applications. Choose this installation class if you need access to advanced configuration options.

During a Desktop Class installation, you make only basic choices. For a Server Class installation, you choose either typical installation (where you make only basic choices) or advanced installation.

During a Desktop Class or a typical installation, Oracle Database automatically installs the sample schemas.

Installation Type

When you install Oracle Database during basic and advanced installations, you need answers for the questions listed in this section. OUI provides default values for every choice.

- What type of database edition installation do you want to perform?

  Your choices are:

  - **Enterprise Edition**—This installation type is the full-featured Oracle Database product that provides data management for enterprise-level applications. It is intended for mission-critical, high-security online transaction processing (OLTP) and data warehousing environments.
  - **Standard Edition**—This installation type is suitable for workgroup or department-level applications, and for small to medium-sized enterprises. It provides core relational database management services and options and includes an integrated set of management tools, replication, Web features, and facilities for building business-critical applications.
  - **Standard One Edition**—This installation type is suitable for workgroup, department, or web applications. It provides core relational database management services for single-server environments or highly distributed branch environments. Oracle Standard Edition One includes all the facilities necessary to build business-critical applications.
  - **Personal Edition** (Microsoft Windows operating systems only)—This installation type installs the same software as the Enterprise Edition, but supports only a single-user, development and deployment environment.
- What are your database configuration options?

Software Installation Directories

You must specify the directory in which the Oracle Database software is installed, or the location where the product binary files are copied from the installation media. You must choose a location that has enough disk space to contain the software and is accessible by the operating system user performing the installation.

You also specify the location of the Oracle base directory, which is used by all Oracle software products installed on the server. The first time you install Oracle Database software on a server, you are prompted to specify the location of the inventory directory, called oraInventory. This directory provides a centralized inventory of all Oracle software products installed on the server. You should use the same value for the Oracle inventory directory each time you perform an Oracle software installation on the server.

Database File Location

A database includes several files that store the user data, database metadata, and information required to recover from failures. As an administrator, you decide what kind of storage subsystem to use for these files. You can select from the following options:

- **File System**—This default option creates database files that are managed by the file system of your operating system. You can specify the directory path where database files are to be stored. Oracle Database can create and manage the actual files.

  If you are not certain about which option to use, then select File System (the default).

- **Automatic Storage Management**—This option enables you to place your data files in Oracle Automatic Storage Management (Oracle ASM) disk groups. If you choose Oracle ASM, then Oracle Database automatically manages database file placement and naming. For environments with a large number of disks, this option simplifies database administration and maximizes performance. Oracle ASM performs software striping and mirroring at the file level for maximum storage flexibility, performance, and availability.

  Oracle ASM uses an Oracle ASM instance, which is distinct from the database instance, to configure and manage disk groups. A single Oracle ASM instance can provide storage for multiple databases on the same server.

Database Identifiers

These options include your global database name and system identifier (SID). The **SID** is a unique identifier that is used to distinguish this instance from other Oracle Database instances that you may create later and run concurrently on your system.

The global database name is the full name of the database that uniquely distinguishes it from any other database. The global database name is in the form database_name.database_domain, for example sales.example.com. The database name portion sales is a simple name you call your

database. The database domain portion example.com specifies the database domain in which the database is located. Together, the database name and domain form the global database name.

About Advanced Installation

During advanced installations using the Server Class method you are prompted to make the additional choices listed in this section, and the choices for a typical installation. The installation process provides default values for every choice.

This guide describes, but does not document, these additional advanced installation choices. For more information, see *Oracle Database Installation Guide* for your platform.

- Product Languages

  You choose which language the software should use after it is installed. You can select multiple languages. The default value is English. If you choose a value other than English, it does not change the language used by the installation.

- Database Configuration Type

  You select a template to use when configuring the database. You can choose either either General Purpose/Transaction Processing or Data Warehousing.

- Database Configuration Options

  You can choose how to configure the database created by the installer. You can select the memory size and management options, the character sets used to store data, the security options for database access, and whether the sample schemas should be installed.

  To complete the exercises in this guide and related course material, you must install the sample schemas. This data is also used in most examples throughout Oracle Database documentation. Oracle recommends that you install the sample schemas.

  This choice is a configuration option only during advanced installation. Sample schemas are installed by default during typical or Desktop class installations.

- Database Management Options

  You specify whether to manage your database centrally or locally using Oracle Enterprise Manager. Central management enables you to manage multiple targets, such as databases and application servers, using a single interface. Using local management you can manage only a single database instance at a time.

  To use central management, you must have an Oracle Enterprise Management agent on each *host*, or computer that has Oracle Database software installed on it. These agents are

responsible for monitoring all components on that host. If an agent is not found on the local host, then this option is disabled during installation.

If you are setting up a single database for the first time, then it is recommended that you configure local management with Oracle Enterprise Manager, which is the default. You can later install additional databases and configure central management using Enterprise Manager.

- Recovery Options

You specify whether automated backups should be configured for the database. If you choose this option, you must specify whether the recovery area should be stored on the local file system or in an Oracle ASM disk group. You must also specify the operating system credentials the backup job uses when performing backups.

Overview of the Oracle Instance and Instance Management

An Oracle database system consists of an Oracle database and an Oracle instance (in an Oracle Real Application Clusters environment, there can be more than one instance).

A **database** consists of a set of disk files that store user data and metadata. **Metadata**, or "data about the data," consists of structural, configuration, and control information about the database.

An **Oracle instance** (also known as a **database instance**) contains the set of Oracle Database background processes that operate on the stored data and the shared allocated memory that those processes use to do their work.

Each instance has an instance ID, also known as a system ID (SID). Because there can be multiple Oracle databases on a host computer, each with its own set of data files, you must identify the instance to which you want to connect. For a local connection, you identify the instance by setting operating system environment variables ORACLE_SID and ORACLE_HOME. For a remote connection, you identify the instance by specifying a network address and a database service name.

An Oracle instance must be started to read and write information to the database. The Oracle instance creates the database upon receipt of instructions from the Oracle Database Configuration Assistant (DBCA) utility or the CREATE DATABASE SQL statement.

When the Oracle instance is not available, your data is safe in the database, but it cannot be accessed by any user or application.

The properties of an Oracle instance are specified using instance initialization parameters. When the instance is started, an initialization parameter file is read, and the instance is configured accordingly.

Managing an Oracle instance includes configuring parameters that affect the basic operation of the Oracle instance. These parameters are called initialization parameters. The Oracle instance reads initialization parameters from a file at startup.

During installation, when you select a preconfigured database workload available in Database Configuration Assistant (DBCA), the initialization parameters are optimized for typical use in the environment that you specified. As the number of database users increases and the workload increases, you might have to alter some initialization parameters. You can make these changes using the Initialization Parameter page in Oracle Enterprise Manager Database Express (EM Express), or by using an advisor provided by Oracle Database, such as the Memory Advisor. See "Optimizing Memory Usage with the Memory Advisors" for more information.

After being read from a file, initialization parameters are retained in memory, where the values for many of them can be changed dynamically. There are two types of parameter files. The type of file used to start the instance determines if dynamic initialization parameter changes persist across database shutdown and startup. The parameter file types are:

- Server parameter file

  The **server parameter file**, commonly known as the SPFILE, is the preferred form of initialization parameter file, and is a binary file that can be written to and read by the database. It *must not* be edited manually. It is stored on the host computer on which Oracle Database is running. Changes are made when you use EM Express or SQL*Plus to modify one or more initialization parameters, or when Oracle Database itself makes changes for self-tuning purposes. Any changes to it persist across database shutdown and startup operations.

- Text initialization parameter file

  A **text initialization parameter file** is a text file that can be read by the Oracle instance, but it is not written to by the instance. You can change a text initialization parameter file with a text editor, but changes do not take effect until you restart the Oracle instance. When you start the instance with this type of file, you can still change many initialization parameters dynamically with EM Express, but only for the current instance. Unless you also edit the text initialization parameter file and make the same change, the change is lost when you restart the database instance.

You can use SQL statements to create the following:

- A server parameter file from a text initialization file
- A server parameter file from the current (in-memory) values of all initialization parameters
- A text initialization parameter file from a server parameter file

When you create the database with DBCA, a server parameter file is created. This file is then used each time the database is started.

Creating a Database Using DBCA

DBCA enables you to create a database from predefined templates provided by Oracle or from templates that you or others have created. A template is a description of a database. Templates are described in more detail in "Managing DBCA Templates".

Selecting the Template

DBCA displays the templates that are available, which includes templates that Oracle ships with the DBCA product. These templates are described in "DBCA Templates Provided by Oracle". If you or others have created templates, those will be displayed also. You select the appropriate template for the database that you want to create. Clicking the "Show Details..." button displays specific information about the database defined by a template.

Including Datafiles

When you select a template, you also specify whether the database definition is to include datafiles. This determines whether you use a seed template (includes datafiles), or a non-seed template (does not include datafiles), to create your database.

Specifying Global Database Name and Database Features

The next page that DBCA displays enables you provide a global database name and SID.

Specifying Database Features

The "Database Features" page is presented only when you select a non-seed template. It enables you to include optional database features.

The following is a representative list of Oracle features that you can install in your database. Some of the listed options might already be included depending upon the database template that you selected. Those options that are already installed are noted as such (grayed out).

- Oracle Spatial
- Oracle Ultra Search

- Oracle Label Security
- Oracle Data Mining
- Oracle OLAP Services
- Sample Schemas

You can also display a list of standard database features. These are features that Oracle recommends you always install, but you have the option of excluding them. These include:

- Oracle JVM
- Oracle Text
- Oracle *inter*Media
- XDB Protocol

Specifying Mode, Initialization Parameters, and Datafiles

The next pages enable you to further define your database. You specify mode (dedicated server of shared server), set initialization parameters, and specify datafile locations. Oracle can determine specific values for you based upon your description of the database you are trying to create. For example, Oracle can choose appropriate settings for SGA memory sizing parameters depending upon whether you select a typical or custom database.

Completing Database Creation

After you have completed the specification of the parameters that define your database you can:

- Create the database now
- Save the description as a database template
- Generate database creation scripts

If you choose to generate scripts, you can use them to create the database later without not previously been configured for use with your database. This provides you the opportunity to add options and features that you did not include when you created the database. These options are discussed in "Specifying Database Features".

Deleting a Database Using DBCA

DBCA enables you to delete a database. When you do so, you delete the database instance and its control file(s), redo log files, and datafiles. Any server parameter file (SPFILE) or initialization parameter file used by the database is also deleted.

Introduction to the Data Dictionary

One of the most important parts of an Oracle database is its **data dictionary**, which is a **read-only** set of tables that provides information about the database. A data dictionary contains:

- The definitions of all schema objects in the database (tables, views, indexes, clusters, synonyms, sequences, procedures, functions, packages, triggers, and so on)
- How much space has been allocated for, and is currently used by, the schema objects
- Default values for columns
- Integrity constraint information
- The names of Oracle users
- Privileges and roles each user has been granted
- Auditing information, such as who has accessed or updated various schema objects
- Other general database information

The data dictionary is structured in tables and views, just like other database data. All the data dictionary tables and views for a given database are stored in that database's SYSTEM tablespace.

Not only is the data dictionary central to every Oracle database, it is an important tool for all users, from end users to application designers and database administrators. Use SQL statements to access the data dictionary. Because the data dictionary is read-only, you can issue only queries (SELECT statements) against it's tables and views.

Structure of the Data Dictionary

The data dictionary consists of the following:

Base Tables

The underlying tables that store information about the associated database. Only Oracle should write to and read these tables. Users rarely access them directly because they are normalized, and most of the data is stored in a cryptic format.

User-Accessible Views

The views that summarize and display the information stored in the base tables of the data dictionary. These views decode the base table data into useful information, such as user or table names, using joins and WHERE clauses to simplify the information. Most users are given access to the views rather than the base tables.

SYS, Owner of the Data Dictionary

The Oracle user SYS owns all base tables and user-accessible views of the data dictionary. No Oracle user should *ever* alter (UPDATE, DELETE, or INSERT) any rows or schema objects contained in the SYS schema, because such activity can compromise data integrity. The security administrator must keep strict control of this central account.

Caution:

Altering or manipulating the data in data dictionary tables can permanently and detrimentally affect the operation of a database.

How the Data Dictionary Is Used

The data dictionary has three primary uses:

- Oracle accesses the data dictionary to find information about users, schema objects, and storage structures.
- Oracle modifies the data dictionary every time that a data definition language (DDL) statement is issued.
- Any Oracle user can use the data dictionary as a read-only reference for information about the database.

How Oracle Uses the Data Dictionary

Data in the base tables of the data dictionary *is necessary for Oracle to function***.** Therefore, only Oracle should write or change data dictionary information. Oracle provides scripts to modify the data dictionary tables when a database is upgraded or downgraded.

During database operation, Oracle reads the data dictionary to ascertain that schema objects exist and that users have proper access to them. Oracle also updates the data dictionary continuously to reflect changes in database structures, auditing, grants, and data.

For example, if user Kathy creates a table named parts, then new rows are added to the data dictionary that reflect the new table, columns, segment, extents, and the privileges that Kathy has on the table. This new information is then visible the next time the dictionary views are queried.

Public Synonyms for Data Dictionary Views

Oracle creates public synonyms for many data dictionary views so users can access them conveniently. The security administrator can also create additional public synonyms for schema objects that are used systemwide. Users should avoid naming their own schema objects with the same names as those used for public synonyms.

Cache the Data Dictionary for Fast Access

Much of the data dictionary information is kept in the SGA in the **dictionary cache**, because Oracle constantly accesses the data dictionary during database operation to validate user access and to verify the state of schema objects. All information is stored in memory using the least recently used (LRU) algorithm.

Parsing information is typically kept in the caches. The COMMENTS columns describing the tables and their columns are not cached unless they are accessed frequently.

Other Programs and the Data Dictionary

Other Oracle products can reference existing views and create additional data dictionary tables or views of their own. Application developers who write programs that refer to the data dictionary should refer to the public synonyms rather than the underlying tables: the synonyms are less likely to change between software releases.

How to Use the Data Dictionary

The views of the data dictionary serve as a reference for all database users. Access the data dictionary views with SQL statements. Some views are accessible to all Oracle users, and others are intended for database administrators only.

The data dictionary is always available when the database is open. It resides in the SYSTEM tablespace, which is always online.

The data dictionary consists of sets of views. In many cases, a set consists of three views containing similar information and distinguished from each other by their prefixes:

Table 4-1 Data Dictionary View Prefixes

Prefix	**Scope**
USER	User's view (what is in the user's schema)
ALL	Expanded user's view (what the user can access)
DBA	Database administrator's view (what is in all users' schemas)

The set of columns is identical across views, with these exceptions:

- Views with the prefix USER usually exclude the column OWNER. This column is implied in the USER views to be the user issuing the query.
- Some DBA views have additional columns containing information useful to the administrator.

See Also:

Oracle9i Database Reference for a complete list of data dictionary views and their columns

Views with the Prefix USER

The views most likely to be of interest to typical database users are those with the prefix USER. These views:

- Refer to the user's own private environment in the database, including information about schema objects created by the user, grants made by the user, and so on
- Display only rows pertinent to the user
- Have columns identical to the other views, except that the column OWNER is implied
- Return a subset of the information in the ALL views
- Can have abbreviated PUBLIC synonyms for convenience

For example, the following query returns all the objects contained in your schema:

```
SELECT object_name, object_type FROM USER_OBJECTS;
```

Views with the Prefix ALL

Views with the prefix ALL refer to the user's overall perspective of the database. These views return information about schema objects to which the user has access through public or explicit grants of privileges and roles, in addition to schema objects that the user owns. For example, the following query returns information about all the objects to which you have access:

```
SELECT owner, object_name, object_type FROM ALL_OBJECTS;
```

Views with the Prefix DBA

Views with the prefix DBA show a global view of the entire database. Synonyms are not created for these views, because DBA views should be queried only by administrators. Therefore, to query the DBA views, administrators must prefix the view name with its owner, SYS, as in the following:

SELECT owner, object_name, object_type FROM SYS.DBA_OBJECTS;

Oracle recommends that you implement data dictionary protection to prevent users having the ANY system privileges from using such privileges on the data dictionary. If you enable dictionary protection (O7_DICTIONARY_ACCESSIBILITY is false), then access to objects in the SYS schema (dictionary objects) is restricted to users with the SYS schema. These users are SYS and those who connect as SYSDBA.

The DUAL Table

The table named DUAL is a small table in the data dictionary that Oracle and user-written programs can reference to guarantee a known result. This table has one column called DUMMY and one row containing the value X.

Dynamic Performance Tables

Throughout its operation, Oracle maintains a set of virtual tables that record current database activity. These tables are called **dynamic performance tables**.

Dynamic performance tables are not true tables, and they should not be accessed by most users. However, database administrators can query and create views on the tables and grant access to those views to other users. These views are sometimes called **fixed views** because they cannot be altered or removed by the database administrator.

SYS owns the dynamic performance tables; their names all begin with V_$. Views are created on these tables, and then public synonyms are created for the views. The synonym names begin with V$. For example, the V$DATAFILE view contains information about the database's datafiles, and the V$FIXED_TABLE view contains information about all of the dynamic performance tables and views in the database.

Database Object Metadata

The DBMS_METADATA package provides interfaces for extracting complete definitions of database objects. The definitions can be expressed either as XML or as SQL DDL. Two styles of interface are provided:

- A flexible, sophisticated interface for programmatic control
- A simplified interface for ad hoc querying using DBCA, or you can use them as a checklist

What Is a Control File?

Every Oracle Database has a **control file**, which is a small binary file that records the physical structure of the database. The control file includes:

- The database name
- Names and locations of associated datafiles and redo log files
- The timestamp of the database creation
- The current log sequence number
- Checkpoint information

The control file must be available for writing by the Oracle Database server whenever the database is open. Without the control file, the database cannot be mounted and recovery is difficult.

The control file of an Oracle Database is created at the same time as the database. By default, at least one copy of the control file is created during database creation. On some operating systems the default is to create multiple copies. You should create two or more copies of the control file during database creation. You can also create control files later, if you lose control files or want to change particular settings in the control files.

Provide Filenames for the Control Files

You specify control file names using the CONTROL_FILES initialization parameter in the database initialization parameter file (see "Creating Initial Control Files"). The instance recognizes and opens all the listed file during startup, and the instance writes to and maintains all listed control files during database operation.

If you do not specify files for CONTROL_FILES before database creation:

- If you are not using Oracle-managed files, then the database creates a control file and uses a default filename. The default name is operating system specific.
- If you are using Oracle-managed files, then the initialization parameters you set to enable that feature determine the name and location of the control files, as described in Chapter 15, "Using Oracle-Managed Files".
- If you are using Automatic Storage Management, you can place incomplete ASM filenames in the DB_CREATE_FILE_DEST and DB_RECOVERY_FILE_DEST initialization parameters. ASM then automatically creates control files in the appropriate places. See the sections "About ASM Filenames" and "Creating a Database That Uses ASM" in *Oracle Database Storage Administrator's Guide* for more information.

Multiplex Control Files on Different Disks

Every Oracle Database should have at least two control files, each stored on a different physical disk. If a control file is damaged due to a disk failure, the associated instance must be shut down. Once the disk drive is repaired, the damaged control file can be restored using the intact copy of the control file from the other disk and the instance can be restarted. In this case, no media recovery is required.

The behavior of multiplexed control files is this:

- The database writes to all filenames listed for the initialization parameter CONTROL_FILES in the database initialization parameter file.
- The database reads only the first file listed in the CONTROL_FILES parameter during database operation.
- If any of the control files become unavailable during database operation, the instance becomes inoperable and should be aborted.

One way to multiplex control files is to store a control file copy on every disk drive that stores members of redo log groups, if the redo log is multiplexed. By storing control files in these locations, you minimize the risk that all control files and all groups of the redo log will be lost in a single disk failure.

Back Up Control Files

It is very important that you back up your control files. This is true initially, and every time you change the physical structure of your database. Such structural changes include:

- Adding, dropping, or renaming datafiles
- Adding or dropping a tablespace, or altering the read/write state of the tablespace
- Adding or dropping redo log files or groups

The methods for backing up control files are discussed in "Backing Up Control Files".

Manage the Size of Control Files

The main determinants of the size of a control file are the values set for the MAXDATAFILES, MAXLOGFILES, MAXLOGMEMBERS, MAXLOGHISTORY, and MAXINSTANCES parameters in the CREATE DATABASE statement that created the associated database. Increasing the values of these parameters increases the size of a control file of the associated database.

What is the Redo Log?

Redo Logs consist of two or more pre-allocated files that store all changes made to the database as they occur. Every instance of an Oracle Database has associated online redo logs to protect the database in case of an instance failure.

Redo log files are filled with redo records. A redo record also called a redo entry, is made up of a group of change vectors, each of which is a description of a change made to a single block in the database.

Redo entries record data that you can use to reconstruct all changes made to the database, including the undo segments. Therefore, the redo log also protects rollback data. When you recover the database using redo data, the database reads the change vectors in the redo records and applies the changes to the relevant blocks.

Whenever a transaction is committed, LGWR writes the transaction redo records from the redo log buffer of the SGA to a redo log file and assigns a system change number (SCN) to identify the redo records for each committed transaction. Only when all redo records associated with a given transaction are safely on disk in the online logs is the user process notified that the transaction has been committed.

The Oracle Database uses only one redo log file at a time to store redo records written from the redo log buffer. The redo log file that LGWR is actively writing to is called the current redo log file. Redo log files that are required for instance recovery are called active redo log files. Redo log files that are no longer required for instance recovery are called inactive redo log files.
If you have enabled archiving (the database is in ARCHIVELOG mode), then the database cannot reuse or overwrite an active online log file until one of the archiver background processes (ARCn) has archived the file. If archiving is disabled (the database is in NOARCHIVELOG mode), then when the last redo log file is full, LGWR continues by overwriting the first available active file.

What are the Redo Log Data Dictionary Views?

The following views provide information on redo logs.

View	Description
V$LOG	Displays the redo log file information from the control file
V$LOGFILE	Identifies redo log groups and members and member status

V$LOG_HISTORY	Contains log history information

What is the Archived Redo log?

The Oracle Database lets you save filled groups of redo log files to one or more offline destinations, known collectively as the archived redo log. The process of turning redo log files into archived redo log files is called archiving. This process is only possible if the database is running in ARCHIVELOG mode. You can choose automatic or manual archiving.

An archived redo log file is a copy of one of the filled members of a redo log group. It includes the redo entries and the unique log sequence number of the identical member of the redo log group.

What is the difference between NOARCHIVELOG and ARCHIVELOG Mode?

The choice whether or not to enable the archiving of filled groups of redo log files depends on the availability and reliability requirements of the application running on the database. If you cannot afford to lose any data in your database in the event of a disk failure, use ARCHIVELOG mode.

Running a Database in NOARCHIVELOG Mode

– When you run your database in NOARCHIVELOG mode, you disable the archiving of the redo log. The database control file indicates that filled groups are not required to be archived.
– NOARCHIVELOG mode protects a database from instance failure but not from media failure. Only the most recent changes made to the database, which are stored in the online redo log groups, are available for instance recovery.
– In NOARCHIVELOG mode you cannot perform online tablespace backups, nor can you use online tablespace backups taken earlier while the database was in ARCHIVELOG mode.

Running a Database in ARCHIVELOG Mode

– When you run a database in ARCHIVELOG mode, you enable the archiving of the redo log. The database control file indicates that a group of filled redo log files cannot be reused by LGWR until the group is archived. A filled group becomes available for archiving immediately after a redo log switch occurs.
– The archiving of filled groups has these advantages:

- A database backup, together with online and archived redo log files, guarantees that you can recover all committed transactions in the event of an operating system or disk failure.
- If you keep archived logs, you can use a backup taken while the database is open and in normal system use.
- You can keep a standby database current with its original database by continually applying the original archived redo logs to the standby.

Managing Tablespaces and Datafiles in oracle 11g

Let us we discuss about what is tablespace? and it's types? How to create it? Mandatory tablespace in database **Image taken from Oracle Documentation**

(**Note: Here tablespace is logical ,datafile is physical)**

Tablespace: Tablespace is logical storage unit in oracle database which consists of one or more datafiles it can't be visible in the data file system

Tablespace Files :

Tablespace has different types in above picture i.e

1)system

2)sysaux

3)temp

4)undo

5)users

i)System tablespace:

-It is mandatory tablespace -It holds the information about all the data dictionary tables in database and it is created at the time of database creation -We can't offline these tablespace -It's very important tablespace -It 's always online when the database is open

ii)Sysaux tablespace

- The sysaux tablespace is system tablespace and auxiliary tablespace

-It is a new feature tablespace from 1og onwards -Half the size of system tablespace

-Snap shots -It helps on AWR report

iii)Temp Tablespace

-It used for database sorting operations -It store temp files

iv)Undo Tablespace

-Transaction recovery purpose -It has committed and uncommitted data

v)Users

-It store information for all objects created by nonsystem users -User tablespace is to store permanent user objects and data.

Features from 10g

1)Sysaux tablespace

2)We can rename tablespace

3)Temp group

4)Bigfile tablespace

Data blocks:

-It is smallest logical unit to store Oracle Data which means to represent specific number of bytes on physical hard disk

-default block is db_block_size=8k

Extends:

-It is a collection of contiguous data blocks

Segments:

-It's a set of extents allocated for specific data structure (like table or index).

-Various kind of segments are table, index, cluster, rollback, temporary

-Important views for segments are

dba_segments,

user_segments,

all_segments

Extend Managements

SQL> select tablespace_name, extent_management, allocation_type from dba_tablespaces;

TABLESPACE_NAME EXTENT_MAN ALLOCATIO ----------
-------------------- ---------- --------- SYSTEM DICTIONARY
USER SYS_UNDOTS LOCAL SYSTEM TEMP LOCAL
UNIFORM

Dictionary Managed Tablespace(DMT):

-Oracle allocates space segment (like table or index),free extents are stored in the dictionary are called dictionary managed tablespace

SQL> CREATE TABLESPACE ts1 DATAFILE
'u01/app/oracle/product/11.2.0.3/oradata/vip/ts1_01.dbf' SIZE 50M
EXTENT MANAGEMENT DICTIONARY
DEFAULT STORAGE (INITIAL 50K NEXT 50K MINEXTENTS 2 MAXEXTENTS
50 PCTINCREASE 0);

Locally Managed Tablepsace(LMT):

-Oracle allocates space segment (like table or index),free extents are stored in the tablespace header are called Locally managed tablespace

SQL> CREATE TABLESPACE ts2 DATAFILE
'u01/app/oracle/product/11.2.0.3/oradata/vip/ts2_01.dbf' SIZE 50M
EXTENT MANAGEMENT LOCAL AUTOALLOCATE;

SQL> CREATE TABLESPACE ts3 DATAFILE
'u01/app/oracle/product/11.2.0.3/oradata/vip/ts3_01.dbf' SIZE 50M
EXTENT MANAGEMENT LOCAL UNIFORM SIZE 128K;

Auto allocated and Uniform:

-Auto allocate specifies the extend size are system managed .optimal next extend started at 64kb and it ll grow 1MB, 8MB, and eventually to 64MB.

-mostly Recommended for low or unmanaged environment

- Uniform specifies the tablespace is managed with uniform extents of SIZE bytes (use K or M to specify the extent size in kilobytes or megabytes). The default size is 1M

Tablespace Creation syntax:

```
SQL> CREATE [TEMPORARY / UNDO] TABLESPACE  <tblspc_name>
 DATAFILE / TEMPFILE '<datafile01_name and Path where file to create>' SIZE <integer
M>[,
                    '<datafile02_name and Path where file to create>' SIZE <integer M>[,
                    '<datafile0N_name and Path where file to create>' SIZE <integer M>[,...]]]
 BLOCKSIZE <DB_BLOCK_SIZE parameter /2k/4k/8k/16k/32k >
 AUTOEXTEND { [OFF/ON (NEXT <integer K/M > MAXSIZE<integer K/M >) /
UNLIMITED] }
 LOGGING/NOLOGGING (Logging default)
 ONLINE/OFFLINE (Online default)
 EXTENT MANAGEMENT { [DICTIONARY] /
              [LOCAL Default (AUTOALLOCATE / UNIFORM SIZE <integer K/M >)] }
 PERMANENT / TEMPORARY (Permanent default)
 MINIMUM EXTENT
 DEFAULT STORAGE  {    [INITIAL <integer K/M >]
                  [NEXT <integer K/M >]
                  [PCTINCREASE <integer K/M >]
                  [MINEXTENTS <integer>]
                  [MAXEXTENTS <integer> / UNLIMITED]
                  [FREELISTS <integer>]
                  [FREELIST GROUPS <integer>]
                  [OPTIMAL <integer>/NULL]
                  [BUFFER_POOL < DEFAULT/KEEP/RECYCLE >]
 } CHUNK <integer K/M >
 NOCACHE;
```

*BLOCKSIZE - By default block size is defined by this parameter DB_BLOCK_SIZE.In oracle 9i multiple blocksize that is different block size for different tablespaces, can be defined, all datafiles of a same tablespace have the same block size.

*DEFAULT STORAGE :

-INITIAL – Specifies the size of the object's first extent.3 k minmum for Locally and 2 k minimum Dictionary.

-NEXT – Specifies the size of the object's sucessive extent.

-PCTINCREASE – Specifies the ratio of the third or the preceding extent of the object. The default value for PCTINCREASE is 50 % and the minimum value is 0%.

-MINEXTENTS – The total number of extent allocated to the segment at the time of creation

-MAXEXTENTS – The maximum number of extent that can be allocated to the segment .

*MININUM EXTENT – The size is specifies in this clause.The extent are multiple of the size specified in this clause .NEXT and INITIAL extent size specified should be multiple of minmum extent.

*PERMANENT / TEMPORARY – Permanent is default, use to store the table,index etc,Temporary is for temporay segments(sorts in Sql) can not store table,index in temporary tablespace.

*LOGGING / NOLOGGING – Logging is default,the DDL operation & direct insert load are recorded in the redo log file.

*ONLINE / OFFLINE - Online is default,tablespace is available as soon as created.

Bigfile Tablespace:

-It's special kind of tablespace which was introduced on oracle 10g and it have a single datafile.

-Bigfile tablespaces can be up to 4G blocks that 128 TB with 32k blocks

-We can resize the tablespace and datafiles ,but we can't add new datafiles

SQL> create bigfile tablespace Don datafile '/u01/app/oracle/product/11.2.0.3/oradata/vip/don.dbf' size 20m;

Resize:

SQL> alter tablespace don resize 20m;

Rename:

SQL> alter tablespace don rename to sam;

Smallfile Tablespace:

-It are restricted datafiles -We can't resize tablespace size but we can do datafile size

SQL> create tablesspace sam datafile '/u01/app/oracle/product/11.2.0.3/oradata/vip/sam.dbf' size 10m;

Adding Datafiles:

SQL> alter tablespace sam add datafile '/u01/app/oracle/product/11.2.0.3/oradata/vip/sam02.dbf' size 30m;

Resize Existing Datafiles:

SQL> alter database datafile '/u01/app/oracle/product/11.2.0.3/oradata/vip/sam02.dbf' resize 35m;

Drop Tablespace:

Drop Tablespace is remove tablespace from oracle

SQL> DROP TABLESPACE tablespace_name
[INCLUDING CONTENTS [{AND DATAFILES | KEEP DATAFILES
] [CASCADE CONSTRAINTS]] ;

Managing Tables and Views

After creating a database and (optionally) schemas within it, you can create tables and views. You can also create tables and views within the default yellowbrick database.

Database Tables

Yellowbrick supports persistent tables and temporary tables.

- **Persistent tables** remain in the database until they are explicitly dropped.
- **Temporary tables** are *local* to the session in which they are created and are dropped when the session ends.

You can use CREATE TABLE, CTAS, and SELECT INTO statements to create tables.

External Tables

Yellowbrick also supports external tables, which are tables that are stored in files outside the database. You can create an external table by "unloading" the results of any valid Yellowbrick query into a specific file on an NFS-mounted file system. You can also read from an external table by selecting from it directly or by using CTAS or INSERT statements to select external data that creates or updates other tables.

See CREATE EXTERNAL TABLE.

Table Maintenance

Aside from loading tables, you may need to maintain them in a few other ways:

Altering and Dropping Tables

Yellowbrick supports basic ALTER TABLE and DROP TABLE commands. You can add columns, rename a table, and change the owner of the table.

You can also GRANT and REVOKE privileges at the table and column level.

Inserting, Updating, and Deleting Rows

Yellowbrick supports standard SQL INSERT, UPDATE, and DELETE commands.

You can also run the \copy command to load smaller tables from files via ybsql. In general, you should use the bulk loader (ybload) to load large tables.

Note: The Yellowbrick column store is designed to optimally store, compress, and organize data for high-volume read-oriented analytics. As a result, when data is deleted it continues to exist in the system for a period of time until it can be efficiently removed and data can be reorganized. These optimizations are fully automatic and require no administration or manual interaction.

Analyzing Tables

By default, tables are automatically analyzed. This analysis updates column-level statistics that the planner uses to optimize query execution. See Auto-Analyzing Tables.

Flushing Tables

A background flush command periodically updates the Yellowbrick backend storage with rows that are temporarily stored in the front end database. See Flushing Tables.

Table DDL

Use the SMC to copy the DDL for a table to the clipboard. Go to **Databases** and select a database. Then double-click the table ID. On the Details screen, click **Definition** > **Copy to Clipboard**.

Creating Indexes

This section describes how to create indexes. To create an index in your own schema, *at least one* of the following conditions must be true:

- The table or cluster to be indexed is in your own schema.
- You have INDEX privilege on the table to be indexed.
- You have CREATE ANY INDEX system privilege.

To create an index in another schema, *all* of the following conditions must be true:

- You have CREATE ANY INDEX system privilege.
- The owner of the other schema has a quota for the tablespaces to contain the index or index partitions, or UNLIMITED TABLESPACE system privilege.

CREATE INDEX

Purpose

Use the CREATE INDEX statement to create an index on:

- One or more columns of a table, a partitioned table, an index-organized table, or a cluster
- One or more scalar typed object attributes of a table or a cluster
- A nested table storage table for indexing a nested table column

An **index** is a schema object that contains an entry for each value that appears in the indexed column(s) of the table or cluster and provides direct, fast access to rows. Oracle Database supports several types of index:

- Normal indexes. (By default, Oracle Database creates B-tree indexes.)
- **Bitmap indexes**, which store rowids associated with a key value as a bitmap
- **Partitioned indexes**, which consist of partitions containing an entry for each value that appears in the indexed column(s) of the table
- **Function-based indexes**, which are based on expressions. They enable you to construct queries that evaluate the value returned by an expression, which in turn may include built-in or user-defined functions.
- **Domain indexes**, which are instances of an application-specific index of type *indextype*

constraint

Purpose

Use a *constraint* to define an **integrity constraint**--a rule that restricts the values in a database. Oracle Database lets you create six types of constraints and lets you declare them in two ways.

The six types of integrity constraint are described briefly here and more fully in "Semantics":

- A **NOT NULL constraint** prohibits a database value from being null.
- A **unique constraint** prohibits multiple rows from having the same value in the same column or combination of columns but allows some values to be null.
- A **primary key constraint** combines a NOT NULL constraint and a unique constraint in a single declaration. That is, it prohibits multiple rows from having the same value in the same column or combination of columns and prohibits values from being null.
- A **foreign key constraint** requires values in one table to match values in another table.
- A **check constraint** requires a value in the database to comply with a specified condition.
- A REF column by definition references an object in another object type or in a relational table. A **REF constraint** lets you further describe the relationship between the REF column and the object it references.

You can define constraints syntactically in two ways:

- As part of the definition of an individual column or attribute. This is called **inline** specification.
- As part of the table definition. This is called **out-of-line** specification.

NOT NULL constraints must be declared inline. All other constraints can be declared either inline or out of line.

Constraint clauses can appear in the following statements:

- CREATE TABLE (see CREATE TABLE)
- ALTER TABLE (see ALTER TABLE)
- CREATE VIEW (see CREATE VIEW)
- ALTER VIEW (see ALTER VIEW)

View Constraints Oracle Database does not enforce view constraints. However, you can enforce constraints on views through constraints on base tables.

You can specify only unique, primary key, and foreign key constraints on views, and they are supported only in DISABLE NOVALIDATE mode. You cannot define view constraints on attributes of an object column.

Prerequisites

You must have the privileges necessary to issue the statement in which you are defining the constraint.

To create a foreign key constraint, in addition, the parent table or view must be in your own schema or you must have the REFERENCES privilege on the columns of the referenced key in the parent table or view.

NOT NULL Constraints

A NOT NULL constraint prohibits a column from containing nulls. The NULL keyword by itself does not actually define an integrity constraint, but you can specify it to explicitly permit a column to contain nulls. You must define NOT NULL and NULL using inline specification. If you specify neither NOT NULL nor NULL, then the default is NULL.

NOT NULL constraints are the only constraints you can specify inline on XMLType and VARRAY columns.

To satisfy a NOT NULL constraint, every row in the table must contain a value for the column.

Restrictions on NOT NULL Constraints NOT NULL constraints are subject to the following restrictions:

- You cannot specify NULL or NOT NULL in a view constraint.
- You cannot specify NULL or NOT NULL for an attribute of an object. Instead, use a CHECK constraint with the IS [NOT] NULL condition.

See Also:

"Attribute-Level Constraints Example" and "NOT NULL Example"

Unique Constraints

A **unique** constraint designates a column as a unique key. A **composite unique key** designates a combination of columns as the unique key. When you define a unique constraint inline, you need only the UNIQUE keyword. When you define a unique constraint out of line, you must also specify one or more columns. You must define a composite unique key out of line.

To satisfy a unique constraint, no two rows in the table can have the same value for the unique key. However, the unique key made up of a single column can contain nulls. To satisfy a composite unique key, no two rows in the table or view can have the same combination of values in the key columns. Any row that contains nulls in all key columns automatically satisfies the constraint. However, two rows that contain nulls for one or more key columns and the same combination of values for the other key columns violate the constraint.

When you specify a unique constraint on one or more columns, Oracle implicitly creates an index on the unique key. If you are defining uniqueness for purposes of query performance, then Oracle recommends that you instead create the unique index explicitly using a CREATE UNIQUE INDEX statement. You can also use the CREATE UNIQUE INDEX statement to create a unique function-based index that defines a conditional unique constraint. See "Using a Function-based Index to Define Conditional Uniqueness: Example" for more information.

Restrictions on Unique Constraints Unique constraints are subject to the following restrictions:

- None of the columns in the unique key can be of LOB, LONG, LONG RAW, VARRAY, NESTED TABLE, OBJECT, REF, TIMESTAMP WITH TIME ZONE, or user-defined type. However, the unique key can contain a column of TIMESTAMP WITH LOCAL TIME ZONE.
- A composite unique key cannot have more than 32 columns.
- You cannot designate the same column or combination of columns as both a primary key and a unique key.
- You cannot specify a unique key when creating a subview in an inheritance hierarchy. The unique key can be specified only for the top-level (root) view.

Primary Key Constraints

A **primary key** constraint designates a column as the primary key of a table or view.
A **composite primary key** designates a combination of columns as the primary key. When you define a primary key constraint inline, you need only the PRIMARY KEY keywords. When you define a primary key constraint out of line, you must also specify one or more columns. You must define a composite primary key out of line.

A primary key constraint combines a NOT NULL and unique constraint in one declaration. Therefore, to satisfy a primary key constraint:

- No primary key value can appear in more than one row in the table.
- No column that is part of the primary key can contain a null.

Restrictions on Primary Key Constraints Primary constraints are subject to the following restrictions:

- A table or view can have only one primary key.
- None of the columns in the primary key can be LOB, LONG, LONG RAW, VARRAY, NESTED TABLE, BFILE, REF, TIMESTAMP WITH TIME ZONE, or user-defined type. However, the primary key can contain a column of TIMESTAMP WITH LOCAL TIME ZONE.
- The size of the primary key cannot exceed approximately one database block.
- A composite primary key cannot have more than 32 columns.
- You cannot designate the same column or combination of columns as both a primary key and a unique key.
- You cannot specify a primary key when creating a subview in an inheritance hierarchy. The primary key can be specified only for the top-level (root) view.
- **Foreign Key Constraints**
- A **foreign key constraint** (also called a **referential integrity constraint**) designates a column as the foreign key and establishes a relationship between that foreign key and a specified primary or unique key, called the **referenced key**. A **composite foreign key** designates a combination of columns as the foreign key.
- The table or view containing the foreign key is called the **child** object, and the table or view containing the referenced key is called the **parent** object. The foreign key and the referenced key can be in the same table or view. In this case, the parent and child tables are the same. If you identify only the parent table or view and omit the column name, then the foreign key automatically references the primary key of the parent table or view. The corresponding column or columns of the foreign key and the referenced key must match in order and datatype.
- You can define a foreign key constraint on a single key column either inline or out of line. You must specify a composite foreign key and a foreign key on an attribute out of line.
- To satisfy a composite foreign key constraint, the composite foreign key must refer to a composite unique key or a composite primary key in the parent table or view, or the value of at least one of the columns of the foreign key must be null.
- You can designate the same column or combination of columns as both a foreign key and a primary or unique key. You can also designate the same column or combination of columns as both a foreign key and a cluster key.
- You can define multiple foreign keys in a table or view. Also, a single column can be part of more than one foreign key.
- **Restrictions on Foreign Key Constraints** Foreign key constraints are subject to the following restrictions:
- None of the columns in the foreign key can be of LOB, LONG, LONG RAW, VARRAY, NESTED TABLE, BFILE, REF, TIMESTAMP WITH TIME ZONE, or user-defined type. However, the primary key can contain a column of TIMESTAMP WITH LOCAL TIME ZONE.

- The referenced unique or primary key constraint on the parent table or view must already be defined.
- A composite foreign key cannot have more than 32 columns.
- The child and parent tables must be on the same database. To enable referential integrity constraints across nodes of a distributed database, you must use database triggers. See CREATE TRIGGER.
- If either the child or parent object is a view, then the constraint is subject to all restrictions on view constraints. See "View Constraints".
- You cannot define a foreign key constraint in a CREATE TABLE statement that contains an AS *subquery* clause. Instead, you must create the table without the constraint and then add it later with an ALTER TABLE statement.

ON DELETE Clause The ON DELETE clause lets you determine how Oracle Database automatically maintains referential integrity if you remove a referenced primary or unique key value. If you omit this clause, then Oracle does not allow you to delete referenced key values in the parent table that have dependent rows in the child table.

- Specify CASCADE if you want Oracle to remove dependent foreign key values.
- Specify SET NULL if you want Oracle to convert dependent foreign key values to NULL.

Restriction on ON DELETE You cannot specify this clause for a view constraint.

Check Constraints

A **check constraint** lets you specify a condition that each row in the table must satisfy. To satisfy the constraint, each row in the table must make the condition either TRUE or unknown (due to a null). When Oracle evaluates a check constraint condition for a particular row, any column names in the condition refer to the column values in that row.

The syntax for inline and out-of-line specification of check constraints is the same. However, inline specification can refer only to the column (or the attributes of the column if it is an object column) currently being defined, whereas out-of-line specification can refer to multiple columns or attributes.

Oracle does not verify that conditions of check constraints are not mutually exclusive. Therefore, if you create multiple check constraints for a column, design them carefully so their purposes do not conflict. Do not assume any particular order of evaluation of the conditions

Restrictions on Check Constraints Check constraints are subject to the following restrictions:

- You cannot specify a check constraint for a view. However, you can define the view using the WITH CHECK OPTION clause, which is equivalent to specifying a check constraint for the view.
- The condition of a check constraint can refer to any column in the table, but it cannot refer to columns of other tables.

- Conditions of check constraints cannot contain the following constructs: o Subqueries and scalar subquery expressions
  - o Calls to the functions that are not deterministic (CURRENT_DATE, CURRENT_TIMESTAMP, DBTIMEZONE, LOCALTIMESTAMP, SESSIONTIMEZONE, SYSDATE, SYSTIMESTAMP, UID, USER, and USERENV)
  - o Calls to user-defined functions
  - o Dereferencing of REF columns (for example, using the DEREF function) o Nested table columns or attributes
  - o The pseudocolumns CURRVAL, NEXTVAL, LEVEL, or ROWNUM o Date constants that are not fully specified

Managing Users and Security

Users access Oracle Database Express Edition through database user accounts. Some of these accounts are automatically created administrative accounts—accounts with database administration privileges. You log in to these administrative accounts to create and manage other user accounts, maintain database security, and perform other database administration tasks.

About User Accounts

A user account is identified by a user name and defines the user's attributes, including the following:

- Password for database authentication
- Privileges and roles
- Default tablespace for database objects
- Default temporary tablespace for query processing work space

When you create a user, you are also implicitly creating a schema for that user. A **schema** is a logical container for the database objects (such as tables, views, triggers, and so on) that the user creates. The schema name is the same as the user name, and can be used to unambiguously refer to objects owned by the user. For example, **HR.EMPLOYEES** refers to the table

named **EMPLOYEES** in the **HR** schema. (The **EMPLOYEES** table is owned by **HR**.) The terms *database object* and *schema object* are used interchangeably.

When you drop (delete) a user, you must either first drop all the user's schema objects, or use the **cascade** feature of the drop operation, which simultaneously drops a user and all of that user's schema objects.

User Privileges and Roles

When creating a user, you grant privileges to enable the user to connect to the database, to run queries and make updates, and to create schema objects. There are two main types of user privileges:

- **System privileges**—A system privilege is the right to perform a particular action, or to perform an action on any schema objects of a particular type. For example, the privileges to create tables and to delete the rows of any table in a database are system privileges.
- **Object privileges**—An object privilege is a right to perform a particular action on a specific schema object. Different object privileges are available for different types of schema objects. The privilege to delete rows from the **DEPARTMENTS** table is an example of an object privilege.

Managing and controlling privileges is made easier by using **roles**, which are named groups of related privileges. You create roles, grant system and object privileges to the roles, and then grant roles to users. Unlike schema objects, roles are not contained in any schema.

Oracle Database Express Edition comes with some predefined roles:

- The **DBA** role enables a user to perform most administrative functions, including creating users and granting privileges; creating and granting roles; creating and dropping schema objects in other users' schemas; and more. It grants all system privileges, but does not include the privileges to start up or shut down the database. It is by default granted to user SYSTEM. You should be very cautious about assigning the DBA role to any other database users.
- Use of the **CONNECT** and **RESOURCE** roles is discouraged. Instead, grant only those privileges that the specific user will need. For example:

```
grant CREATE SESSION, ALTER SESSION, CREATE DATABASE LINK, -
  CREATE MATERIALIZED VIEW, CREATE PROCEDURE, CREATE
PUBLIC SYNONYM, -
  CREATE ROLE, CREATE SEQUENCE, CREATE SYNONYM, CREATE TABLE, -
  CREATE TRIGGER, CREATE TYPE, CREATE VIEW, UNLIMITED TABLESPACE
-
  to chris;
```

Internal User Accounts

Certain user accounts are created automatically for database administration. Examples are **SYS** and **SYSTEM**. Other accounts are automatically created just so that individual Oracle Database XE features or products can have their own schemas. An example is the **CTXSYS** account, which is used by the Oracle Text product. Oracle Text is used to index the Oracle Database XE online Help. The Help index is stored in the **CTXSYS** schema in the database.

These automatically created accounts are called **internal user accounts**, and their schemas are called **internal schemas**.

The only internal accounts that you may log in with are the **SYS** and **SYSTEM** accounts, although it is recommended that you avoid logging in with the **SYS** account. Do not attempt to log in with other internal accounts. See "The SYS and SYSTEM Users" for more information.

About Administrative Accounts and Privileges

Administrative accounts and privileges enable you to perform administrative functions like managing users, managing database memory, and starting up and shutting down the database.

The SYS and SYSTEM Users

The following administrative user accounts are automatically created when you install Oracle Database Express Edition (Oracle Database XE). They are both created with the password that you supplied upon installation (Windows operating systems) or configuration (Linux operating systems).

- **SYSTEM**

  This is the user account that you log in with to perform all administrative functions other than starting up and shutting down the database.

- **SYS**

  All base tables and views for the database data dictionary are stored in the **SYS** schema. These base tables and views are critical for the operation of Oracle Database XE. To maintain the integrity of the data dictionary, tables in the **SYS** schema are manipulated only by the database. They should never be modified by any user or database administrator. You must not create any tables in the **SYS** schema.

  There is typically no reason to log in as user **SYS**. User **SYSTEM** is preferred for all administrative tasks except starting up and shutting down. See "Starting Up and Shutting Down" for more information.

The SYSDBA System Privilege

SYSDBA is a system privilege that is assigned only to user **SYS**. It enables **SYS** to perform high-level administrative tasks such as starting up and shutting down the database.

Although under typical circumstances it is not necessary to log in to the database as user **SYS**, if you want to log in as **SYS** with the SQL Command Line (SQL*Plus), you must connect to the database "**AS SYSDBA**." Connecting **AS SYSDBA** invokes the **SYSDBA** privilege. If you omit the **AS SYSDBA** clause when logging in as user **SYS**, the SQL Command Line rejects the login attempt.

The following example illustrates how to connect to the database with the **SYSDBA** privilege from the SQL Command Line:

```
SQL > connect sys/password as sysdba
```

password is the password for the **SYS** user account.

Operating System Authentication

Operating system authentication (OS authentication) is a way of using operating system login credentials to authenticate database users. One aspect of OS authentication can be used to authenticate database administrators. If you log in to the Oracle Database XE host computer with a user name that is in a special operating system user group, you are then permitted to connect to the database with the **SYSDBA** privilege. An administrator who is authenticated through OS authentication does not need to know the **SYS** or **SYSTEM** account password.

OS authentication is needed because there must be a way to identify administrative users even if the database is shut down. A user authenticated in this way can then start up the database. (See "Starting Up and Shutting Down" for more information.)

Table:- lists the operating system user groups whose member users can connect to the database with the **SYSDBA** privilege.

Table :- Operating System User Groups for OS Authentication

Platform	Operating System User Group Name
Linux	**Dba**
Windows	**ORA_DBA**

On each platform, if the OS authentication user group does not already exist, it is automatically created when you install Oracle Database XE. In addition, upon installation on the Linux

platform, the user account **oracle** is automatically created and placed in the **dba** group. Upon installation on the Windows platform, the user performing the installation is automatically added to the **ORA_DBA** group. On both platforms, you can add other host users to the OS authentication user group to enable them to connect to the database with the **SYSDBA** privilege.

Logging In as an Administrator

There are three ways to log in to Oracle Database Express Edition (Oracle Database XE) to perform administrative tasks:

- Log in as user **SYSTEM**
- Log in as a user who has been granted the **DBA** role
- Log in and connect to the database as **SYSDBA**

Table 7-2 provides information about each of these login methods.

Table 7-2 Database Administrator Login Methods

Login Method	Permitted In	Notes
Log in to the database as user **SYSTEM**	The Oracle Database XE graphical user interface and the SQL Command Line	For routine administrative tasks like managing memory managing users. You must supply the password for the **SYSTEM** user.
Log in to the database as a user who has been granted the **DBA** role	The Oracle Database XE graphical user interface and the SQL Command Line	For routine administrative tasks like managing users. A administrator must first grant the **DBA** role to the user.
Log in and connect to the database as **SYSDBA**	the SQL Command Line	For high-level administrative tasks like starting up and s down the database, and changing the **SYS** password. Y connect as **SYSDBA** using the **SYS** user name and pass using operating system authentication.

Logging In as User SYSTEM

You can log in as user **SYSTEM** using either of the following methods:

- Using SQL Developer, open a database connection to the SYSTEM user.
- Using the SQL Command Line, enter the following statement:

```
SQL> CONNECT SYSTEM/<password>;
```

Logging In as a User with the DBA Role

The procedures for logging in as a user who has been granted the **DBA** role are the same as those for logging in as user **SYSTEM**, with the following exceptions:

- When logging in, you must supply the user name and password for this user account.
- An administrator must have previously logged in and granted the **DBA** role to this user.

  See "User Privileges and Roles" for more information.

Logging In and Connecting to the Database as SYSDBA

You can log in and connect as **SYSDBA** using either of the following methods:

- Using SQL Developer, open a database connection to the **SYS** user **AS SYSDBA**.
- Using the SQL Command Line, enter one the following

  statements. To use database authentication:

```
SQL> CONNECT SYS/<password> AS SYSDBA;
```

  To use operating system (OS) authentication:

```
SQL> CONNECT / AS SYSDBA;
```

  The slash (/) indicates that the database should authenticate you with operating system (OS) authentication. Remember that when you connect with OS authentication, you are effectively logging in to the database as user **SYS**.

Changing Administrative User Passwords

To change the password for user **SYS** or **SYSTEM**:

1. Using the SQL Command Line, connect to the database as **SYSDBA**.

   See "Logging In and Connecting to the Database as SYSDBA" for instructions.

2. Enter one of the following commands:

```
3. ALTER USER SYS IDENTIFIED BY newpassword;

4. ALTER USER SYSTEM IDENTIFIED BY newpassword;
```

where ***newpassword*** is the desired new password.

Managing Database Users

You can use SQL Developer or the SQL Command Line (SQL*Plus) to manage database users. This section discusses using SQL Developer, and contains the following topics:

- Creating Users
- Altering Users
- Locking and Unlocking User Accounts
- Expiring a User Password
- Dropping Users

To perform these operations, in the SQL Developer Connections navigator, open a connection

1. In the SQL Developer Connections navigator, open a connection to the **SYSTEM** user.
2. In the nodes under this **SYSTEM** connection, expand **Other Users**.

   This displays nodes for all database users, including several Oracle-supplied internal users. The Connections navigator hierarchy may look like this:

   Connections

```
. . .
SYSTEM Views
  Editioning
  Views
  . . .
  Other Users
    ANONYMOU
    S
    APEX_040000
    APEX_PUBLIC_USER
    APQQOSSYS CHRIS

    CTXSYS
    . . .
```

HR

MDSYS

3. To create a new database user, right-click the **Other Users** node in the Connections navigator and select **Create User**.
4. To perform an action on a database user, right-click that user in the hierarchy and select the appropriate command (**Edit User** or **Drop User**).

Creating Users

To create a new database user, right-click the **Other Users** node in the SQL Developer Connections navigator and select **Create User**. Before creating a user, determine the following:

- Whether or not you want to permit the user to create database objects in that user's own schema.

  If so, on the Create Database User page, grant individual session-related and create object system privileges. See the following topics for more information:

  - "User Privileges and Roles" for details on privileges and roles
  - "Configuring Privilege and Role Authorization" in Oracle Database Security Guide for more information on system privileges
  - "Creating and Managing Schema Objects" in Oracle Database Express Edition 2 Day Developer's Guide for more information on database objects
- Whether or not you want to grant the user DBA privileges.

  If so, on the Create Database User page, grant the **DBA** role. See "User Privileges and Roles" for details on the **DBA** role.

  Because DBA privileges include the ability to create database objects in any schema, if you grant the **DBA** role, you do not need to grant individual create object system privileges.

Example: Creating a User

Suppose you want to create a user account for a database application developer named Nick. Because Nick is a developer, you want to grant him all **CREATE** system privileges so that he can create the schema objects that his applications require. In addition, you want to create his account with the password *firesign*.

To create the user **Nick**:

1. Right-click the **Other Users** node in the SQL Developer Connections navigator and select **Create User**.

2. In the Create/Edit User dialog box, for the **User** tab, enter the information shown in the following figure:

   User Name: **NICK**

   New Password and **Confirm Password**: Desired password for the user.

   Password expired (user must change): Select or not, as desired. (It is not selected in the figure.)

   Account is Locked: Select or not, as desired. (It is not selected in the figure.)

   Edition Enabled: Select or not, as desired. (It is not selected in the figure.)

   Default Tablespace: **USERS**

   Temporary Tablespace: **TEMP**

3. In the Create/Edit User dialog box, click the **System Privileges** tab, and under Granted select the following privileges because you want to be sure that **NICK** will have them):

- **ALTER SESSION**
- **CREATE SESSION**
- **CREATE DATABASE LINK**
- **CREATE MATERIALIZED VIEW**
- **CREATE PROCEDURE**
- **CREATE PUBLIC SYNONYM**
- **CREATE ROLE**
- **CREATE SEQUENCE**
- **CREATE SYNONYM**
- **CREATE TABLE**
- **CREATE TRIGGER**
- **CREATE TYPE**
- **CREATE VIEW**
- **UNLIMITED TABLESPACE**

4. In the Create/Edit User dialog box, click **Apply**, then click

Close. Altering Users

You can use the Manage Database Users page to alter a user. Altering a user means changing some of that user's attributes. You can change all user attributes except the user name, default tablespace, and temporary tablespace. If you want to change the user name, you must drop the user and re-create that user with a different name. (Before you drop the user, ensure that the user's schema objects are either no longer needed or are backed up (for example, by exporting them). See "Dropping Users" for more information.)

One of the attributes that you can alter is the user password. If you do this, you must either communicate the new password to the user, or request the new password from the user and then enter it. An easier and more secure way to cause a password change is to expire the password. When you **expire** a password, the user is prompted to change the password at the next login. See "Expiring a User Password" for more information.

Example: Altering a User

Suppose Nick is promoted to senior developer, and he has shown an interest in helping with routine database administration tasks. You decide to grant the **DBA** role to Nick.

To alter Nick's user account:

1. In the SQL Developer Connections navigator, expand the **SYSTEM** connection and right-click the **Other Users** node.
2. Right-click **NICK** and select **Edit User**.
3. In the Create/Edit User dialog box, click the **Roles** tab.
4. Under Granted, select **DBA**.
5. In the Create/Edit User dialog box, click **Apply**, then click **Close**.

Locking and Unlocking User Accounts

To temporarily deny access to the database for a particular user, you can lock the user account. If the user then attempts to connect, the database displays an error message and disallows the connection. You can unlock the user account when you want to allow database access again for that user.

To lock or unlock a user account:

1. In the SQL Developer Connections navigator, expand the **SYSTEM** connection and right-click the **Other Users** node.
2. Right-click the desired user and select **Edit User**.
3. In the Create/Edit User dialog box, in the User tab, select or deselect **Account is Locked**: selecting (checking) causes the account to be locked; deselecting (unchecking) causes the account to be unlocked.
4. In the Create/Edit User dialog box, click **Apply**, then click **Close**.

Expiring a User Password

When you expire a user password, the user is prompted to change the password at the next login. Reasons to expire a password include the following:

- A user password becomes compromised.
- You have a security policy in place that requires users to change their passwords on a regular basis.
- A user has forgotten his or her password.

In this case, you alter the user account, assign a new temporary password, and expire the password. The user then logs in with the temporary password and is prompted to choose a new password.

See "Altering Users" for more information.

Example: Expiring a Password

Suppose Nick's password becomes compromised, and you want to assign him a new one. The easiest way to do this is to expire his current password. The next time that Nick logs in with the compromised password, he is prompted to choose a new password.

To expire Nick's password:

1. In the SQL Developer Connections navigator, expand the **SYSTEM** connection and right-click the **Other Users** node.
2. Right-click **NICK** and select **Edit User**.
3. In the Create/Edit User dialog box, in the User tab, select (check) **Password expired (user must change next login)**.
4. In the Create/Edit User dialog box, click **Apply**, then click **Close**.

Dropping Users

Dropping a user removes the user from the database. Before you can drop a user, you must first drop all the user's schema objects. Or, you can use the **cascade** feature of the drop operation, which simultaneously drops a user and also that user's schema objects. The following are two alternatives to dropping a user and losing all the user's schema objects:

- To temporarily deny access to the database for a particular user while preserving the user's schema objects, you can lock the user account. See "Locking and Unlocking User Accounts" for more information.
- To drop a user but retain the data from the user's tables, export the tables first. See "Exporting and Importing Metadata and Data" for instructions.

Example: Dropping a User

Suppose Nick's project is canceled and Nick takes a position in another department. You want to drop the user **NICK** and all associated schema objects.

To drop user **NICK** and all his owned schema objects:

1. In the SQL Developer Connections navigator, expand the **SYSTEM** connection and right-click the **Other Users** node.
2. Right-click **NICK** and select **Drop User**.
3. In the Drop User dialog box, (check) **Cascade**

This indicates that you want to drop the user's schema objects also. If the user has schema objects and you do not select this option, you receive an error message if you attempt to complete the drop operation.

4. In the Drop User dialog box, click **Apply**.

QUESTION BANK
Unit-2

Q1. What is Oracle Net Architecture. Explain with the help of Diagram.
Q2. Explain the usage of Oracle shared server architecture.
Q3. Explain the process of Backup and recovery in Oracle.
Q4. How to transport data. Also Explain Export utility.
Q5.What is import utility in Oracle?
Q6. Explain the Process of Loading data into database?
Q7. Explain database Performance tuning.

Note:- These are the questions which we can expect in your university exams. you can do your preparation for your university exams based on these questions.

UNIT -2

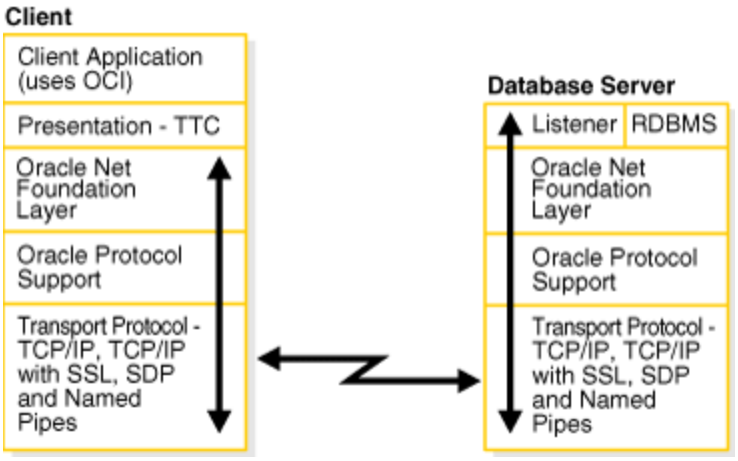
Understanding Oracle Net Architecture

The Oracle Net listener is an application positioned on top of the Oracle Net foundation layer. The database receives an initial connection from a client application through the listener.

The listener brokers client requests, handing off the requests to the Oracle database server. Every time a client requests a network session with a database, the listener receives the initial request.

Figure :illustrates the various layers on the client and database during an initial connection. As shown in the diagram, the listener is at the top layer of the server-side network stack.

Figure : Layers Used in an Initial Connection



Description of "Figure 5-1 Layers Used in an Initial Connection"

Table 7–2 summarizes the options that need to be configured for each of the security mechanisms on the client-side. Each of the columns is briefly discussed after the table.

Table 7–2 Summary of Client-Side Configuration Requirements

Mechanism	**Keystore**	**Truststore**	**Default User**	**SAML Callback Handler**	**STS**	**SSL**	**User in GlassFish**
Username Authentication with Symmetric Keys		X	X				X
Mutual Certificates	X	X					
Transport Security						X	X
Message Authentication over SSL - Username Token			X			X	X
Message Authentication over SSL - X.509 Token	X					X	
SAML Authorization over SSL	X	X		X		X	
Endorsing Certificate	X	X					
SAML Sender	X	X		X			

Mechanism	Keystore	Truststore	Default User	SAML Callback Handler	STS	SSL	User in GlassFish
Vouches with Certificate							
SAML Holder of Key	X	X		X			
STS Issued Token	X	X			X		
STS Issued Token with Service Certificate	X	X			X		
STS Issued Endorsing Token	X	X			X		

- **Keystore**: If this column has an X, configure the keystore to point to the alias for the client certificate. For the GlassFish keystores, the keystore file is keystore.jks and the alias is xws-security-client, assuming that you've updated the GlassFish default certificate stores as described in To Update GlassFish Certificates.
- **Truststore**: If this column has an X, configure the truststore that contains the certificate and trusted roots of the server. For the GlassFish keystores, the file is cacerts.jks and the alias is xws-security-server, assuming that you've updated the GlassFish default certificate stores as described in To Update GlassFish Certificates.

  When using an STS mechanism, the client specifies the truststore and certificate alias for the STS, not the service. For the GlassFish stores, the file is cacerts.jks and the alias is wssip.

- **Default User**: When this column has an X, you must configure either a default username and password, a UsernameCallbackHandler, or leave these options

blank and specify a user at runtime. More information on these options can be found at Configuring Username Authentication on the Client.

- **SAML Callback Handler**: When this column has an X, you must specify a SAML Callback Handler. Examples of SAML Callback Handlers are described in Example SAML Callback Handlers.
- **STS**: If this column has an X, you must have a Security Token Service that can be referenced by the service. An example of an STS can be found in the section To Create and Secure the STS (STS). The STS is secured using a separate (non-STS) security mechanism. The security configuration for the client-side of this application is dependent upon the security mechanism selected for the STS, and not on the security mechanism selected for the application.
- **SSL**: To use a mechanism that uses secure transport (SSL), you must configure the system to point to the client and server keystore and truststore files. Steps for doing this are described in Configuring SSL For Your Applications.
- **User in Glassfish**: To use a mechanism that requires a user database for authentication, you can add a user to the file realm of GlassFish. Instructions for doing this can be found at Adding Users to GlassFish.

Configuring Username Authentication on the Client

On the client side, a user name and password must be configured for some of the security mechanisms. For this purpose, you can use the default Username and Password Callback Handlers (when deploying to GlassFish), specify a SAML Callback Handler, specify a default user name and password for development purposes, create and specify your own Callback Handlers if the container you are using does not provide defaults, or leave all of these options blank and specify the username and password dynamically at runtime. When using any of these options, you must create an authorized user on GlassFish using the Admin Console, as described in Adding Users to GlassFish.

To Configure Username Authentication on the Client

Once you've created an authorized user and determined how your application needs to specify the user, configure the Username Authentication options as follows.

1. In the Projects window, expand the node for the web service client.
2. Expand the Web Service References node.
3. Right-click the node for the web service reference for which you want to configure security options.
4. Select Edit Web Service Attributes.
5. Select the WSIT tab to display the WSIT options.

6. Expand the Username Authentication section to specify the user name and password information as required by the service. The dialog appears as shown in Figure 7–3.

   Figure 7–3 WSIT Configuration - Client - Username Authentication

7. The following options are available.

Note –

Currently the GlassFish CallbackHandler cannot handle the following: SAML Callbacks and Require ThumbPrint Reference assertions under an X.509 Token. This may be addressed in a future milestone.

   - **Authentication Credentials**: Select Static or Dynamic.
   - **Default Username**, **Default Password**: Type the name of an authorized user and the password for this user. This option is best used only in the development environment. When the Default Username and Default Password are specified, the username and password are stored in the wsit-client.xml file in clear text, which presents a security risk. Do not use this option for production.
   - **SAML Callback Handler**: To use a SAML Callback Handler, you need to create one, as there is no default. References to example SAML Callback Handlers are provided in Example SAML Callback Handlers. An example that uses a SAML Callback Handler can be found in Example: SAML Authorization over SSL (SA).

When writing SAML Callback Handlers for different security mechanisms, set the subject confirmation method to SV (Sender Vouches) or HOK (Holder of Key) and the appropriate SAML Assertion version depending on the SAML version and SAML Token Profile selected when setting the security mechanism for the service.

For example, the following code snippet for one of the SAMLCallbackHandlers listed above demonstrates how to set the subject confirmation method and sets the SAMLAssertion version to 1.0, profile 1.0.

```
if (callbacks[i] instanceof SAMLCallback) {
   try {

      SAMLCallback samlCallback = (SAMLCallback)callbacks[i];

      /*
       Set confirmation Method to SV [SenderVouches] or HOK[Holder of Key]
      */
      samlCallback.setConfirmationMethod
         (samlCallback.SV_ASSERTION_TYPE);

      if (samlCallback.getConfirmationMethod().equals(
            samlCallback.SV_ASSERTION_TYPE)) {
         samlCallback.setAssertionElement
            (createSVSAMLAssertion());

         svAssertion_saml10 =
             samlCallback.getAssertionElement();
         /*
         samlCallback.setAssertionElement
            (createSVSAMLAssertion20());
         svAssertion_saml20 =
             samlCallback.getAssertionElement();
         */
      } else if (samlCallback.getConfirmationMethod().equals(
            samlCallback.HOK_ASSERTION_TYPE)) {
         samlCallback.setAssertionElement
            (createHOKSAMLAssertion());
         hokAssertion_saml10 =
             samlCallback.getAssertionElement();
         /*
         samlCallback.setAssertionElement
            (createHOKSAMLAssertion20());
         hokAssertion_saml20 =
             samlCallback.getAssertionElement();
         */
      }
   } catch (Exception e) {
      e.printStackTrace();
   }
} else {
   throw unsupportedCallback;

}
```

Oracle shared server architecture

Oracle's shared server architecture increases the scalability of applications and the number of clients that can be simultaneously connected to the database. The shared server architecture also enables existing applications to scale up without making any changes to the application itself.

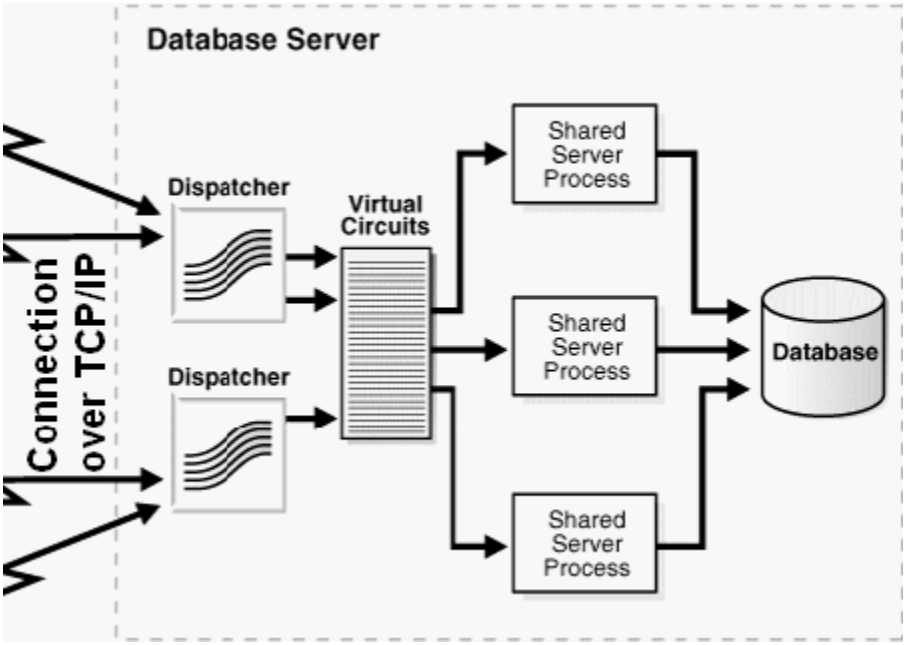
When using shared server, clients do not communicate directly with a database's server process—a database process that handles a client's requests on behalf of a database. Instead, client requests are routed to one or more dispatchers. The dispatchers place the client requests on a common queue. An idle shared server from the shared pool of server

processes picks up and processes a request from the queue. This means a small pool of server processes can serve a large number of clients.

Basic difference between the shared and dedicated server

The following two figures show the basic difference between the shared server connection model and the traditional dedicated server connection model. In the shared server model, a dispatcher can support multiple client connections concurrently. In the dedicated server model, there is one server process for each client. Each time a connection request is received, a server process is started and dedicated to that connection until completed. This introduces a processing delay.

Shared server is ideal in configurations with a large number of connections because it reduces the server's memory requirements. Shared server is well suited for both Internet and intranet environments.



Shared servers vs dedicated servers considerations

Using shared servers reduces the number of processes and the amount of memory consumed on the database host. Shared servers are beneficial for databases where there are many OLTP users performing intermittent transactions.

Using shared servers rather than dedicated servers is also generally better for systems that have a high connection rate to the database. With shared servers, when a connect request is received, a dispatcher is available to handle concurrent connection requests. With dedicated servers, however, a connection-specific dedicated server is sequentially initialized for each connection request.

Performance of certain database features can improve when a shared server architecture is used, and performance of certain database features can degrade slightly when a shared server architecture is used.

For example, a session can be prevented from migrating to another shared server while parallel execution is active (then not good for a datawarehouse database)

A session can remain nonmigratable even after a request from the client has been processed, because not all the user information has been stored in the UGA. If a server were to process the request from the client, then the part of the user state that was not stored in the UGA would be inaccessible. To avoid this situation, individual shared servers often need to remain bound to a user session.

When using some features, you may need to configure more shared servers, because some servers might be bound to sessions for an excessive amount of time.

Backup and Recovery

Backup and recovery procedures protect your database against data loss and reconstruct the data, should loss occur. The reconstructing of data is achieved through media recovery, which refers to the various operations involved in restoring, rolling forward, and rolling back a backup of database files. This chapter introduces concepts fundamental to designing a backup and recovery strategy.

Introduction to Backup

A backup is a copy of data. This copy can include important parts of the database, such as the control file and datafiles. A backup is a safeguard against unexpected data loss and application errors. If you lose the original data, then you can reconstruct it by using a backup.

Backups are divided into physical backups and logical backups. Physical backups, which are the primary concern in a backup and recovery strategy, are copies of physical database files. You can make physical backups with either the Recovery Manager (RMAN) utility or operating system utilities. In contrast, logical backups contain logical data (for example, tables and stored procedures) extracted with an Oracle utility and stored in a binary file. You can use logical backups to supplement physical backups.

There are two ways to perform Oracle backup and recovery: Recovery Manager and user-managed backup and recovery.

Recovery Manager (RMAN) is an Oracle utility that can back up, restore, and recover database files. It is a feature of the Oracle database server and does not require separate installation.

You can also use operating system commands for backups and SQL*Plus for recovery. This method, also called user-managed backup and recovery, is fully supported by Oracle, although use of RMAN is highly recommended because it is more robust and greatly simplifies administration.

Whether you use RMAN or user-managed methods, you can supplement your physical backups with logical backups of schema objects made using the Export utility. The utility writes data from an Oracle database to binary operating system files. You can later use Import to restore this data into a database.

Consistent and Inconsistent Backups

A consistent backup is one in which the files being backed up contain all changes up to the same **system change number (SCN)**. This means that the files in the backup contain all the data taken from a same point in time. Unlike an inconsistent backup, a consistent whole database backup does not require recovery after it is restored.

An inconsistent backup is a backup of one or more database files that you make while the database is open or after the database has shut down abnormally.

Overview of Consistent Backups

A consistent backup of a database or part of a database is a backup in which all read/write datafiles and control files are checkpointed with the same SCN.

The only way to make a consistent whole database backup is to shut down the database with the NORMAL, IMMEDIATE, or TRANSACTIONAL options and make the backup while the database is closed. If a database is not shut down cleanly, for example, an instance fails or you issue a SHUTDOWN ABORT statement, then the database's datafiles are always inconsistent—unless the database is a **read-only database**.

Oracle makes the control files and datafiles consistent to the same SCN during a database **checkpoint**. The only tablespaces in a consistent backup that are allowed to have older SCNs are read-only and offline normal tablespaces, which are still consistent with the other datafiles in the backup because no changes have been made to them.

The important point is that you can open the database after restoring a consistent whole database backup *without needing recovery* because the data is already consistent: no action is required to make the data in the restored datafiles correct. Hence, you can restore a year-old consistent backup of your database without performing media recovery and without Oracle performing instance recovery. Of course, when you restore a consistent whole database backup without applying redo, you lose all transactions that were made since the backup was taken.

A consistent whole database backup is the only valid backup option for databases operating in NOARCHIVELOG mode, because otherwise recovery is necessary for consistency. In NOARCHIVELOG mode, Oracle does not archive the redo logs, and so the required redo logs might not exist on disk. A consistent whole backup is also a valid backup option for databases operating in ARCHIVELOG mode. When this type of backup is restored and archived logs are available, you have the option of either opening the database immediately and losing transactions that were made since the backup was taken, or applying the archived logs to recover those transactions.

Overview of Inconsistent Backups

An inconsistent backup is a backup in which the files being backed up do not contain all the changes made at all the SCNs. In other words, some changes are missing. This means that the files in the backup contain data taken from different points in time. This can occur because the datafiles are being modified as backups are being taken. Oracle recovery makes inconsistent backups consistent by reading all archived and online redo logs, starting with the earliest SCN in any of the datafile headers, and applying the changes from the logs back into the datafiles.

If the database must be up and running 24 hours a day, seven days a week, then you have no choice but to perform inconsistent backups of the whole database. A backup of online datafiles is called an **online backup**. This requires that you run your database in ARCHIVELOG mode.

If you run the database in ARCHIVELOG mode, then you do not have to back up the whole database at one time. For example, if your database contains seven tablespaces, and if you back up the control file as well as a different tablespace each night, then in a week you will back up all tablespaces in the database as well as the control file. You can consider this staggered backup as a whole database backup. However, if such a staggered backup must be restored, then you need to recover using all archived redo logs that were created since the earliest backup was taken.

Archiving Unarchived Redo Log Files

After an **online backup** or inconsistent closed backup, always ensure that you have the redo necessary to recover the backup by archiving the unarchived redo logs.

Backing Up the Archived Logs and the Control File

After open or inconsistent closed backups, Oracle recommends backing up all archived logs produced during the backup, and then backing up the control file after the backup completes. If you do not have all archived redo logs produced during the backup, then you cannot recover the backup because you do not have all the redo records necessary to make it consistent.

Introduction to Recovery

To restore a physical backup of a datafile or control file is to reconstruct it and make it available to the Oracle database server. To recover a restored datafile is to update it by applying archived redo logs and online redo logs, that is, records of changes made to the database after the backup was taken. If you use RMAN, then you can also recover datafiles with incremental backups, which are backups of a datafile that contain only blocks that changed after a previous incremental backup.

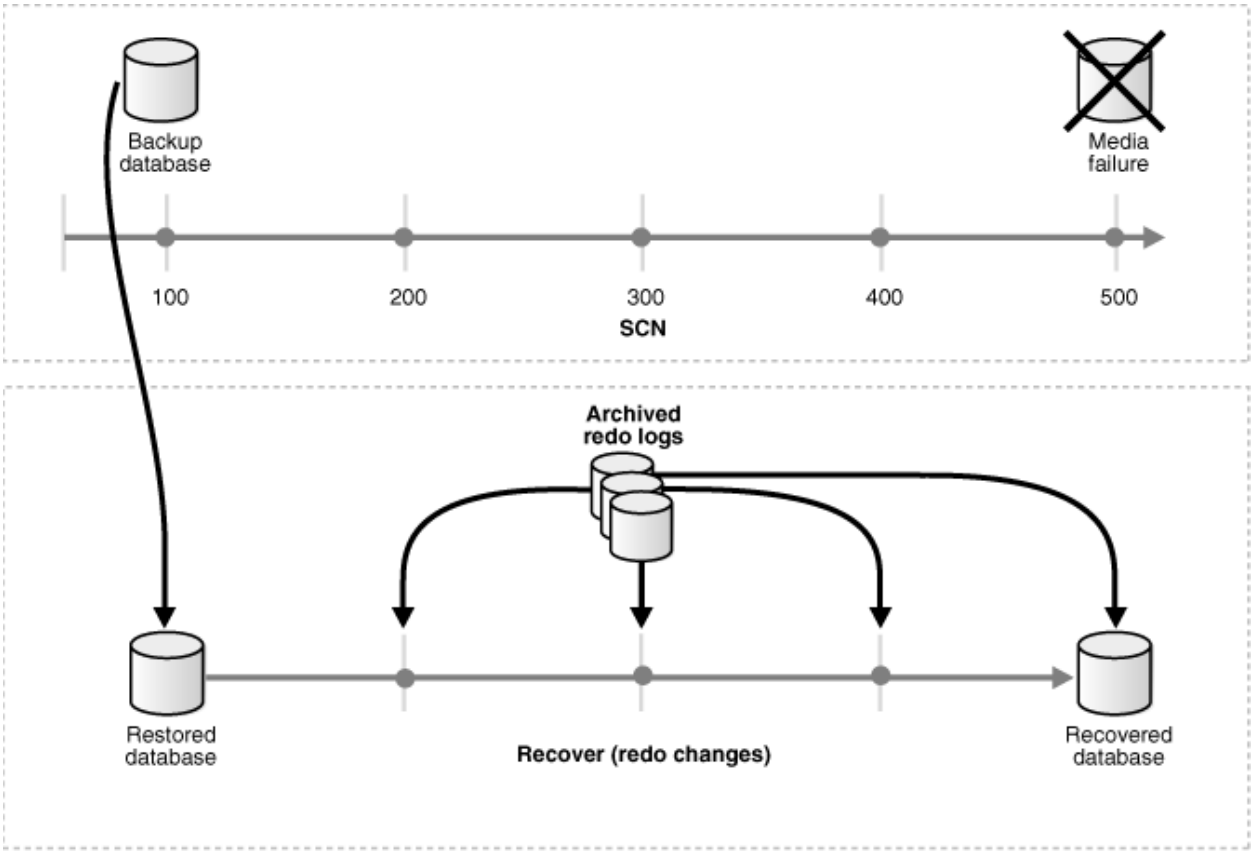
After the necessary files are restored, media recovery must be initiated by the user. Media recovery involves various operations to restore, roll forward, and roll back a backup of database files.

Media recovery applies archived redo logs and online redo logs to recover the datafiles. Whenever a change is made to a datafile, the change is first recorded in the online redo logs. Media recovery selectively applies the changes recorded in the online and archived redo logs to the restored datafile to roll it forward.

To correct problems caused by logical data corruptions or user errors, you can use Oracle Flashback. Oracle Flashback Database and Oracle Flashback Table let you quickly recover to a previous time.

Figure : illustrates the basic principle of backing up, restoring, and performing media recovery on a database.

Figure : Media Recovery



Description of "Figure 15-2 Media Recovery"

Unlike media recovery, Oracle performs crash recovery and instance recovery automatically after an instance failure. Crash and instance recovery recover a database to its transaction-consistent state just before instance failure. By definition, crash recovery is the recovery of a database in a single-instance configuration or an Oracle Real Application Clusters configuration in which all instances have crashed. In contrast, instance recovery is the recovery of one failed instance by a live instance in an Oracle Real Application Clusters configuration.

Overview of Media Recovery

The type of recovery that takes a backup and applies redo is called media recovery. Media recovery updates a backup to either to the current or to a specified prior time. Typically, the term "media recovery" refers to recovery of datafiles. Block media recovery is a more specialized operation that you use when just a few blocks in one or more files have been corrupted. In any case, you always use a restored backup to perform the recovery.

This section contains the following topics:

- Complete Recovery
- Incomplete Recovery
- Datafile Media Recovery
- Block Media Recovery

Complete Recovery

Complete recovery involves using redo data or incremental backups combined with a backup of a database, tablespace, or datafile to update it to the most current point in time. It is called *complete* because Oracle applies *all* of the redo changes contained in the archived and online logs to the backup. Typically, you perform complete media recovery after a media failure damages datafiles or the control file.

You can perform complete recovery on a database, tablespace, or datafile. If you are performing complete recovery on the whole database, then you must:

- Mount the database
- Ensure that all datafiles you want to recover are online
- Restore a backup of the whole database or the files you want to recover
- Apply online or archived redo logs, or a combination of the two

If you are performing complete recovery on a tablespace or datafile, then you must:

- Take the tablespace or datafile to be recovered offline if the database is open
- Restore a backup of the datafiles you want to recover
- Apply online or archived redo logs, or a combination of the two

Incomplete Recovery

Incomplete recovery, or point-in-time recovery, uses a backup to produce a noncurrent version of the database. In other words, you do not apply all of the redo records generated after the most recent backup. You usually perform incomplete recovery of the whole database in the following situations:

- Media failure destroys some or all of the online redo logs.
- A user error causes data loss, for example, a user inadvertently drops a table.
- You cannot perform complete recovery because an archived redo log is missing.
- You lose your current control file and must use a backup control file to open the database.

To perform incomplete media recovery, you must restore all datafiles from backups created prior to the time to which you want to recover and then open the database with the RESETLOGS option when recovery completes. The RESETLOGS operation creates a new incarnation of the database—in other words, a database with a new stream of log sequence numbers starting with log sequence 1.

Before using the OPEN RESETLOGS command to open the database in read/write mode after an incomplete recovery, it is a good idea to first open the database in read-only mode, and inspect the data to make sure that the database was recovered to the correct point. If the recovery was done to the wrong point, then it is easier to re-run the recovery if no OPEN RESETLOGS has been done. If you open the database read-only and discover that not enough recovery was done, then just run the recovery again to the desired time. If you discover that too much recovery was done, then you must restore the database again and re-run the recovery.

Tablespace Point-in-Time Recovery

The tablespace point-in-time recovery (TSPITR) feature lets you recover one or more tablespaces to a point in time that is different from the rest of the database. TSPITR is most useful when you want to:

- Recover from an erroneous drop or truncate table operation
- Recover a table that has become logically corrupted
- Recover from an incorrect batch job or other DML statement that has affected only a subset of the database
- Recover one independent schema to a point different from the rest of a physical database (in cases where there are multiple independent schemas in separate tablespaces of one physical database)

- Recover a tablespace on a very large database (VLDB) rather than restore the whole database from a backup and perform a complete database roll-forward

TSPITR has the following limitations:

- You cannot use it on the SYSTEM tablespace, an UNDO tablespace, or any tablespace that contains rollback segments.
- Tablespaces that contain interdependent data must be recovered together. For example, if two tables are in separate tablespaces and have a foreign key relationship, then both tablespaces must be recovered at the same time; you cannot recover just one of them. Oracle can enforce this limitation when it detects data relationships that have been explicitly declared with database constraints. There could be other data relationships that are not declared with database constraints. Oracle cannot detect these, and the DBA must be careful to always restore a consistent set of tablespaces.

Incomplete Media Recovery Options

Because you are not completely recovering the database to the most current time, you must tell Oracle when to terminate recovery. You can perform the following types of media recovery.

Type of Recovery	Function
Time-based recovery	Recovers the data up to a specified point in time.
Cancel-based recovery	Recovers until you issue the CANCEL statement (not available when using Recovery Manager).
Change-based recovery	Recovers until the specified SCN.
Log sequence recovery	Recovers until the specified log sequence number (only available when using Recovery Manager).

Datafile Media Recovery

Datafile media recovery is used to recover from a lost or damaged current datafile or control file. It is also used to recover changes that were lost when a tablespace went offline without the OFFLINE NORMAL option. Both datafile media recovery and instance recovery must repair database integrity. However, these types of recovery differ with respect to their additional features. Media recovery has the following characteristics:

- Applies changes to restored backups of damaged datafiles.
- Can use archived logs as well as online logs.
- Requires explicit invocation by a user.
- Does not detect media failure (that is, the need to restore a backup) automatically. After a backup has been restored, however, detection of the need to recover it through media recovery *is* automatic.
- Has a recovery time governed solely by user policy (for example, frequency of backups, parallel recovery parameters, number of database transactions since the last backup) rather than by Oracle internal mechanisms.

The database cannot be opened if any of the online datafiles needs media recovery, nor can a datafile that needs media recovery be brought online until media recovery is complete. The following scenarios necessitate media recovery:

- You restore a backup of a datafile.
- You restore a backup control file (even if all datafiles are current).
- A datafile is taken offline (either by you or automatically by Oracle) without the OFFLINE NORMAL option.

Unless the database is not open by any instance, datafile media recovery can only operate on offline datafiles. You can initiate datafile media recovery before opening a database even when crash recovery would have sufficed. If so, crash recovery still runs automatically at database open.

Note that when a file requires media recovery, you *must* perform media recovery even if all necessary changes are contained in the online logs. In other words, you must still run recovery even though the archived logs are not needed. Media recovery could find nothing to do — and signal the "no recovery required" error — if invoked for files that do not need recovery.

Block Media Recovery

Block media recovery is a technique for restoring and recovering individual data blocks while all database files remain online and available. If corruption is limited to only a few blocks among a subset of database files, then block media recovery might be preferable to datafile recovery.

The interface to block media recovery is provided by RMAN. If you do not already use RMAN as your principal backup and recovery solution, then you can still perform block media recovery by cataloging into the RMAN repository the necessary user-managed datafile and archived redo log backups.

Overview of RMAN and User-Managed Restore and Recovery

You have a choice between two basic methods for recovering physical files. You can:

- Use the RMAN utility to restore and recover the database
- Restore backups by means of operating system utilities, and then recover by running the SQL*Plus `RECOVER` command

Whichever method you choose, you can recover a database, tablespace, or datafile. Before performing media recovery, you need to determine which datafiles to recover. Often you can use the fixed view `V$RECOVER_FILE`. This view lists all files that require recovery and explains the error that necessitates recovery.

RMAN Restore and Recovery

The basic RMAN recovery commands are `RESTORE` and `RECOVER`. Use `RESTORE` to restore datafiles from backup sets or from image copies on disk, either to their current location or to a new location. You can also restore backup sets containing archived redo logs, but this is usually unnecessary, because RMAN automatically restores the archived logs that are needed for recovery and deletes them after the recovery is finished. Use the RMAN `RECOVER` command to perform media recovery and apply archived logs or incremental backups.

RMAN automates the procedure for recovering and restoring your backups and copies.

User-Managed Restore and Recovery

If you do not use RMAN, then you can restore backups with operating system utilities and then run the SQL*Plus `RECOVER` command to recover the database. You should follow these basic steps:

1. After identifying which files are damaged, place the database in the appropriate state for restore and recovery. For example, if some but not all datafiles are damaged, then take the affected tablespaces offline while the database is open.
2. Restore the files with an operating system utility. If you do not have a backup, it is sometimes possible to perform recovery if you have the necessary redo logs dating from the time when the datafiles were first created and the control file contains the name of the damaged file.

   If you cannot restore a datafile to its original location, then relocate the restored datafile and change the location in the control file.

3. Restore any necessary archived redo log files.
4. Use the SQL*Plus `RECOVER` command to recover the datafile backups.

Recovery Using Oracle Flashback Technology

To correct problems caused by logical data corruptions or user errors, you can use Oracle Flashback. Flashback Database and Flashback Table let you quickly recover to a previous time.

This section contains the following topics:

- Overview of Oracle Flashback Database
- Overview of Oracle Flashback Table

Overview of Oracle Flashback Database

Oracle Flashback Database lets you quickly recover an Oracle database to a previous time to correct problems caused by logical data corruptions or user errors.

If an Oracle managed disk area, called a flash recovery area is configured, and if you have enabled the Flashback functionality, then you can use the RMAN and SQL FLASHBACK DATABASE commands to return the database to a prior time. Flashback Database is not true media recovery, because it does not involve restoring physical files. However, Flashback is preferable to using the RESTORE and RECOVER commands in some cases, because it is faster and easier, and does not require restoring the whole database.

To Flashback a database, Oracle uses past block images to back out changes to the database. During normal database operation, Oracle occasionally logs these block images in Flashback logs. Flashback logs are written sequentially, and they are not archived. Oracle automatically creates, deletes, and resizes Flashback logs in the flash recovery area. You only need to be aware of Flashback logs for monitoring performance and deciding how much disk space to allocate to the flash recovery area for Flashback logs.

The amount of time it takes to Flashback a database is proportional to how far back you need to revert the database, rather than the time it would take to restore and recover the whole database, which could be much longer. The before images in the Flashback logs are only used to restore the database to a point in the past, and forward recovery is used to bring the database to a consistent state at some time in the past. Oracle returns datafiles to the previous point-in-time, but not auxiliary files, such as initialization parameter files.

Overview of Oracle Flashback Table

Oracle Flashback Table lets you recover tables to a specified point in time with a single statement. You can restore table data along with associated indexes, triggers, and constraints, while the database is online, undoing changes to only the specified tables. Flashback Table does not address physical corruption; for example, bad disks or data segment and index inconsistencies.

Flashback Table works like a self-service repair tool. Suppose a user accidentally deletes some important rows from a table and wants to recover the deleted rows. You can restore the table to the time before the deletion and see the missing rows in the table with the FLASHBACK TABLE statement.

You can revert the table and its contents to a certain wall clock time or user-specified system change number (SCN). Use Flashback Table with Oracle Flashback Version query and Flashback Transaction Query to find a time to which the table should be restored back to.

For Flashback Table to succeed, the system must retain enough undo information to satisfy the specified SCN or timestamp, and the integrity constraints specified on the tables cannot be violated. Also, row movement must be enabled.

The point of time in the past that you use Flashback Table to go to is controlled by the undo retention of the system. Oracle Database 10g automatically tunes a parameter called the undo retention period. The undo retention period indicates the amount of time that must pass before old undo information—that is, undo information for committed transactions—can be overwritten. The database collects usage statistics and tunes the undo retention period based on these statistics and on undo tablespace size.

Other Types of Oracle Recovery

This section contains the following topics:

- Overview of Redo Application
- Overview of Instance and Crash Recovery

Overview of Redo Application

Database buffers in the buffer cache in the SGA are written to disk only when necessary, using a least-recently-used (LRU) algorithm. Because of the way that the database writer process uses this algorithm to write database buffers to datafiles, datafiles could contain some data blocks modified by uncommitted transactions and some data blocks missing changes from committed transactions.

Two potential problems can result if an instance failure occurs:

- Data blocks modified by a transaction might not be written to the datafiles at commit time and might only appear in the redo log. Therefore, the redo log contains changes that must be reapplied to the database during recovery.
- After the roll forward phase, the datafiles could contain changes that had not been committed at the time of the failure. These uncommitted changes must be rolled back to ensure transactional consistency. These changes were either saved to the datafiles before the failure, or introduced during the roll forward phase.

To solve this dilemma, two separate steps are generally used by Oracle for a successful recovery of a system failure: rolling forward with the redo log (cache recovery) and rolling back with the rollback or undo segments (transaction recovery).

Overview of Cache Recovery

The **online redo log** is a set of operating system files that record all changes made to any database block, including data, index, and rollback segments, *whether the changes are committed or uncommitted.* All changes to Oracle blocks are recorded in the online log.

The first step of recovery from an instance or disk failure is called **cache recovery** or **rolling forward**, and involves reapplying all of the changes recorded in the redo log to the datafiles. Because rollback data is also recorded in the redo log, rolling forward also regenerates the corresponding rollback segments

Rolling forward proceeds through as many redo log files as necessary to bring the database forward in time. Rolling forward usually includes online redo log files (instance recovery or media recovery) and could include archived redo log files (media recovery only).

After rolling forward, the data blocks contain all committed changes. They could also contain uncommitted changes that were either saved to the datafiles before the failure, or were recorded in the redo log and introduced during cache recovery.

Overview of Transaction Recovery

You can run Oracle in either **manual undo management mode** or **automatic undo management mode**. In manual mode, you must create and manage **rollback segment**s to record the before-image of changes to the database. In automatic undo management mode, you create one or more undo tablespaces. These undo tablespaces contain undo segments similar to traditional rollback segments. The main difference is that Oracle manages the undo for you.

Undo blocks (whether in rollback segments or automatic undo tablespaces) record database actions that should be undone during certain database operations. In database recovery, the undo blocks roll back the effects of uncommitted transactions previously applied by the rolling forward phase.

After the roll forward, any changes that were not committed must be undone. Oracle applies undo blocks to roll back uncommitted changes in data blocks that were either written before the failure or introduced by redo application during cache recovery. This process is called **rolling back** or **transaction recovery**.

Oracle can roll back multiple transactions simultaneously as needed. All transactions systemwide that were active at the time of failure are marked as terminated. Instead of waiting for SMON to roll back terminated transactions, new transactions can recover blocking transactions themselves to get the row locks they need.

Overview of Instance and Crash Recovery

Crash recovery is used to recover from a failure either when a single-instance database fails or all instances of an Oracle Real Application Clusters database fail. Instance recovery refers to the case where a surviving instance recovers a failed instance in an Oracle Real Application Clusters database.

The goal of crash and instance recovery is to restore the data block changes located in the cache of the terminated instance and to close the redo thread that was left open. Instance and crash recovery use only online redo log files and current online datafiles. Oracle recovers the **redo thread**s of the terminated instances together.

Crash and instance recovery involve two distinct operations: rolling forward the current, online datafiles by applying both committed and uncommitted transactions contained in online redo records, and then rolling back changes made in uncommitted transactions to their original state.

Crash and instance recovery have the following shared characteristics:

- Redo the changes using the current online datafiles (as left on disk after the failure or `SHUTDOWN ABORT`)
- Use only the online redo logs and never require the use of the archived logs
- Have a recovery time governed by the number of terminated instances, amount of redo generated in each terminated redo thread since the last checkpoint, and by user-configurable factors such as the number and size of redo log files, checkpoint frequency, and the parallel recovery setting

Oracle performs this recovery automatically on two occasions:

- At the first database open after the failure of a single-instance database or all instances of an Oracle Real Applications Cluster database (crash recovery).
- When some but not all instances of an Oracle Real Application Clusters configuration fail (instance recovery). The recovery is performed automatically by a surviving instance in the configuration.

The important point is that in both crash and instance recovery, Oracle applies the redo automatically: no user intervention is required to supply redo logs. However, you can set parameters in the database server that can tune the duration of instance and crash recovery performance. Also, you can tune the rolling forward and rolling back phases of instance recovery separately.

Deciding Which Recovery Technique to Use

When to Use Media Recovery

Use media recovery when one or more datafiles has been physically damaged. This can happen due to hardware errors or user errors, such as accidentally deleting a file. Complete media recovery is used with individual datafiles, tablespaces, or the entire database.

Use incomplete media recovery when the database has been logically damaged. This can happen due to application error or user error, such as accidentally deleting a table or tablespace. Incomplete media recovery is used only with the whole database, not with individual datafiles or tablespaces. (If you do not want to do incomplete media recovery of the entire database, you can do tablespace point-in-time recovery with individual tablespaces.)

Use block media recovery when a small number of blocks in one or more files have been physically damaged. This usually happens due to hardware errors, such as a bad disk controller, or operating system I/O errors. Block media recovery is used with individual data blocks, and the remainder of the database remains online and available during the recovery.

When to Use Oracle Flashback

Flashback Table is a push button solution to restore the contents of a table to a given point in time. An application on top of Flashback Query can achieve this, but with less efficiency.

Flashback Database applies to the entire database. It requires configuration and resources, but it provides a fast alternative to performing incomplete database recovery.

Flashback Table uses information in the undo tablespace to restore the table. This provides significant benefits over media recovery in terms of ease of use, availability, and faster restoration.

Flashback Database and Flashback Table differ in granularity, performance, and restrictions. For a primary database, consider using Flashback Database rather than Flashback Table in the following situations:

- There is a logical data corruption, particularly undo corruption.
- A user error affected the whole database.
- A user error affected a table or a small set of tables, but the impact of reverting this set of tables is not clear because of the logical relationships between tables.
- A user error affected a table or a small set of tables, but using Flashback Table would fail because of its DDL restrictions.
- Flashback Database works through all DDL operations, whereas Flashback Table does not. Also, because Flashback Database moves the entire database back in time, constraints are not an issue, whereas they are with Flashback Table. Flashback Table cannot be used on a standby database

When to Use CREATE TABLE AS SELECT Recovery

To do an out of place restore of the data, perform a CTAS (`CREATE TABLE AS SELECT … AS OF …`) using the Flashback Query SQL "AS OF …" clause. For example, to create a copy of the table as of a specific time:

```
CREATE TABLE old_emp AS SELECT *

FROM employees AS OF TIMESTAMP '2002-02-05  14:15:00'
```

For out of place creation of the table, you only get data back. Constraints, indexes, and so on are not restored. This could take significantly more time and space than Flashback Table. However, Flashback Table only restores rows in blocks that were modified after the specified time, making it more efficient.

When to Use Import/Export Utilities Recovery

In contrast to physical backups, **logical backups** are exports of schema objects, like tables and stored procedures, into a binary file. Oracle utilities are used to move Oracle schema objects in and out of Oracle. Export, or Data Pump Export, writes data from an Oracle database to binary operating system files. Import, or Data Pump Import, reads export files and restores the corresponding data into an existing database.

Although import and export are designed for moving Oracle data, you can also use them as a supplemental method of protecting data in an Oracle database. You should not use Oracle import and export utilities as the sole method of backing up your data.

Oracle import and export utilities work similarly to CTAS, but they restore constraints, indexes, and so on. They effectively re-create the whole table if an export was performed earlier corresponding to the Flashback time. Flashback Table is more performance efficient than import/export utilities, because it restores only the subset of rows that got modified.

When to Use Tablespace Point-in-Time Recovery

Use tablespace point-in-time recovery when one or more tablespaces have been logically damaged, and you do not want to do incomplete media recovery of the entire database. Tablespace point-in-time recovery is used with individual tablespaces.

Flash Recovery Area

The flash recovery area is an Oracle-managed directory, file system, or Automatic Storage Management disk group that provides a centralized disk location for backup and recovery files. Oracle creates archived logs in the flash recovery area. RMAN can store its backups in the flash recovery area, and it uses it when restoring files during media recovery. The flash recovery area also acts as a disk cache for tape.

Oracle recovery components interact with the flash recovery area ensuring that the database is completely recoverable using files in flash recovery area. All files necessary to recover the database following a media failure are part of flash recovery area.

Following is a list of recovery-related files in flash recovery area:

- Current control file
- Online logs
- Archived logs
- Flashback logs
- Control file autobackups
- Control file copies
- Datafile copies
- Backup pieces

Flash Recovery Area Disk Limit

Oracle lets you define a disk limit, which is the amount of space that Oracle can use in the flash recovery area. A disk limit lets you use the remaining disk space for other purposes and not to dedicate a complete disk for the flash recovery area. It does not include any overhead that is not known to Oracle. For example, the flash recovery area disk limit does not include the extra size of a file system that is compressed, mirrored, or some other redundancy mechanism.

Oracle and RMAN create files in the flash recovery area until the space used reaches the flash recovery area disk limit. Then, Oracle deletes the minimum set of existing files from the flash recovery area that are obsolete, redundant copies, or backed up to tertiary storage. Oracle warns the user when available disk space is less than 15%, but it continues to fill the disk to 100% of the flash recovery area disk limit.

The bigger the flash recovery area, the more useful it becomes. The recommended disk limit is the sum of the database size, the size of incremental backups, and the size of all archive logs that have not been copied to tape.

If the flash recovery area is big enough to keep a copy of the tablespaces, then those tablespaces do not need to access tertiary storage. The minimum size of the flash recovery area should be at least large enough to contain archive logs that have not been copied to tape. For example, if an ASM disk group of size 100 GB is used with normal redundancy for the flash recovery area, then the flash recovery area disk limit must be set to 50 GB.

Transporting data into databases

What Are the Export and Import Utilities?

The Export and Import utilities provide a simple way for you to transfer data objects between Oracle databases, even if they reside on platforms with different hardware and software configurations.

When you run Export against an Oracle database, objects (such as tables) are extracted, followed by their related objects (such as indexes, comments, and grants), if any. The extracted data is written to an export dump file. The Import utility reads the object definitions and table data from the dump file.

An export file is an Oracle binary-format dump file that is typically located on disk or tape. The dump files can be transferred using FTP or physically transported (in the case of tape) to a different site. The files can then be used with the Import utility to transfer data between databases that are on systems not connected through a network. The files can also be used as backups in addition to normal backup procedures.

Export dump files can only be read by the Oracle Import utility. The version of the Import utility cannot be earlier than the version of the Export utility used to create the dump file.

You can also display the contents of an export file without actually performing an import. To do this, use the Import SHOW parameter. See SHOW for more information.

To load data from ASCII fixed-format or delimited files, use the SQL*Loader utility.

Verifying Access Privileges for Export and Import Operations

To use Export and Import, you must have the CREATE SESSION privilege on an Oracle database. This privilege belongs to the CONNECT role established during database creation. To export tables owned by another user, you must have the EXP_FULL_DATABASE role enabled. This role is granted to all database administrators (DBAs).

If you do not have the system privileges contained in the EXP_FULL_DATABASE role, you cannot export objects contained in another user's schema. For example, you cannot export a table in another user's schema, even if you created a synonym for it.

The following schema names are reserved and will not be processed by Export:

- ORDSYS
- MDSYS
- CTXSYS
- ORDPLUGINS
- LBACSYS

You can perform an import operation even if you did not create the export file. However, keep in mind that if the export file was created by a user with the EXP_FULL_DATABASE role, then you must have the IMP_FULL_DATABASE role to import it. Both of these roles are typically assigned to database administrators (DBAs).

Invoking Export and Import

You can invoke Export and Import, and specify parameters by using any of the following methods:

- Command-line entries
- Parameter files
- Interactive mode

Before you use one of these methods, be sure to read the descriptions of the available parameters. See Export Parameters and Import Parameters.

Invoking Export and Import As SYSDBA

SYSDBA is used internally and has specialized functions; its behavior is not the same as for generalized users. Therefore, you should not typically need to invoke Export or Import as SYSDBA, except in the following situations:

- At the request of Oracle technical support
- When importing a transportable tablespace set

To invoke Export or Import as SYSDBA, use the following syntax (substitute exp for imp if you are using Export). Add any desired parameters or parameter filenames:

```
imp \'username/password AS SYSDBA\'
```

Optionally, you could also specify an instance name:

```
imp \'username/password@instance AS SYSDBA\'
```

If either the username or password is omitted, you will be prompted you for it.

This example shows the entire connect string enclosed in quotation marks and backslashes. This is because the string, AS SYSDBA, contains a blank, a situation for which most operating systems require that the entire connect string be placed in quotation marks or marked as a literal by some method. Some operating systems also require that quotation marks on the command line be preceded by an escape character. In this example, backslashes are used as the escape character. If the backslashes were not present, the command-line parser that Export and Import use would not understand the quotation marks and would remove them.

Command-Line Entries

You can specify all valid parameters and their values from the command line using the following syntax:

```
exp username/password PARAMETER=value
```

or

```
exp username/password PARAMETER=(value1,value2,...,valuen)
```

The number of parameters cannot exceed the maximum length of a command line on the system. Note that the examples could use imp to invoke Import rather than exp to invoke Export.

Parameter Files

The information in this section applies to both Export and Import, but the examples show use of the Export command, exp.

You can specify all valid parameters and their values in a parameter file. Storing the parameters in a file allows them to be easily modified or reused, and is the recommended method for invoking Export. If you use different parameters for different databases, you can have multiple parameter files.

Create the parameter file using any flat file text editor. The command-line option PARFILE=*filename* tells Export to read the parameters from the specified file rather than from the command line. For example:

```
exp PARFILE=filename

exp username/password PARFILE=filename
```

The first example does not specify the *username/password* on the command line to illustrate that you can specify them in the parameter file, although, for security reasons, this is not recommended.

The syntax for parameter file specifications is one of the following:

```
PARAMETER=value

PARAMETER=(value)

PARAMETER=(value1, value2, ...)
```

The following example shows a partial parameter file listing:

```
FULL=y

FILE=dba.dmp

GRANTS=y

INDEXES=y

CONSISTENT=y
```

You can add comments to the parameter file by preceding them with the pound (#) sign. Export ignores all characters to the right of the pound (#) sign.

You can specify a parameter file at the same time that you are entering parameters on the command line. In fact, you can specify the same parameter in both places. The position of the PARFILE parameter and other parameters on the command line determines which parameters take precedence. For example, assume the parameter file params.dat contains the parameter INDEXES=y and Export is invoked with the following line:

```
exp username/password PARFILE=params.dat INDEXES=n
```

In this case, because INDEXES=n occurs *after* PARFILE=params.dat, INDEXES=n overrides the value of the INDEXES parameter in the parameter file.

Importing Objects into Your Own Schema

Table 19-1 lists the privileges required to import objects into your own schema. All of these privileges initially belong to the RESOURCE role.

Table 19-1 Privileges Required to Import Objects into Your Own Schema

Object	Required Privilege (Privilege Type, If Applicab
Clusters	CREATE CLUSTER (System) or UNLIMITED TABLESPACE System). The user must also be assigned a tablespace quota.
Database links	CREATE DATABASE LINK (System) and CREATE SESSION (System) on remote database
Triggers on tables	CREATE TRIGGER (System)
Triggers on schemas	CREATE ANY TRIGGER (System)
Indexes	CREATE INDEX (System) or UNLIMITED TABLESPACE (System). The user must also be assigned a tablespace quota.
Integrity constraints	ALTER TABLE (Object)
Libraries	CREATE ANY LIBRARY (System)
Packages	CREATE PROCEDURE (System)
Private synonyms	CREATE SYNONYM (System)
Sequences	CREATE SEQUENCE (System)
Snapshots	CREATE SNAPSHOT (System)
Stored functions	CREATE PROCEDURE (System)
Stored procedures	CREATE PROCEDURE (System)
Table data	INSERT TABLE (Object)
Table definitions (including comments and audit options)	CREATE TABLE (System) or UNLIMITED TABLESPACE (System). The user must also be assigned a tablespace quota.
Views	CREATE VIEW (System) and SELECT (Object) on the base table, or SELECT ANY TABLE (System)
Object types	CREATE TYPE (System)
Foreign function libraries	CREATE LIBRARY (System)
Dimensions	CREATE DIMENSION (System)

Object	Required Privilege (Privilege Type, If Applicab
Operators	CREATE OPERATOR (System)
Indextypes	CREATE INDEXTYPE (System)

Export and Import Modes

The Export and Import utilities support four modes of operation:

- Full: Exports and imports a full database. Only users with the EXP_FULL_DATABASE and IMP_FULL_DATABASE roles can use this mode. Use the FULL parameter to specify this mode.
- Tablespace: Enables a privileged user to move a set of tablespaces from one Oracle database to another. Use the TRANSPORT_TABLESPACE parameter to specify this mode.
- User: Enables you to export and import all objects that belong to you (such as tables, grants, indexes, and procedures). A privileged user importing in user mode can import all objects in the schemas of a specified set of users. Use the OWNER parameter to specify this mode in Export, and use the FROMUSER parameter to specify this mode in Import.
- Table: Enables you to export and import specific tables and partitions. A privileged user can qualify the tables by specifying the schema that contains them. Use the TABLES parameter to specify this mode.

- ***Table describes Objects Exported and Imported in Each Mode***

Object	Table Mode	User Mode	Full Database Mode	Tablespace
Analyze cluster	No	Yes	Yes	No
Analyze tables/statistics	Yes	Yes	Yes	Yes
Application contexts	No	No	Yes	No
Auditing information	Yes	Yes	Yes	No
B-tree, bitmap, domain function-based indexes	Yes[Foot 1]	Yes	Yes	Yes
Cluster definitions	No	Yes	Yes	Yes

Oracle SQL*Loader

Introduction to SQL*Loader tool

SQL*Loader allows you to load data from an external file into a table in the database. It can parse many delimited file formats such as CSV, tab-delimited, and pipe-delimited.

SQL*Loader provides the following methods to load data:

- Conventional path loads – construct INSERT statements from the contents of the input datafile based on the predefined specification and execute the inserts.

- Direct path loads – creates data blocks in Oracle database block format from the datafile and directly write the data block to the database. This way is much faster than the conventional path but subject to some restrictions.
- External table loads – create an external table for the data stored in the datafile and execute INSERT statements to insert the data from the datafile into the target table. The external table loads support parallel loading if datafile is big enough.

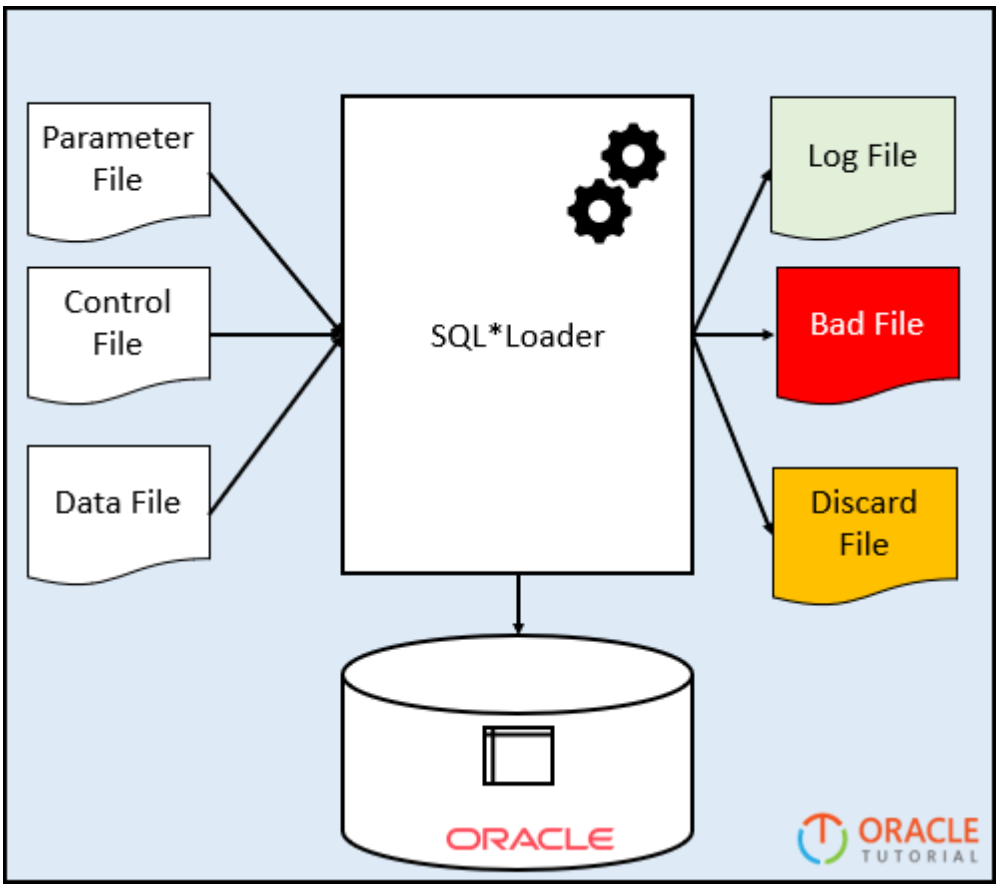
To execute the SQL*Load tool, you need at least three files:

- The input data file stores delimited or raw data
- The parameter file stores the location of the input/output files
- The control file contains the specification on how data is loaded.

After that, you execute the command sqlldr from the command line on Windows or Terminal on GNU/Linux:

```
1 >sqlldr parfile=parameter_file.par
```

The following picture illustrates the SQL*Loader process:



Let's take the example of using the SQL*Load tool.

SQL*Loader example

We will load email data CSV file format into the emails table in the database.

Prepare the input files

1. The following is the content of the email.dat file:

```
1 1,john.doe@example.com
2 2,jane.doe@example.com
3 3,peter.doe@example.com
```

2. The contents of the control file (email.ctl) is as follows:

```
1 load data into table emails
2 insert
3 fields terminated by ","
4 (
5 email_id,
6 email
7 )
```

In the control file:

- The load data into table emails insert instruct the SQL*Loader to load data into the emails table using the INSERT statement.
- The fields terminated by "," (email_id,email) specifies that each row in the file has two columns email_id and email separated by a comma (,).

3. Here is the content of the parameter file (email.par):

```
1 userid=ot@pdborc/Abcd1234
2 control=email.ctl
3 log=email.log
4 bad=email.bad
5 data=email.dat
6 direct=true
```

In this parameter file, we specify the user (userid) that will connect to the Oracle database, the control file (email.ctl), log file (email.log), and data file (email.dat).
The email.bad file stores invalid data. And the last line direct=true instructs the SQL*Loader to use the direct path load method.
Note that there is no space between the parameter and value, for example:

```
1 control=email.ctrl
```

After having three files, you can place it in a directory e.g., C:\loader.

Here is the link to download the three files in a zip file format:

Download the data, control and parameter files

Load data from a flat file into the table

First, create a new table to store the email data from the input datafile:

```
1 CREATE TABLE emails(
2     email_id NUMBER PRIMARY KEY,
3     email VARCHAR2(150) NOT NULL
4 );
```

Second, launch the SQL*Loader program from the directory that you store the files using the sqlldr command:

```
1 C:\loader> sqlldr parfile=dept_loader.par
```

Here is the output:

```
1   Path used:      Direct
2
3   Load completed - logical record count 3.
4
5   Table EMAILS:
```

```
6  3 Rows successfully loaded.
7
8  Check the log file:
9  email.log
10 for more information about the load.
```

Third, log in to the Oracle and verify the contents of the emails table:

```
1 SELECT * FROM emails;
```

Here is the result set:

EMAIL_ID	EMAIL
1	john.doe@example.com
2	jane.doe@example.com
3	peter.doe@example.com

Finally, review the log file:

```
1
2  SQL*Loader: Release 12.1.0.2.0 - Production on Tue Jul 23 08:04:46 2019
3
4  Copyright (c) 1982, 2014, Oracle and/or its affiliates.  All rights reserved.
5
6  Control File:   email.ctl
7  Data File:      email.dat
8    Bad File:     email.bad
9    Discard File:  none specified
10  (Allow all discards)
11
12 Number to load: ALL
13 Number to skip: 0
14 Errors allowed: 50
15 Continuation:    none specified
16 Path used:      Direct
17
18 Table EMAILS, loaded from every logical record.
19 Insert option in effect for this table: INSERT
20
21   Column Name                  Position   Len  Term Encl Datatype
22 ------------------------------ ---------- ----- ---- ---- ---------------------
23 EMAIL_ID                        FIRST     *   ,      CHARACTER
24 EMAIL                          NEXT      *   ,      CHARACTER
25
26 The following index(es) on table EMAILS were processed:
27 index OT.SYS_C0010446 loaded successfully with 3 keys
28
29 Table EMAILS:
30   3 Rows successfully loaded.
31   0 Rows not loaded due to data errors.
32   0 Rows not loaded because all WHEN clauses were failed.
33   0 Rows not loaded because all fields were null.
34
35 Bind array size not used in direct path.
36 Column array  rows :    5000
37 Stream buffer bytes:  256000
38 Read   buffer bytes: 1048576
39
40 Total logical records skipped:          0
41 Total logical records read:             3
42 Total logical records rejected:         0
43 Total logical records discarded:        0
44 Total stream buffers loaded by SQL*Loader main thread:        2
45 Total stream buffers loaded by SQL*Loader load thread:        0
46
47 Run began on Tue Jul 23 08:04:46 2019
48 Run ended on Tue Jul 23 08:04:48 2019
49
50 Elapsed time was:     00:00:01.47
51 CPU time was:         00:00:00.07
```

DataBase Performance Tuning

Monitoring for performance issues and doing regular maintenance of databases will keep them tuned and performing better.

4.1 Better-performing Systems

Planning the initial database design, monitoring the performance, and maintaining it are the proactive steps to achieving better-performing systems. This applies to any database environment. Along with proactive monitoring and tuning for performance, DBAs need to deal with performance issues that arise at times.

Oracle has a view of the sessions and a way to see the current statements that are running against the database. Looking at the queries running and validating the statistics that are currently on the tables are the beginning steps. With the **cost-based optimizer**, current statistics are important for the queries to choose the right plan. In OEM alerts, waits are shown with alerts, and OEM provides a list of top queries running and also shows if any process is being blocked. In Oracle, blocking normally is checked after figuring out what is running and validating the statistics.

Become an Oracle DBA professional with this complete Oracle DBA Training!

4.2 Indexes

Indexes exist to help speed up queries. Having proper columns indexed can reduce the logical I/Os for queries. There are costs associated with data changes when indexes are involved. Maintenance requirements should also be considered. The performance gains of adding an index should be more than the cost of maintaining it. Also, it is important to know that too many indexes can add to performance issues instead of resolving them. Hence, indexes should be used selectively and their usage should be monitored appropriately.

- **Index Monitoring**

By enabling index monitoring, we can see which indexes are being used in Oracle. The owner of an index can alter the index to enable monitoring and leave it on for a set period. The v$object_usage table will show whether an index is used.

```
SQLPLUS> alter index IDX1_EMP_DEPT monitoring usage;
```

With this, the index is altered.

```
SQLPLUS> select empno from emp where deptno=10;
```

The rows are returned.

```
SQLPLUS> select index_name,used, monitoring
from v$object_usage;
```

INDEX_NAME	USED	MONITORING
IDX1_EMP_DEPT	YES	YES

The v$object_usage table has two other columns that show the time when the monitoring is started and when it is stopped. Meanwhile, to end the monitoring of an index, we have to use this statement:

```
alter index index_name no monitoring usage
```

Index monitoring will not track how many times an index is used, but it does offer a way to find out if there are unused indexes on a table.

Check out the Top Oracle DBA Interview Questions to enter into the Oracle DBA career!

- **Index Types**

Indexes are definitely a useful tool for improving access to data in the database. Understanding which type of index is being used and how to improve that index will help in performance tuning. Knowing how various index types affect data changes and improve SELECT statements will help us decide if the benefits of an index outweigh the costs of retaining it in place.

SQL Server Index Types	Oracle Index Types
Unique clustered	B-tree
Nonunique clustered	Function-based
Unique Nonclustered	Reverse key
Nonunique nonclustered	Index-organized tables (IOT)
Indexed views	Bitmap
Full text	Bitmap join
Spatial	Compressed
Filtered	Descending
XML	Partitioned
	Domain
	Invisible
	Intermedia (for LOBs and text)

-
  - **Primary Key Indexes**: The primary key index is created for a table when a constraint is added, and we can either use an existing index or create a new one.
  - **Function-based Indexes**: Oracle's function-based index type can dramatically reduce query time. They are useful for large tables even with simple functions like UPPER to do string comparisons.
  - **Indexes for Views**: Views use indexes on their associated tables to build information, but there might be a need for an index for selecting from a view. Oracle has materialized views, similar to views, but are a snapshot of the data. These materialized views can have functions and aggregations, along with subqueries and other views, including self-joins.
  - **Bitmap Indexes**: Bitmap indexes are stored differently than b-tree indexes. Instead of storing the row ID, a bitmap for each key is used. Because of this, these indexes are typically smaller in size and are useful for columns that have a low cardinality.
  - **Reverse Key Indexes**: They are used as a nice little trick to spread out index blocks for a sequenced column. Reversing the numbers will allow the index to have different beginning values and use different blocks in the index b-tree structure. This is especially useful for RAC environments. When we are doing inserts, the reverse index will minimize the concurrency on the index blocks.
  - **Partitioned Indexes**: Partitioning is a useful way to tune a large database environment. Oracle offers options for partitioning tables, such as LIST, HASH, RANGE, and COMPOSITE. The partition key is how the table is partitioned. We can

create partitioned indexes for these tables. An index can be a locally partitioned index based on the partition key and setup used for each partition. Local indexes are easier to manage because they are handled with each partition, as partitions might be added, dropped, or merged.

- **Invisible Indexes**: Invisible indexes are hidden from the optimizer, but not from being maintained, so as rows are changed. One reason to use an invisible index is to test the performance of the queries without an index.

If you aspire to be a certified Oracle DBA professional, then learn Oracle DBA from experts in this Oracle DBA Certification!

4.3 Locking

Holding locks on a database object will also cause another concurrent session to wait. Waiting to acquire a lock or perform a transaction could even cause blocking, depending on the locks required to perform a select transaction.

A **deadlock** is when two or more users are waiting to access data locked by each other. When the deadlock occurs, Oracle chooses the victim, rolls back the transaction, and allows the other process to continue.

Because of how Oracle handles locking, blocking is not always the first area that we should check for performance unless the application is trying to explicitly handle the locking outside Oracle. Access outside of the application, such as using query tools for ad-hoc queries, could open a transaction, and since the flow of the query is waiting on the user, Oracle Database would also wait on the user and hold onto the locks. So, if an UPDATE, INSERT, or DELETE statement is open in such a tool, there is no auto-commit that will release the locks. If the user does not issue a commit or rollback, this would leave an uncommitted transaction open, which could block others.

Lock Type	Description
Row	No limit. Readers do not wait for writers, and writers do not wait for readers. If attempting to update the same row at the same time, writers will wait for writers.
Table	DML statements—`INSERT`, `UPDATE`, `DELETE`, and `SELECT FOR UPDATE`. Table locks prevent DDL and structure changes while the transaction is occurring.
Row share table	Lock with intent to update data. This is the least restrictive lock and allows for other transactions to have the same row share lock.
Row exclusive table	Changes being made—`INSERT`, `UPDATE`, `DELETE`. This is slightly more restrictive than a row share lock. It allows other transactions on the same table.
Share table	Locks the table for updates. It allows reads of the table but no other writes.
Share row exclusive table	Only one transaction at a time can acquire a shared row lock on a table.
Exclusive table	Most restrictive lock. Only one transaction can have this lock on the table.
DDL	Dictionary lock for the structure of the objects, indexes, table, and view definitions.
Internal lock and latch	Lock on datafiles and internal structures.

4.4 Current Activity Views

Oracle has various system views that provide the current session and wait for information. These are very helpful for performance tuning and troubleshooting.

- **Current Sessions**

Obviously, when there are performance issues, it is necessary to take a look at the current sessions on the database. There is no sp_who, sp_who2, or sp_lock in Oracle, but there is the v$session view. This view shows which sessions are active. We can join this with another view to see all the queries that a session is running.

- **Activity Monitors**

There are statistics gathered as part of Automatic Workload Repository (AWR) to provide reports for analyzing the health of the database and look for performance issues. Historical views are based on snapshots that have been gathered. Viewing these areas of activity can help us

troubleshoot performance issues by pointing to an area that might be responding slowly or experiencing an overload, such as too many physical I/Os or hard parsing of SQL statements.

The Instance Activity section shows values since the database has been up and running or back until the last snapshot that is available.

- **Waits**

Another area to check into Oracle for performance issues is waiting for events. This information will be available in OEM views. The SQL*Net message from the client event is the wait for the client to tell the database server to do something. It is just waiting for instructions and really isn't contributing to issues. There might be applications that open sessions and then just wait for responses before getting the data from the database.

4.5 Automatic Workload Repository

Automatic Workload Repository (AWR) contains significant information that can be helpful when it comes to tuning the database environment. The database takes regular snapshots to get information about the database settings and the workload in the environment and stores it in AWR metadata tables (WRM$_) and historical statistics tables (WRH$_).

In **Oracle Database 11g,** these reports and information are part of Oracle Diagnostic Pack, which provides an automated gathering of the information and ways to pull the information out of the workload and also history tables for review and evaluation of performance issues. We can also create baseline templates to compare the information.

- **AWR Reports**

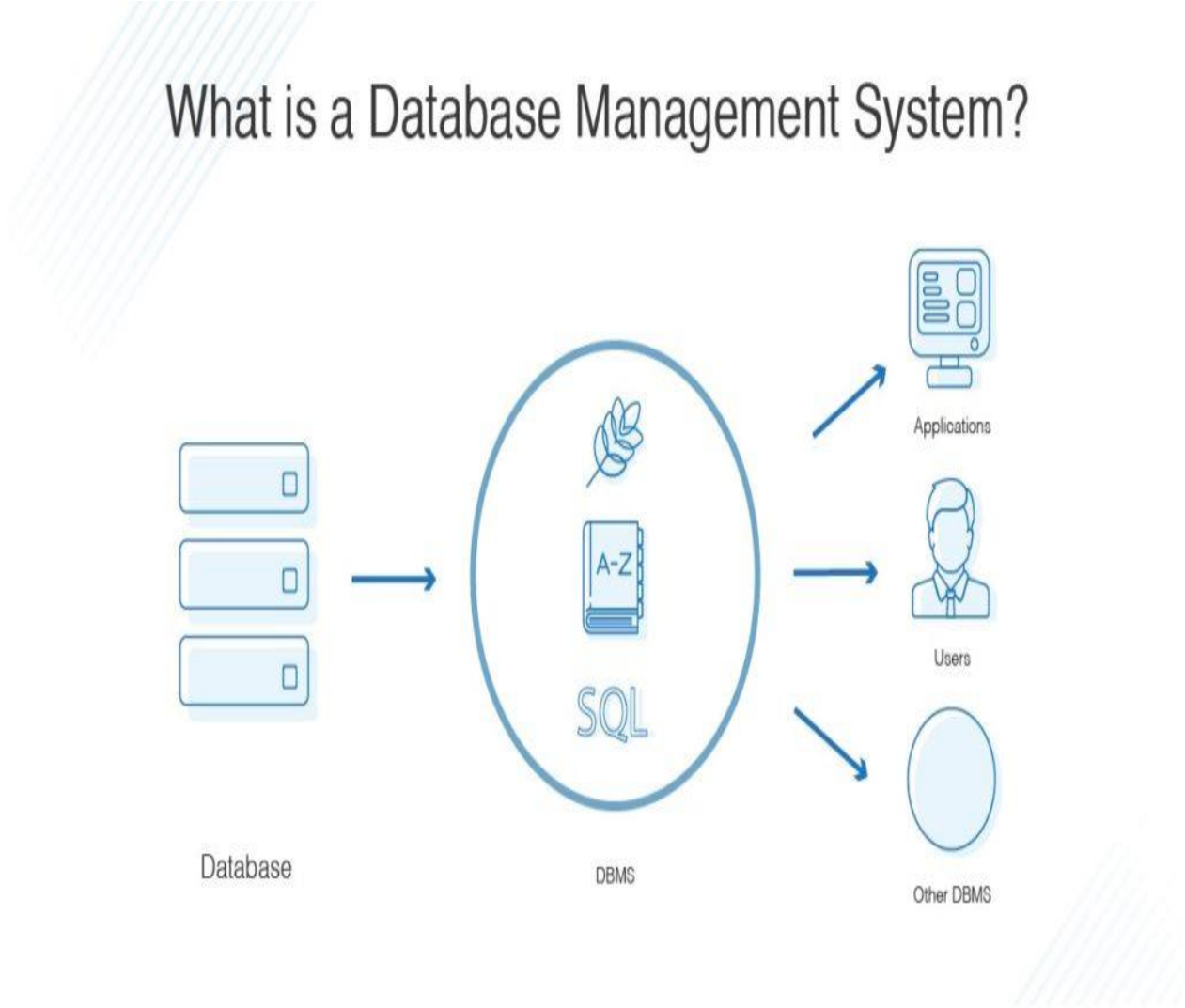
AWR reports have information about different waits. The reports list the top waits, providing a quick way to determine the areas where to start looking for bottlenecks.

AWR reports can be viewed in OEM. The reports are based on snapshot times. If different intervals are needed, different reports can be generated. In OEM, we can view the details in the list or see the reports in the HTML format.

How to Do Performance Tuning in Oracle

Oracle performance tuning is a crucial step in ensuring speedy application function and data retrieval. Here's what you need to know to improve database performance.

All database administrators (DBAs) are familiar with the onerous task of increasing database performance. To accelerate application function, DBAs have to expedite query response time, which means DBAs must have a clear understanding of how their database is organized and how it serves its purpose. That is to say, DBAs must understand not just the database itself but the dedicated computer language that accesses the database to retrieve, manipulate, or delete information.

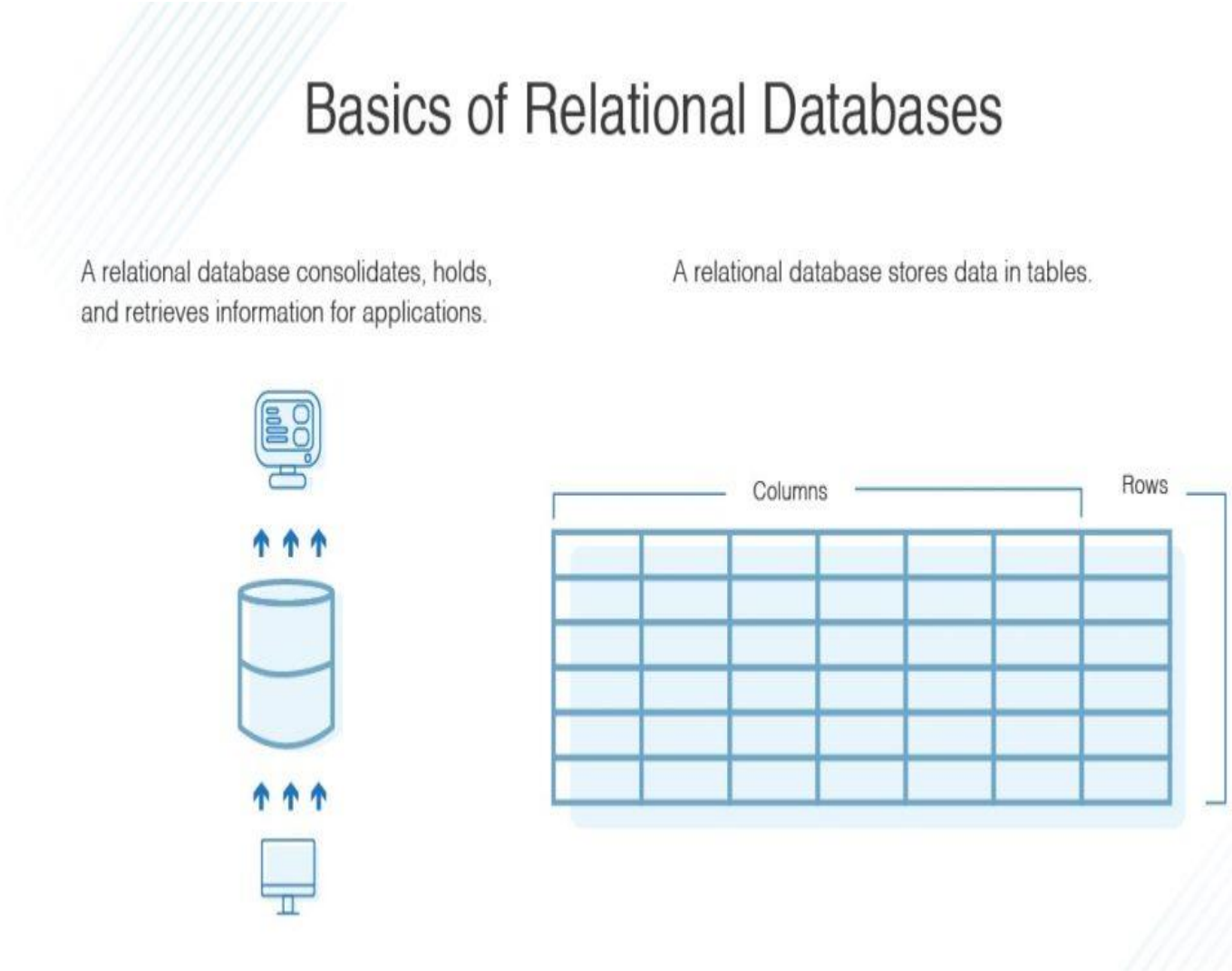


What Is Oracle Performance Tuning?

Wondering how to do performance tuning in Oracle, specifically? Oracle is a relational database management system (RDBMS), and it utilizes Structured Query Language (SQL) to enable communication between applications and the database. Performance tuning is the process of optimizing Oracle performance by streamlining the execution of SQL statements. In other words, performance tuning simplifies the process of accessing and altering information contained by the database with the intention of improving query response times and application operations.

To break this down further, let's first analyze the different components at play in a database management system, beginning with the database itself. As mentioned above, Oracle utilizes the relational database model.

What Is a Relational Database?



- **A relational database is an information system**. That is, it's a computerized method of storing and using information.
- **A relational database exists to consolidate, hold, and retrieve information as needed by applications**. Those application will then use that data for some purpose. A company, for example, might use a database to store customers' personal information. A segmentation app may query the database for the email addresses of buyers who have made a purchase in the last month — personal information the company can use to target a specific customer segment with an email blast.
- **A relational database stores data in tables.** These tables are called "relations" and are configured much like a spreadsheet: the columns are "fields" that contain different attributes, and the rows are specific entries. A client base information table might have rows 1 to 27, each row corresponding to a different customer record and each column designating an attribute of personal information (name, email address, date of last purchase, etc.). Row 1 might read: "John," "john@email.com," "1/1/2019," etc.

This database model is the foundation of the management system. The management system itself is software that creates relational databases and manages their organization and interaction with applications that use its data. As outlined by Oracle itself, the defining features of a database management system (DBMS) are:

- **Kernel code** – determines settings and allocates memory and storage for the system.

- **A data dictionary –** a collection of metadata. This repository offers a read-only overview of the data within the database, showing tables and views in addition to reference information regarding the database itself and its users.
- **A specified query language –** enables applications to access the database information.

As stated, Oracle uses SQL as its database language. SQL is distinguished by the ability to input, manipulate, retrieve, and delete data in a database. It also allows administrators to assign and revoke access as well as create their own views and functions. SQL is nonprocedural, which means that SQL statements inform the DBMS of *what* needs to be done, but they do not prescribe a course of action. Basically, the software in the DBMS figures out how to execute a statementby analyzing available options and electing the best sequence of actions. Not only is SQL a quick language to pick up, but users can embed SQL within other host languages.

How Does Oracle Database Performance Tuning Work?

Performance tuning considers the many elements in an RDBMS to troubleshoot the source of performance problems. Often, DBAs are faced with a difficult task—network users report experiencing app delays and slow-loading pages, but administrators cannot pinpoint the source of database bottlenecks. Is it an optimizer issue? The coding of query statements? A problem with the computer itself?

To identify the root cause, performance-tuning administrators must consider the many elements in an RDBMS to troubleshoot database operation lags. Not only can performance tuning be quite time-intensive, but it can be difficult to know where to begin. An intensive performance tuning process takes a systems-level approach and considers RDBMS components from top to bottom.

As many database experts note, tuning SQL statements one by one will have little effect if the administrator hasn't first performed system-level tuning on the server, instance(s), and objects. I recommend assessing input and output (I/O) measures, optimizer parameters and statistics, and instance settings before tuning individual SQL statements. Otherwise, meticulous SQL tuning may be reversed later by the optimizer as it determines execution protocol contrary to designed execution plans.

10 Steps for Effective Oracle Performance Tuning

Once DBAs have done a systems-level check, they can proceed to SQL query tuning. Generally speaking, SQL tuning seeks to minimize the number of steps—"database touches"—a query entails, thus decreasing time cost and wait time. There are a lot of little SQL quirks and best practices to keep in mind, and while this list is by no means comprehensive or universally applicable, I have found that the following guidelines are helpful pointers in most situations.

Here are my top 10 SQL query performance tuning tips:

1. Begin by identifying the most cost-intensive queries to appropriately allocate your tuning efforts. The truth is that SQL query performance tuning is an ongoing process; there's always room for improvement, there's always more code to optimize, and there's always monitoring and upkeep to be done—which can make it feel like it never ends. For this reason, it's important to isolate the high-impact SQL statements—those that are executed most frequently and require the most database and I/O activity. These statements offer the biggest returns in terms of database performance improvement, so targeting them will optimize the amount of tuning work put in relative to performance improvements.
2. Always minimize the amount of data that must be scanned in an operation. Many query statements will prompt the database to perform full-table scans, which incur much more I/O and can degrade

performance by slowing down operations and carrying out unnecessarily broad searches. To streamline data retrieval:

- Add indexes to tables if you need to access under 5% of their data, except in the case of fairly small tables (which are more expediently searched in full whether or not you need much data).
- Do not include * in your SELECT statement queries unless necessary to fetch data, as this symbol will load the system.
- Use filters in WHERE clauses to restrict the size of the data set.
- Conversely, in a column-oriented system, only select the columns you need for the query.
- Cull unnecessary tables from query statements. Occasionally developers may forget to remove JOINS that do not serve the query. While this can be innocuous in the testing stage, once the system goes into effect, JOINS to tables that do not contribute to the retrieved data can greatly increase processing time.
- Utilize EXISTS in subqueries. This communicates to Oracle that it can stop the search, rather than complete a full-table scan by default, when it finds the match.

3. Do not use indexes to tables that undergo more UPDATE or INSERT operations, as indexes can slow data input. In that same vein, you might consider dropping your indexes when striving for batch updates or insertions. In this case, it might be best to recreate the indexes after a single batch event, or simply avoid indexes on tables frequently experiencing batch data loads to begin with.
4. Don't mix data types, and do not convert numbers to characters. Their comparison can slow operations and impact performance.
5. In some scenarios, it may be easier to create a new field than to perform a calculation function on a JOIN or WHERE clause. In this situation, the new field would contain the calculated value, which the statement would SELECT rather than calculate itself. To accomplish this, the person tuning the code would have to have permission to alter the datasets, of course—but this should be no problem for a DBA or other IT administrator.
6. More broadly, align your SQL statements and datasets. Basically, comb through SQL syntax to ensure that you've written your statements in ways that match up with the data structure and allow easy access.
7. Institute protocol by using procedures instead of individual statements. A procedure is a collection of statements, and they downsize the cost of executing a recurring query. For example, if you use an app that requires you to pull data weekly, this query may account for a significant amount of database activity. You could use a procedure to certify that the query executes quickly, and according to your execution plan, because database engines carry out procedures without optimizing them.
8. Use global temporary tables (GTT) whenever possible to simplify complication summarization queries. By breaking up the work of work-intensive subqueries, GTT have been shown to improve database performance significantly.
9. Use hints. Oracle makes a list of hints available online to help application designers and DBAs. The purpose, as explained by the database itself, is to allow administrators and developers to "alter execution plans" and "force various approaches." This allows designers who are tuning their SQL statements to take the reins from the optimizer in specific scenarios where the humans know more about the data than the optimizer. In these situations, they can ensure their execution plan gets carried out rather than overwritten by the optimizer, which might choose a data access path that does not optimize speed and performance.
10. Finally, make tuning a routine. While you don't want to have to allot redundant labor to tuning the same queries over and over, performance tuning in SQL does require habitual maintenance to prevent database performance deterioration over time as both datasets and RDBMS software evolve. With this in mind, commit to normalizing and defragmenting the database routinely.

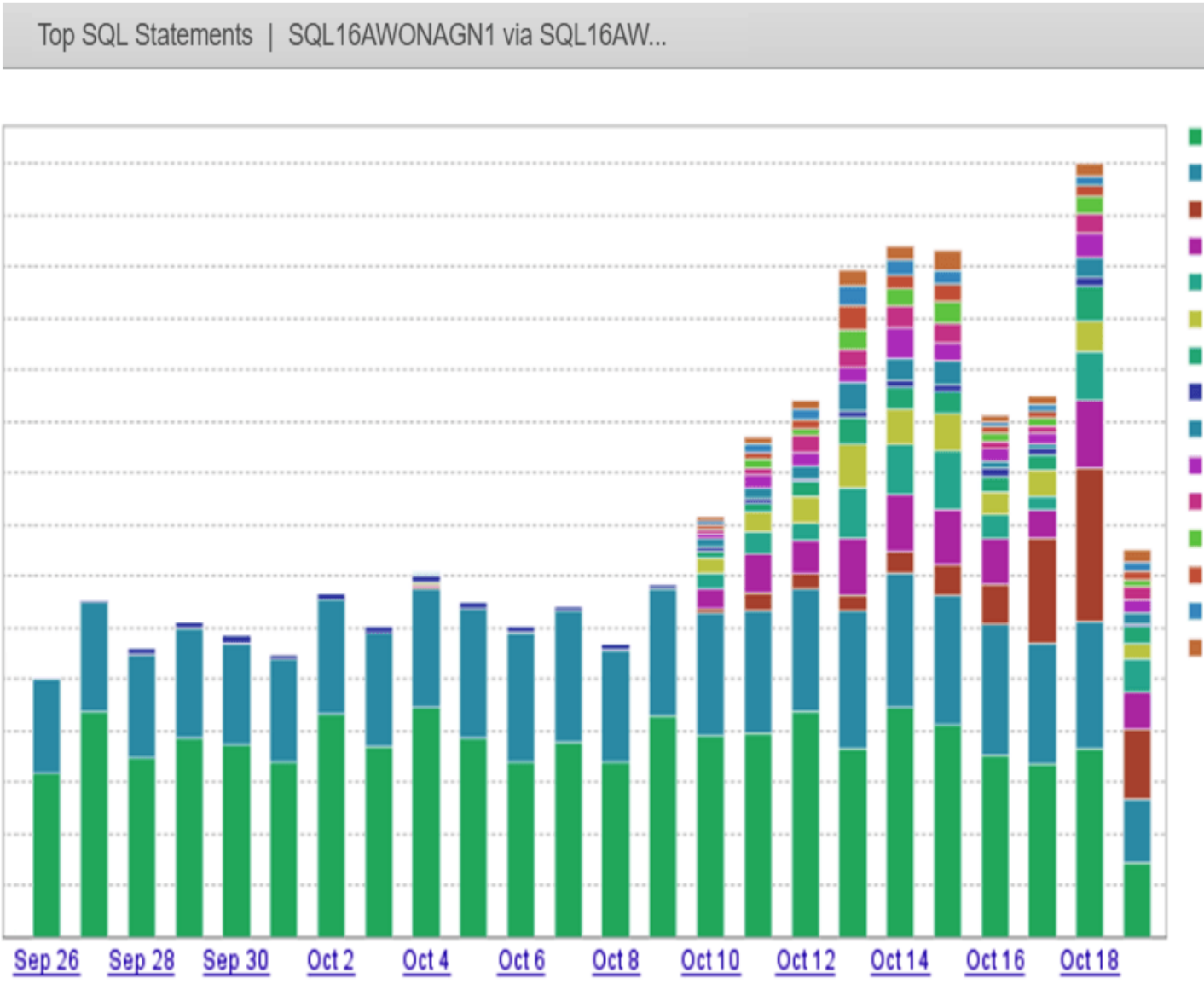
Oracle Performance Tuning Software Recommendations

Of course, all of this Oracle performance monitoring and assessment work is nearly impossible to do manually on an ad hoc basis. Plenty of useful and affordable software exists that can automate database performance tracking and help DBAs identify the source of performance problems. This saves IT departments a good deal

of time. Rather than noting a lag and having to root through their code and database set up to find the root cause, they can check their software to drill down into the queries with the highest performance impact. Here are the Oracle database tools I rate most highly:

1. **Database Performance Analyzer**

SolarWinds® Database Performance Analyzer® (DPA) is one of the most powerful solutions for those wondering how to check SQL query performance in Oracle. The dashboard was designed with the user experience in mind, and it provides comprehensible, informative visualizations breaking down SQL statement activity—data which can be dense and difficult to parse on other platforms.

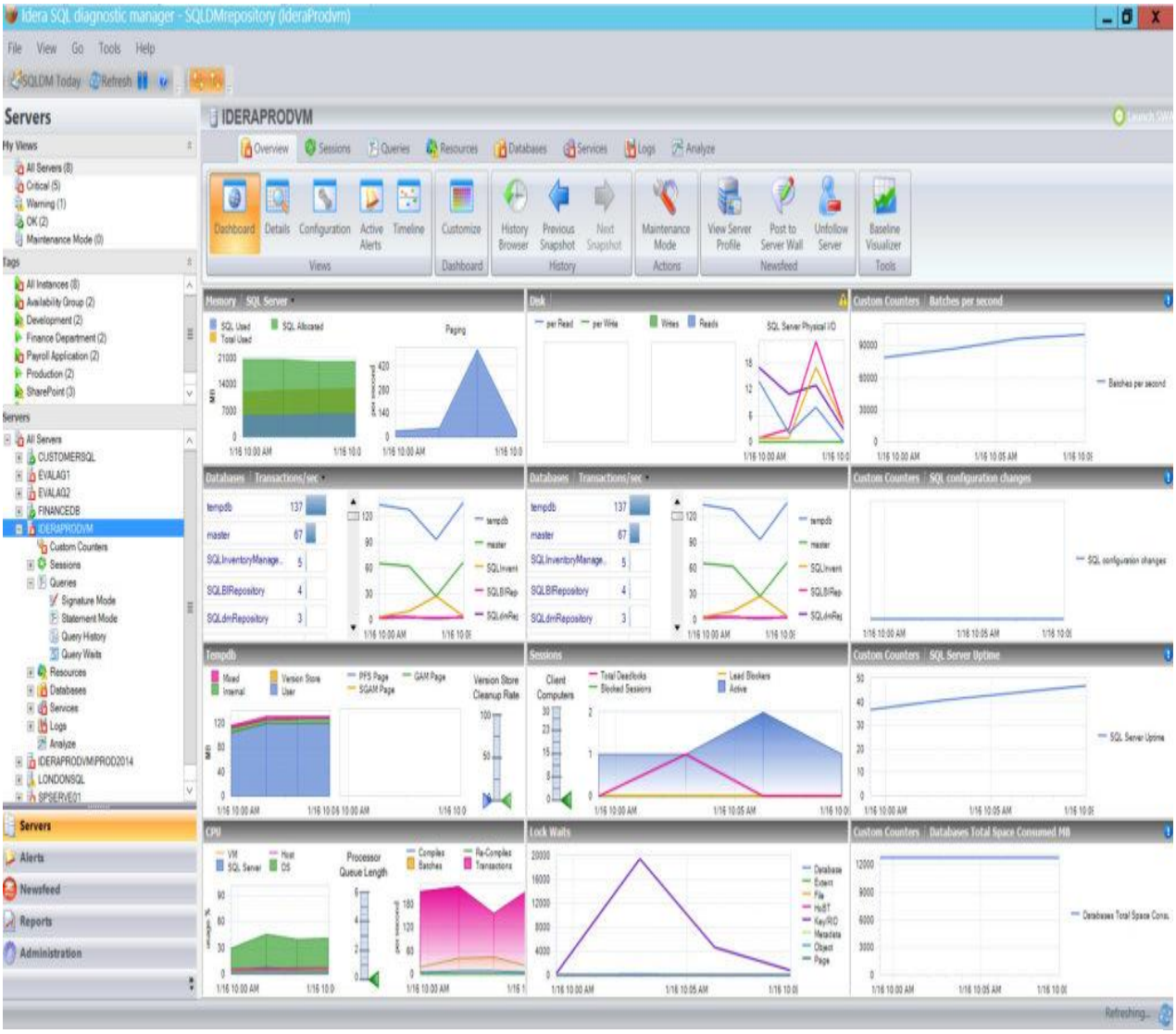


The dashboard displays a time analysis graph which plots the day against total query response time (the amount of time between a query request and a query response; basically, the cumulative processing time and resource availability wait time). The bar for each individual day is color-coded by query statement, which allows DBAs to easily drill down into metrics for each query simply by clicking.

DPA collates data in real time, so its information is up-to-date from one second to the next, and it shows alerts on the dashboard (or delivers them via mobile notifications or email) when it detects critical issues. This helps DBAs hierarchize their chaotic workload, taking care of potential problems and query wait times that have the biggest impact on system performance before focusing their attention elsewhere.

This function is particularly useful because it focuses performance-tuning work—which can require high investment for low returns when mismanaged—on the priority: wait time. Many tools and even IT departments make the mistake of prioritizing resource utilization during performance tuning, but this can obfuscate the *source* of a performance problem. It's guesswork, more than anything—something which is not true a database performance analytics which cross-check specific queries, wait time events, I/O and storage stats, and database activity against one another to determine causality and find performance solutions.
Basically, SolarWinds DPA is unparalleled in the way it consolidates and analyzes data to make it truly actionable. Its graph of worst-performing SQL statements enables users to learn the specific wait types and events that are causing application wait times. If they glean that a wait is due to the CPU, for example, DBAs have a place to start performance tuning: the actual *root cause*. Fixing this issue will immediately yield a performance boost with tangible benefits for network users and applications. Further, DPA improves group communication and institutional memory. Not only does DPA preserve historical data, but it produces custom reports, alerts, and metrics for DBAs to present internally to colleagues or externally to other departments or even clients.

2. **Idera SQL Diagnostic Manager**



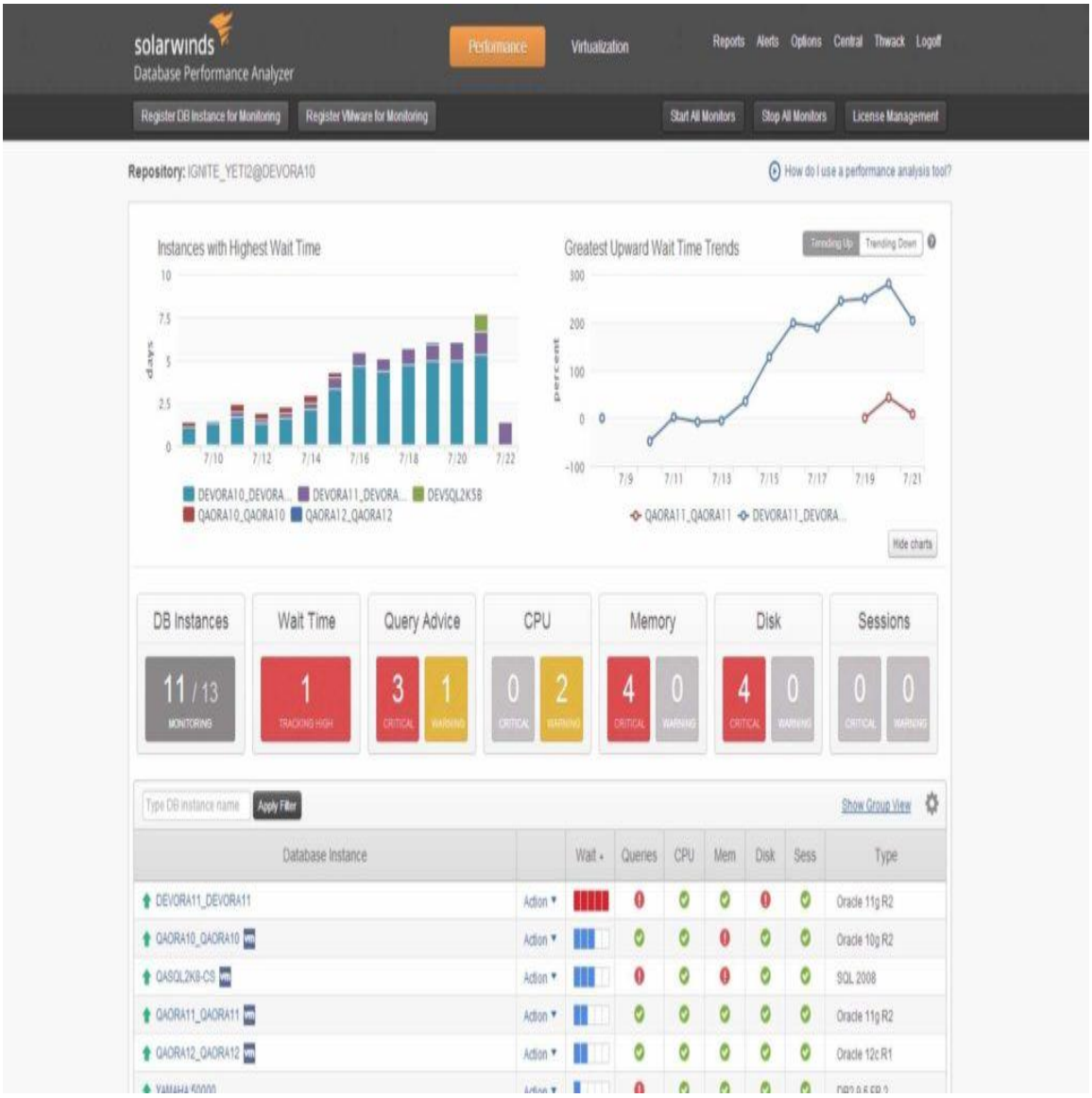
Another excellent tool for those learning how to check Oracle database performance is the Idera SQL Diagnostic Manager. It includes some features similar to the SolarWinds DPA tool, like predictive alerts and customizable monitoring settings for multiple servers.

I found that its main strength are its prescriptive suggestions. That is, if you have a performance problem, the tool will flag the issue and recommend a solution like executable script. This is particularly helpful for IT personnel who are still getting their footing in SQL query performance tuning, although of course they'll need their own expertise to troubleshoot complex bottlenecks.

In my opinion, the interface is a bit too cluttered, but it's definitely a good option for Windows users. The only other problem I'd note is that it's configurable reports and monitoring don't feel as robust as they could be.

3. **SolarWinds Database Performance Analyzer Free**

Also notable is the free version of SolarWinds DPA. No, it doesn't include all the amenities of the paid version, but it's pretty remarkable for being the only response time monitoring tool for databases.



DPA Free has some pretty great capabilities—it monitors response time and displays both wait events and wait types, providing a data-driven springboard and useful visualization for DBAs. Further, like the premium version of DPA, it's agentless, supports unlimited users, and has less than 1% load on servers.

Given its nonexistent price and these specs, DPA Free has a lot of value, and it's an excellent place to start for those getting their feet wet. Still, for DBAs seeking more in-depth capabilities that center around response time and identify the root causes of database underperformance, I'd recommend the premium version of DPA hands down.

Digest

Database Administration
CSE 8th SEM

DBA(DATABASE ADMINISTRATION) CSE402B

QUESTION BANK

Unit-3

Q1. What is PL/SQL? What are its advantages and disadvantages?
Q2. What is difference between SQL and PL/SQL?
Q3. What is PL/SQL block?
Q4. What is PL/SQL cursor?
Q5. What are the functions of PL/SQL cursor?
Q6.What are the types of PL/SQL cursor?
Q7. What are the PL/SQL procedures?
Q8.What are the PL/SQL functions?

Note:- These are the questions which we can expect in your university exams. you can do your preparation for your university exams based on these questions.

Unit-3

PL/SQL Introduction

PL/SQL is a block structured language that enables developers to combine the power of SQL with procedural statements.All the statements of a block are passed to oracle engine all at once which increases processing speed and decreases the traffic.

Disadvantages of SQL:

- SQL doesn't provide the programmers with a technique of condition checking, looping and branching.
- SQL statements are passed to Oracle engine one at a time which increases traffic and decreases speed.
- SQL has no facility of error checking during manipulation of data.

Features of PL/SQL:

1. PL/SQL is basically a procedural language, which provides the functionality of decision making, iteration and many more features of procedural programming languages.

- PL/SQL can execute a number of queries in one block using single command.
- One can create a PL/SQL unit such as procedures, functions, packages, triggers, and types, which are stored in the database for reuse by applications.
- PL/SQL provides a feature to handle the exception which occurs in PL/SQL block known as exception handling block.
- Applications written in PL/SQL are portable to computer hardware or operating system where Oracle is operational.
- PL/SQL Offers extensive error checking.

Differences between SQL and PL/SQL:

SQL	PL/SQL
SQL is a single query that is used to perform DML and DDL operations.	PL/SQL is a block of codes that used to write the entire program blocks/procedure/ function, etc.
It is declarative, that defines what needs to be done, rather than how things need to be done.	PL/SQL is procedural that defines how the things needs to be done.
Execute as a single statement.	Execute as a whole block.

Mainly used to manipulate data.	Mainly used to create an application.
Cannot contain PL/SQL code in it.	It is an extension of SQL, so it can contain SQL inside it.

Structure of PL/SQL Block:

PL/SQL extends SQL by adding constructs found in procedural languages, resulting in a structural language that is more powerful than SQL. The basic unit in PL/SQL is a block. All PL/SQL programs a made up of blocks, which

can be nested within each other.

Typically, each block performs a logical action in the program. A block has the following structure:

```
DECLARE
    declaration statements;

BEGIN
    executable statements

EXCEPTIONS
    exception handling statements

END;
```

- Declare section starts with **DECLARE** keyword in which variables, constants, records as cursors can be declared which stores data temporarily. It basically consists definition of PL/SQL identifiers. This part of the code is optional.
- Execution section starts with **BEGIN** and ends with **END** keyword.This is a mandatory section and here the program logic is written to perform any task like loops and conditional statements. It supports all DML commands, DDL commands and SQL*PLUS built-in functions as well.

- Exception section starts with **EXCEPTION** keyword.This section is optional which contains statements that are executed when a run-time error occurs. Any exceptions can be handled in this section.

PL/SQL identifiers

There are several PL/SQL identifiers such as variables, constants, procedures, cursors, triggers etc.

- **Variables**:
  Like several other programming languages, variables in PL/SQL must be declared prior to its use. They should have a valid name and data type as well.
  Syntax for declaration of variables:

```
variable_name datatype [NOT NULL := value ];
```

Example to show how to declare variables in PL/SQL :

filter_none

brightness_4

```
SQL> SET SERVEROUTPUT ON;


SQL> DECLARE
        var1 INTEGER;
        var2 REAL;
        var3 varchar2(20) ;


BEGIN
        null;
END;
/
```

Cursors in PL/SQL

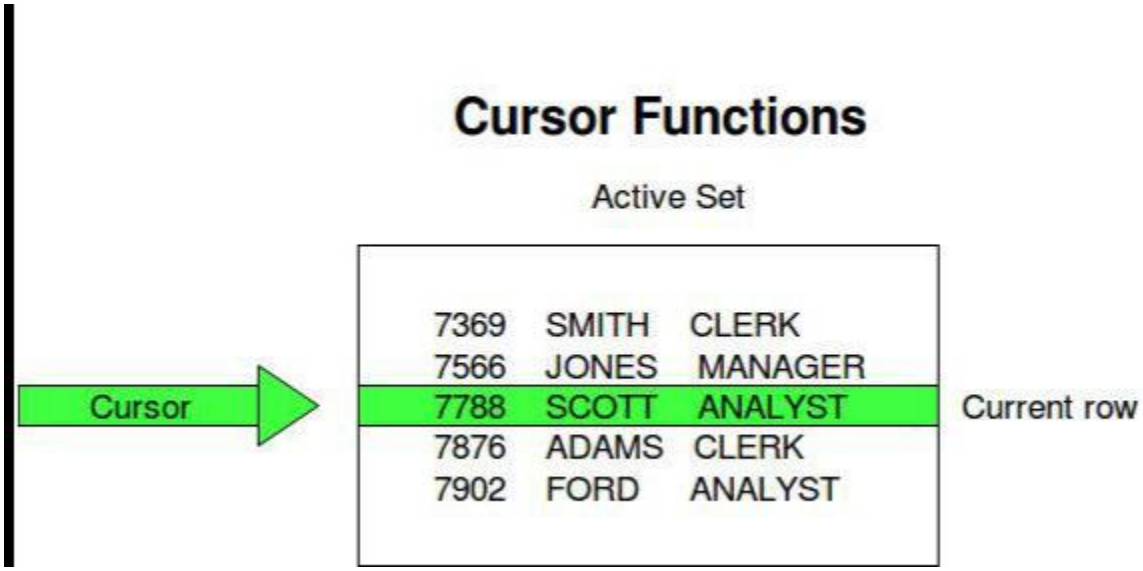
Cursor in SQL

To execute SQL statements, a work area is used by the Oracle engine for its internal processing and storing the information. This work area is private to SQL's operations. The 'Cursor' is the PL/SQL construct that allows the user to name the work area and access the stored information in it.

Use of Cursor

The major function of a cursor is to retrieve data, one row at a time, from a result set, unlike the SQL commands which operate on all the rows in the result set at one time. Cursors are used when the user needs to update records in a singleton fashion or in a row by row manner, in a database table.

The Data that is stored in the Cursor is called the *Active Data Set*. Oracle DBMS has another predefined area in the main memory Set, within which the cursors are opened. Hence the size of the cursor is limited by the size of this pre-defined area.



- **Declare Cursor:** A cursor is declared by defining the SQL statement that returns a result set.
- **Open:** A Cursor is opened and populated by executing the SQL statement defined by the cursor.
- **Fetch:** When the cursor is opened, rows can be fetched from the cursor one by one or in a block to perform data manipulation.
- **Close:** After data manipulation, close the cursor explicitly.

- **Deallocate:** Finally, delete the cursor definition and release all the system resources associated with the cursor.

Cursors are classified depending on the circumstances in which they are opened.

- **Implicit Cursor:** If the Oracle engine opened a cursor for its internal processing it is known as an Implicit Cursor. It is created "automatically" for the user by Oracle when a query is executed and is simpler to code.
- **Explicit Cursor:** A Cursor can also be opened for processing data through a PL/SQL block, on demand. Such a user-defined cursor is known as an Explicit Cursor.

Explicit cursor

An explicit cursor is defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement which returns more than one row. A suitable name for the cursor.

General syntax for creating a cursor:

```
CURSOR cursor_name IS select_statement;
```

cursor_name – A suitable name for the cursor.

select_statement – A select query which returns multiple rows

How to use Explicit Cursor?

There are four steps in using an Explicit Cursor.

- DECLARE the cursor in the Declaration section.
- OPEN the cursor in the Execution Section.
- FETCH the data from the cursor into PL/SQL variables or records in the Execution Section.
- CLOSE the cursor in the Execution Section before you end the PL/SQL Block.

Syntax:

```
DECLARE variables;
  records;
  create a cursor;
  BEGIN
```

```
OPEN cursor;
FETCH cursor;
  process the records;
  CLOSE cursor;
END;
```

SQL | Procedures in PL/SQL

PL/SQL is a block-structured language that enables developers to combine the power of SQL with procedural statements.
A stored procedure in PL/SQL is nothing but a series of declarative SQL statements which can be stored in the database catalogue. A procedure can be thought of as a function or a method. They can be invoked through triggers, other procedures, or applications on Java, PHP etc.
All the statements of a block are passed to Oracle engine all at once which increases processing speed and decreases the traffic.

Advantages:

- p They result in performance improvement of the application. If a procedure is being called frequently in an application in a single connection, then the compiled version of the procedure is delivered.
- q They reduce the traffic between the database and the application, since the lengthy statements are already fed into the database and need not be sent again and again via the application.
- r They add to code reusability, similar to how functions and methods work in other languages such as C/C++ and Java.

- p Stored procedures can cause a lot of memory usage. The database administrator should decide an upper bound as to how many stored procedures are feasible for a particular application.
- q MySQL does not provide the functionality of debugging the stored procedures.

Syntax to create a stored procedure

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
```

```
GO

-- Comments --

CREATE PROCEDURE procedure_name
   p  ,
   q  ,
r

AS
BEGIN
-- Query --
END

GO
```

Example:

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

CREATE PROCEDURE GetStudentDetails
        @StudentID int = 0
AS
BEGIN
        SET NOCOUNT ON;
```

```
SELECT FirstName, LastName, BirthDate, City, Country FROM
Students WHERE StudentID=@StudentID
```

```
END
GO
```

Syntax to modify an existing stored procedure

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

-- Comments --

ALTER PROCEDURE procedure_name
    ,
    ,


AS
BEGIN
-- Query --
END

GO
```

Example:

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

ALTER PROCEDURE GetStudentDetails
        @StudentID int = 0
```

```
AS
BEGIN
        SET NOCOUNT ON;
        SELECT FirstName, LastName, City
        FROM Students WHERE StudentID=@StudentID
END
GO
```

Syntax to drop a Procedure:

```
DROP PROCEDURE procedure_name
```

Example:

```
DROP PROCEDURE GetStudentDetails
```

Creating a Function

A standalone function is created using the **CREATE FUNCTION** statement. The simplified syntax for the **CREATE OR REPLACE PROCEDURE** statement is as follows −

```
CREATE [OR REPLACE] FUNCTION function_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
RETURN return_datatype
{IS | AS}
BEGIN
   function_body > END
[function_name];
```

Where,

- *function-name* specifies the name of the function.
- [OR REPLACE] option allows the modification of an existing function.
- The optional parameter list contains name, mode and types of the parameters. IN represents the value that will be passed from outside and

OUT represents the parameter that will be used to return a value outside of the procedure.

- The function must contain a **return** statement.
- The *RETURN* clause specifies the data type you are going to return from the function.
- *function-body* contains the executable part.
- The AS keyword is used instead of the IS keyword for creating a standalone function.

Example

The following example illustrates how to create and call a standalone function. This function returns the total number of CUSTOMERS in the customers table.

We will use the CUSTOMERS table, which we had created in the PL/SQL Variables chapter −

Select * from customers;

```
+----    +----------        +-----    +-----------        +----------        +
   | ID | NAME                | AGE | ADDRESS        | SALARY         |
+----    +----------        +-----    +-----------        +----------        +
|      1 | Ramesh        |       32 | Ahmedabad      |          2000.00  |
|      2 | Khilan        |       25 | Delhi          |          1500.00  |
|      3 | kaushik       |       23 | Kota           |          2000.00  |
|      4 | Chaitali      |       25 | Mumbai         |          6500.00  |
|      5 | Hardik        |       27 | Bhopal         |          8500.00  |
|      6 | Komal         |       22 | MP             |          4500.00  |
+----    +----------        +-----    +-----------        +----------        +
```

```
CREATE OR REPLACE FUNCTION totalCustomers
RETURN number IS
    total number(2) := 0;
BEGIN
    SELECT count(*) into total
    FROM customers;

    RETURN total;
END;
/
```

When the above code is executed using the SQL prompt, it will produce the following result −

Function created.

Calling a Function

While creating a function, you give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task. When a program calls a function, the program control is transferred to the called function.

A called function performs the defined task and when its return statement is executed or when the **last end statement** is reached, it returns the program control back to the main program.

To call a function, you simply need to pass the required parameters along with the function name and if the function returns a value, then you can store the returned value. Following program calls the function **totalCustomers** from an anonymous block −

```
DECLARE
    □ number(2);
BEGIN
    c := totalCustomers();
    dbms_output.put_line('Total no. of Customers: ' || c);
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result −

Total no. of Customers: 6

PL/SQL procedure successfully completed.

The following example demonstrates Declaring, Defining, and Invoking a Simple PL/SQL Function that computes and returns the maximum of two values.

```
DECLARE
    □ number;
    □ number;
    □ number;
FUNCTION findMax(x IN number, y IN number)
```

```
RETURN number
IS
        number;
BEGIN
      IF x > y THEN z:=
            x;
      ELSE Z:= y;
      END IF;
      RETURN z;
END;
BEGIN       a:=
      23;       b:=
      45;
      c := findMax(a, b);
      dbms_output.put_line(' Maximum of (23,45): ' || c); END;
/
```

When the above code is executed at the SQL prompt, it produces the following result −

```
Maximum of (23,45): 45

PL/SQL procedure successfully completed.
```

PL/SQL Recursive Functions

We have seen that a program or subprogram may call another subprogram. When a subprogram calls itself, it is referred to as a recursive call and the process is known as **recursion**.

To illustrate the concept, let us calculate the factorial of a number. Factorial of a number n is defined as −

```
n! = n*(n-1)!
      n*(n-1)*(n-2)!
            ...
      n*(n-1)*(n-2)*(n-3)... 1
```

The following program calculates the factorial of a given number by calling itself recursively −

```
DECLARE
```

```
    num number;
    factorial number;

FUNCTION fact(x number)
RETURN number
IS
    □ number;
BEGIN
    IF x=0 THEN f
        := 1;
    ELSE
        f := x * fact(x-1);
    END IF;
RETURN f;
END;

BEGIN
    num:= 6;
    factorial := fact(num);
    dbms_output.put_line(' Factorial '|| num || ' is '
□   factorial);
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result −

Factorial 6 is 720

DBA(DATABASE ADMINISTRATION) CSE402B

QUESTION BANK

Unit-4

Q1. What is active database . Explain its architecture ?
Q2. What are ECA rules?
Q3. How to implement ECA rules?
Q4. What is cube technology explain in detail?
Q5. Categorize cube technology in brief?
Q6.What is Data warehouse. Explain in detail.
Q7. Explain top down approach of data warehouse.
Q8.Explain bottom up approach of data warehouse.

UNIT-4

What does Active Database Management System (ADBMS) mean?

An active database management system (ADBMS) is an event-driven system in which schema or data changes generate events monitored by active rules. Active database management systems are invoked by synchronous events generated by user or application programs as well as external asynchronous data change events such as a change in sensor value or time.

Active database management systems support event monitoring. They store events in event history as an event type and time; the former represents any kind of primitive event, while the latter represents time the event occurred. ADMSs clearly define rule semantics such as event consumption policy, event detection and coupling modes along with instance or set oriented semantics.

A common event consumption policy includes the following parameter contexts:

- Cumulative: All instances of primitive event are consumed if a complex event occurs.
- Chronicle: Events are consumed in time order.
- Recent: The latest instances of primitive events that are part of complex events are consumed in time order.

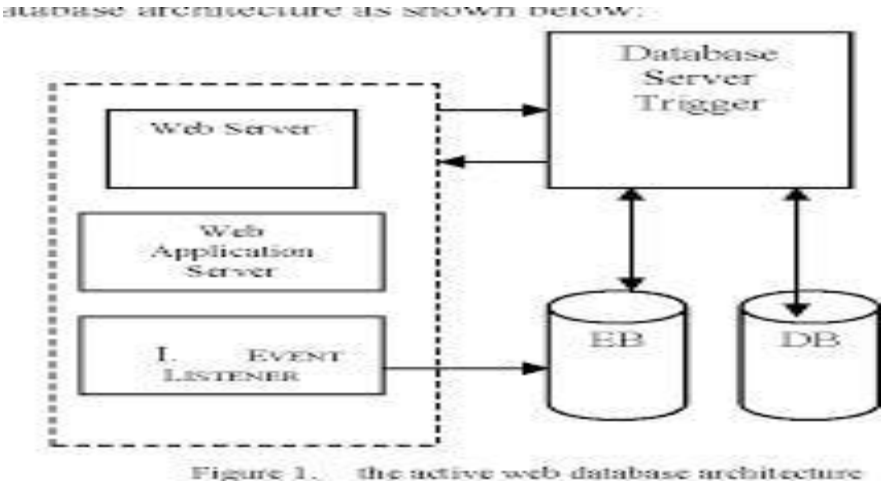


Figure 1. the active web database architecture

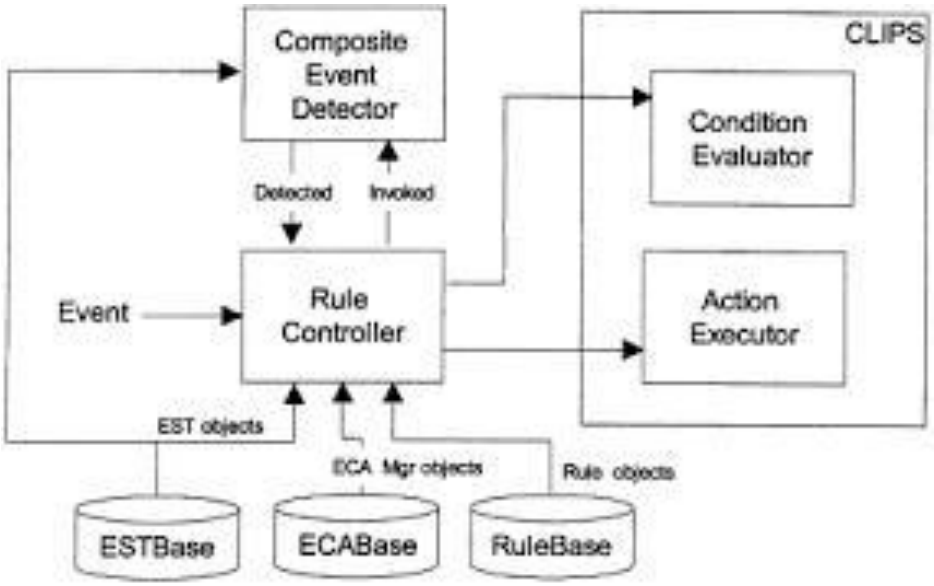
Event-condition-action rule (ECA rule)

An event-condition-action rule (ECA rule) is the method underlying event-driven computing, in which actions are triggered by events, given the existence of specific conditions.

Events with significance to the system are identified within an event-driven program. An event could be some user action, a transmission of sensor data or a message from some other program or system, among an almost infinite number of other possibilities. The ECA rule specifies how events drive the desired program responses. When an event with significance for the system occurs, the conditions are checked for or evaluated; if the conditions exist or meet pre-established criteria, the appropriate action is executed.

ECA rules originated in active databases and have since been used in areas including personalization, big data management and business process automation. The model is being explored for M2M (machine-to-machine) networking, Internet of Things (IoT), cognitive computing and the semantic web.

Implementation of ECA rules



Event condition action (**ECA**) is a short-cut for referring to the structure of active rules in event driven architecture and active database systems.

Such a rule traditionally consisted of three parts:

- The *event* part specifies the signal that triggers the invocation of the rule
- The *condition* part is a logical test that, if satisfied or evaluates to true, causes the action to be carried out
- The *action* part consists of updates or invocations on the local data

This structure was used by the early research in active databases which started to use the term ECA. Current state of the art ECA rule engines use many variations on rule structure. Also other features not considered by the early research is introduced, such as strategies for event selection into the event part.

In a memory-based rule engine, the condition could be some tests on local data and actions could be updates to object attributes. In a database system, the condition could simply be a query to the database, with the result set (if not null) being passed to the action part for changes to the database. In either case, actions could also be calls to external programs or remote procedures.

Note that for database usage, updates to the database are regarded as internal events. As a consequence, the execution of the action part of an active rule can match the event part of the same or another active rule, thus triggering it. The equivalent in a memory-based rule engine would be

to invoke an external method that caused an external event to trigger another ECA rule.

ECA rules can also be used in rule engines that use variants of the Rete algorithm for rule processing.

Cube technology

Data cube is a multi-dimensional structure. Data cube is a data abstraction to view aggregated data from a number of perspectives. The dimensions are aggregated as the 'measure' attribute, as the remaining dimensions are known as the 'feature' attributes.

Data is viewed on a cube in a multidimensional manner. The aggregated and summarized facts of variables or attributes can be viewed. This is the requirement where OLAP plays a role.

What is data cube technology used for?

Data cubes are commonly used for easy interpretation of data. It is used to represent data along with dimensions as some measures of business needs. Each dimension of the cube represents some attribute of the database, e.g Sales per day, month or year.

In computer programming contexts, a **data cube** (or **datacube**) is a multi-dimensional ("n-D") array of values. Typically, the term datacube is applied in contexts where these arrays are massively larger than the hosting computer's main memory; examples include multi-terabyte/petabyte data warehouses and time series of image data.

The data cube is used to represent data (sometimes called facts) along some measure of interest. For example, in OLAP such measures could be the subsidiaries a company has, the products the company offers, and time; in this setup, a fact would be a sales event where a particular product has been sold in a particular subsidiary at a particular time. In satellite image timeseries measures would be Latitude and Longitude coordinates and time; a fact would be a pixel at a given space/time coordinate as taken by the satellite (following some processing that is not of concern here). Even though it is called a *cube* (and the examples provided above happen to be 3-dimensional for brevity), a data cube generally is a multi-dimensional concept which can be 1-dimensional, 2-dimensional, 3-

dimensional, or higher-dimensional. In any case, every dimension represents a separate measure whereas the cells in the cube represent the facts of interest. Sometimes cubes hold only few values with the rest being *empty*, i.e.: undefined, sometimes most or all cube coordinates hold a cell value. In the first case such data are called *sparse*, in the second case they are called *dense*, although there is no hard delineation between both.

A data cube is generally used to easily interpret data. It is especially useful when representing data together with dimensions as certain measures of business requirements. A cube's every dimension represents certain characteristic of the database, for example, daily, monthly or yearly sales. The data included inside a data cube makes it possible analyze almost all the figures for virtually any or all customers, sales agents, products, and much more. Thus, a data cube can help to establish trends and analyze performance.

Data cubes are mainly categorized into two categories:

- Multidimensional Data Cube: Most OLAP products are developed based on a structure where the cube is patterned as a multidimensional array. These multidimensional OLAP (MOLAP) products usually offers improved performance when compared to other approaches mainly because they can be indexed directly into the structure of the data cube to gather subsets of data. When the number of dimensions is greater, the cube becomes sparser. That means that several cells that represent particular attribute combinations will not contain any aggregated data. This in turn boosts the storage requirements, which may reach undesirable levels at times, making the MOLAP solution untenable for huge data sets with many dimensions. Compression techniques might help; however, their use can damage the natural indexing of MOLAP.

- Relational OLAP: Relational OLAP make use of the relational database model. The ROLAP data cube is employed as a bunch of relational tables (approximately twice as many as the quantity of

dimensions) compared to a multidimensional array. Each one of these tables, known as a cuboid, signifies a specific view.

What is a Data Warehouse?

A data warehouse is a system that pulls together data from many different sources within an organization for reporting and analysis. The reports created from complex queries within a data warehouse are used to make business decisions.In more comprehensive terms, a data warehouse is a consolidated view of either a physical or logical data repository collected from various systems. The primary focus of a data warehouse is to provide a correlation between data from existing systems, i.e., product inventory stored in one system purchase orders for a specific customer, stored in another system. Data warehouses are used for online analytical processing (OLAP), which uses complex queries to analyze rather than process transactions.

What is a Database?

A database contains information organized in columns, rows, and tables that is periodically indexed to make accessing relevant information more accessible.

Many enterprises and organizations create and manage databases using a database management system. Special DBMS software can be used create and store product inventory and customer information, for example.

Organizations most often use databases for online transaction processing (OLTP). Database software needs to provide easy access to information and fast querying so that transactions can be carried out efficiently. Databases are often referred to as operational systems, meaning they are used to process day-to-day transactions in an organization.

While most read, write, and report generation is usually managed by a Database Administrator, some transactional databases provide atomicity, consistency,

isolation, and durability (ACID) compliance to ensure that the information contains in the database is consistent and any transactions that take place, are complete.

Data Warehouse vs Database

Data warehouses and databases are both relational data systems, but were built to serve different purposes. A data warehouse is built to store large quantities of historical data and enable fast, complex queries across all the data, typically using Online Analytical Processing (OLAP). A database was built to store current transactions and enable fast access to specific transactions for ongoing business processes, known as Online Transaction Processing (OLTP).

Optimization

A database is optimized to maximize the speed and efficiency with which data is updated (added, modified, or deleted) and enable faster analysis and data access. Databases use Online Transactional Processing (OLTP) to delete, insert, replace, and update large numbers of short online transactions. Other features include fast query processing, multi-access data integrity, and a number of processed transactions per second. Databases performing OLTP transactions contain and maintain current, and detailed data from a single source. However, due to the number of table joins, performing analytical queries is difficult and requires an experienced database administrator or developer familiar with the application, to write queries that result in any meaningful analysis.

Response times from databases need to be extremely quick for efficient transaction processing. The most important aspect of a database is that the write operation is recorded in the system. A business that sells products online wouldn't be in business very long if its database didn't make a record of every purchase.

Data warehouses use Online Analytical Processing (OLAP) that is optimized to handle a low number of complex queries on aggregated large historical data sets. Tables are denormalized and transformed to yield summarized data, multidimensional views, and faster query response times. Additionally, query response times are used to measure an OLAP system's effectiveness. As a function of business intelligence, OLAP allows managers and analysts to select, extract,

view, and analyze corporate data to identify and obtain insights on corporate trends as well as identify potential issues.

Data Structure

Most databases use a normalized data structure. Data normalization means reorganizing data so that it contains no redundant data, and all related data items are stored together, with related data separated into multiple tables. Normalizing data ensures the database takes up minimal disk space while response times are maximized.

Product

ProductID	Desc
1	Mtn Bike #726
2	Road Bike #871
3	Touring Bike #378

Color

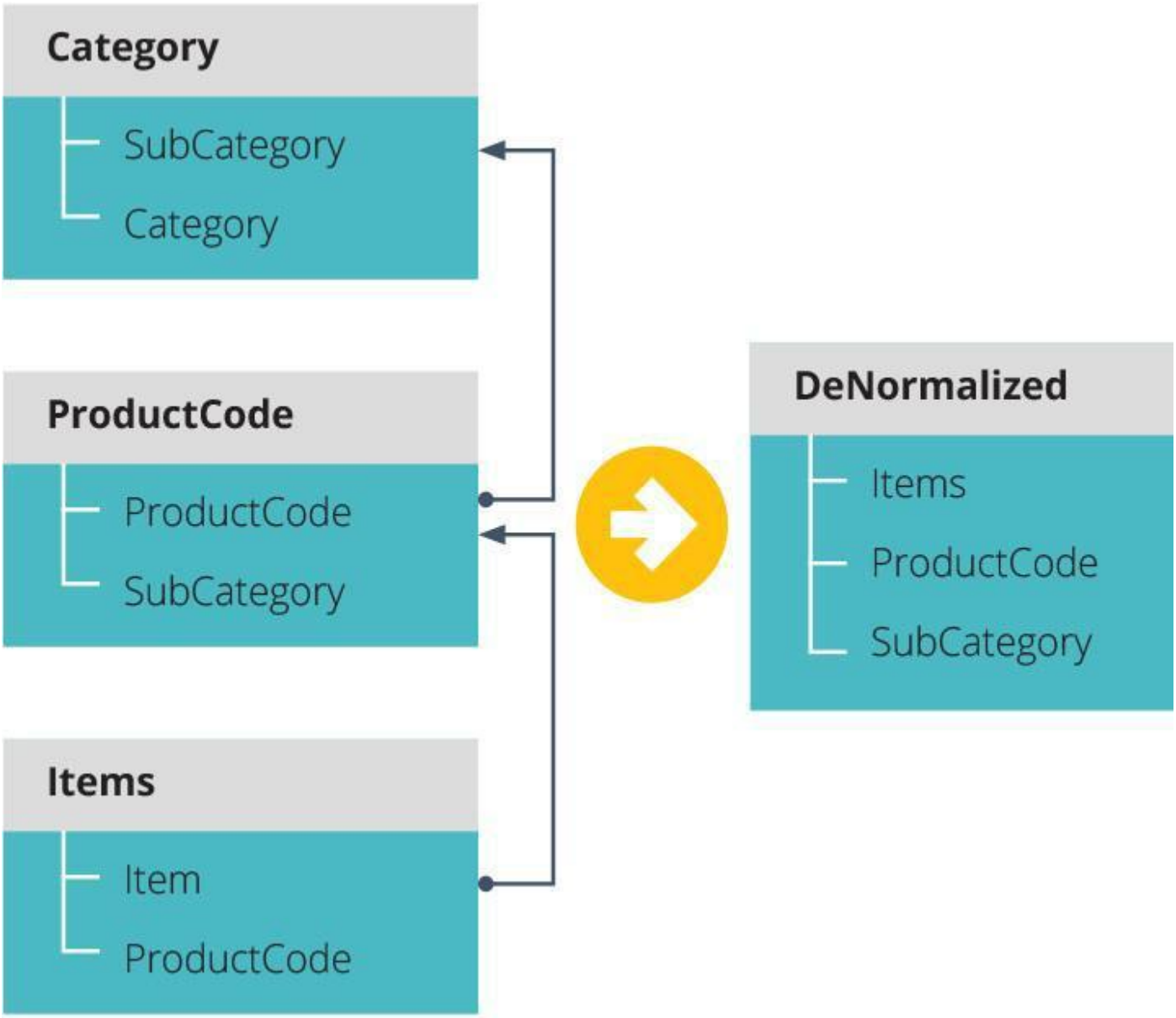
ColorID	Desc
1	Red
2	Black
3	Silver
4	Mauve

Product-Color

ProductID	ColorID
1	1
1	2
2	1
2	2
2	3
3	1
3	3
3	4

The more normalized your data is, the more complex the queries needed to read the data because a single query combines data from many tables. This puts a huge strain on computing resources.

The data in a data warehouse does not need to be organized for quick transactions. Therefore, data warehouses normally use a denormalized data structure.
A denormalized data structure uses fewer tables because it groups data and doesn't exclude data redundancies. Denormalization offers better performance when reading data for analytical purposes.



Data Timeline

 database processes day-to-day transactions within an organization. Therefore, databases typically don't contain historical data—current data is all that matters in a normalized relational database.

Data warehouses are used for analytical purposes and business reporting. Data warehouses typically store historical data by integrating copies of transaction data from disparate sources. Data warehouses can also use real-time data feeds for reports that use the most current, integrated information.

Analysis

While databases are normally used for transactional purposes, analytical queries can still be performed on the data. The problem is that the complexity of the data's normalized organization makes analytical queries difficult to carry out. A skilled developer or analyst will be required to create such analytical queries. The depth of analysis is limited to static one-time reports because databases just give a snapshot overview of data at a specific time.

The structure of data warehouses makes analytical queries much simpler to perform. No advanced knowledge of database applications is required. Analytics in data warehouses is dynamic, meaning it takes into account data that changes over time.

Concurrent Users

An OLTP database supports thousands of concurrent users. Many users must be able to interact with the database simultaneously without it affecting the system's performance.

Data warehouses support a limited number of concurrent users compared to operational systems. The data warehouse is separated from front-end applications and it relies on complex queries, thus necessitating a limit on how many people can use the system simultaneously.

Database vs. Data Warehouse Applications

With databases, there is a one-to-one relationship with a single application as its source. A credit card processing application is an excellent example of a single data source that can run on an OLTP database. This type of database contains highly detailed data as well as a detailed relational views. Tables are normalized to achieve efficient storage, concurrent transaction processing, as well as return quick query results.

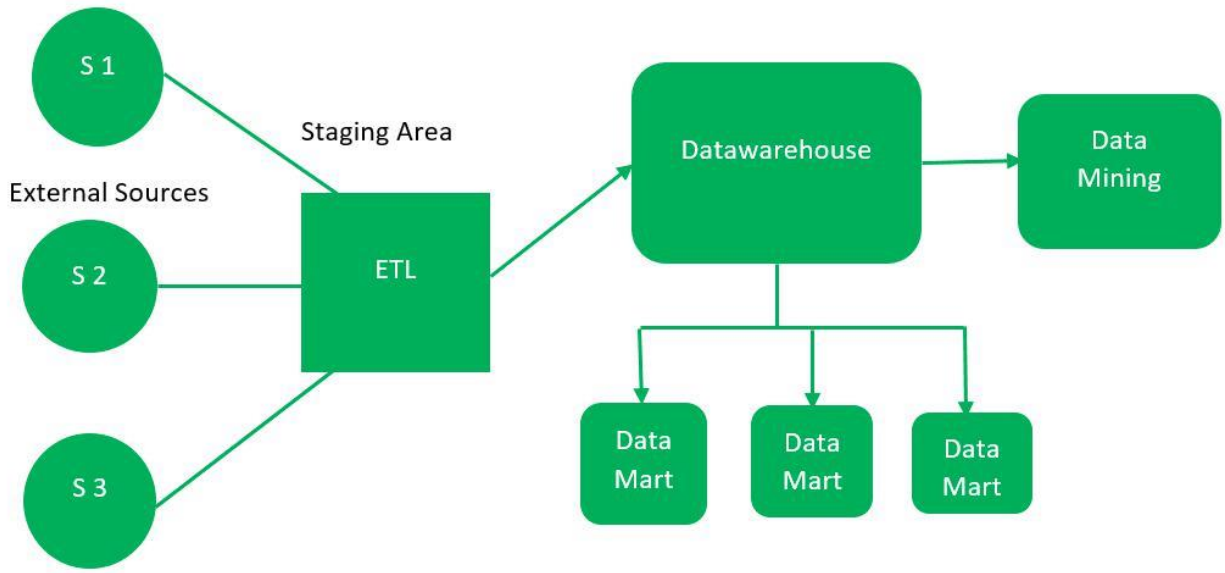
Data warehouses store summarized historical data from many different applications. There is a one to many relationships between a data warehouses and

the applications that serve as data sources. Examples of data sources include but are not limited to customer relationship management (CRM), enterprise resource management (ERP), or even social media data.

Data Warehouse Architecture

A **data-warehouse** is a heterogeneous collection of different data sources organised under a unified schema. There are 2 approaches for constructing data-warehouse: Top-down approach and Bottom-up approach are explained as below.

1. Top-down approach:



the essential components are discussed below:

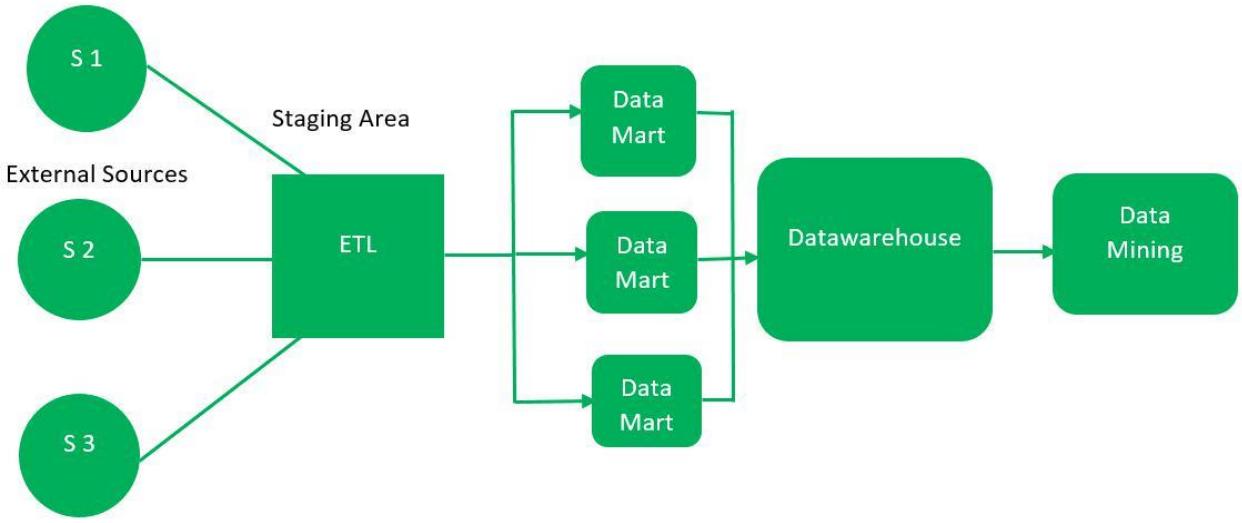
- **External Sources –**
  External source is a source from where data is collected irrespective of the type of data. Data can be structured, semi structured and unstructured as well.

- **Stage Area –**
  Since the data, extracted from the external sources does not follow a particular format, so there is a need to validate this data to load into datawarehouse. For this purpose, it is recommended to use **ETL** tool.
  E(Extracted): Data is extracted from External data source.
  T(Transform): Data is transformed into the standard format.
  L(Load): Data is loaded into datawarehouse after transforming it into the standard format.
- **Data-warehouse –**
  After cleansing of data, it is stored in the datawarehouse as central repository. It actually stores the meta data and the actual data gets stored in the data marts. **Note** that datawarehouse stores the data in its purest form in this top-down approach.
- **Data Marts –**
  Data mart is also a part of storage component. It stores the information of a particular function of an organisation which is handled by single authority. There can be as many number of data marts in an organisation depending upon the functions. We can also say that data mart contains subset of the data stored in data warehouse.
- **Data Mining –**
  The practice of analysing the big data present in datawarehouse is data mining. It is used to find the hidden patterns that are present in the database or in datawarehouse with the help of algorithm of data mining. This approach is defined by **Inmon** as – datawarehouse as a central repository for the complete organisation and data marts are created
  from it after the complete datawarehouse has been created.

- Since the data marts are created from the datawarehouse, provides consistent dimensional view of data marts.
- Also, this model is considered as the strongest model for business changes. That's why, big organisations prefer to follow this approach.
- Creating data mart from datawarehouse is easy.

- The cost, time taken in designing and its maintainence is very high.

Bottom-up approach:



- First, the data is extracted from external soures (same as happens in top-down approach).
- Then, the data go through the staging area (as explained above) and loaded into data marts instead of data warehouse. The data marts are created first and provide reporting capability. It addresses a single business area.
- These data marts are then integrated into data warehouse.

This approach is given by **Kinball** as – data marts are created first and provides a thin view for analyses and data warehouse is created after complete data marts have been created.

- As the data marts are created first, so the reports are quickly generated.
- We can accommodate more number of data marts here and in this way data warehouse can be extended.
- Also, the cost and time taken in designing this model is low comparatively.

\endash This model is not strong as top-down approach as dimensional view of data marts is not consistent as it is in above approach.

CSE-B.tech 8th Sem-Data base Administration (DBA)
CSE-402-B

Prepared By: Ms. Deepa