BCA Semester – 3

CS-15 RDBMS Using Oracle

ORACLE

H. & H. B. Kotak Institute of Science, Rajkot
BCA Department

Unit – 1

DBMS Overview, SQL, SQL*Plus

Introduction to DBMS

The **database** is a collection of inter-related data which is used to retrieve, insert and delete the data efficiently. It is also used to organize the data in the form of a table, schema, views, and reports, etc. For example: The college Database organizes the data about the admin, staff, students and faculty etc. Using the database, you can easily retrieve, insert, and delete the information.

Database management system (DBMS) is a software which is used to manage the database. For example: MySQL, Oracle, etc are a very popular commercial database which is used in different applications. DBMS provides an interface to perform various operations like database creation, storing data in it, updating data, creating a table in the database and a lot more. It provides protection and security to the database. In the case of multiple users, it also maintains data consistency. DBMS allows users the following tasks:

- **Data Definition**: It is used for creation, modification, and removal of definition that defines the organization of data in the database.
- **Data Updation**: It is used for the insertion, modification, and deletion of the actual data in the database.
- **Data Retrieval**: It is used to retrieve the data from the database which can be used by applications for various purposes.
- **User Administration**: It is used for registering and monitoring users, maintain data integrity, enforcing data security, dealing with concurrency control, monitoring performance and recovering information corrupted by unexpected failure.

Introduction to RDBMS

RDBMS stands for Relational Database Management Systems. All modern database management systems like SQL, MS SQL Server, IBM DB2, ORACLE, My-SQL and Microsoft Access are based on RDBMS. It is called Relational Data Base Management System (RDBMS) because it is based on relational model introduced by Dr. E. F. Codd. Data is represented in terms of tuples (rows) in RDBMS. Relational database is most commonly used database. It contains number of tables and each table has its own primary key. Due to a collection of organized set of tables, data can be accessed easily in RDBMS.

DBMS v/s RDBMS

DBMS	RDBMS
DBMS applications store **data as file**.	RDBMS applications store **data in a tabular form**.
In DBMS, data is generally stored in either a hierarchical form or a navigational form.	In RDBMS, the tables have an identifier called primary key and the data values are stored in the form of tables.
Normalization is not present in DBMS.	**Normalization is** present in RDBMS.
DBMS does **not apply any security** with regards to data manipulation.	RDBMS **defines the integrity constraint** for the purpose of ACID (Atomocity, Consistency, Isolation and Durability) property.
DBMS uses file system to store data, so there will be **no relation between the tables**.	in RDBMS, data values are stored in the form of tables, so a **relationship** between these data values will be stored in the form of a table as well.

DBMS has to provide some uniform methods to access the stored information.	RDBMS system supports a tabular structure of the data and a relationship between them to access the stored information.
DBMS **does not support distributed database**.	RDBMS **supports distributed database**.
DBMS is meant to be for small organization and **deal with small data**. it supports **single user**.	RDBMS is designed to **handle large amount of data**. it supports **multiple users**.
Examples of DBMS are file systems, **xml** etc.	Example of RDBMS are **mysql**, **postgre**, **sql server**, **oracle** etc.

Dr.E.F.Codd Rules

Dr E.F.Codd, also known to the world as the 'Father of Database Management Systems' had propounded 12 rules which are in-fact 13 in number. The rules are numbered from zero to twelve. According to him, a DBMS is fully relational if it abides by all his twelve rules. Till now, only few databases abide by all the eleven rules. His twelve rules are fondly called 'E.F.Codd's Twelve Commandments'. His brilliant and seminal research paper 'A Relational Model of Data for Large Shared Data Banks' in its entirety is a visual treat to eyes. Here is brief note on E.F Codd's Twelve rules:

- **Rule 0 – Foundation rule:** Any relational database management system that is propounded to be RDBMS or advocated to be a RDBMS should be able to manage the stored data in its entirety through its relational capabilities.
- **Rule 1 – Rule of Information:** Relational Databases should store the data in the form of relations. Tables are relations in Relational Database Management Systems. Be it any user defined data or meta-data, it is important to store the value as an entity in the table cells.
- **Rule 2 – Rule of Guaranteed Access:** The use of pointers to access data logically is strictly forbidden. Every data entity which is atomic in nature should be accessed logically by using a right combination of the name of table, primary key represented by a specific row value and column name represented by attribute value.
- **Rule 3 – Rule of Systematic Null Value Support:** Null values are completely supported in relational databases. They should be uniformly considered as 'missing information'. Null values are independent of any data type. They should not be mistaken for blanks or zeroes or empty strings. Null values can also be interpreted as 'inapplicable data' or 'unknown information.'
- **Rule 4 – Rule of Active and online relational Catalog:** In the Database Management Systems lexicon, 'metadata' is the data about the database or the data about the data. The active online catalog that stores the metadata is called 'Data dictionary'. The so called data dictionary is accessible only by authored users who have the required privileges and the query languages used for accessing the database should be used for accessing the data of data dictionary.
- **Rule 5 – Rule of Comprehensive Data Sub-language:** A single robust language should be able to define integrity constraints, views, data manipulations, transactions and authorizations. If the database allows access to the aforementioned ones, it is violating this rule.
- **Rule 6 – Rule of Updating Views:** Views should reflect the updates of their respective base tables and vice versa. A view is a logical table which shows restricted data. Views generally make the data readable but not modifiable. Views help in data abstraction.

- **Rule 7 – Rule of Set level insertion, update and deletion:** A single operation should be sufficient to retrieve, insert, update and delete the data.
- **Rule 8 – Rule of Physical Data Independence:** Batch and end user operations are logically separated from physical storage and respective access methods.
- **Rule 9 – Rule of Logical Data Independence:** Batch and end users can change the database schema without having to recreate it or recreate the applications built upon it.
- **Rule 10 – Rule of Integrity Independence:** Integrity constraints should be available and stored as metadata in data dictionary and not in the application programs.
- **Rule 11 – Rule of Distribution Independence:** The Data Manipulation Language of the relational system should not be concerned about the physical data storage and no alterations should be required if the physical data is centralized or distributed.
- **Rule 12 – Rule of Non Subversion:** Any row should obey the security and integrity constraints imposed. No special privileges are applicable.

Almost all full scale DBMSs are RDMSs. Oracle implements 11+ rules and so does Sybase. SQL Server also implements 11+ rules while FoxPro implements 7+ rules.

Importance of E. R. Diagram in Relational DBMS

ER Diagram stands for Entity Relationship Diagram, also known as ERD is a diagram that displays the relationship of entity sets stored in a database. In other words, ER diagrams help to explain the logical structure of databases. ER diagrams are created based on three basic concepts: entities, attributes and relationships. ER Diagrams contain different symbols that use rectangles to represent entities, ovals to define attributes and diamond shapes to represent relationships.

ER diagrams are a visual tool which is helpful to represent the ER model. It was proposed by Peter Chen in 1971 to create a uniform convention which can be used for relational database and network. He aimed to use an ER model as a conceptual modelling approach.

Entity Relationship Diagram Symbols & Notations: mainly contains three basic symbols which are rectangle, oval and diamond to represent relationships between elements, entities and attributes. There are some sub-elements which are based on main elements in ERD Diagram. ER Diagram is a visual representation of data that describes how data is related to each other using different ERD Symbols and Notations. Following are the main components and its symbols in ER Diagrams:

- Rectangles: This Entity Relationship Diagram symbol represents entity types
- Ellipses : Symbol represent attributes
- Diamonds: This symbol represents relationship types
- Lines: It links attributes to entity types and entity types with other relationship types
- Primary key: attributes are underlined
- Double Ellipses: Represent multi-valued attributes

Entity: A real-world thing either living or non-living that is easily recognizable and nonrecognizable. It is anything in the enterprise that is to be represented in our database. It may be a physical thing or simply a fact about the enterprise or an event that happens in the real world. An entity can be place, person, object, event or a concept, which stores data in the database. The characteristics of entities

are must have an attribute, and a unique key. Every entity is made up of some 'attributes' which represent that entity.

Attributes: It is a single-valued property of either an entity-type or a relationship-type. For example, a lecture might have attributes: time, date, duration, place, etc. An attribute in ER Diagram examples, is represented by an Ellipse.

Relationship: Relationship is nothing but an association among two or more entities. E.g., Tom works in the Chemistry department. Entities take part in relationships. We can often identify relationships with verbs or verb phrases. Defines the numerical attributes of the relationship between two entities or entity sets. Different types of cardinal relationships are:

One-to-One Relationships: One entity from entity set X can be associated with at most one entity of entity set Y and vice versa. Example: One student can register for numerous courses. However, all those courses have a single line back to that one student.

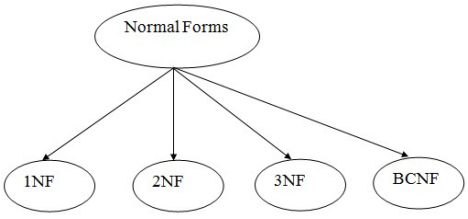
One-to-Many Relationships: One entity from entity set X can be associated with multiple entities of entity set Y, but an entity from entity set Y can be associated with at least one entity. For example, one class is consisting of multiple students.

May to One Relationships: More than one entity from entity set X can be associated with at most one entity of entity set Y. However, an entity from entity set Y may or may not be associated with more than one entity from entity set X. For example, many students belong to the same class.

Many-to-Many Relationships: One entity from X can be associated with more than one entity from Y and vice versa. For example, Students as a group are associated with multiple faculty members, and faculty members can be associated with multiple students.

Normalization

Normalization is the process of organizing the data in the database. Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate the undesirable characteristics like Insertion, Update and Deletion Anomalies. Normalization divides the larger table into the smaller table and links them using relationship. The normal form is used to reduce redundancy from the database table. There are the four types of normal forms:



Normal Form	Description
1NF	A relation is in 1NF if it contains an atomic value.
2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.
3NF	A relation will be in 3NF if it is in 2NF and no transition dependency exists.
4NF	A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
5NF	A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.

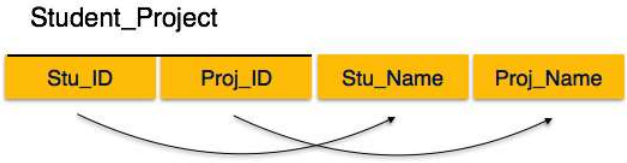
First Normal Form (1NF): A relation will be 1NF if it contains an atomic value. It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute. First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

Course	Content
Programming	Java, c++
Web	HTML, PHP, ASP

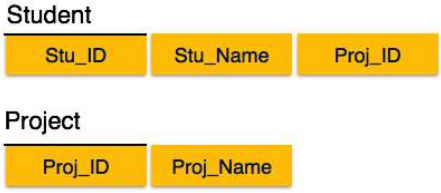
We re-arrange the relation (table) as below, to convert it to First Normal Form.

Course	Content
Programming	Java
Programming	c++
Web	HTML
Web	PHP
Web	ASP

Second Normal Form (2NF): In the 2NF, relational must be in 1NF. In the second normal form, all non-key attributes are fully functional dependent on the primary key.



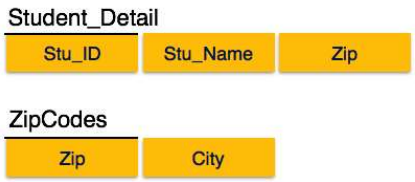
We see here in Student_Project relation that the prime key attributes are Stu_ID and Proj_ID. According to the rule, non-key attributes, i.e. Stu_Name and Proj_Name must be dependent upon both and not on any of the prime key attribute individually. But we find that Stu_Name can be identified by Stu_ID and Proj_Name can be identified by Proj_ID independently. This is called partial dependency, which is not allowed in Second Normal Form.



We broke the relation in two as depicted in the above picture. So there exists no partial dependency.

Third Normal Form (3NF): A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency. 3NF is used to reduce the data duplication. It is also used to achieve the data integrity. If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form. A relation is in third normal form if it holds at least one of the following conditions for every non-trivial function dependency X → Y. X is a super key. Y is a prime attribute, i.e., each element of Y is part of some candidate key.

We find that in the above Student_detail relation, Stu_ID is the key and only prime key attribute. We find that City can be identified by Stu_ID as well as Zip itself. Neither Zip is a superkey nor is City a prime attribute. Additionally, Stu_ID → Zip → City, so there exists transitive dependency. To bring this relation into third normal form, we break the relation into two relations as follows –



Boyce Codd normal form (BCNF): BCNF is the advance version of 3NF. It is stricter than 3NF. A table is in BCNF if every functional dependency X → Y, X is the super key of the table. For BCNF, the table should be in 3NF, and for every FD, LHS is super key. In the above image, Stu_ID is the super-key in the relation Student_Detail and Zip is the super-key in the relation ZipCodes. So, Stu_ID → Stu_Name, Zip and Zip → City Which confirms that both the relations are in BCNF.

Introduction to SQL

SQL stands for Structured Query Language. It is designed for managing data in a relational database management system (RDBMS). It is pronounced as S-Q-L or sometime See-Qwell. SQL is a database language, it is used for database creation, deletion, fetching rows, and modifying rows, etc. SQL is based on relational algebra and tuple relational calculus. All DBMS like MySQL, Oracle, MS Access, Sybase, Informix, PostgreSQL, and SQL Server use SQL as standard database language.

SQL is a programming language for Relational Databases. It is designed over relational algebra and tuple relational calculus. SQL comes as a package with all major distributions of RDBMS. SQL comprises both data definition and data manipulation languages. Using the data definition properties of SQL, one can design and modify database schema, whereas data manipulation properties allows SQL to store and retrieve data from database.

SQL Commands/Statements

SQL statements are started with any of the SQL commands/keywords like SELECT, INSERT, UPDATE, DELETE, ALTER, DROP etc. and the statement ends with a semicolon (;). SQL is not case sensitive. Generally SQL keywords are written in uppercase. SQL statements are dependent on text lines. We can place a single SQL statement on one or multiple text lines. You can perform most of the action in a database with SQL statements. SQL depends on relational algebra and tuple relational calculus. The following are the category of SQL Statements:

Data Definition Language (DDL): SQL uses the following set of commands to define database schema:

- CREATE: Creates new databases, tables and views from RDBMS.
- DROP: Drops commands, views, tables, and databases from RDBMS.
- ALTER: Modifies database schema.

Data Manipulation Language (DML): SQL is equipped with data manipulation language (DML). DML modifies the database instance by inserting, updating and deleting its data. DML is responsible for all forms data modification in a database. SQL contains the following set of commands in its DML section:

- INSERT: This command is used for inserting values into the rows of a table (relation).
- UPDATE: This command is used for updating or modifying the values of columns in a table (relation).
- DELETE: This command is used for removing one or more rows from a table (relation).

Data Query Language (DQL): SELECT This is one of the fundamental query command of SQL. It is similar to the projection operation of relational algebra. It selects the attributes based on the condition described by WHERE clause. FROM − This clause takes a relation name as an argument from which attributes are to be selected/projected. In case more than one relation names are given, this clause corresponds to Cartesian product. WHERE − This clause defines predicate or conditions, which must match in order to qualify the attributes to be projected.

Transaction Control Language (TCL): Transaction Control Language commands are used to manage transactions in the database. These are used to manage the changes made by DML-statements. It also allows statements to be grouped together into logical transactions.

- COMMIT: Commit command is used to permanently save any transaction into the database.
- ROLLBACK: This command restores the database to last committed state. It is also used with savepoint command to jump to a savepoint in a transaction.
- SAVEPOINT: Savepoint command is used to temporarily save a transaction so that you can rollback to that point whenever necessary.

Data Control Language (DCL): A Data Control Language is a syntax similar to a computer programming language used to control access to data stored in a database (Authorization). In particular, it is a component of Structured Query Language (SQL).

- GRANT: allow specified users to perform specified tasks.
- REVOKE: cancel previously granted or denied permissions.

SQL Datatypes

Data types are used to represent the nature of the data that can be stored in the database table. For example, in a particular column of a table, if we want to store a string type of data then we will have to declare a string data type of this column. Data types mainly classified into three categories for every database.

Oracle String data types

CHAR(size)	It is used to store character data within the predefined length. It can be stored up to 2000 bytes.
NCHAR(size)	It is used to store national character data within the predefined length. It can be stored up to 2000 bytes.
VARCHAR2(size)	It is used to store variable string data within the predefined length. It can be stored up to 4000 byte.
VARCHAR(SIZE)	It is the same as VARCHAR2(size). You can also use VARCHAR(size), but it is suggested to use VARCHAR2(size)

NVARCHAR2(size)	It is used to store Unicode string data within the predefined length. We have to must specify the size of NVARCHAR2 data type. It can be stored up to 4000 bytes.

Oracle Numeric Data Types

NUMBER(p, s)	It contains precision p and scale s. The precision p can range from 1 to 38, and the scale s can range from -84 to 127.
FLOAT(p)	It is a subtype of the NUMBER data type. The precision p can range from 1 to 126.
BINARY_FLOAT	It is used for binary precision(32-bit). It requires 5 bytes, including length byte.
BINARY_DOUBLE	It is used for double binary precision (64-bit). It requires 9 bytes, including length byte.

Oracle Date and Time Data Types

DATE	It is used to store a valid date-time format with a fixed length. Its range varies from January 1, 4712 BC to December 31, 9999 AD.
TIMESTAMP	It is used to store the valid date in YYYY-MM-DD with time hh:mm:ss format.

Oracle Large Object Data Types (LOB Types)

BLOB	It is used to specify unstructured binary data. Its range goes up to 2^{32}-1 bytes or 4 GB.
BFILE	It is used to store binary data in an external file. Its range goes up to 2^{32}-1 bytes or 4 GB.
CLOB	It is used for single-byte character data. Its range goes up to 2^{32}-1 bytes or 4 GB.
NCLOB	It is used to specify single byte or fixed length multibyte national character set (NCHAR) data. Its range is up to 2^{32}-1 bytes or 4 GB.
RAW(size)	It is used to specify variable length raw binary data. Its range is up to 2000 bytes per row. Its maximum size must be specified.

Introduction to SQL*Plus

SQL*Plus is the command-line interface to the Oracle database. Its fundamental reason for existence is to allow you to enter and execute ad hoc SQL statements and PL/SQL code blocks. SQL*Plus is essentially an interactive query tool, with some scripting capabilities. It is a non-GUI, character-based tool that has been around since the dawn of the Oracle age. Using SQL*Plus, you can enter an SQL statement, such as a SELECT query, and view the results. You can also execute Data Definition Language (DDL) commands that allow you to maintain and modify your database. You can even enter and execute PL/SQL code. In spite of SQL*Plus's age and lack of "flash," it is a workhorse tool used day in and day out by database administrators, developers, and yes, even end users. As a database administrator, it is my tool of choice for managing the databases under my care. I use it to peek under the hood — to explore the physical implementation of my database, and to create and manage users,

tables, and tablespaces. In my role as a developer, SQL*Plus is the first tool that I fire up when I need to develop a query. In spite of all the fancy, GUI-based SQL generators contained in products such as PowerBuilder, Clear Access, and Crystal Reports, I still find it quicker and easier to build up and test a complex query in SQL*Plus before transferring it to whatever development tool I am using.

SQL*Plus formatting commands

SQL*Plus contains several commands that can alter the appearance of the output. These commands are only in effect for the current SQL*Plus session. They can also be included in SQL script files and can be executed using the START command. The formatting commands include:

- **BREAK** – Set the formatting behavior for records that have the same values for some columns
- **BTITLE** – Place a title on the bottom of each page in the printout from a SQL statement
- **COLUMN** – Change the appearance of an output column from a query
- **REMARK** – Place a comment following the REMARK keyword
- **SET** – Set a SQL*Plus variable to a new value
- **SHOW** – Show the current value of a SQL*Plus variable
- **TTITLE** – Place a title on the top of each page in the printout from a SQL statement
- **UNDEFINE** – Delete a user defined variable

Note that none of these SQL*Plus formatting commands changes the underlying table structures.

SQL Operator

SQL statements generally contain some reserved words or characters that are used to perform operations such as comparison and arithmetical operations etc. These reserved words or characters are known as operators. Generally there are three types of operators in SQL:

SQL Arithmetic Operators:

Operators	Descriptions	Examples
+	It is used to add containing values of both operands	a+b will give 150
-	It subtracts right hand operand from left hand operand	a-b will give -50
*	It multiply both operand's values	a*b will give 5000
/	It divides left hand operand by right hand operand	b/a will give 2
%	It divides left hand operand by right hand operand and returns reminder	b%a will give 0

SQL Comparison Operators:

Operator	Description	Example
=	Examine both operands value that are equal or not, if yes condition become true.	(a=b) is not true
!=	This is used to check the value of both operands equal or not, if not condition become true.	(a!=b) is true

< >	Examines the operand's value equal or not, if values are not equal condition is true	(a<>b) is true
>	Examine the left operand value is greater than right Operand, if yes condition becomes true	(a>b) is not true
<	Examines the left operand value is less than right Operand, if yes condition becomes true	(a<="" td="">
>=	Examines that the value of left operand is greater than or equal to the value of right operand or not, if yes condition become true	(a>=b) is not true
<=	Examines that the value of left operand is less than or equal to the value of right operand or not, if yes condition becomes true	(a<=b) is true
!<	Examines that the left operand value is not less than the right operand value	(a!<="" td="">
!>	Examines that the value of left operand is not greater than the value of right operand	(a!>b) is true

SQL Logical Operators:

Operator	Description
ALL	this is used to compare a value to all values in another value set.
AND	this operator allows the existence of multiple conditions in an SQL statement.
ANY	this operator is used to compare the value in list according to the condition.
BETWEEN	this operator is used to search for values, that are within a set of values
IN	this operator is used to compare a value to that specified list value
NOT	the NOT operator reverse the meaning of any logical operator
OR	this operator is used to combine multiple conditions in SQL statements
EXISTS	the EXISTS operator is used to search for the presence of a row in a specified table
LIKE	this operator is used to compare a value to similar values using wildcard operator

SQL Expression

An expression is a combination of one or more values, operators and SQL functions that evaluate to a value. These SQL EXPRESSIONs are like formulae and they are written in query language. You can also use them to query the database for a specific set of data. Consider the basic syntax of the SELECT statement as follows:

```
SELECT column1, column2, columnN
FROM table_name
WHERE [CONDITION|EXPRESSION];
```

There are different types of SQL expressions, Boolean, Numeric, Date.

SQL v/s SQL*Plus

SQL: As we all know, SQL is a language. SQL is based on the ANSI SQL standard. SQL is the query language used for communication with Oracle server to access and modify the data. SQL is a language which is invented by IBM. SQL can be simply used to ask queries, i.e. it involves DML, DDL and DCL. In SQL, there is no continuation character. Keywords cannot be abbreviated in SQL. SQL uses functions to manipulate the data. Process the data and defines the objects from the database.

SQL *Plus: While SQL*Plus is a tool. SQL *Plus is an interface specifies the Oracle system for executing SQL statements. SQL* Plus is a command line tool with which you can send SQL queries to the server. Also, it can help you format the query result. SQL * Plus is a tool to use SQL language for a database from Oracle corporation. SQL * Plus is command line tool which doesn't involve DML, DDL and DCL. Whereas, in SQL * Plus there is a continuation character. But keywords can be abbreviated in SQL*Plus. SQL * plus uses commands to manipulate the data. Not permit the processing of information in the database

Unit – 2

Managing Tables and Data, Data Control and Transaction Control Command

Creating, Altering & Dropping tables

Oracle object types are user-defined types that make it possible to model real-world entities, such as customers and purchase orders, as objects in the database. New object types can be created from any built-in database types and any previously created object types, object references, and collection types. Object types can work with complex data, such as images, audio, and video. Oracle Database stores metadata for user-defined types in a schema that is available to SQL, PL/SQL, Java, and other languages. The following are the commands to manage Table object in Oracle databases.

Create Table: To create a new table in Oracle Database, we use the CREATE TABLE statement. The following illustrates the basic syntax of the CREATE TABLE statement:

```
CREATE TABLE table_name (
    column_1 data_type column_constraint,
    column_2 data_type column_constraint,
    ...
    table_constraint
 );
```

First, specify the table name and schema name to which the new table belongs on the CREATE TABLE clause. Second, list all columns of the table within the parentheses. In case a table has multiple columns, you need to separate them by commas (,). A column definition includes the column name followed by its data type e.g., NUMBER, VARCHAR2, and a column constraint such as NOT NULL, primary key, check. Third, add table constraints if applicable e.g., primary key, foreign key, check. For example:

```
CREATE TABLE persons (
    person_id NUMBER,
    first_name VARCHAR2(50) NOT NULL,
    last_name VARCHAR2(50) NOT NULL,
    PRIMARY KEY(person_id) );
```

Alter Table: To modify the structure of an existing table, you use the ALTER TABLE statement. The following illustrates the syntax:

```
ALTER TABLE table_name action;
```

In this statement, First, specify the table name which you want to modify. Second, indicate the action that you want to perform after the table name. The ALTER TABLE statement allows you to:

- Add one or more columns
- Modify column definition
- Drop one or more columns
- Rename columns
- Rename table

To **add a new column** to a table, you use the following syntax:

```
ALTER TABLE table_name
    ADD column_name type constraint;
```

For example, the following statement adds a new column named birthdate to the persons table:

```
ALTER TABLE persons
   ADD birthdate DATE NOT NULL;
```

To **modify the attributes** of a column, you use the following syntax:

```
ALTER TABLE table_name
  MODIFY column_name type constraint;
```

For example, the following statement changes the birthdate column to a null-able column:

```
ALTER TABLE persons MODIFY birthdate DATE NULL;
```

Drop Table: To move a table to the recycle bin or remove it entirely from the database, you use the DROP TABLE statement:

```
DROP TABLE table_name
[CASCADE CONSTRAINTS | PURGE];
```

In this statement, First, indicate the table and its schema that you want to drop after the DROP TABLE clause. If you don't specify the schema name explicitly, the statement assumes that you are removing the table from your own schema. Second, specify CASCADE CONSTRAINTS clause to remove all referential integrity constraints which refer to primary and unique keys in the table. In case such referential integrity constraints exist and you don't use this clause, Oracle returns an error and stops removing the table. Third, specify PURGE clause if you want to drop the table and release the space associated with it at once. By using the PURGE clause, Oracle will not place the table and its dependent objects into the recycle bin. The following example drops the persons table from the database:

```
DROP TABLE persons;
```

Data Manipulation Language Command

DML modifies the database instance by inserting, updating and deleting its data. DML is responsible for all forms data modification in a database. SQL contains the following set of commands in its DML section:

Insert: To insert a new row into a table, you use the Oracle INSERT statement as follows:

```
INSERT INTO table_name (column_list)
VALUES(value_list);
```

In this statement: First, specify the name of the table into which you want to insert. Second, specify a list of comma-separated column names within parentheses. Third, specify a list of comma-separated values that corresponds to the column list. If the value list has the same order as the table columns, you can skip the column list although this is not considered as a good practice. The following statement inserts a new row into the DEPT table:

```
insert into DEPT (DEPTNO, DNAME, LOC)
values (10, 'ACCOUNTING', 'NEW YORK')
```

Update: To changes existing values in a table, you use the following Oracle UPDATE statement:

```
UPDATE table_name
SET column1 = value1, column2 = value2, column3 = value3
WHERE condition;
```

Let's examine the UPDATE statement in detail. First, you specify the name of the table which you want to update. Second, you specify the name of the column whose values are to be updated and the new value. If you update more than two columns, you separate each expression column = value by a comma. The value1, value2, or value3 can be literals or a subquery that returns a single value. Note that the UPDATE statement allows you to update as many columns as you want. Third, the WHERE clause determines which rows of the table should be updated. The WHERE clause is optional. If you omit it, the UPDATE statement will update all rows of the table. The following UPDATE statement changes the cost of the part with id 1:

```
UPDATE parts SET cost = 130
WHERE part_id = 1;
```

Delete: To delete one or more rows from a table, you use the Oracle DELETE statement as follows:

```
DELETE FROM table_name WHERE condition;
```

In this statement, First, you specify the name of the table from which you want to delete data. Second, you specify which row should be deleted by using the condition in the WHERE clause. If you omit the WHERE clause, the Oracle DELETE statement removes all rows from the table. Note that it is faster and more efficient to use the TRUNCATE TABLE statement to delete all rows from a large table. The following statement deletes a row whose order id is 1 and item id is 1:

```
DELETE FROM sales WHERE order_id = 1;
```

Truncate: Oracle introduced the TRUNCATE TABLE statement that allows you to delete all rows from a big table. The following illustrates the syntax of the Oracle TRUNCATE TABLE statement:

```
TRUNCATE TABLE table_name;
```

In this case, because we don't specify the schema name explicitly, Oracle assumes that we truncate the table from our own schema. To delete all rows from the customers_copy table you use the following TRUNCATE TABLE statement:

```
TRUNCATE TABLE customers_copy;
```

Rename Table: To rename a table, you use the following Oracle RENAME table statement as follows:

```
RENAME table_name TO new_name;
```

In the RENAME table statement: First, specify the name of the existing table which you want to rename. Second, specify the new table name. The new name must not be the same as another table in the same schema. To rename the promotions table to campaigns table, you use the following statement:

```
RENAME promotions TO campaigns;
```

Database constraints

Use a constraint to define an integrity constraint—a rule that restricts the values in a database. Oracle Database lets you create five types of constraints and lets you declare them in two ways. The five types of integrity constraint are described briefly here and more fully in "Semantics":

- A **NULL/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. 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.

You can define constraints syntactically in two ways: As part of the definition of an individual column or attribute. This is called inline specification or **Column Level Constraint**. Second, as part of the table definition. This is called out-of-line specification or **Table Level Constraint**. For example:

```
create table dept(
  deptno     number(2,0),
  dname      varchar2(14),
  loc        varchar2(13),
  constraint pk_dept primary key (deptno)  );

create table emp(
  empno    number(4,0),
  ename    varchar2(10),
```

```
job      varchar2(9),
mgr      number(4,0),
hiredate date,
sal      number(7,2),
comm     number(7,2),
deptno   number(2,0),
constraint pk_emp primary key (empno),
constraint fk_deptno foreign key (deptno) references dept (deptno) );
```

SELECT statement: To retrieve data from one or more columns of a table, you use the SELECT statement with the following syntax:

```
SELECT * | [ DISTINCT ] [ Column List] FROM table_name
 [ WHERE <Condition>]
 [ GROUP BY <Column List>]
 [ HAVING <Condition>]
 [ ORDER BY columns [ASC | DESC] ];
```

In this SELECT statement: First, specify the table name from which you want to query the data. Second, indicate the columns from which you want to return the data. If you have more than one column, you need to separate each by a comma (,).

Join: Oracle join is used to combine columns from two or more tables based on values of the related columns. The related columns are typically the primary key column(s) of the first table and foreign key column(s) of the second table. Oracle supports **inner join, left join, right join, full outer join** and join that you can join a table to itself to query hierarchical data using an inner join, left join, or right join. This kind of join is known as **self-join**.

Subquery: A subquery is a SELECT statement nested inside another statement such as SELECT, INSERT, UPDATE, or DELETE. Typically, you can use a subquery anywhere that you use an expression. Consider this following subquery example that uses the products table:

```
SELECT product_id, product_name, list_price
FROM products
WHERE list_price = (SELECT MAX( list_price ) FROM products );
```

Operator to get combine result of multiple queries:

- **Minus:** The Oracle MINUS operator compares two queries and returns distinct rows from the first query that are not output by the second query. In other words, the MINUS operator subtracts one result set from another.
- **Intersect:** The Oracle INTERSECT operator compares the result of two queries and returns the distinct rows that are output by both queries.
- **Union:** The UNION operator is a set operator that combines result sets of two or more SELECT statements into a single result set.

Built in functions

Numeric Function:

Function	Description and Example
addition	The addition operator (+). SELECT NortheastSales + SoutheastSales AS EastTotalSales
subtraction	The subtraction operator (-). SELECT SalesRevenue - TotalCosts AS Profit
multiplication	The multiplication operator (*). SELECT Price * 0.7 AS SalePrice
division	The division operator (/). SELECT YearTotal / 4 AS QuarterAvg
ABS	Returns the absolute value of n. If n is 0 or a positive integer, returns n. Otherwise, n is multiplied by -1. SELECT ABS(-1) AS one **RESULT:** one = 1
CEIL	Returns the smallest integer value not less than n. SELECT CEIL(123.45) AS x, CEIL(32) AS y, CEIL(-123.45) AS z **RESULT:** x = 124, y = 32, z = -123
EXP	Exponentiation, where the base is e. Returns the value of e (the base of natural logarithms) raised to the power n. SELECT EXP(1.0) AS baseE **RESULT:** baseE = e^1.0 = 2.71828182845905
FLOOR	Returns the largest integer value not greater than n. SELECT FLOOR(123.45) AS x, FLOOR(32) AS y, FLOOR(-123.45) AS z **RESULT:** x = 123, y = 32, z = -124
LN	Natural logarithm. Computes the logarithm of its single argument, the base of which is e. SELECT LN(1.0) AS baseE **RESULT:** baseE = e^1.0 = 0
LOG	Logarithm. log(n, m) takes two arguments, where n is the base, and m is the value you are taking the logarithm of. Log(10,1000) = 3
MOD	Modulo. Returns the remainder of n divided by m.

Function	Description and Example
	Mod(10,3) = 1 EQL uses the fmod floating point remainder, as defined in the C/POSIX standard.
ROUND	Returns a number rounded to the specified decimal place. The unary (one argument) version takes only one argument (the number to be rounded) and drops the decimal (non-integral) portion of the input. For example: ROUND(8.2) returns 8 ROUND(8.7) returns 9 The binary (two argument) version takes two arguments (the number to be rounded and a positive or negative integer that allows you to set the number of spaces at which the number is rounded). The binary version always returns a double: • Positive second arguments correspond to the number of places that must be returned ***after*** the decimal point. For example: ROUND(123.4567, 3) returns 123.457 • Negative second arguments correspond to the number of places that must be returned ***before*** the decimal point. For example: • ROUND(123.4, -3) returns 0 ROUND(1234.56, -3) returns 1000
SIGN	Returns the sign of the argument as -1, 0, or 1, depending on whether n is negative, zero, or positive. The result is always a double. SELECT SIGN(-12) AS x, SIGN(0) AS y, SIGN(12) AS z **RESULT:** x = -1, y = 0, z = 1
SQRT	Returns the nonnegative square root of n. SELECT SQRT(9) AS x **RESULT:** x = 3
TRUNC	Returns the number n truncated to m decimal places. If m is 0, the result has no decimal point or fractional part. The unary (one argument) version drops the decimal (non-integral) portion of the input. For example: SELECT TRUNC(3.14159265) AS x **RESULT:** x = 3 The binary (two argument) version allows you to set the number of spaces at which the number is truncated. The binary version always returns a double. For example: SELECT TRUNC(3.14159265, 3) AS y **RESULT:** y = 3.141
SIN	The sine of n, where the angle of n is in radians.

Function	Description and Example
	SIN(3.14159/6) = 0.499999616987256
COS	The cosine of n, where the angle of n is in radians. COS(3.14159/3) = 0.500000766025195
TAN	The tangent of n, where the angle of n is in radians. TAN(3.14159/4) = 0.999998673205984
POWER	Returns the value (as a double) of n raised to the power of m. Power(2,8) = 256
TO_DURATION	Casts a string representation of a timestamp into a number of milliseconds so that it can be used as a duration.
TO_DOUBLE	Casts a string representation of an integer as a double.
TO_INTEGER(boolean)	Casts TRUE/FALSE to 1/0.

Character Function:

Function	Example	Result	Purpose
ASCII	ASCII('A')	65	Returns an ASCII code value of a character.
CHR	CHR('65')	'A'	Converts a numeric value to its corresponding ASCII character.
CONCAT	CONCAT('A','BC')	'ABC'	Concatenate two strings and return the combined string.
INITCAP	INITCAP('hi there')	'Hi There'	Converts the first character in each word in a specified string to uppercase and the rest to lowercase.
LENGTH	LENGTH('ABC')	3	Return the number of characters (or length) of a specified string
LOWER	LOWER('Abc')	'abc'	Return a string with all characters converted to lowercase.
LPAD	LPAD('ABC',5,'*')	'**ABC'	Return a string that is left-padded with the specified characters to a certain length.
LTRIM	LTRIM(' ABC ')	'ABC '	Remove spaces or other specified characters in a set from the left end of a string.
REPLACE	REPLACE('JACK AND JOND','J','BL');	'BLACK AND BLOND'	Replace all occurrences of a substring by another substring in a string.
RPAD	RPAD('ABC',5,'*')	'ABC**'	Return a string that is right-padded with the specified characters to a certain length.
RTRIM	RTRIM(' ABC ')	' ABC'	Remove all spaces or specified character in a set from the right end of a string.
SOUNDEX	SOUNDEX('sea')	'S000'	Return a phonetic representation of a specified string.

Function	Example	Result	Purpose
SUBSTR	SUBSTR('Oracle Substring', 1, 6)	'Oracle'	Extract a substring from a string.
TRIM	TRIM(' ABC ')	'ABC'	Remove the space character or other specified characters either from the start or end of a string.
UPPER	UPPER('Abc')	'ABC'	Convert all characters in a specified string to uppercase.

Date Function:

Function	Example	Result	Description
ADD_MONTHS	ADD_MONTHS(DATE '2016-02-29', 1)	31-MAR-16	Add a number of months (n) to a date and return the same day which is n of months away.
LAST_DAY	LAST_DAY(DATE '2016-02-01')	29-FEB-16	Gets the last day of the month of a specified date.
MONTHS_BETWEEN	MONTHS_BETWEEN(DATE '2017-07-01', DATE '2017-01-01')	6	Return the number of months between two dates.
NEXT_DAY	NEXT_DAY(DATE '2000-01-01', 'SUNDAY')	02-JAN-00	Get the first weekday that is later than a specified date.
ROUND	ROUND(DATE '2017-07-16', 'MM')	01-AUG-17	Return a date rounded to a specific unit of measure.
SYSDATE	SYSDATE	01-AUG-17	Return the current system date and time of the operating system where the Oracle Database resides.
SYSTIMESTAMP	SELECT SYSTIMESTAMP FROM dual;	01-AUG-17 01.33.57.929000000 PM -07:00	Return the system date and time that includes fractional seconds and time zone.
TO_CHAR	TO_CHAR(DATE'2017-01-01', 'DL')	Sunday, January 01, 2017	Convert a DATE or an INTERVAL value to a character string in a specified format.
TO_DATE	TO_DATE('01 Jan 2017', 'DD MON YYYY')	01-JAN-17	Convert a date which is in the character string to a DATE value.
TRUNC	TRUNC(DATE '2017-07-16', 'MM')	01-JUL-17	Return a date truncated to a specific unit of measure.

Aggregate function:

Function	Description
AVG	Return the average of values of a set
COUNT	Return the number of values in a set or number of rows in a table

Function	Description
MAX	Return the maximum value in a set of values
MIN	Return the minimum value in a set of values
SUM	Returns the sum of values in a set of values

General Functions: CASE WHEN

COALESCE – show you how to substitute null with a more meaningful alternative.

DECODE – learn how to add if-then-else logic to a SQL query.

Creating User

The CREATE USER statement allows you to create a new database user which you can use to log in to the Oracle database. The basic syntax of the CREATE USER statement is as follows:

```
CREATE USER username
    IDENTIFIED BY password
    [DEFAULT TABLESPACE tablespace]
    [QUOTA {size | UNLIMITED} ON tablespace]
    [PROFILE profile]
    [PASSWORD EXPIRE]
    [ACCOUNT {LOCK | UNLOCK}];
```

CREATE USER username: Specify the name of the user to be created. IDENTIFIED BY password: Specify a password for the local user to use to log on to the database. Note that you can create an external or global user, which is not covered in this tutorial. DEFAULT TABLESPACE: Specify the tablespace of the objects such as tables and views that the user will create. If you skip this clause, the user's objects will be stored in the database default tablespace if available, typically it is USERS tablespace; or the SYSTEM tablespace in case there is no database default tablespace.

Creating Role

A role is a group of privileges. Instead of granting individual privileges to users, you can group related privileges into a role and grant this role to users. Roles help manage privileges more efficiently. To create a new role, you use the CREATE ROLE statement. The basic syntax of the CREATE ROLE statement is as follows:

```
CREATE ROLE role_name
[IDENTIFIED BY password]
[NOT IDENTIFIED]
```

In this syntax: First, specify the name of the role that you want to create. Second, use IDENTIFIED BY password option to create a local role and indicate that the user, who was granted the role, must provide the password to the database when enabling the role. Third, use NOT IDENTIFIED to indicate that the role is authorized by the database and the user, who was granted this role, don't need a password to enable the role. After a role is created, it is empty. To grant privileges to a role, you use the GRANT statement.

Data Control Language (DCL) Statement

DCL is used to control privileges in Database. To perform any operation in the database, such as for creating tables, sequences or views, a user needs privileges.

Grant: After creating a user, you need to decide which actions the user can do in the Oracle database. By definition, a **privilege** is a right to execute an SQL statement or a right to access an object of another user. Oracle defines two main types of privileges: system privileges and object privileges.

- **System privileges:** System privileges determine what a user can do in the database. They mainly allow a user to add or modify schema objects in the database like creating tables, creating views, and removing tablespaces. The most important system privileges are: CREATE SESSION, CREATE TABLE, CREATE VIEW, CREATE PROCEDURE, SYSDBA, SYSOPER.
- **Object privileges:** Object privileges decide how a user can access the data in the database. The object privileges apply to rows in tables or views. Here are some common object privileges: INSERT, UPDATE, DELETE, INDEX, EXECUTE. To grant one or more privileges to a user, you use the GRANT statement

The GRANT statement assigns one or more privileges to a specific user. The following illustrates the basic syntax of the GRANT statement:

```
GRANT {system_privileges | object_privileges }
TO user
[WITH ADMIN OPTION]
```

In this syntax: First, specify the system or object privileges that you want to assign to a user after the GRANT keyword. If you assign more than one privilege, you use a comma-separated list of privileges. Second, specify the user that receives the privileges after the TO keyword.

Revoke: The Oracle REVOKE statement revokes system and object privileges from a user. Here is the basic syntax of the Oracle REVOKE statement:

```
REVOKE {system_privilege | object_privilege } FROM user;
```

In this syntax: First, specify the system or object privileges that you want to revoke from the user. Second, specify the user from which you want to revoke the privileges.

What is transaction?

A transaction is a logical unit of work that contains one or more SQL statements. A transaction begins with the first executable SQL statement. A transaction ends when it is committed or rolled back, either explicitly with a COMMIT or ROLLBACK statement or implicitly when a DDL statement is issued. Transaction Control Language(TCL) commands are used to manage transactions in the database.

Commit: COMMIT command is used to permanently save any transaction into the database. When we use any DML command like INSERT, UPDATE or DELETE, the changes made by these commands are not permanent, until the current session is closed, the changes made by these commands can be rolled back. To avoid that, we use the COMMIT command to mark the changes as permanent. Following is commit command's syntax:

```
COMMIT;
```

Rollback: This command restores the database to last commited state. It is also used with SAVEPOINT command to jump to a savepoint in an ongoing transaction. If we have used the UPDATE command to make some changes into the database, and realise that those changes were not required, then we can use the ROLLBACK command to rollback those changes, if they were not commited using the COMMIT command. Following is rollback command's syntax:

```
ROLLBACK;
ROLLBACK TO savepoint_name;
```

Savepoint: SAVEPOINT command is used to temporarily save a transaction so that you can rollback to that point whenever required. Following is savepoint command's syntax:

```
SAVEPOINT savepoint_name;
```

In short, using this command we can name the different states of our data in any table and then rollback to that state using the ROLLBACK command whenever required.

Unit – 3

Other ORACLE Database Objects, Concurrency control using lock

View

A view is a virtual table because you can use it like a table in your SQL queries. Every view has columns with data types so you can execute a query against views or manage their contents (with some restrictions) using the INSERT, UPDATE, DELETE, and MERGE statements. You can expose the data from underlying tables to the external applications via views. Whenever the structures of the base tables change, you just need to update the view. The interface between the database and the external applications remains intact. The beauty is that you don't have to change a single line of code to keep the external applications up and running.

Unlike a table, a view does not store any data. To be precise, a view only behaves like a table. And it is just a named query stored in the database. When you query data from a view, Oracle uses this stored query to retrieve the data from the underlying tables. To create a new view in a database, you use the following Oracle CREATE VIEW statement:

```
CREATE [OR REPLACE] VIEW view_name  AS
    SQL-query;
```

The OR REPLACE option replaces the definition of existing view. It is handy if you have granted various privileges on the view. The defining query is a SELECT statement that defines the columns and rows of the view. The following statement creates a view named employee_yos based on the employees table. The view shows the employee id, name and years of service:

```
CREATE VIEW employee_yos AS
SELECT
    employee_id, first_name || ' ' || last_name full_name,
    FLOOR( months_between( CURRENT_DATE, hire_date )/ 12 ) yos
FROM employees;
```

To remove a view from the database, you use the following DROP VIEW statement:

```
DROP VIEW view_name;
```

For Example:

```
DROP VIEW employee_yos;
```

Sequence

A sequence is a list of integers in which their orders are important. For example, the (1,2,3,4,5) and (5,4,3,2,1) are totally different sequences even though they have the same members. The CREATE SEQUENCE statement allows you to create a new sequence object in Oracle Database. Here is the basic syntax of the CREATE SEQUENCE statement:

```
CREATE SEQUENCE schema_name.sequence_name
[INCREMENT BY interval]
[START WITH first_number]
[MAXVALUE max_value | NOMAXVALUE]
[MINVALUE min_value | NOMINVALUE]
[CYCLE | NOCYCLE]
[CACHE cache_size | NOCACHE]
[ORDER | NOORDER];
```

Specify the name of the sequence after the CREATE SEQUENCE keywords. If you want to create a sequence in a specific schema, you can specify the schema name in along with the sequence name. INCREMENT BY: Specify the interval between sequence numbers after the INCREMENT BY keyword. START WITH: Specify the first number in the sequence. MAXVALUE: Specify the maximum value of

the sequence. MINVALUE: Specify the minimum value of the sequence. CYCLE: Use CYCLE to allow the sequence to generate value after it reaches the limit, min value for a descending sequence and max value for an ascending sequence. CACHE: Specify the number of sequence values that Oracle will pre-allocate and keep in the memory for faster access.

The following statement creates an ascending sequence called id_seq, starting from 10, incrementing by 10, minimum value 10, maximum value 100. The sequence returns 10 once it reaches 100 because of the CYCLE option.

```
CREATE SEQUENCE id_seq
    INCREMENT BY 10
    START WITH 10
    MINVALUE 10
    MAXVALUE 100
    CYCLE
    CACHE 2;
```

To get the next value of the sequence, you use the NEXTVAL pseudo-column:

```
SELECT id_seq.NEXTVAL FROM dual;
```

To get the current value of the sequence, you use the CURRVAL pseudo-column:

```
SELECT id_seq.CURRVAL FROM dual;
```

Synonyms

Synonyms provide a level of security by hiding the name and owner of a schema object such as a table or a view. On top of that, they provide location transparency for remote objects of a distributed database.

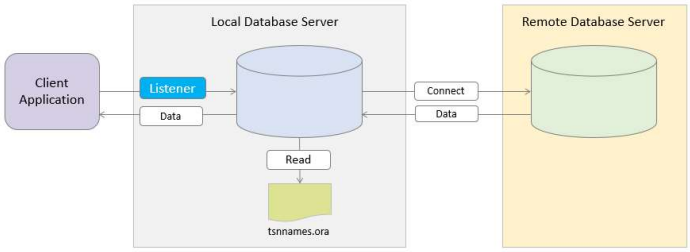
Synonyms create a level of abstraction of the underlying schema objects so that you can rename and move of the underlying objects without affecting the applications based on the synonyms. Note that synonyms themselves are not secured. When you grant object privileges on a synonym, you are granting privileges on the underlying object, and the synonym only acts as an alias in the GRANT statement. The CREATE SYNONYM statement allows you to create a synonym which is an alternative name for a database object such as a table, view, sequence, procedure, stored function, and materialized view. Here is the basic syntax of creating a new synonym:

```
CREATE [OR REPLACE] [PUBLIC] SYNONYM schema.synonym_name
FOR schema.object;
```

In this syntax: First, specify the name of the synonym and its schema. If you skip the schema, Oracle will create the synonym in your own schema. Second, specify the object for which you want to create the synonym after the FOR keyword. Note that the schema object (schema.object) cannot be contained in a package. Third, use the OR REPLACE option if you want to re-create the synonym if it already exists. In case the synonym does not exist, the OR REPLACE has no effect. Fourth, use the PUBLIC keyword to create a public synonym which is a synonym that will be accessible to all users. Note that users must have sufficient privileges on the underlying objects to use the public synonyms. This example uses the CREATE SYNONYM statement to create a synonym for the inventories table from the sample database:

```
CREATE SYNONYM stocks FOR inventories;
```

Database Links



A database link is a connection from the Oracle database to another remote database. The remote database can be an Oracle Database or any ODBC compliant database such as SQL Server or MySQL. A database link allows a user or program to access database objects such as tables and views from another database. Once you create a database link, you can access the tables or views from the remote database. To create a private database link, you use the CREATE DATABASE LINK statement as follows:

```
CREATE DATABASE LINK dblink
    CONNECT TO remote_user IDENTIFIED BY password
    USING '(DESCRIPTION =(ADDRESS = (PROTOCOL = TCP)(HOST =
oracledb.example.com)(PORT=1521))
        (CONNECT_DATA=(SERVICE_NAME=service_name)) )';
```

In this syntax: First, specify the name of the database link after the CREATE DATABASE LINK keywords. Second, provide user and password of the remote database after the CONNECT TO and IDENTIFIED BY keywords. Finally, specify the service name of the remote database. If you specify only the database name, Oracle will append the database domain to the connect string to form a complete service name. In this example, we will create a database link to a remote Oracle Database server located in the server 10.50.100.143 with the port 1521 and service name SALES. First, add the following entry to tnsnames.ora file in the local Oracle Database server. Typically, the tnsnames.ora is located in the directory /NETWORK/ADMIN/ under ORACLE_HOME:

```
CREATE DATABASE LINK sales
    CONNECT TO bob IDENTIFIED BY Abcd1234
    USING ' (DESCRIPTION =
    (ADDRESS =(PROTOCOL = TCP)(HOST = 10.50.100.143)(PORT =
1521))(CONNECT_DATA = (SERVER = DEDICATED) (SERVICE_NAME =
SALES_PRD) ) ) ';
```

A database link allows a user or program to access database objects such as tables and views from another database. Once you create a database link, you can access the tables or views from the remote database using the following pattern:

```
table_name@database_link
```

For example, you can query data from a table in the remote database as if it was in the local server:

```
SELECT * FROM remote_table@database_link;
```

Index

Oracle index is one of the effective tools for boost the query performance. However, in order to use it effectively, you must understand it correctly. This section helps you understand and use Oracle indexes to speed up your queries.

Indexes are used to search the rows in the oracle table quickly. If the index is not present the select query has to read the whole table and returns the rows. With Index, the rows can be retrieved quickly We should create Indexes when retrieving a small number of rows from a table. or to retrieve the first set of rows as fast as possible from some query that will ultimately return a large number of rows. It also depends on the data distribution i.e clustering factor Indexes are logically and physically

independent of the data in the associate table. Indexes are optional structures associated with tables and clusters. You can create indexes on one or more columns of a table to speed SQL statement execution on that table. Indexes are the primary means of reducing disk I/O when properly used. The query decides at the beginning whether to use index or no The best thing with indexes is that retrieval performance of indexed data remains almost constant, even as new rows are inserted. However, the presence of many indexes on a table decreases the performance of updates, deletes, and inserts because Oracle must also update the indexes associated with the table. The following are the types of index in Oracle:

B*Tree Indexes: Regular indexes are B-trees. B stands for balanced. This means all the leaf nodes are at the same depth. So all entries take the same amount of work to access. B-Tree Indexes (balanced tree) are the most common type of index. B-Tree index stored the ROWID and the index key value in a tree structure. When creating an index, a ROOT block is created, then BRANCH blocks are created and finally LEAF blocks. Each branch holds the range of data its leaf blocks hold, and each root holds the range of data its branches hold. B-Tree indexes are most useful on columns that appear in the where clause (SELECT … WHERE EMPNO=1). The Oracle server, keeps the tree balanced by splitting index blocks, when new data is inserted to the table.

```
CREATE <UNIQUE|NON UNIQUE>  INDEX <index_name>
ON <table_name> (<column_name>,<column_name>…)
TABLESPACE <tablespace_name>;
```

Example:

```
Create index exp_idx on table emp(name);
```

Bitmap Indexes: use bitmap index on a column or columns that have few distinct values, or low cardinality. Bitmap Indexes are most appropriate on low distinct cardinality data (as opposed to B-Tree indexes). This type of index, creates a binary map of all index values, and store that map in the index blocks, this means that the index will require less space than B-Tree index. Each bit in the bitmap corresponds to a possible rowid. If the bit is set, then it means that the row with the corresponding rowid contains the key value. A mapping function converts the bit position to an actual rowid, so the bitmap index provides the same functionality as a regular index even though it uses a different representation internally. If the number of different key values is small, then bitmap indexes are very space efficient.

```
CREATE BITMAP  INDEX <index_name>
ON <table_name> (<column_name>, <column_name>…)
PCTFREE <integer>;
```

Example:

```
CREATE BITMAP INDEX ON emp_data(gender);
```

Function-Based Indexes: speed up queries that involve expression which consists of functions. Function-Based Indexes are indexes created on columns that a function is usually applied on. When using a function on an indexed column, the index is ignored, therefore a function-based index is very useful for these operations.

```
CREATE INDEX <index_name> ON <table_name> [ Function
(<column_name>, <column_name.)];
```

Example:

```
CREATE INDEX EMP_IDX on EMP(UPPER(ENAME));
```

Application Domain Indexes: Oracle provides extensible indexing to accommodate indexes on complex data types such as documents, spatial data, images, and video clips and to make use of specialized indexing techniques. With extensible indexing, you can encapsulate application-specific index management routines as an indextype schema object and define a domain index (an application-specific index) on table columns or attributes of an object type. Extensible indexing also provides efficient processing of application-specific operators.

The application software, called the cartridge, controls the structure and content of a domain index. The Oracle server interacts with the application to build, maintain, and search the domain index. The index structure itself can be stored in the Oracle database as an index-organized table or externally as a file.

Cluster

Using clusters can improve performance and reduce disk space requirements. A cluster provides an optional method of storing table data. A cluster is made up of a group of tables that share the same data blocks. The tables are grouped together because they share common columns and are often used together.

For example, the emp and dept table share the deptno column. When you cluster the emp and dept tables, Oracle Database physically stores all rows for each department from both the emp and dept tables in the same data blocks. Because clusters store related rows of different tables together in the same data blocks, properly used clusters offer two primary benefits: Disk I/O is reduced and access time improves for joins of clustered tables. And second is, the cluster key is the column, or group of columns, that the clustered tables have in common. You specify the columns of the cluster key when creating the cluster. You subsequently specify the same columns when creating every table added to the cluster. Each cluster key value is stored only once each in the cluster and the cluster index, no matter how many rows of different tables contain the value.

You create a cluster using the CREATE CLUSTER statement. The following statement creates a cluster named emp_dept, which stores the emp and dept tables, clustered by the deptno column:

```
CREATE CLUSTER emp_dept (deptno NUMBER(3))
   SIZE 600
   TABLESPACE users
   STORAGE (INITIAL 200K NEXT 300K MINEXTENTS 2
      PCTINCREASE 33);
```

You create a table in a cluster using the CREATE TABLE statement with the CLUSTER clause. To create a table in a cluster, you must have either the CREATE TABLE or CREATE ANY TABLE system privilege. You do not need a tablespace quota or the UNLIMITED TABLESPACE system privilege to create a table in a cluster. For example, the emp and dept tables can be created in the emp_dept cluster using the following statements:

```
CREATE TABLE emp (
   empno NUMBER(5) PRIMARY KEY,
   ename VARCHAR2(15) NOT NULL,     . . .
   deptno NUMBER(3) REFERENCES dept)
   CLUSTER emp_dept (deptno);

CREATE TABLE dept (
   deptno NUMBER(3) PRIMARY KEY, . . . )
   CLUSTER emp_dept (deptno);
```

Snapshot

A snapshot is a staged copy of data in a Data Store that is used in one or more processes. Note that you do not have to copy the data that you are working with, but doing so allows you greater access to Director's results browsing functionality, as you are able to drill down on processor metrics to see the data itself, at each stage in your processing.

Commonly, you might take a copy of the data when working on an audit process, or when defining the rules for data cleansing, but you might run a process in streaming mode (that is, without copying data into the repository) when you run a data cleansing process in production, in order to save time in execution. See the Performance Tuning Guide for more information.

There are two types of snapshot: **Server-side snapshots** may be reloaded either manually or automatically (for example, as part of a scheduled job) whenever the server has access to the data source. This means that when a process is scheduled for execution, it can automatically rerun the snapshot and pick up any changes in the data if required. **Client-side snapshots** are used for sources of data that are accessed via the client rather than the server. The following syntax can be used to create snapshot:

```
CREATE SNAPSHOT snapshot_name
AS snapshot_query;
```

What Are Locks?

Locks are mechanisms that prevent destructive interaction between transactions accessing the same resource—either user objects such as tables and rows or system objects not visible to users, such as shared data structures in memory and data dictionary rows.

In all cases, Oracle automatically obtains necessary locks when executing SQL statements, so users need not be concerned with such details. Oracle automatically uses the lowest applicable level of restrictiveness to provide the highest degree of data concurrency yet also provide fail-safe data integrity. Oracle also allows the user to lock data manually.

Locking Policy

A locking policy is an important component of any multiuser Oracle application. When users share objects in an application, a locking policy ensures that two or more users do not attempt to modify the same object or its underlying data simultaneously. Oracle TopLink works with relational databases to provide support for several types of locking policy, including:

- **Lost Updates:** The application does not verify that data is current.
- **Pessimistic Locking:** When a user accesses an object to update it, the database locks the object until the update is completed. No other user can read or update the object until the first user releases the lock. The database offers this locking type. Pessimistic locking locks objects when the transaction accesses them, before commit time, ensuring that only one client is editing the object at any given time. Pessimistic locking detects locking violations at object read time. The Oracle implementation of pessimistic locking uses database row-level locks, such that attempts to read a locked row either fail or are blocked until the row is unlocked, depending on the database.
- **Optimistic Locking:** All users have read access to the object. When a user attempts to write a change, the application checks to ensure that the object has not changed since the last read. Oracle TopLink provides this locking policy. Optimistic locking, also known as write locking, allows unlimited read access to a given object, but allows a client to modify the object only if the object has not changed since the client last read it. Optimistic locking checks a version of an object at transaction commit time against the version read during the transaction. This

check ensures that no other client modified the data after it was read by the current transaction. If this check detects stale data, the check raises an OptimisticLockException, and the commit fails.

Locking Issues

It is important to understand how locking works in a concurrent application before continuing with a description of the concurrency mechanisms makes Oracle available to you. Blocking and deadlocking have important performance implications for your application. Consequently, this section provides a fundamental description of these concepts, and how they affect Oracle operations.

Blocking: Simply put, a thread of control is blocked when it attempts to obtain a lock, but that attempt is denied because some other thread of control holds a conflicting lock. Once blocked, the thread of control is temporarily unable to make any forward progress until the requested lock is obtained or the operation requesting the lock is abandoned.

Be aware that when we talk about blocking, strictly speaking the thread is not what is attempting to obtain the lock. Rather, some object within the thread (such as a cursor) is attempting to obtain the lock. However, once a locker attempts to obtain a lock, the entire thread of control must pause until the lock request is in some way resolved.

Deadlocks: A deadlock occurs when two or more threads of control are blocked, each waiting on a resource held by the other thread. When this happens, there is no possibility of the threads ever making forward progress unless some outside agent takes action to break the deadlock.

For example, if Txn A is blocked by Txn B at the same time Txn B is blocked by Txn A then the threads of control containing Txn A and Txn B are deadlocked; neither thread can make any forward progress because neither thread will ever release the lock that is blocking the other thread.

Lock Escalation: Lock escalation occurs when numerous locks are held at one level of granularity (for example, rows) and a database raises the locks to a higher level of granularity (for example, table). For example, if a single user locks many rows in a table, some databases automatically escalate the user's row locks to a single table. The number of locks is reduced, but the restrictiveness of what is being locked is increased.

Oracle never escalates locks. Lock escalation greatly increases the likelihood of deadlocks. Imagine the situation where the system is trying to escalate locks on behalf of transaction T1 but cannot because of the locks held by transaction T2. A deadlock is created if transaction T2 also requires lock escalation of the same data before it can proceed.

Lock Types

Oracle automatically uses different types of locks to control concurrent access to data and to prevent destructive interaction between users. Oracle automatically locks a resource on behalf of a transaction to prevent other transactions from doing something also requiring exclusive access to the same resource. The lock is released automatically when some event occurs so that the transaction no longer requires the resource. Throughout its operation, Oracle automatically acquires different types of locks at different levels of restrictiveness depending on the resource being locked and the operation being performed. Oracle locks fall into one of three general categories.

DML Locks (data locks): DML locks protect data. For example, table locks lock entire tables, row locks lock selected rows. The purpose of a DML lock (data lock) is to guarantee the integrity of data being

accessed concurrently by multiple users. DML locks prevent destructive interference of simultaneous conflicting DML or DDL operations. DML statements automatically acquire both table-level locks and row-level locks.

DDL Locks (dictionary locks): DDL locks protect the structure of schema objects—for example, the definitions of tables and views. A data dictionary lock (DDL) protects the definition of a schema object while that object is acted upon or referred to by an ongoing DDL operation. Recall that a DDL statement implicitly commits its transaction. For example, assume that a user creates a procedure. On behalf of the user's single-statement transaction, Oracle automatically acquires DDL locks for all schema objects referenced in the procedure definition. The DDL locks prevent objects referenced in the procedure from being altered or dropped before the procedure compilation is complete. Oracle acquires a dictionary lock automatically on behalf of any DDL transaction requiring it. Users cannot explicitly request DDL locks. Only individual schema objects that are modified or referenced are locked during DDL operations. The whole data dictionary is never locked. DDL locks fall into three categories: exclusive DDL locks, share DDL locks, and breakable parse locks.

Latches and Internal Locks: Latches and internal locks protect internal database and memory structures. Both are inaccessible to users, because users have no need to control over their occurrence or duration. The following section helps to interpret the Enterprise Manager LOCKS and LATCHES monitors.

- **Latches:** Latches are simple, low-level serialization mechanisms to protect shared data structures in the system global area (SGA). For example, latches protect the list of users currently accessing the database and protect the data structures describing the blocks in the buffer cache. A server or background process acquires a latch for a very short time while manipulating or looking at one of these structures. The implementation of latches is operating system dependent, particularly in regard to whether and how long a process will wait for a latch.
- **Internal Locks:** Internal locks are higher-level, more complex mechanisms than latches and serve a variety of purposes.

Manual Locking and User-Defined Locks

Oracle always performs locking automatically to ensure data concurrency, data integrity, and statement-level read consistency. However, you can override the Oracle default locking mechanisms. Overriding the default locking is useful in situations such as these:

- Applications require transaction-level read consistency or repeatable reads. In other words, queries in them must produce consistent data for the duration of the transaction, not reflecting changes by other transactions. You can achieve transaction-level read consistency by using explicit locking, read-only transactions, serializable transactions, or by overriding default locking.
- Applications require that a transaction have exclusive access to a resource so that the transaction does not have to wait for other transactions to complete.

Oracle's automatic locking can be overridden at the transaction level or the session level. At the transaction level, transactions that include the following SQL statements override Oracle's default locking:

- The SET TRANSACTION ISOLATION LEVEL statement
- The LOCK TABLE statement (which locks either a table or, when used with views, the underlying base tables)
- The SELECT ... FOR UPDATE statement

Locks acquired by these statements are released after the transaction commits or rolls back.

Unit – 4

Introduction to PL/SQL,

Advanced PL/SQL

Introduction to PL/SQL

PL/SQL stands for "Procedural Language extensions to the Structured Query Language". SQL is a popular language for both querying and updating data in the relational database management systems (RDBMS). PL/SQL adds many procedural constructs to SQL language to overcome some limitations of SQL. Besides, PL/SQL provides a more comprehensive programming language solution for building mission-critical applications on Oracle Databases. PL/SQL is a highly structured and readable language. Its constructs express the intent of the code clearly. Also, PL/SQL is a straightforward language to learn.

PL/SQL is a standard and portable language for Oracle Database development. If you develop a program that executes on an Oracle Database, you can quickly move it to another compatible Oracle Database without any changes.

SQL v/s PL/SQL

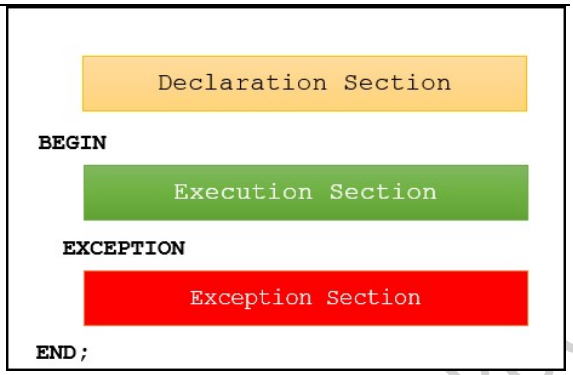
SQL, Structural Query Language is a standard database language which is used create, maintain and retrieve the relational database whereas PL/SQL, Procedural Language extension to SQL, it extends SQL and provide it procedural capabilities. Following are the important differences between SQL and PL/SQL.

Key	SQL	PL/SQL
Definition	SQL, is Structural Query Language for database.	PL/SQL is a programming language using SQL for a database.
Variables	SQL has no variables.	PL/SQL has variables, data types etc.
Control Structures	SQL has no FOR loop, if control and similar structures.	PL/SQL has FOR loop, while loop, if controls and other similar structures.
Operations	SQL can execute a single operation at a time.	PL/SQL can perform multiple operation at a time.
Language Type	SQL is a declarative language.	PL/SQL is a procedural language.
Embedded	SQL can be embedded in a PL/SQL block.	PL/SQL can also be embedded in SQL code.
Interaction	SQL directly interacts with database server.	PL/SQL does not directly interacts with database server.
Orientation	SQL is data oriented language.	PL/SQL is application oriented language.
Objective	SQL is used to write queries, create and execute DDL and DML statements.	PL/SQL is used to write program blocks, functions, procedures, triggers and packages.

PL/SQL Block Structure

PL/SQL is a block-structured language whose code is organized into blocks. A PL/SQL block consists of three sections: declaration, executable, and exception-handling sections. In a block, the executable section is mandatory while the declaration and exception-handling sections are optional. A PL/SQL block has a name. Functions or Procedures is an example of a named block. A named block is stored into the Oracle Database server and can be reused later.

A block without a name is an anonymous block. An anonymous block is not saved in the Oracle Database server, so it is just for one-time use. However, PL/SQL anonymous blocks can be useful for testing purposes. The following picture illustrates the structure of a PL/SQL block:



Declaration section: A PL/SQL block has a declaration section where you declare variables, allocate memory for cursors, and define data types.

Executable section: A PL/SQL block has an executable section. An executable section starts with the keyword BEGIN and ends with the keyword END. The executable section must have a least one executable statement, even if it is the NULL statement which does nothing.

Exception-handling section: A PL/SQL block has an exception-handling section that starts with the keyword EXCEPTION. The exception-handling section is where you catch and handle exceptions raised by the code in the execution section. Note a block itself is an executable statement, therefore you can nest a block within other blocks.

Language construct of PL/SQL

Variables: In PL/SQL, a variable is named storage location that stores a value of a particular data type. The value of the variable changes through the program. Before using a variable, you must declare it in the declaration section of a block.

Each value in PL/SQL such as a constant, variable and parameter has a data type that determines the storage format, valid values, and allowed operations. PL/SQL has two kinds of data types: scalar and composite. The scalar types are types that store single values such as number, Boolean, character, and datetime whereas the composite types are types that store multiple values, for example, record and collection. This tutorial explains the scalar data types that store values with no internal components. The syntax for a variable declaration is as follows:

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

In this syntax: First, specify the name of the variable. The name of the variable should be as descriptive as possible, e.g., l_total_sales, l_credit_limit, and l_sales_revenue. Second, choose an appropriate data type for the variable, depending on the kind of value which you want to store, for example, number, character, Boolean, and datetime.

Basic and Composite Data type: PL/SQL divides the scalar data types into four families: Number, Boolean, Character, and Datetime. A scalar data type may have subtypes. A subtype is a data type that is a subset of another data type, which is its base type. A subtype further defines a base type by restricting the value or size of the base data type.

Comments: When compiling the PL/SQL code, the Oracle precompiler ignores comments. However, you should always use comments to make your code more readable and to help you and other developers understand it better in the future. PL/SQL has two comment styles: single-line and multi-line comments. A single-line comment starts with a double hyphen (--) that can appear anywhere on a line and extends to the end of the line. A multi-line comment starts with a slash-asterisk (/*) and ends with an asterisk-slash (*/), and can span multiple lines.

Constants: Unlike a variable, a constant holds a value that does not change throughout the execution of the program. Constants make your code more readable. To declare a constant, you specify the name, CONSTANT keyword, data type, and the default value. The following illustrates the syntax of declaring a constant:

```
constant_name CONSTANT datatype [NOT NULL]  := expression
```

Conditions

in this section you will learn how to use the PL/SQL IF statement to either execute or skip a sequence of statements based on a specified condition. The IF statement allows you to either execute or skip a sequence of statements, depending on a condition. The IF statement has the three forms:

- IF THEN
- IF THEN ELSE
- IF THEN ELSIF

PL/SQL IF THEN statement: The following illustrates the structure of the IF THEN statement:

```
IF condition THEN
    statements;
END IF;
```

The condition is a Boolean expression that always evaluates to TRUE, FALSE, or NULL. If the condition evaluates to TRUE, the statements after the THEN execute. Otherwise, the IF statement does nothing. The IF THEN ELSE statement has the following structure:

```
IF condition THEN
    statements;
ELSE
    else_statements;
END IF;
```

If the condition evaluates to TRUE, then the statements between THEN and ELSE execute. In case the condition evaluates to FALSE or NULL, the else_statements between ELSE and END IF executes. The following illustrates the structure of the IF THEN ELSIF statement:

```
IF condition_1 THEN
  statements_1
ELSIF condition_2 THEN
  statements_2
[ ELSIF condition_3 THEN
    statements_3
]
...
[ ELSE
    else_statements
]
END IF;
```

In this structure, the condition between IF and THEN, which is the first condition, is always evaluated. Each other condition between ELSEIF and THEN is evaluated only if the preceding condition is FALSE. For example, the condition_2 is evaluated only if the condition_1 is false, the condition_3 is evaluated only if the condition_2 is false, and so on. If a condition is true, other subsequent conditions are not evaluated. If no condition is true, the else_statements between the ELSE and ENDIF execute. In case you skip the the ELSE clause and no condition is TRUE, then the IF THEN ELSIF does nothing.

CASE statement: A simple CASE statement evaluates a single expression and compares the result with some values. The simple **CASE** statement has the following structure:

```
CASE selector
WHEN selector_value_1 THEN
    statements_1
WHEN selector_value_1 THEN
    statement_2
...
ELSE
    else_statements
END CASE;
```

Let's examine the syntax of the simple CASE statement in detail: The selector is an expression which is evaluated once. The result of the selector is used to select one of the several alternatives e.g., selector_value_1 and selector_value_2. WHEN selector_value THEN statements, The selector values i.e., selector_value_1, selector_value_2, etc., are evaluated sequentially. If the result of a selector value equals the result of the selector, then the associated sequence of statements executes and the CASE statement ends. In addition, the subsequent selector values are not evaluated. ELSE else_statements, If no values in WHERE clauses match the result of the selector in the CASE clause, the sequence of statements in the ELSE clause executes.

Looping

PL/SQL FOR LOOP statement: PL/SQL FOR LOOP executes a sequence of statements a specified number of times. The PL/SQL FOR LOOP statement has the following structure:

```
FOR index IN [REVERSE] lower_bound .. upper_bound
LOOP
    statements;
END LOOP;
```

The index is an implicit variable. It is local to the FOR LOOP statement. In other words, you cannot reference it outside the loop. Inside the loop, you can reference index but you cannot change its value. After the FOR LOOP statement executes, the index becomes undefined. Both lower_bound and upper_bound are numbers or expressions that evaluate to numbers. The lower_bound and upper_bound are evaluated once when the FOR LOOP statement starts. Their values are stored as temporary PLS_INTEGER values. The results of lower_bound and upper_bound are rounded to the nearest integer if necessary.

If you modify the values of lower_bound or upper_bound inside the loop, the change will have no effect because they are evaluated once only before the first loop iteration starts. Typically, lower_bound is less than upper_bound. In this case, index is set to lower_bound, the statements execute, and control returns to the top of the loop, where index is compared to upper_bound. If index is less than upper_bound, index is incremented by one, the statements execute, and control again returns to the top of the loop. When index is greater than upper_bound, the loop terminates, and control transfers to the statement after the FOR LOOP statement.If lower_bound is equal to upper_bound, the statements execute only once. When lower_bound is greater than upper_bound, the statements do not execute at all.

WHILE loop: Here is the syntax for the WHILE loop statement:

```
WHILE condition
LOOP
    statements;
END LOOP;
```

The condition in the WHILE is a Boolean expression that evaluates to TRUE, FALSE or NULL. The WHILE loop statement continues to execute the statements between the LOOP and END LOOP as long as the condition in the WHILE clause evaluates to TRUE.

PL/SQL evaluates the condition in the WHILE clause before each loop iteration. If the condition is TRUE, then the loop body executes. In case it is FALSE or NULL, the loop terminates. If the condition is FALSE before entering the loop, the WHILE loop does not execute at all. This behavior is different from the LOOP statement whose loop body always executes once. To terminate the loop prematurely, you use an EXIT or EXIT WHEN statement.

SELECT INTO Statement

PL/SQL SELECT INTO statement is the simplest and fastest way to fetch a single row from a table into variables. The following illustrates the syntax of the PL/SQL SELECT INTO statement:

```
SELECT select_list INTO variable_list
FROM table_name WHERE   condition;
```

In this syntax, the number of columns in the variable_list must be the same as the number of variables (or the number of components of a record) in the select_list. In addition, their corresponding data type must be compatible.

%TYPE

The %TYPE attribute allow you to declare a constant, variable, or parameter to be of the same data type as previously declared variable, record, nested table, or database column.

Syntax: identifier Table.column_name%TYPE;

Example:

```
declare
     v_name employee.lastname%TYPE;
     v_dep  number;
     v_min_dep v_dep%TYPE:=31;
begin
     select   lastname   into   v_name   from   EMPLOYEE   where
     DEPARTMENTID=v_min_dep;
     DBMS_OUTPUT.PUT_LINE('v_name: '||v_name);
end;
```

%ROWTYPE

To declare a table-based record, you use the %ROWTYPE attribute with a table name. A table-based record has each field corresponding to a column in a table. The %ROWTYPE attribute provides a record type that represents a row in a database table. The record can store an entire row of data selected from the table or fetched from a cursor or cursor variable. Fields in a record and corresponding columns in a row have the same names and datatypes. You can use the %ROWTYPE attribute in variable declarations as a datatype specifier. Variables declared using %ROWTYPE are treated like those declared using a datatype name.

```
record_name table_name%ROWTYPE;
```

The following example declares a record named r_contact with the same structure as the contacts table in the sample database:

```
r_contact contacts%ROWTYPE;
```

Using Cursor

A cursor is a pointer that points to a result of a query. PL/SQL controls the context area through a cursor. A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the **active set (Result Set)**. You can name a cursor so that it could be referred to in a program to fetch and process the rows returned by the SQL statement, one at a time. PL/SQL has two types of cursors: implicit cursors and explicit cursors.

Implicit cursors: Whenever Oracle executes an SQL statement such as SELECT INTO, INSERT, UPDATE, and DELETE, it automatically creates an implicit cursor. Oracle internally manages the whole execution cycle of implicit cursors and reveals only the cursor's information and statuses such as SQL%ROWCOUNT, SQL%ISOPEN, SQL%FOUND, and SQL%NOTFOUND. The implicit cursor is not elegant when the query returns zero or multiple rows which cause NO_DATA_FOUND or TOO_MANY_ROWS exception respectively.

Attribute & Description
%FOUND: Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE.
%NOTFOUND: The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE.
%ISOPEN: Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement.
%ROWCOUNT: Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement.

Explicit cursors: An explicit cursor is an SELECT statement declared explicitly in the declaration section of the current block or a package specification. For an explicit cursor, you have control over its execution cycle from OPEN, FETCH, and CLOSE. Oracle defines an execution cycle that executes an SQL statement and associates a cursor with it.

Declaring the Cursor: Declaring the cursor defines the cursor with a name and the associated SELECT statement. For example −

```
CURSOR c_customers IS
   SELECT id, name, address FROM customers;
```

Opening the Cursor: Opening the cursor allocates the memory for the cursor and makes it ready for fetching the rows returned by the SQL statement into it. For example, we will open the above defined cursor as follows −

```
OPEN c_customers;
```

Fetching the Cursor: Fetching the cursor involves accessing one row at a time. For example, we will fetch rows from the above-opened cursor as follows −

```
FETCH c_customers INTO c_id, c_name, c_addr;
```

Closing the Cursor: Closing the cursor means releasing the allocated memory. For example, we will close the above-opened cursor as follows −

```
CLOSE c_customers;
```

Example: Following is a complete example to illustrate the concepts of explicit cursors &minua;

```
DECLARE
   c_id customers.id%type;
   c_name customers.name%type;
   c_addr customers.address%type;
   CURSOR c_customers is
      SELECT id, name, address FROM customers;
BEGIN
   OPEN c_customers;
   LOOP
   FETCH c_customers into c_id, c_name, c_addr;
      EXIT WHEN c_customers%notfound;
      dbms_output.put_line(c_id ||' '|| c_name ||' '||c_addr);
   END LOOP;
   CLOSE c_customers;
END;
```

PL/SQL cursor FOR LOOP statement: The cursor FOR LOOP statement is an elegant extension of the numeric FOR LOOP statement. The numeric FOR LOOP executes the body of a loop once for every integer value in a specified range. Similarly, the cursor FOR LOOP executes the body of the loop once for each row returned by the query associated with the cursor. A nice feature of the cursor FOR LOOP statement is that it allows you to fetch every row from a cursor without manually managing the execution cycle i.e., OPEN, FETCH, and CLOSE.

The cursor FOR LOOP implicitly creates its loop index as a record variable with the row type in which the cursor returns and then opens the cursor. In each loop iteration, the cursor FOR LOOP statement fetches a row from the result set into its loop index. If there is no row to fetch, the cursor FOR LOOP closes the cursor. The cursor is also closed if a statement inside the loop transfers control outside the loop, e.g., EXIT and GOTO, or raises an exception. The following illustrates the syntax of the cursor FOR LOOP statement:

```
FOR record IN cursor_name
LOOP
    process_record_statements;
END LOOP;
```

PL/SQL Cursor with Parameters: An explicit cursor may accept a list of parameters. Each time you open the cursor, you can pass different arguments to the cursor, which results in different result sets. The following shows the syntax of a declaring a cursor with parameters:

```
CURSOR cursor_name (parameter_list) IS cursor_query;
```

In the cursor query, each parameter in the parameter list can be used anywhere which a constant is used. The cursor parameters cannot be referenced outside of the cursor query. To open a cursor with parameters, you use the following syntax:

```
OPEN cursor_name (value_list);
```

Exception Handling

PL/SQL treats all errors that occur in an anonymous block, procedure, or function as exceptions. The exceptions can have different causes such as coding mistakes, bugs, even hardware failures. It is not possible to anticipate all potential exceptions; however, you can write code to handle exceptions to enable the program to continue running as normal. The code that you write to handle exceptions is called an exception handler. A PL/SQL block can have an exception-handling section, which can have one or more exception handlers. Here is the basic syntax of the exception-handling section:

```
BEGIN
    -- executable section
    ...
    -- exception-handling section
    EXCEPTION
        WHEN e1 THEN
            -- exception_handler1
        WHEN e2 THEN
            -- exception_handler1
        WHEN OTHERS THEN
            -- other_exception_handler
END;
```

In this syntax, e1, e2 are exceptions. When an exception occurs in the executable section, the execution of the current block stops and control transfers to the exception-handling section. If the exception e1 occurred, the exception_handler1 runs. If the exception e2 occurred, the exception_handler2 executes. In case any other exception raises, then the other_exception_handler runs.

After an exception handler executes, control transfers to the next statement of the enclosing block. If there is no enclosing block, then the control returns to the invoker if the exception handler is in a subprogram or host environment (SQL Developer or SQL*Plus) if the exception handler is in an anonymous block. Let's take some examples of handling exceptions. The following block accepts a customer id as an input and returns the customer name:

```
DECLARE
    l_name customers.NAME%TYPE;
    l_customer_id customers.customer_id%TYPE := &customer_id;
BEGIN
    -- get the customer name by id
    SELECT name INTO l_name
    FROM customers
    WHERE customer_id = l_customer_id;

    -- show the customer name
    dbms_output.put_line('Customer name is ' || l_name);
END;
```

Creating and Using Procedure

A PL/SQL procedure is a reusable unit that encapsulates specific business logic of the application. Technically speaking, a PL/SQL procedure is a named block stored as a schema object in the Oracle Database. The following illustrates the basic syntax of creating a procedure in PL/SQL:

```
CREATE [OR REPLACE ] PROCEDURE procedure_name (parameter_list)
IS
     [declaration statements]
BEGIN
     [execution statements]
EXCEPTION
     [exception handler]
END [procedure_name ];
```

A procedure begins with a header that specifies its name and an optional parameter list. Each parameter can be in either IN, OUT, or INOUT mode. The parameter mode specifies whether a parameter can be read from or write to. An IN parameter is read-only. You can reference an IN parameter inside a procedure, but you cannot change its value. Oracle uses IN as the default mode. It means that if you don't specify the mode for a parameter explicitly, Oracle will use the IN mode.

An OUT parameter is writable. Typically, you set a returned value for the OUT parameter and return it to the calling program. Note that a procedure ignores the value that you supply for an OUT parameter. An INOUT parameter is both readable and writable. The procedure can read and modify it. Note that OR REPLACE option allows you to overwrite the current procedure with the new code.

Similar to an anonymous block, the procedure body has three parts. The executable part is mandatory whereas the declarative and exception-handling parts are optional. The executable part must contain at least one executable statement. The following procedure accepts a customer id and prints out the customer's contact information including first name, last name, and email:

```
CREATE OR REPLACE PROCEDURE print_contact (in_customer_id NUMBER)
IS
  r_contact contacts%ROWTYPE;
BEGIN
  -- get contact based on customer id
  SELECT * INTO r_contact FROM contacts
  WHERE customer_id = p_customer_id;

  -- print out contact's information
  dbms_output.put_line( r_contact.first_name || ' ' ||
  r_contact.last_name || '<' || r_contact.email ||'>' );
EXCEPTION
   WHEN OTHERS THEN
      dbms_output.put_line( SQLERRM );
END;
```

Executing a PL/SQL procedure: The following shows the syntax for executing a procedure:

```
EXECUTE procedure_name(arguments);
```

Or

```
EXEC procedure_name(arguments);
```

For example, to execute the print_contact procedure that prints the contact information of customer id 100, you use the following statement:

```
EXEC print_contact(100);
```

To delete a procedure, you use the DROP PROCEDURE followed by the procedure's name that you want to drop as shown in the following syntax:

```
DROP PROCEDURE procedure_name;
```

For example, the following statement drops the print_contact procedure:

```
DROP PROCEDURE print_contact;
```

Creating and Using Functions

Similar to a procedure, a PL/SQL function is a reusable program unit stored as a schema object in the Oracle Database. The following illustrates the syntax for creating a function:

```
CREATE [OR REPLACE] FUNCTION function_name (parameter_list)
RETURN return_type
IS
     [declarative section]
BEGIN
     [executable section]

[EXCEPTION]
     [exception-handling section]
END;
```

A function consists of a header and body. The function header has the function name and a RETURN clause that specifies the datatype of the returned value. Each parameter of the function can be either in the IN, OUT, or INOUT mode. For more information on the parameter mode, check it out the PL/SQL procedure tutorial

The function body is the same as the procedure body which has three sections: declarative section, executable section, and exception-handling section. The declarative section is between the IS and BEGIN keywords. It is where you declare variables, constants, cursors, and user-defined types. The executable section is between the BEGIN and END keywords. It is where you place the executable statements. Unlike a procedure, you must have at least one RETURN statement in the executable statement. The exception-handling section is where you put the exception handler code. In these three sections, only the executable section is required, the others are optional. The following example creates a function that calculates total sales by year.

```
CREATE OR REPLACE FUNCTION get_total_sales(
    in_year PLS_INTEGER)
RETURN NUMBER
IS
    l_total_sales NUMBER := 0;
BEGIN
    -- get total sales
    SELECT SUM(unit_price * quantity)INTO l_total_sales
    FROM order_items INNER JOIN orders USING (order_id)
    WHERE status = 'Shipped'
    GROUP BY EXTRACT(YEAR FROM order_date)
    HAVING EXTRACT(YEAR FROM order_date) = in_year;

    -- return the total sales
    RETURN l_total_sales;
END;
```

You use a function anywhere that you use an expression of the same type. You can call a function in various places such as:

```
DECLARE
    l_sales_2017 NUMBER := 0;
BEGIN
    l_sales_2017 := get_total_sales (2017);
    DBMS_OUTPUT.PUT_LINE('Sales 2017: ' || l_sales_2017);
END;
```

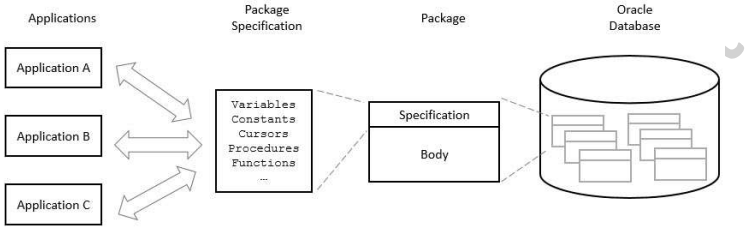
The DROP FUNCTION deletes a function from the Oracle Database. The syntax for removing a function is straightforward:

```
DROP FUNCTION function_name;
```

Followed by the DROP FUNCTION keywords is the function name that you want to drop. For example, the following statement drops the GET_TOTAL_SALES function:

```
DROP FUNCTION get_total_sales;
```

Package



In PL/SQL, a package is a schema object that contains definitions for a group of related functionalities. A package includes variables, constants, cursors, exceptions, procedures, functions, and subprograms. It is compiled and stored in the Oracle Database. Typically, a package has a specification and a body. A package specification is mandatory while the package body can be required or optional, depending on the package specification. The picture illustrates PL/SQL packages.

Package specification: The package specification declares the public objects that are accessible from outside the package. If a package specification whose public objects include cursors and subprograms, then it must have a body which defines queries for the cursors and code for the subprograms. To create a new package specification, you use the CREATE PACKAGE statement as shown below:

```
CREATE [OR REPLACE] PACKAGE [schema_name.] <package_name> IS | AS
    declarations;
END [<package_name>];
```

In this syntax: Follow the CREATE PACKAGE clause is the name of the package which you want to create. The OR REPLACE option allows you to replace the package if it exists and recompile it. The schema_name is the schema to which the package belongs. By default, it is your schema. Between the AS and END keywords, you declare the public items of the package specification. See the following package specification example:

```
CREATE OR REPLACE PACKAGE order_mgmt AS
      gc_shipped_status  CONSTANT VARCHAR(10) := 'Shipped';
      gc_pending_status CONSTANT VARCHAR(10) := 'Pending';
      gc_canceled_status CONSTANT VARCHAR(10) := 'Canceled';

      -- cursor that returns the order detail
      CURSOR g_cur_order(p_order_id NUMBER)
      IS
      SELECT  customer_id,  status,  salesman_id,  order_date,  item_id,
      product_name, quantity, unit_price FROM order_items INNER JOIN orders
      USING (order_id) INNER JOIN products USING (product_id) WHERE order_id
      = p_order_id;

        -- get net value of a order
        FUNCTION get_net_value( p_order_id NUMBER)
          RETURN NUMBER;

        -- Get net value by customer
        FUNCTION get_net_value_by_customer(p_customer_id NUMBER,
            p_year        NUMBER)
          RETURN NUMBER;
END order_mgmt;
```

Package body: A package body contains the implementation of the cursors or subprograms declared in the package specification. In the package body, you can declare or define private variables, cursors, etc., used only by package body itself. A package body can have an initialization part whose statements initialize variables or perform other one-time setups for the whole package. A package body can also have an exception-handling part used to handle exceptions. To create a package body, you use the CREATE PACKAGE BODY as shown below:

```
CREATE [OR REPLACE] PACKAGE BODY <package_name> IS | AS
    declarations
    implementations;
[BEGIN EXCEPTION]
END [<package_name>];
```

In this syntax: First, you specify the package name after the CREATE PACKAGE BODY keywords. The schema name is optional. By default, it is your schema. Second, the OR REPLACE option replaces the existing package body definition or do nothing if the package body does not exist. Third, between the AS and END keywords are the declarations of private items and the implementations of the public items declared in the package specification. Note that you can use either IS or AS keyword. The following example illustrates how to create the body of the order_mgmt package:

```
CREATE OR REPLACE PACKAGE BODY order_mgmt
AS
  -- get net value of a order
  FUNCTION get_net_value(p_order_id NUMBER)
    RETURN NUMBER
  IS
    ln_net_value NUMBER
  BEGIN
    SELECT SUM(unit_price * quantity) INTO ln_net_value
    FROM order_items WHERE order_id = p_order_id;
  RETURN p_order_id;
  EXCEPTION
      WHEN no_data_found THEN
            DBMS_OUTPUT.PUT_LINE(SQLERRM);
  END get_net_value;

  -- Get net value by customer
  FUNCTION   get_net_value_by_customer   (p_customer_id   NUMBER,   p_year
NUMBER)
    RETURN NUMBER
  IS
    ln_net_value NUMBER
  BEGIN
    SELECT SUM(quantity * unit_price)INTO ln_net_value FROM order_items
    INNER JOIN orders USING (order_id)
    WHERE extract(YEAR FROM order_date) = p_year
    AND customer_id = p_customer_id AND status = gc_shipped_status;
  RETURN ln_net_value;
  EXCEPTION
        WHEN no_data_found THEN DBMS_OUTPUT.PUT_LINE(SQLERRM);
 END get_net_value_by_customer;

END order_mgmt;
```

To drop a package, you use the DROP PACKAGE statement with the following syntax:

```
DROP PACKAGE [BODY] package_name;
```

Triggers

A trigger is a named PL/SQL block stored in the Oracle Database and executed automatically when a triggering event takes place. The event can be any of the following:

- A data manipulation language (DML) statement executed against a table e.g., INSERT, UPDATE, or DELETE. For example, if you define a trigger that fires before an INSERT statement on the customers table, the trigger will fire once before a new row is inserted into the customers table.
- A data definition language (DDL) statement executes e.g., CREATE or ALTER statement. These triggers are often used for auditing purposes to record changes of the schema.
- A system event such as startup or shutdown of the Oracle Database.
- A user event such as login or logout.

The act of executing a trigger is also known as firing a trigger. We say that the trigger is fired. Oracle triggers are useful in many cases such as the following:

- Enforcing complex business rules that cannot be established using integrity constraint such as UNIQUE, NOT NULL, and CHECK.
- Preventing invalid transactions.
- Gathering statistical information on table accesses.
- Generating value automatically for derived columns.
- Auditing sensitive data.

To create a new trigger in Oracle, you use the following CREATE TRIGGER statement:

```
CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE | AFTER } triggering_event ON table_name
[FOR EACH ROW]
[FOLLOWS | PRECEDES another_trigger]
[ENABLE / DISABLE ]
[WHEN condition]
DECLARE
    declaration statements
BEGIN
    executable statements
EXCEPTION
    exception_handling statements
END;
```

Let's examine the syntax of the CREATE TRIGGER statement in more detail. A trigger has two main parts: header and body. The following illustrates the trigger header:

```
CREATE [OR REPLACE] TRIGGER trigger_name
    {BEFORE | AFTER } triggering_event ON table_name
    [FOR EACH ROW]
    [FOLLOWS | PRECEDES another_trigger]
    [ENABLE / DISABLE ]
    [WHEN condition]
```

And this is the trigger body:

```
DECLARE
    declaration statements
BEGIN
    executable statements
EXCEPTION
    exception_handling statements
END;
```

As you can see, the trigger body has the same structure as an anonymous PL/SQL block. The DROP TRIGGER statement allows you to remove a trigger from the database. Here is the basic syntax of the DROP TRIGGER statement:

```
DROP TRIGGER [schema_name.]trigger_name;
```

Creating Objects

Nested Tables: Nested tables are single-dimensional, unbounded collections of homogeneous elements. First, a nested table is single-dimensional, meaning that each row has a single column of data like a one-dimension array. Second, a nested table is unbounded. It means that the number of elements of a nested table is predetermined. Third, homogeneous elements mean that all elements of a nested table have the same data type. Noted that a nested table is initially dense. However, it can become sparse through the removal of elements. Declaring a nested table is a two-step process. First, declare the nested table type using this syntax:

```
TYPE nested_table_type
    IS TABLE OF element_datatype [NOT NULL];
```

Then, declare the nested table variable based on a nested table type:

```
nested_table_variable nested_table_type;
```

It is possible to create a nested table type located in the database:

```
CREATE [OR REPLACE] TYPE nested_table_type
    IS TABLE OF element_datatype [NOT NULL];
```

If you want to drop a type, use the following DROP TYPE statement:

```
DROP TYPE type_name [FORCE];
```

Varrays: A VARRAY is single-dimensional collections of elements with the same data type. Unlike an associative array and nested table, a VARRAY always has a fixed number of elements(bounded) and never has gaps between the elements (not sparse). To declare a VARRAY type, you use this syntax:

```
TYPE type_name IS VARRAY(max_elements)
    OF element_type [NOT NULL];
```

In this declaration: type_name is the type of the VARRAY. max_elements is the maximum number of elements allowed in the VARRAY. NOT NULL specifies that the element of the VARRAY of that type cannot have NULL elements. Note that a VARRAY variable can be null, or uninitialized. element_type is the type of elements of the VARRAY type's variable. To create a VARRAY type which is accessible globally in the database, not just in your PL/SQL code, you use the following syntax:

```
CREATE [OR REPLACE ] TYPE type_name AS | IS
    VARRAY(max_elements) OF element_type [NOT NULL];
```

In this declaration, the OR REPLACE modifies existing type while keeping all existing grants of privileges.Once you created your own VARRAY type, you can declare a VARRAY instance of that type by referencing the VARRAY type. The basic syntax for VARRAY declaration is:

```
varray_name type_name [:= type_name(...)];
```

In this syntax: The varray_name is the name of the VARRAY. The type_name is the VARRAY type. The type_name(...) is the constructor of the VARRAY type, which accepts a comma-separated list of elements as arguments. It has the same name as the VARRAY type.

Unit – 5

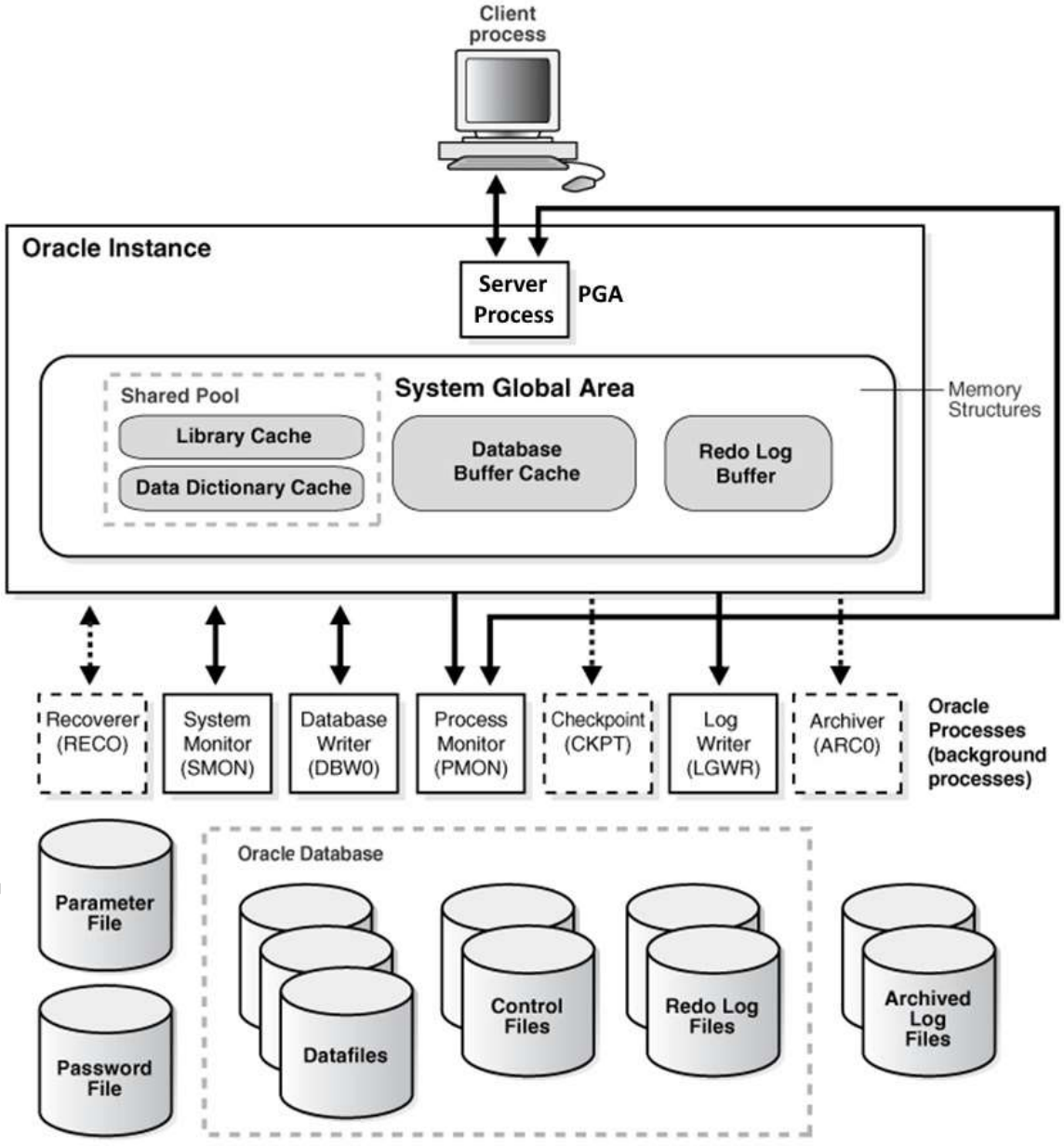
Oracle Database Structure and Storage Database, Resource Management and Task Scheduling

Instance Architecture

A database instance is a set of memory structures that manage database files. A Database Instance is an interface between client applications (users) and the database. An Oracle instance consists of three main parts: **System Global Area (SGA)**, **Program Global Area (PGA)**, and **background processes**. When an instance is started, Oracle Database allocates a memory area called the SGA and starts one or more background processes. The SGA serves various purposes, including the following:

- Maintaining internal data structures that many processes and threads access concurrently
- Caching data blocks read from disk
- Buffering redo data before writing it to the online redo log files
- Storing SQL execution plans

Oracle processes running on a single computer share the SGA. The way in which Oracle processes associate with the SGA varies according to operating system. A database instance includes background processes. Server processes, and the process memory allocated in these processes, also exist in the instance. The instance continues to function when server processes terminate. The following graphic shows the main components of an Oracle database instance.



Database Processes

An Oracle database uses memory structures and processes to manage and access the database. All memory structures exist in the main memory of the computers that constitute the database system. Processes are jobs that work in the memory of these computers. The architectural features discussed in this section enable the Oracle database to support:

- Many users concurrently accessing a single database
- The high performance required by concurrent multiuser, multiapplication database systems

Oracle creates a set of background processes for each instance. The background processes consolidate functions that would otherwise be handled by multiple Oracle programs running for each user process. They asynchronously perform I/O and monitor other Oracle process to provide increased parallelism for better performance and reliability.

The mandatory background processes are present in all typical database configurations. These processes run by default in a database instance started with a minimally configured initialization parameter file. These are some of the mandatory background processes:

Process Monitor Process (PMON): The process monitor (PMON) performs process recovery when a user process fails. PMON is responsible for cleaning up the database buffer cache and freeing resources that the user process was using. For example, it resets the status of the active transaction table, releases locks, and removes the process ID from the list of active processes. PMON periodically checks the status of dispatcher and server processes, and restarts any that have stopped running (but not any that Oracle has terminated intentionally). PMON also registers information about the instance and dispatcher processes with the network listener.

System Monitor Process (SMON): The system monitor process (SMON) performs recovery, if necessary, at instance startup. SMON is also responsible for cleaning up temporary segments that are no longer in use and for coalescing contiguous free extents within dictionary managed tablespaces. If any terminated transactions were skipped during instance recovery because of file-read or offline errors, SMON recovers them when the tablespace or file is brought back online. SMON checks regularly to see whether it is needed. Other processes can call SMON if they detect a need for it.

Database Writer Process (DBWn): The database writer process (DBWn) writes the contents of buffers to datafiles. The DBWn processes are responsible for writing modified (dirty) buffers in the database buffer cache to disk. Although one database writer process (DBW0) is adequate for most systems, you can configure additional processes (DBW1 through DBW9 and DBWa through DBWj) to improve write performance if your system modifies data heavily. These additional DBWn processes are not useful on uniprocessor systems.

When a buffer in the database buffer cache is modified, it is marked dirty. A cold buffer is a buffer that has not been recently used according to the least recently used (LRU) algorithm. The DBWn process writes cold, dirty buffers to disk so that user processes are able to find cold, clean buffers that can be used to read new blocks into the cache. As buffers are dirtied by user processes, the number of free buffers diminishes. If the number of free buffers drops too low, user processes that must read blocks from disk into the cache are not able to find free buffers. DBWn manages the buffer cache so that user processes can always find free buffers.

Log Writer Process (LGWR): The log writer process (LGWR) is responsible for redo log buffer management—writing the redo log buffer to a redo log file on disk. LGWR writes all redo entries that have been copied into the buffer since the last time it wrote.

The redo log buffer is a circular buffer. When LGWR writes redo entries from the redo log buffer to a redo log file, server processes can then copy new entries over the entries in the redo log buffer that have been written to disk. LGWR normally writes fast enough to ensure that space is always available in the buffer for new entries, even when access to the redo log is heavy.

Checkpoint Process (CKPT): When a checkpoint occurs, Oracle must update the headers of all datafiles to record the details of the checkpoint. This is done by the CKPT process. The CKPT process does not write blocks to disk; DBWn always performs that work. The statistic DBWR checkpoints displayed by the System_Statistics monitor in Enterprise Manager indicates the number of checkpoint requests completed.

Archiver Processes (ARCn): The archiver process (ARCn) copies redo log files to a designated storage device after a log switch has occurred. ARCn processes are present only when the database is in ARCHIVELOG mode, and automatic archiving is enabled. An Oracle instance can have up to 10 ARCn processes (ARC0 to ARC9). The LGWR process starts a new ARCn process whenever the current number of ARCn processes is insufficient to handle the workload. The alert log keeps a record of when LGWR starts a new ARCn process.

Recoverer Process (RECO): The recoverer process (RECO) is a background process used with the distributed database configuration that automatically resolves failures involving distributed transactions. The RECO process of a node automatically connects to other databases involved in an in-doubt distributed transaction. When the RECO process re-establishes a connection between involved database servers, it automatically resolves all in-doubt transactions, removing from each database's pending transaction table any rows that correspond to the resolved in-doubt transactions.

If the RECO process fails to connect with a remote server, RECO automatically tries to connect again after a timed interval. However, RECO waits an increasing amount of time (growing exponentially) before it attempts another connection. The RECO process is present only if the instance permits distributed transactions. The number of concurrent distributed transactions is not limited.

Memory Structure

To Understand Oracle Database basically we need to understand the Memory Architecture i.e. how the memory is used and for what purpose along with the basic processes (Oracle Process Architecture) that uses this memory for Inter process Communication. In this post we will understand the Oracle Memory Architecture in detail. Oracle memory architecture is divided in following memory structure:

System Global Area (SGA): This is a large, shared memory segment that virtually all Oracle processes will access at one point or another. SGA is used to store database information that is shared by database processes. It contains data and control information for the Oracle Server and is allocated in the virtual memory if the computer where Oracle resides. SGA consists of several memory structures:

- **Redo Buffer:** The redo buffer is where data that needs to be written to the online redo logs will be cached temporarily, before it is written to disk. Since a memory-to-memory transfer is much faster than a memory-to-disk transfer, use of the redo log buffer can speed up database operation. The data will not reside in the redo buffer for very long. In fact, LGWR initiates a flush of this area in one of the following scenarios: Every three seconds, Whenever someone commits, When LGWR is asked to switch log files, or When the redo buffer gets one-third full or contains 1MB of cached redo log data.

- **Buffer Cache:** The block buffer cache is where Oracle stores database blocks before writing them to disk and after reading them in from disk. There are three places to store cached blocks from individual segments in the SGA: Default pool (hot cache): The location where all segment blocks are normally cached. Keep pool (warm cache): An alternate buffer pool where by convention you assign segments that are accessed fairly frequently, but still get aged out of the default buffer pool due to other segments needing space. Recycle pool (do not care to cache): An alternate buffer pool where by convention you assign large segments that you access very randomly, and which would therefore cause excessive buffer flushing of many blocks from many segments. There's no benefit to caching such segments because by the time you wanted the block again, it would have been aged out of the cache. You would separate these segments out from the segments in the default and keep pools so they would not cause those blocks to age out of the cache.
- **Shared Pool:** The shared pool is where Oracle caches many bits of "program" data. When we parse a query, the parsed representation is cached there. Before we go through the job of parsing an entire query, Oracle searches the shared pool to see if the work has already been done. PL/SQL code that you run is cached in the shared pool, so the next time you run it, Oracle doesn't have to read it in from disk again. PL/SQL code is not only cached here, it is shared here as well. If you have 1,000 sessions all executing the same code, only one copy of the code is loaded and shared among all sessions. Oracle stores the system parameters in the shared pool. The data dictionary cache (cached information about database objects) is stored here. Shared Pool manages memory on an LRU basis, similar to buffer cache, which is perfect for caching and reusing data.
- **Large Pool:** The large pool is not so named because it is a "large" structure (although it may very well be large in size). It is so named because it is used for allocations of large pieces of memory that are bigger than the shared pool is designed to handle. Large memory allocations tend to get a chunk of memory, use it, and then be done with it. There was no need to cache this memory as in buffer cache and Shared Pool, hence a new pool was allocated. So basically Shared pool is more like Keep Pool whereas Large Pool is similar to the Recycle Pool. Large pool is used specifically by: Shared server connections, to allocate the UGA region in the SGA. Parallel execution of statements, to allow for the allocation of interprocess message buffers, which are used to coordinate the parallel query servers. Backup for RMAN disk I/O buffers in some cases.
- **Java Pool:** The Java pool is used in different ways, depending on the mode in which the Oracle server is running. In dedicated server mode the total memory required for the Java pool is quite modest and can be determined based on the number of Java classes you'll be using. In shared server connection the java pool includes shared part of each java class and Some of the UGA used for per-session state of each session, which is allocated from the JAVA_POOL within the SGA.
- **Streams Pool:** The Streams pool (or up to 10 percent of the shared pool if no Streams pool is configured) is used to buffer queue messages used by the Streams process as it moves or copies data from one database to another.

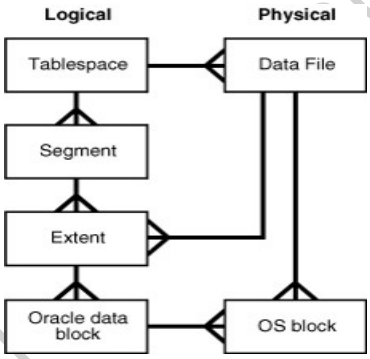
Process Global Area (PGA): This is memory that is private to a single process or thread; it is not accessible from other processes/threads. PGA is the memory reserved for each user process connecting to an Oracle Database and is allocated when a process is created and deallocated when a process is terminated. Contents of PGA: Private SQL Area: Contains data such as bind information and run-time memory structures. It contains Persistent Area which contains bind information and is freed only when the cursor is closed and Run time Area which is created as the first step of an execute request. This area is freed only when the statement has been executed. The number of Private SQL areas that can be allocated to a user process depends on the OPEN_CURSORS initialization

parameter. Session Memory: Consists of memory allocated to hold a session's variable and other info related to the session. SQL Work Areas: Used for memory intensive operations such as: Sort, Hash-join, Bitmap merge, Bitmap Create.

User Global Area (UGA): This is memory associated with your session. It is located either in the SGA or the PGA, depending whether you are connected to the database using a shared server (it will be in the SGA), or a dedicated server (it will be in the PGA).

Oracle Logical Storage Structures

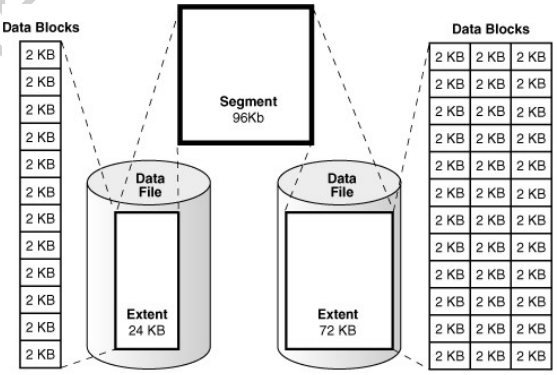
Oracle Database allocates logical space for all data in the database. The logical units of database space allocation are data blocks, extents, segments, and tablespaces. At a physical level, the data is stored in data files on disk. The data in the data files is stored in operating system blocks.



The relationships among data blocks, extents, and segments within a tablespace. In this example, a segment has two extents stored in different data files.

At the finest level of granularity, Oracle Database stores data in data blocks. One logical data block corresponds to a specific number of bytes of physical disk space, for example, 2 KB. Data blocks are the smallest units of storage that Oracle Database can use or allocate.

An **extent** is a set of logically contiguous data blocks allocated for storing a specific type of information. In Figure, the 24 KB extent has 12 data blocks, while the 72 KB extent has 36 data blocks.



A **segment** is a set of extents allocated for a specific database object, such as a table. For example, the data for the employees table is stored in its own data segment, whereas each index for employees is stored in its own index segment. Every database object that consumes storage consists of a single segment.

Each segment belongs to one and only one **tablespace**. Thus, all extents for a segment are stored in the same tablespace. Within a tablespace, a segment can include extents from multiple data files, as shown in Figure 12-2. For example, one extent for a segment may be stored in users01.dbf, while another is stored in users02.dbf. A single extent can never span data files.

Oracle Database manages the logical storage space in the data files of a database in units called **data blocks**, also called Oracle blocks or pages. A data block is the minimum unit of database I/O. At the physical level, database data is stored in disk files made up of operating system blocks. An operating system block is the minimum unit of data that the operating system can read or write. In contrast, an Oracle block is a logical storage structure whose size and structure are not known to the operating system.

Oracle Database files (Oracle Physical Storage Structure)

A tablespace in an Oracle database consists of one or more physical datafiles. A datafile can be associated with only one tablespace and only one database.

Oracle creates a datafile for a tablespace by allocating the specified amount of disk space plus the overhead required for the file header. When a datafile is created, the operating system under which Oracle runs is responsible for clearing old information and authorizations from a file before allocating it to Oracle. If the file is large, this process can take a significant amount of time. The first tablespace in any database is always the SYSTEM tablespace, so Oracle automatically allocates the first datafiles of any database for the SYSTEM tablespace during database creation.

Data File: When a datafile is first created, the allocated disk space is formatted but does not contain any user data. However, Oracle reserves the space to hold the data for future segments of the associated tablespace—it is used exclusively by Oracle. As the data grows in a tablespace, Oracle uses the free space in the associated datafiles to allocate extents for the segment. The data associated with schema objects in a tablespace is physically stored in one or more of the datafiles that constitute the tablespace. Note that a schema object does not correspond to a specific datafile; rather, a datafile is a repository for the data of any schema object within a specific tablespace. Oracle allocates space for the data associated with a schema object in one or more datafiles of a tablespace. Therefore, a schema object can span one or more datafiles. Unless table striping is used (where data is spread across more than one disk), the database administrator and end users cannot control which datafile stores a schema object.

The database control file is a small binary file necessary for the database to start and operate successfully. A control file is updated continuously by Oracle during database use, so it must be available for writing whenever the database is open. If for some reason the control file is not accessible, then the database cannot function properly. Each control file is associated with only one Oracle database.

Control File: A control file contains information about the associated database that is required for access by an instance, both at startup and during normal operation. Information about the database is stored in different sections of the control file. Each section is a set of records about an aspect of the database. For example, one section in the control file tracks data files and contains a set of records, one for each data file. Each section is stored in multiple logical control file blocks. Records can span blocks within a section. The control file contains the following types of records:

- Circular reuse records: These records contain noncritical information that is eligible to be overwritten if needed. When all available record slots are full, the database either expands the control file to make room for a new record or overwrites the oldest record. Examples include records about archived redo log files and RMAN backups.
- Noncircular reuse records: These records contain critical information that does not change often and cannot be overwritten. Examples of information include tablespaces, data files, online redo log files, and redo threads. Oracle Database never reuses these records unless the corresponding object is dropped from the tablespace.

Redo Log File: The most crucial structure for recovery is the online redo log, which consists of two or more pre-allocated files that store changes to the database as they occur. The online redo log records changes to the data files. The database maintains online redo log files to protect against data loss. Specifically, after an instance failure the online redo log files enable Oracle Database to recover committed data not yet written to the data files. Oracle Database writes every transaction synchronously to the redo log buffer, which is then written to the online redo logs. The contents of the log include uncommitted transactions, undo data, and schema and object management statements.

Oracle Database uses the online redo log only for recovery. However, administrators can query online redo log files through a SQL interface in the Oracle LogMiner utility. Redo log files are a useful source of historical information about database activity.

Creating & Altering Database

Use the CREATE DATABASE statement to create a database, making it available for general use. This statement erases all data in any specified data files that already exist in order to prepare them for initial database use. If you use the statement on an existing database, then all data in the data files is lost.

After creating the database, this statement mounts it in either exclusive or parallel mode, depending on the value of the CLUSTER_DATABASE initialization parameter and opens it, making it available for normal use. You can then create tablespaces for the database. To create a database, you must have the SYSDBA system privilege. An initialization parameter file with the name of the database to be created must be available, and you must be in STARTUP NOMOUNT mode.

Opening & shutdown Database

Opening: To start up a database instance, you use the STARTUP command. When the Oracle Database starts an instance, it goes through the following stages: NOMOUNT, MOUNT, and OPEN. The STARTUP command allows you to control the stage of the database instance.

NOMOUNT stage: In the NOMOUNT stage, Oracle carries the following steps:

- First, search for a server parameter file in the default location. You can override the default behavior by using the SPFILE or PFILE parameters in the STARTUP command.
- Next, read the parameter file to get the values of the initialization parameters.
- Then, allocate the system global area (SGA) based on the initialization parameter settings.
- After that, start the Oracle background processes such as SMON, PMON, and LGWR.
- Finally, open the alert log and trace files and record all explicit parameters to the alert log in the valid parameter syntax.

At the NOMOUNT stage, Oracle does not associate the database with the instance.

MOUNT stage: In the MOUNT stage, Oracle associates a database with an instance. In other words, the instance mounts the database. The instance carries the following steps to mount a database:

- First, get the name of the database control files specified in the CONTROL_FILE initialization parameter.
- Second, open the control files.
- Third, find the name of data files and the online redo log files.

When a database is mounted, the database is only available to database administrators, not all users.

OPEN stage: In the OPEN stage, Oracle performs the following actions:

- First, open the online data files in tablespaces other than the undo tablespaces.
- Then, select an undo tablespace. The instance uses default undo tablespace if an undo tablespace is specified in the UNDO_TABLESPACE initialization parameter. Otherwise, it will select the first available undo tablespace.
- Finally, open the online redo log files.

When Oracle opens a mounted database, the database is available for normal operations.

Shutdown: To shut down a currently running Oracle Database instance, you use the SHUTDOWN command as follows:

```
SHUTDOWN [ABORT | IMMEDIATE | NORMAL | TRANSACTIONAL [LOCAL]]
```

The **SHUTDOWN NORMAL** option waits for the current users to disconnect from the database before shutting down the database. The database instance will not accept any further database connection. The SHUTDOWN NORMAL does not require an instance recovery on the next database startup.

The **SHUTDOWN TRANSACTIONAL** waits for all uncommitted transactions to complete before shutting down the database instance. This saves the work for all users without requesting them to log off.

The database instance also does not accept any new transaction after a SHUTDOWN TRANSACTIONAL . When completing all transactions, the database instance disconnects all the currently connected users from the database and shuts down.

The **SHUTDOWN ABORT** is not recommended and only used on some occasions. The SHUTDOWN ABORT has a similar effect as you unplug the power of the server. The database will be in an inconsistent state. Therefore, you should never use the SHUTDOWN ABORT command before backing up the database. If you try to do so, you may not be able to recover the backup.

The **SHUTDOWN IMMEDIATE** is the most common and practical way to shut down the Oracle database. The SHUTDOWN IMMEDIATE does not wait for the current users to disconnect from the database or current transactions to complete. During the SHUTDOWN IMMEDIATE, all the connected sessions are disconnected immediately, all uncommitted transactions are rolled back, and the database completely shuts down.

Initialization Parameter

When the Oracle database is started, one of the first things it needs to do is read the database initialization parameter file. The parameter file **(init.ora**) is created by the DBA and defines the overall instance configuration, such as how much memory should be allocated to the instance, the file locations, and internal optimization parameters. The initialization parameters are a very important part of the Oracle database. Oracle reads the initialization parameter values from either a PFILE or SPFILE as the database is starting. The parameters tell the Oracle programs how much memory to allocate, where to put files related to the database and the location of existing datafiles, the following are the list of parameters used int init.ora:

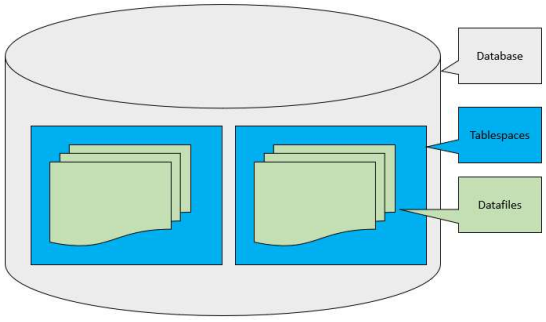
NAME	VALUE	DESCRIPTION
ALWAYS_ANTI_JOIN	NESTED_LOOPS	Always use this anti-join when possible
ALWAYS_SEMI_JOIN	standard	Always use this semi-join when possible
AQ_TM_PROCESSES	0	Number of AQ Time Managers to start
AUDIT_FILE_DEST	(PD)	Destination for audit files
AUDIT_TRAIL	NONE	Enable system auditing
BACKGROUND_CORE_DUMP	PARTIAL	Sets whether SGA is dumped with core file dump, PARTIAL means don't dump SGA.
BACKGROUND_DUMP_DEST	(PD)	Detached process dump directory
BACKUP_TAPE_IO_SLAVES	FALSE	BACKUP Tape I/O slaves

BITMAP_MERGE_AREA_SIZE	1048576	Maximum memory allow for BITMAP MERGE
BLANK_TRIMMING	FALSE	Blank trimming semantics parameter
BUFFER_POOL_KEEP	0	Number of database blocks/latches in KEEP buffer pool
BUFFER_POOL_RECYCLE	0	Number of database blocks/latches in recycle buffer pool
COMMIT_POINT_STRENGTH	1	Bias this node has toward not preparing in a two-phase commit
COMPATIBLE	8.1.0	Database will be compatible with this software version
CONTROL_FILE_RECORD_KEEP_TIME	7	Control file record keep time in days
CONTROL_FILES	(PD)	Control file names list
CORE_DUMP_DEST	(PD)	Destination for core dump files.
CPU_COUNT	(PD)	Number of cpu's for this instance
CREATE_BITMAP_AREA_SIZE	8388608	Size of create bitmap buffer for bitmap index
CURSOR_SPACE_FOR_TIME	FALSE	Use more memory in order to get faster execution
DB_BLOCK_BUFFERS	8000	Number of database blocks cached in memory
DB_BLOCK_CHECKING	TRUE	Data and index block checking overrides events 10210 and 10211
DB_BLOCK_CHECKSUM	FALSE	Store checksum in db blocks and check during reads
DB_BLOCK_LRU_LATCHES	1|CPU_COUNT/2	Number of lru latches
DB_BLOCK_MAX_DIRTY_TARGET	DB_BLOCK_BUFFERS	Upper bound on modified buffers/recovery reads
DB_BLOCK_SIZE	(PD)	Size of database block in bytes
DB_DOMAIN	WORLD	Directory part of global database name stored with CREATE DATABASE
DB_FILE_DIRECT_IO_COUNT	64	Sequential I/O block count
DB_FILE_MULTIBLOCK_READ_COUNT	8	Db blocks read for each IO
DB_FILE_NAME_CONVERT	NULL	Datafile name convert pattern and string for standby/clone database
DB_FILES	MAXDATAFILES	Max allowable # db files
DB_NAME	(PD)	Database name specified in CREATE DATABASE
DB_WRITER_PROCESSES	1	Number of background database writer processes to start

DBLINK_ENCRYPT_LOGIN	FALSE	Enforce password for distributed login always be encrypted
DBWR_IO_SLAVES	0	Number of DBWR I/O slaves
DELAYED_LOGGING_BLOCK_CLEANOUTS	TRUE	Turns delayed block cleanout on or off
DISK_ASYNCH_IO	TRUE	Use asynch I/O for random access devices
DISTRIBUTED_TRANSACTIONS	(PD)	Max. number of concurrent distributed transactions
DML_LOCKS	4*Trans.	Dml locks - one for each table modified in a transaction
ENQUEUE_RESOURCES	Derived	Resources for enqueues
EVENT	NULL	Debug event control
FIXED_DATE	NULL	Fixed SYSDATE value
FREEZE_DB_FOR_FAST_INSTANCE_RECOVERY	FALSE	Freeze database during instance recovery (OPS)
GC_DEFER_TIME	10	How long to defer down converts for hot buffers (DFS)(OPS)
GC_FILES_TO_LOCKS	NULL	Mapping between file numbers and lock buckets (DFS)(OPS)
GC_RELEASABLE_LOCKS	0	Number of releasable locks (DFS)(OPS)
GC_ROLLBACK_LOCKS	20	Locks for the rollback segments (DFS)(OPS)
GLOBAL_NAMES	TRUE	Enforce that database links have same name as remote database
HASH_JOIN_ENABLED	TRUE	Enable/disable hash join
HASH_MULTIBLOCK_IO_COUNT	1	Number of blocks hash join will read/write at once
HI_SHARED_MEMORY_ADDRESS	0	SGA starting address (high order 32-bits on 64-bit platforms)
HS_AUTOREGISTER	TRUE	Enable automatic server DD updates in HS agent self-registration
IFILE	NULL	Include file in init.ora
INSTANCE_GROUPS	NULL	List of instance group names
INSTANCE_NAME	NULL	Instance name supported by the instance
INSTANCE_NUMBER	0	Instance number
JAVA_POOL_SIZE	10000K	Size in bytes of the Java pool
JOB_QUEUE_INTERVAL	60	Wakeup interval in seconds for job queue processes
JOB_QUEUE_KEEP_CONNECTIONS	FALSE	Keep network connections between execution of jobs
JOB_QUEUE_PROCESSES	0	Number of job queue processes to start

LARGE_POOL_SIZE	0	Size in bytes of the large allocation pool (auto set at 600k)
LICENSE_MAX_SESSIONS	0	Maximum number of non-system user sessions allowed
LICENSE_MAX_USERS	0	Maximum number of named users that can be created in the database
LICENSE_SESSIONS_WARNING	0	Warning level for number of non-system user sessions
LM_LOCKS	12000	Number of locks configured for the lock manager (OPS)
LM_PROCS	64	Number of client processes configured for the lock manager (OPS)
LM_RESS	6000	Number of resources configured for the lock manager (OPS)
LOCAL_LISTENER	NULL	Local listener
LOCK_NAME_SPACE	NULL	Lock name space used for generating lock names for standby/clone database
LOCK_SGA	FALSE	Lock entire SGA in physical memory

Tablespace (Create, Alter, Drop)



Oracle divides a database into one or more logical storage units called tablespaces. Each tablespace consists of one or more files called datafiles. A datafile physically stores the data objects of the database such as tables and indexes on disk. In other words, Oracle logically stores data in the tablespaces and physically stores data in datafiles associated with the corresponding tablespaces. The picture illustrates the relationship between a database, tablespaces, and datafiles. The **CREATE TABLESPACE** statement allows you to create a new tablespace. The following illustrates how to create a new tablespace named tbs1 with size 1MB:

```
CREATE TABLESPACE tbs1
   DATAFILE 'tbs1_data.dbf'
   SIZE 1m;
```

In this statement: First, specify the name of the tablespace after the CREATE TABLESPACE keywords. In this example, the tablespace name is tbs1. Second, specify the path to the data file of the tablespace in the DATAFILE clause. In this case, it is tbs1.dbf. Note that you can use the datafile full path. Third, specify the size of the tablespace in the SIZE clause. In this example, 1m stands for 1MB, which is quite small.

The **DROP TABLESPACE** allows you to remove a tablespace from the database. Here is the basic syntax of the DROP TABLESPACE statement:

```
DROP TABLESPACE tablespace_name
    [INCLUDING CONTENTS [AND | KEEP] DATAFILES]
```

```
[CASCADE CONSTRAINTS];
```

In this syntax: First, specify the name of the tablespace that you want to drop after the DROP TABLESPACE keywords. Second, use the INCLUDE CONTENTS to delete all contents of the tablespace. If the tablespace has any objects, you must use this option to remove the tablespace. Any attempt to remove a tablespace that has objects without specifying the INCLUDING CONTENTS option will result in an error. Third, use AND DATAFILES option to instruct Oracle to delete the datafiles of the tablespace and KEEP DATAFILES option to leave the datafiles untouched. Fourth, if the tablespace that has objects such as tables whose primary keys are referenced by referential integrity constraints from tables outside the tablespace, you must use the CASCADE CONSTRAINTS option to drop these constraints. If you omit the CASCACDE CONSTRAINTS clause in such situations, Oracle returns an error and does not remove the tablespace.

```
DROP TABLESPACE tbs1;
```

The way to alter a tablespace is to add a new datafile by using the **ALTER TABLESPACE** statement:

```
ALTER TABLESPACE tablespace_name
    ADD DATAFILE 'path_to_datafile'
    SIZE size
    AUTOEXTEND ON;
```

If you use the AUTOEXTEND ON clause, Oracle will automatically extend the size of the datafile when needed. Use the ATLER TABLESPACE statement to add one more datafile whose size is 10MB with the AUTOEXTEND ON option:

```
ALTER TABLESPACE tbs10
    ADD DATAFILE 'tbs10_2.dbf'
    SIZE 10m
    AUTOEXTEND ON;
```

Rollback Segment (Create, After)

Each database has one or more rollback segments. A rollback segment is an Oracle database structure that stores undo information for transactions. Undo information is the original information that was changed during a transaction. It restores the changed database information back to what it was before a transaction changed it. When a transaction changes data, the Oracle server assigns the transaction to the next available segment and stores the undo information there. The undo information ensures multi-version read consistency, which means a query gathers all its data consistent to a point in time despite the fact that another session may be simultaneously making uncommitted changes to that data. While rollback segments store undo information to ensure multi-version read consistency, redo log groups protect rollback data and make it possible to reconstruct the entire database.

```
CREATE ROLLBACK SEGMENT rbs_one
  TABLESPACE rbs_ts
  STORAGE (INITIAL 10K NEXT 10K MAXEXTENTS UNLIMITED);
```

To alter Rollback Segment we can use following syntax:

```
ALTER ROLLBACK SEGMENT segment_name
  [STORAGE Storage_Clause]
  [ONLINE | OFFLINE]
  [SHRINK];
```

Import

The Import utility reads the object definitions and table data from an Export dump file. It inserts the data objects into an Oracle database. 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 specify all valid parameters and their values from the command line using the following syntax:

```
imp username/password PARAMETER=value
```

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

The number of parameters cannot exceed the maximum length of a command line on the system. The Import utility provides four modes of import.

- **Full**--Only users with the IMP_FULL_DATABASE role can import in this mode, which imports a full database export dump file. Use the FULL parameter to specify this mode.
- **Tablespace**--allows 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 (Owner)-**-allows you to 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 FROMUSER parameter to specify this mode.
- **Table**--allows you to 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.

Export

The Export utility provides 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 file. You can specify all valid parameters and their values from the command line using the following syntax:

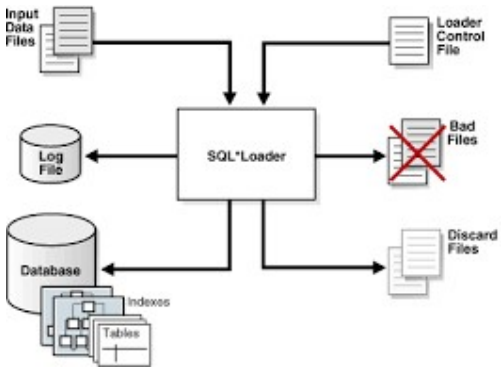
```
exp username/password PARAMETER=value
```

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

The number of parameters cannot exceed the maximum length of a command line on the system. The Export utility provides four modes of export: Full, User (Owner), Table, and Tablespace. To specify one of these modes, use the appropriate parameter (FULL, OWNER, TABLES, or TABLESPACES) when you invoke Export.

SQL* Loader

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

You execute the command **sqlldr** from the command line on Windows or Terminal on GNU/Linux:

```
sqlldr parfile=parameter_file.par
```

Managing Automated Database Maintenance Tasks: Oracle Database has automated several common maintenance tasks typically performed by database administrators. These automated maintenance tasks are performed when the system load is expected to be light. You can enable and disable individual maintenance tasks, and can configure when these tasks run and what resource allocations they are allotted.

Managing Resources with Oracle Database Resource Manager: Oracle Database Resource Manager (the Resource Manager) enables you to manage multiple workloads within a database that are contending for system and database resources. An overview of the Resource Manager like: Elements of the Resource Manager, The Types of Resources Managed by the Resource Manager, About Resource Manager Administration Privileges

What Solutions Does the Resource Manager Provide for Workload Management? When database resource allocation decisions are left to the operating system, you may encounter problems with workload management like: Excessive overhead, Excessive overhead results from operating system context switching between Oracle Database server processes when the number of server processes is high, Inefficient scheduling, The operating system reschedules database servers while they hold latches, which is inefficient, Inappropriate allocation of resources, The operating system distributes resources equally among all active processes and cannot prioritize one task over another, Inability to manage database-specific resources, such as parallel execution servers and active sessions.

The Resource Manager helps to overcome these problems by allowing the database more control over how hardware resources are allocated. In an environment with multiple concurrent user sessions that run jobs with differing priorities, all sessions should not be treated equally. The Resource Manager enables you to classify sessions into groups based on session attributes, and to then allocate resources to those groups in a way that optimizes hardware utilization for your application environment.

Oracle Scheduler Concepts: Oracle Database includes Oracle Scheduler, an enterprise job scheduler to help you simplify the scheduling of hundreds or even thousands of tasks. Oracle Scheduler (the Scheduler) is implemented by the procedures and functions in the DBMS_SCHEDULER PL/SQL package. The Scheduler enables you to control when and where various computing tasks take place in the enterprise environment. The Scheduler helps you effectively manage and plan these tasks. By ensuring that many routine computing tasks occur without manual intervention, you can lower operating costs, implement more reliable routines, minimize human error, and shorten the time windows needed. The Scheduler provides sophisticated, flexible enterprise scheduling functionality, which you can use to: Run database program units, Run external executables, (executables that are external to the database), Prioritize jobs based on business requirements, and Schedule job execution.

Scheduling Jobs with Oracle Scheduler: You operate Oracle Scheduler by creating and managing a set of Scheduler objects. Each Scheduler object is a complete database schema object of the form [schema.]name. Scheduler objects follow the naming rules for database objects exactly and share the SQL namespace with other database objects. Follow SQL naming rules to name Scheduler objects

in the DBMS_SCHEDULER package. By default, Scheduler object names are uppercase unless they are surrounded by double quotes. For example, when creating a job, job_name => 'my_job' is the same as job_name => 'My_Job' and job_name => 'MY_JOB', but different from job_name => '"my_job"'. These naming rules are also followed in those cases where comma-delimited lists of Scheduler object names are used within the DBMS_SCHEDULER package.

Administering Oracle Scheduler: You must have the SCHEDULER_ADMIN role to perform all Oracle Scheduler administration tasks. Typically, database administrators already have this role with the ADMIN option as part of the DBA role. For example, users SYS and SYSTEM are granted the DBA role. You can grant this role to another administrator by issuing the following statement:

```
GRANT SCHEDULER_ADMIN TO username;
```

Because the SCHEDULER_ADMIN role is a powerful role allowing a grantee to execute code as any user, you should consider granting individual Scheduler system privileges instead. Object and system privileges are granted using regular SQL grant syntax. An example is if the database administrator issues the following statement:

```
GRANT CREATE JOB TO scott;
```

After this statement is executed, scott can create jobs, schedules, programs, file watchers, and credentials in his schema. Another example is if the database administrator issues the following statement:

```
GRANT MANAGE SCHEDULER TO adam;
```

After this statement is executed, adam can create, alter, or drop windows, job classes, or window groups. He will also be able to set and retrieve Scheduler attributes and purge Scheduler logs.