



---

# RDBMS USING ORACLE

---

Paper Code: CS-15 (BCA)



PREPARED BY:

LT. M. J. KUNDALIYA ARTS, COMMERCE & COMPUTER SCIENCE MAHILA COLLEGE  
Rashtriya Shala Campus, Rajkot

---

**Table of Contents**

1.1 Introduction to DBMS .....	3
1.2 Introduction to RDBMS? .....	3
1.3 Dr.E.F.CODD Rules.....	3
1.4 Entity-Relationship Diagrams.....	5
1.5 Normalization.....	7
1.6 Introduction to Structured Query Language (SQL) .....	10
1.7 SQL Commands and Data types:.....	10
1.8 Introduction to SQL * Plus .....	12
1.9 SQL * Plus formatting commands.....	13
1.10 Operator and Expression .....	16
1.11 SQL v/s SQL * Plus .....	18
2.1 Creating, Altering, and Dropping tables .....	19
2.2 Data Manipulation Command like Insert, Update, Delete .....	21
2.3 Different types of constraints and applying of constration.....	22
2.4 Select statement .....	27
2.5 Join .....	30
2.6 SubQuery,minus,intersect,union .....	31
2.7 Numeric functions:.....	34
2.8 Character Function: .....	37
2.9 DATE FUNCTION:.....	38
2.10 AGGREGATE FUNCTION: .....	40
2.11 General Functions .....	41
2.12 Creating users and Role, Grant, Revoke .....	43
2.13 Grant, Revoke command .....	44
2.14 What is transaction? .....	46
2.15 Starting and Ending of Transaction.....	46
2.16 Commit, Rollback, Savepoint .....	47
3.1 View.....	48
3.2 Sequence:.....	49
3.3 Synonyms .....	52
3.4 Database Links: .....	52
3.5 Index:.....	53

---

## INDEX

3.6 Cluster .....	54
3.7 Snapshot .....	55
3.8 What Are Locks? .....	55
3.9 Locking Issue: .....	56
3.10 Lock Types .....	58
4.1 Introduction of PL/SQL:.....	62
4.2 PL/SQL Block Structure: .....	62
4.3 Language Construct Of PL/SQL: .....	64
4.4 %Type and %Rowtype .....	73
4.5 Using Cursor .....	74
4.6 Exception Handling: .....	76
4.7 Creating and using Procedure,Functions: .....	80
4.8 PACKAGE: .....	83
4.9 Triggers.....	85
4.10 Creating Objects.....	87
4.11 Collections.....	87
5.1 Instance Architecture (Database processes, Memory Structure, Data files) .....	92
5.2 Creating and Altering Database .....	95
5.3 Opening and Shutdown Database .....	95
5.4 Initialization Parameter.....	95
5.5 Control Files and Redo Log Files .....	96
5.6 Tablespace (Create, Alter, Drop) .....	97
5.7 Rollback Segments (Create, Alter) .....	98
5.8 Oracle Blocks.....	99
5.9 Import .....	99
5.10 Export.....	100
5.11 SQL *Loader: .....	101
5.12 Managing Automated Database Maintenance Task .....	102
5.13 Managing Resources with Oracle Database Resource Manager .....	103
5.14 Oracle Scheduler Concepts .....	104
5.15 Scheduling Jobs with Oracle Scheduler .....	106
5.16 Administering Oracle Scheduler .....	107

## Unit 1: DBMS Overview, SQL, SQL\*PLUS

---

### 1.1 Introduction to DBMS

#### ❖ What is Database?

- A database is a shared collection of logically related data in a systematic manner, that is stored to meet the requirements of different users of an organization, that can easily be accessed, managed and updated.

#### ❖ What is DBMS?

- Database + Management + System.
- A DBMS is system software for creating and managing databases.
- The DBMS provides users and programmers with a systematic way to create, retrieve, update and manage data.
- A DBMS makes it possible for end users to create, read, update and delete data in a database.

### 1.2 Introduction to RDBMS?

- Stands for Relational Database Management System.
- RDBMS is a Database Management System that is based on the relational model as introduced by E.F.Codd.
- RDBMS is the basis for SQL, and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.

#### ❖ Difference Between DBMS and RDBMS:

DBMS	RDBMS
DBMS application store data as file.	RDBMS applications store data in a tabular form.
Speed of operation is very slow.	Speed of operation is very fast.
Hardware and software requirements are less.	Hardware and software requirements are high.
Facilities and Utilities offered are limited.	Facilities and Utilities offered are many.
Platform is used is normally DOS.	Platform used can be any DOS, UNIX, VAX, VMS etc.
Examples are Dbase, FOXBASE etc.	Examples are ORACLE, INGRES etc.
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.
Normalization is not present in DBMS.	Normalization is present in RDBMS.
DBMS uses file System to store data, so there will be no relation between the tables.	RDBMS 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.

### 1.3 Dr.E.F.CODD Rules

Dr. Edgar Frank Codd is the Father of Database Management System. In 1985 Dr. E.F. Codd gives 12 Rules for relational database model. A relational database management system uses only its relational capabilities to manage the information stored in its database.

## **Unit 1: DBMS Overview, SQL, SQL\*PLUS**

---

### **1.Information Rule**

All information stored in a relational database is represented only by data item values, which are stored in the tables that make up the database. Associations between data items are not logically represented in any other way, such as, by the use of pointers from one table to the other.

### **2.Guaranteed Access Rule**

Every data item value stored in a relational database is accessible by stating the name of the table it is stored in, the name of the column under which it is stored and the value of the primary key that defines the row in which it is stored.

### **3.Systematic treatment of null values**

The database management system has a consistent method for representing null values. For example, null values for numeric data must be distinct from zero or any other numeric value and for character data it must be different from a string of blanks or any other character value.

### **4.Dynamic Online catalog based on the Relational Model**

The logical description of a relational database is represented in the same manner as ordinary data. This is done so that the facilities of the relational database management system itself can be used to maintain database description.

### **5.Comprehensive Data Sublanguage Rule**

A relational database management system may support many types of languages for describing data and accessing the database. However, there must be at least one language that uses ordinary character strings to support the definition of data, the definition of views, the manipulation of data, constraints on data integrity, information concerning authorization and the boundaries for recovery of units.

### **6.View Updatibility**

Any view that can be defined combinations of base tables, which are theoretically updateable, is capable of being updated by the relational database management system.

### **7.Insert, Update and Delete**

Any operand that describes the results of a single retrieval operation is capable of being applied to an insert, update or delete operation as well.

### **8.Physical Data independence**

Changes made to physical storage representations or access methods do not require changes to be made to application programs.

### **9.Logical data independence**

Changes made to tables, that do not modify any data stored in that table, do not require changes to be made to application programs.

### 10.Integrity Constraints

Constraints that apply to entity integrity and referential integrity are specifiable by the data language implemented by the database management system and not by the statements coded into the applications program.

### 11.Database Distribution


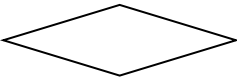
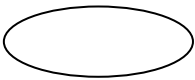

The data language implemented by the relational database management system supports the ability to distribute the database without requiring changes to be made to application programs. This facility must be provided in the data language, Whether or not the database management system itself supports distributed databases.

### 12.Non-Subversion

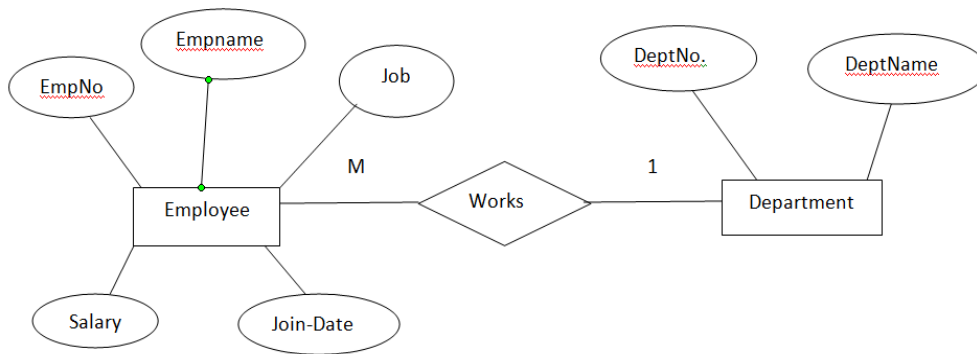
If the relational database management system supports facilities that allow application programs to operate on the tables a row at a time, an application program using this type of database access is prevented from bypassing entity integrity or referential integrity constraints that are defined for the database.

### 1.4 Entity-Relationship Diagrams

Diagrams are one of the better ways to communicate different of a components of a system. They are also too easy to understand by everyone. They offer an overview of the entire system. An E-R diagram is graphical method of representing entity classes, attributes and relationships. An E-R diagram uses six basic symbols:

	A rectangle to denote an entity or entity set.
	A diamond to denote a relationship between two entities.
	An oval to denote attributes.
	A line which links attributes to an entity or entity set and entity sets to relationships.
<b>1</b>	A '1' to denote a single occurrence.A '1' to denote a single occurrence.
<b>M</b>	An 'M' to denote multiple occurrences.

When an E-R diagram is built. The first step is defining entities. The next step is to define the relationship between the entities. The final step to identify the attributes that belong to each entity.Once the E-R diagram is completed. The entities will become the files (or table). Figure illustrates a many-to-one relationship between the entity sets Employee and Department. The next process is that of normalization. Which will be covered in detail in the next topic.

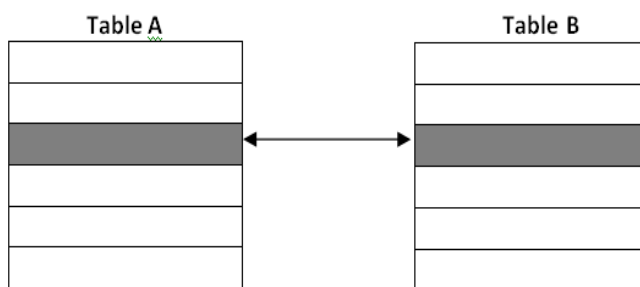


### There are 3 types of relationship

- 1) One to One [1:1]
- 2) One to Many [1:M]
- 3) Many to Many [M:M]

#### 1).One to One [1:1]

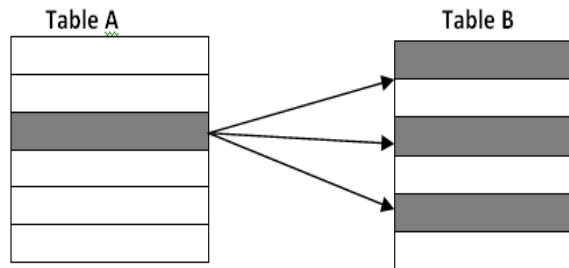
A pair of tables bears a one-to-one relationship when a single record in the first table is related to only one record in the second table, and a single record in the second table is related to only one record in the first table.



As you can see, a single record in TABLE A is related to only one record in TABLE B, and a single record in TABLE B is related to only one record in TABLE A. A one-to-one relationship usually (but not always) involves a subset table.

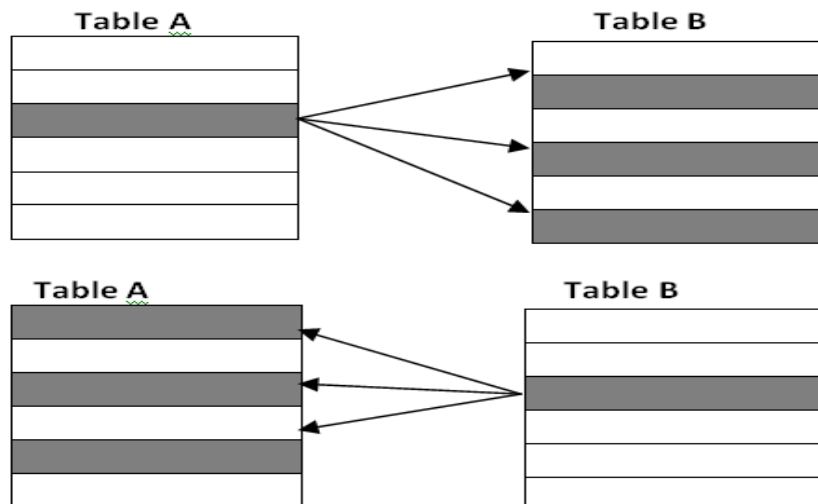
#### 2).One to Many [1:M]

A one-to-many relationship exists between a pair of tables when a single record in the first table can be related to one or more records in the second table, but a single record in the second table can be related to only one record in the first table. Let's look at a generic example of this type of relationship. Say you're working with two tables, TABLE A and TABLE B, that have a one-to-many relationship between them. Because of the relationship, a single record in TABLE A can be related to one or more records in TABLE B.



### 3). Many to Many [M:M]

A pair of tables bears a many-to-many relationship when a single record in the first table can be related to one or more records in the second table and a single record in the second table can be related to one or more records in the first table. Assume once again that you're working with TABLE A and TABLE B and that there is a many-to-many relationship between them. Because of the relationship, a single record in TABLE A can be related to one or more records (but not necessarily all) in TABLE B. Conversely, a single record in the TABLE B can be related to one or more records (but not necessarily all) in TABLE A.



### 1.5 Normalization

It is one of the most important concepts in the study of the RDBMS. The case with which information is stored and retrieved depends a lot on the way the tables have been defined. Tables that are haggard and bulky often defeat the purpose of having an RDBMS all together. This is because such tables may not be easy to maintain.

In this section, we will discuss the concept of normalization in detail. It can be defined as a process of putting data right-making it normal. Normalization is important from the database design designer's point of view as it enables him to design better. It is concerned with database design. There are two ways of approaching logical database design-

The top down approach

The bottom-up approach

In the top down approach, first entities and relationships are identified; the ER diagram is made and mapped with the tables. The ER modeling technique uses the top down approach. Normalization



## Unit 1: DBMS Overview, SQL, SQL\*PLUS

---

uses the bottom up approach. Normalization is the technique that makes the relational data files differ from other data files, which are referred to as flat files we will understand this with the help of an example.

A library maintains a register of all books issued to its members. The register contain the following column

No.	Name of the Book	Borrowed by	Date of issue	Date of return

Every time a member failed to return the book, a letter is sent a telephone call is made to that member's house for reminding. In separate part of the register. A list of all members name along with their address and telephone numbers is maintain.

Member name	Address and telephone number.

Now every time a member defaults in returning a book, the librarian looks of the name of that member in the list and makes a reminder call. In this case we can say that the librarian has a relational database on paper. This is called a normalized data. Normalization is here referring to data being collect and stored in natural grouping. Issue detail of book and member details are stored as separate groups or lists.

Like this library register even in RDBMS, it is imperative to have normalized table. Normalization can be defined as the process of the restructuring a relation (table). For reducing it to a form where each domain would consist of single non composite values

### [2] Benefits of normalization

Normalization reduces repetition for example data redundancy. When the same data is repeatedly stored, it causes storage and access problems.

- Inconsistency in data retrieval
- Error while updating data tables

Suppose the address and the telephone nos. of the members were grouped together in one table along with book details. How would it make difference? First let take a look at the table below:

No.	Name of the Book	Borrowed by	Address	Telephone	Issue Date	Return Date

As you can see there are no groups everything is put in one table now consider a member who borrow above four books every weeks, it's now easy to see how many times the address and telephone no of this member will be stored. In the table leading to a huge amount of data. Suppose one member is changed his address. The no. of rows were the address will have to be changed will

## Unit 1: DBMS Overview, SQL, SQL\*PLUS

---

be many. This duplication of effort due to poor database design. As it can be seen data that has not been normalized can lead to several problems two of which we have discussed.

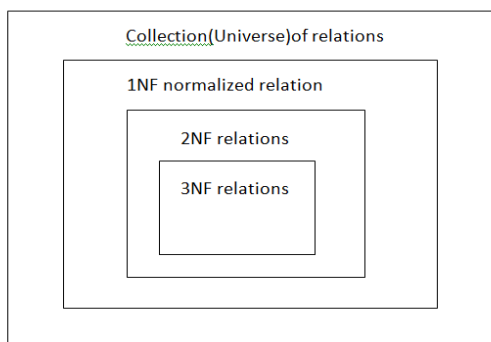
Having understood normalization and its benefits let's proceed what happens after a table is normalized.

### Normal Form

Dr. Codd originally defines three levels of normalization. These three levels were called first normal form. Second normal form and third normal form respectively. Normalization is usually discussed in terms of forms. Normal forms are table structures with minimum redundancy. Normal forms that have been identified are:

- First normal form (1st NF)
- Second normal form (2nd NF)
- Third normal form (3rd NF)
- Boyce-Codd normal form

The first three forms were defined by Dr. Codd. Later Dr. Codd and Boyce introduced one more normal form, which called the Boyce-Codd normal form. Theory of normalization is based on the concept of function dependency. The diagram below illustrates the levels of normalization.



In order to understand more about normal form we must have understood what is meant by functional dependency.

Establish deletion rules

Data integrity is one of the main advantages that relationships offer in an RDBMS. Deletion rules must be established for a relationship. This rule states what will happen if a record has to be deleted. Defining a deletion rule prevents records from being orphaned; i.e. the record will exist in a subordinate table but will not exist in the main table. There can be two options applicable for deletion rule.

**Restrict:** when deletion is restricted, a record in a subordinate table of a one to one or many to one relationship cannot be deleted for example a publisher whose books are still found in the books table cannot be deleted. There are several examples where significance of such a rule can be appreciated further.

In an employ table, an employee cannot be deleted because a salary table still holds his record. In a pending order table, pending order cannot be deleted until the order has been serviced. Customer who has not paid these dues cannot be deleted from the customer table.

## Unit 1: DBMS Overview, SQL, SQL\*PLUS

---

**Cascade:** In this type of rule when a record is deleted, all related records in all subordinate tables will also be deleted. In the first session we had talk about RDBMS maintaining data integrity. This is how it is possible.

### 1.6 Introduction to Structured Query Language (SQL)

SQL is Non Procedural Language in the conventional programming language, coding is essential to achieve a given task. SQL is a language that provides an interface to relational database system. SQL was developed by IBM in the 1970. SQL can be used by a range of users, including those with little or no programming experience. It reduces the amount of time required for creating and maintaining systems. It is an English-like language.

In SQL only the task that has to be achieved needs to specify. For example, to retrieve rows from a table we simply use the SELECT command. A Data Sub language SQL does not support programming language constructs. Conditional statements like 'If Then', 'While' cannot be used in SQL. Not a database management system SQL is an important tool for communication with the DBMS and supports database management statements.

**SQL - A Brief History:** SQL was first introduced by Dr. E. F. Codd in his pioneering work 'A relational Model for Large Shared Data Banks'. Since introduction many researchers at the San Jose Research Laboratories made efforts in implementing Dr. Codd's ideas, the mid of 70s saw the development of many computer programming language based on this relational model. One of these was called Structure English Query Language of SEQUEL language.

#### SQL

- ✓ SQL stands for Structured Query Language.
- ✓ SQL is used to communicate with the database.
- ✓ SQL provides interface to relational database system.
- ✓ SQL is a standard language for Relational Database Management System (RDBMS).
- ✓ SQL is used to perform tasks such as update, insert, delete or retrieve data from database.

#### Features of SQL

- ✓ SQL can be used by the users, including those with little or no programming experience.
- ✓ It is a non procedural language.
- ✓ It reduces amount of time required for creating and maintaining system.

#### Rules for SQL

- ✓ A ';' is used at the end of the sql statements.
- ✓ Statements may be split across lines but Keywords may not.
- ✓ '/\* \*/' is used as a multi lines comments and '-' is used for single line comments.

### 1.7 SQL Commands and Data types:

- DDL (Data Definition Language)
  - ✓ CREATE, ALTER, DROP, TRUNCATE
- DML (Data Manipulation Language)
  - ✓ INSERT, UPDATE, DELETE, LOCK
- DQL (Data Query Language)
  - ✓ SELECT

## Unit 1: DBMS Overview, SQL, SQL\*PLUS

---

- TCL (Transaction Control Language)
  - ✓ COMMIT, ROLLBACK, SAVEPOINT
- DCL (Decision Control Language)
  - ✓ GRANT, REVOKE

### DDL:

- It contains commands like CREATE, ALTER, DROP, TRUNCATE, GRANT, REVOKE.
- **CREATE:** It is used create table, views, procedure, trigger etc.
- **ALTER:** It is used to change the structure of the Table.
- **DROP:** It is used to delete objects from the database.
- **TRUNCATE:** It is used to remove all the records from the table, including space that was allocated.
- **GRANT:** It is used to give different command grant (access right) to the user.
- **REVOKE:** It is use to take given grant back from the user.

#### 1) DML:

- ✓ It contains commands like Insert, update, delete, call.
- ✓ **INSERT:** Use to insert data into a table.
- ✓ **UPDATE:** It is use to update existing data within the table.
- ✓ **DELETE:** It is use to delete all the records from the table, the space will be remaining.
- ✓ **CALL:** It is used to call a PL/SQL program.

#### 2) DCL:

- ✓ It contains commands like Commit, Savepoint, and Rollback.
- ✓ **COMMIT:** It is used to save the work.
- ✓ **SAVEPOINT:** It is used to identify a point in transaction.
- ✓ **ROLLBACK:** Restore the database to original since the last commit (like UNDO).

#### 3) DQL:

- ✓ DQL contains Select Statements.
- ✓ **SELECT:** It is used to retrieve data from the database.

### Datatypes:

#### [1] Char (Size)

This data type is used to store alphanumeric data. The size given in bracket determines the number of characters the cell can hold. Storage length is between 1 and 255 characters. Spaces are padded to the right of the value to supplement the total allocated length of the column. I.e. if a value that is inserted in a cell of CHAR data type is shorter than the size it is defined for then it will be padded with spaces on the right until it reaches the size characters in length.

#### [2] Varchar (Size)/ Varchar2 (Size)

This data type is used to store variable length alphanumeric data. The maximum this data type can hold is 2000 characters. One difference between this data type and the CHAR data type is ORACLE compares VARCHAR values using non-padded comparison semantics i.e. the inserted values will not be padded with spaces.

### [3] Date

Date data type stores date as well as time data. The standard format is DD-MMM-YY i.e. 20-Dec-07 use to represent 20/12/07. Date Time stores date in the 24-hour format. By default, the time in a date field is 12:00:00 am, if no time portion is specified. The default date for a date field is the first day of the current month. Valid range is 1-January-4712 B.C. to 31-December-4712 A.D.

### [4] Number (P, S)

The NUMBER data type is used to fixed or floating-point positive or negative values. Numbers as large as  $9.99 \times 10^{124}$  i.e. 1 followed by 125 zeros can be stored. The precision (P) determines the maximum length of the data, whereas the scale (S) determines the number of places to the right of the decimal. If scale is omitted then the default is zero. The maximum precision is 38 digits.

### [5] Long

This data type is used to store alphanumeric data up to 2GB. LONG data can be used to store array of binary data in ASCII format. LONG values cannot be indexed, and the normal character functions such as SUBSTR can't be applied to LONG values.

### [6] Long Varchar: Same as Long

### [7] Raw/ Long Raw

The long raw is also known as Binary Large Object (BLOB) in other database management systems. It is typically used to store graphic, sound or video data. LONG RAW data type can contain up to 2GB. Values stored in columns having LONG RAW data type cannot be indexed.

**[8] BLOB:** A binary large Object with a limit of 4GB in length

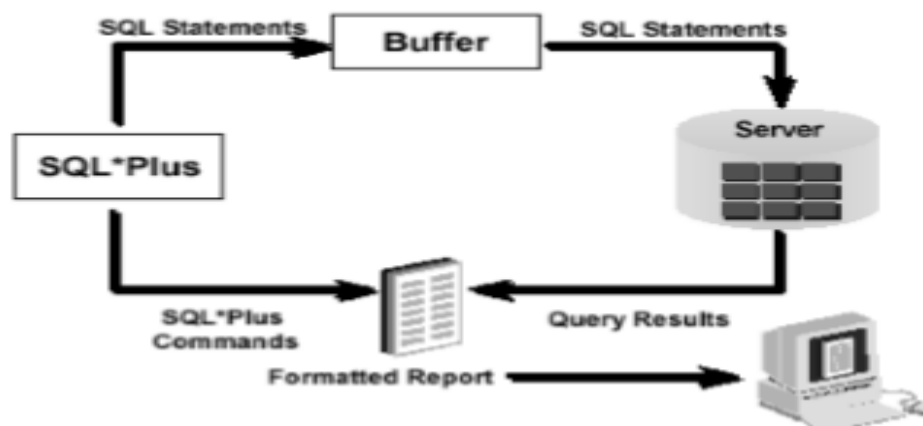
**[9] CLOB:** A Character large Object with a limit of 4 GB in length.

## 1.8 Introduction to SQL \* Plus

SQL\*Plus (pronounced "sequel plus") is an interactive tool for the Oracle RDBMS environment. SQL\*Plus can be used simply to process SQL statements one at a time, process SQL statements interactively with end users, utilize PL/SQL for procedural processing of SQL statements, list and print query results, format query results into reports, describe the contents of a given table, and copy data between databases.

User type in SQL statements in SQL\*Plus that send statements to Oracle server. Oracle server then validates and executes the

statements on its databases. The query results are returned to SQL\*Plus and displayed to the user. Besides sending SQL statements to the server, SQL\*Plus also saves them into a local buffer and allow users to view and change the statements.



After login into SQL\*Plus, at the SQL prompt, type any SQL command. By hitting the ENTER key the SQL prompt will change to line number prompts. When one finished typing a command, type / (slash) or ; (semicolon) at the end of the SQL command. This will execute the command immediately after hitting ENTER key.

### 1.9 SQL \* Plus formatting commands

The SQL\*Plus formatting commands are used to manipulate the result set from a SQL query.

**TTITLE:1)** Ttitle on/off

2) Ttitle <Align><Print> [Skip n]

TTITLE is used to place text at the top of each page. There are various print options that position text at various locations. TTITLE will center the text and add date and page numbers if no print options are specified. print options include BOLD, CENTER, COL, FORMAT, LEFT, RIGHT, SKIP, and TAB. TTITLE with no options at all will display the current text setting. Other options that can be specified are ON and OFF. TTITLE is ON by default.

**BTITLE:** 1) Btitle on/off

2) Btitle <Align><Print> [Skip n]

BTITLE is used to place text at the bottom of each page. There are various print options that position text at various locations. BTITLE will simply center the text if no print options are specified. Print options include BOLD, CENTER, COL, FORMAT, LEFT, RIGHT, SKIP, and TAB. BTITLE spelled out by itself, it will display the current text setting. Other options that can be specified are ON and OFF. BTITLE is ON by default.

**COLUMN:** Column <name> Heading <heading> Format <format>

COLUMN is used to alter the default display attributes for a given column (column name) of a SQL query. There are a variety of options, but the more common ones are FORMAT, HEADING, JUSTIFY, NEWLINE. The FORMAT option is useful in applying editing to numeric fields, date masks to date fields, and specific lengths to variable-length character fields. The HEADING option overrides the SQL\*Plus default heading for the particular column. The JUSTIFY option overrides the SQL\*Plus column alignment to the heading default. The NEWLINE option will print the column on the beginning of the next line.

**BREAK ON:** Break on <element> [action]

## Unit 1: DBMS Overview, SQL, SQL\*PLUS

---

Element: Column/Row/Report

Action: Skip n/Skip Page/ Duplicate / No Duplicate

This command controls the organization of rows returned by the query. BREAK can manipulate the appearance of the output by specifying under what conditions a BREAK should occur and what actions should be taken at the BREAK. The appearance of the output can be controlled by skipping a line or skipping to top of next page and providing totals when used in conjunction with COMPUTE. Any number of lines can be skipped at a BREAK point. BREAK points can be defined at the column level, for multiple columns, on a row, on a page, or on a report. See the COMPUTE command for BREAK examples. Entering BREAK by itself at the SQL prompt will display the current BREAK settings.

**COMPUTE:** Compute <function> OF <column> ON <element>

Element: Column / Row / Report

Function: Avg / Count / Max / Min / Sum

COMPUTE calculates and prints totals for groupings of rows defined by the BREAK command. A variety of standard functions can be utilized. The most common option is the name of the column in the query on which the total is to be calculated. The break option determines where the totals are to be printed and reset, as defined by the BREAK command.

**CLEAR** and options CLEAR resets any of the SQL\*Plus formatting commands. You can also use it to clear the screen. The options include BREAKS, BUFFER, COLUMNS, COMPUTES, SCREEN, SQL, and TIMING.

Example\_1: COLUMN sal FORMAT \$99,999.00 HEADING Salary; SELECT ENAME,SAL FROM EMP;

Example\_2: BREAK ON sales\_rep SKIP 2 BREAK ON REPORT COMPUTE SUM OF monthly\_sales ON sales\_rep COMPUTE SUM OF commissions ON sales\_rep COMPUTE SUM OF monthly\_sales ON REPORT COMPUTE SUM OF commissions ON REPORT Clear Buffer **LIST:** The LIST command lists the most recently executed SQL statement in the buffer.the will simply be the displayed statement. Example: SQL>list o/p: select \* from stud;

**L command** You can move to specific line from the buffer line from the buffer by placing a line number after the l;

Example: SQL>l2

### Miscellaneous Commands

This section presents a variety of commands that enable you to interact with the user, comment on the code, and enhance coding options.

#### ACCEPT variable number or char PROMPT text

ACCEPT receives input from the terminal and places the contents in variable. This variable can already have been defined with the DEFINE command. If the PROMPT option is specified, then the text will be displayed after skipping a line. The variable attributes of number or char can be defined at this time. The variable will be a char if not otherwise defined.

### **DEFINE variable**

DEFINE creates a user-defined variable and assigns it to be of char (character) format. This variable can be assigned a default value at this time.

### **DESC or DESCRIBE database\_object**

DESCRIBE displays the columns associated with a table, view, or synonym.

### **PAUSE text**

PAUSE prints the contents of text after skipping a line, and then waits for the Return or Enter key to be pressed.

### **PROMPT text**

PROMPT simply skips a line and prints the contents of text.

### **REM or REMARK**

SQL \*Plus will ignore the contents of this line when it is used in SQL \*Plus command files. REMARK enables documentation or other comments to be contained in these SQL \*Plus command files.

### **SET SQL\*Plus System Variable**

The SET command controls the default settings for the SQL\*Plus environment. You can automatically alter these settings for each SQL\*Plus session by including them in the LOGIN.SQL file, discussed earlier in this chapter. The following are some common SET options that suppress various SQL \*Plus output:

**SET SERVEROUTPUT OFF/ON:** Whether to display the output of stored procedures (or PL/SQL blocks). By default OFF.

#### **Example:**

```
SQL> SET SERVEROUTPUT ON;
```

### **SET AUTOCOMMIT {OFF|ON}:**

Commits after each SQL command or PL/SQL block automatically. By Default OFF.

#### **Example:**

```
SQL> SET AUTOCOMMIT ON;
```

**SET COLSEP {| text}:** The text to be printed between columns retrieved by SELECT statement. Columns normally separated by a space.

#### **Example:**

```
SQL> SET COLSEP "|";
```

```
SQL > SELECT * FROM EMP;
```

Above command will display "|" sign as column separator.

**SET FEEDBACK OFF/OFF:**Suppresses the number of query rows returned.

**SET VERIFY OFF/OFF:**Suppresses the attribution text when using variables, including command line variables.



## Unit 1: DBMS Overview, SQL, SQL\*PLUS

---

**SET TERMOUT OFF/ON:** Suppresses all terminal output, this is particularly useful in conjunction with the SPOOL command.

**SET ECHO on/off:** Suppress the display of SQL \*Plus commands.

**SET HEADING on/off:** Set printing of column heading on or off in report.

**SET WRAP on/off:** If it is OFF, it truncates the display of a SELECT row in case of too long for line width printing of column heading on or off in report.

**SET SQLPROMPT SQL/Text:** Sets the SQL\*Plus command prompt.

**SET PAGESIZE n:** Set the number of lines per page.

**SET LINESIZE n:** Set the width of the output report line. By default is 80.

**SPOOL:**

- 1) Spool <filename>
- 2) Spool off
- 3) Spool out

The SPOOL command is used to open, close, or print an operating system-dependent file. Specifying SPOOL filename will create an operating system-dependent file; filename can contain the full pathname of the file and extension. If no file extension is given, the file suffix, LST, will be appended (filename.LST). Options include OFF or OUT. If OFF is specified, then the operating system-dependent file is simply closed. If OUT is specified, then the operating system-dependent file is closed and sent to the operating system-dependent printer assigned as the default printer to the user's operating system environment. Example: Simple SQL\*Plus report code.

```
SET FEEDBACK OFF
```

```
SET VERIFY OFF
```

```
SET TERMOUT OFF
```

```
TTITLE 'Salary Information'
```

```
COLUMN SAL FORMAT 9, 99,999.99 HEADING "BASIC SALARY" COLUMN ENAME  
FORMAT A12 HEADING "EMPLOYEE NAME" SPOOL REPO.TXT
```

```
SELECT ENAME, SAL FROM EMP;
```

```
SPOOL OFF;
```

### 1.10 Operator and Expression

**OPERATORS:**

Arithmetic: + - \* / %

Logical: AND ,OR, NOT

Like: LIKE

Relational: ><>= <= = != <>

Miscellaneous: BETWEEN IS IN ANY ALL THE

### Logical Operator “AND”:

The Oracle engine will display only those rows of a table which satisfies all the condition specified in 'where' clause using AND operator.

Example:

```
SELECT * FROM student WHERE course = 'BCA' and SEM='3';
```

Above example will retrieve all the row of student table whose course is BCA and semester is 3.

### Logical Operator “OR”:

The Oracle engine will display only those rows of a table which satisfies at least one condition specified in 'where' clause using OR operator.

Example:

```
SELECT * FROM EMP WHERE city = 'bombay' or city = 'pune';
```

Above example will retrieve all the rows of EMP table whose city is either Bombay or pune.

### NOT Operator:

Operation will be TRUE when condition becomes FALSE.

Example:

```
SELECT * FROM EMP WHERE NOT (city = 'bombay' or city = 'pune');
```

Above example will display all the rows of EMP table that are NOT in Bombay or pune.

### BETWEEN Operator:

To select the data within range, BETWEEN operators is used.

Example:

```
SELECT * FROM empdet WHERE salary BETWEEN 2000 AND 7000;
```

The above statement will display all the records whose salary is between 2000 and 7000. This will include both the limit value.

### IN and NOT IN:

Operator = compares a single value to another single value.

If a value needs to compare with list of values IN is used.

By using IN, comparison becomes easy compare to doing with OR operator.

Example:

```
SELECT * FROM EMP WHERE city IN ('rajkot', 'bombay', 'pune');
```

Above example will return all the rows of EMP table whose city is either Rajkot or Bombay or pune.

Example:

```
SELECT * FROM EMP WHERE city NOT IN ('rajkot', 'bombay', 'pune');
```

Above example will return all the rows of EMP table whose city is other than Rajkot, Bombay and pune.

## Unit 1: DBMS Overview, SQL, SQL\*PLUS

---

### EXPRESSION:

The definition of an expression is simple: An expression returns a value. Expression types are very broad, covering different data types such as String, Numeric, and Boolean. In fact, pretty much anything following a clause (SELECT or FROM, for example) is an expression. In the following example amount is an expression that returns the value contained in the amount column.

Example: Select amount from emp;

### 1.11SQL v/s SQL \* Plus

SQL	SQL * Plus
It is a language for communication with the Oracle Server to access data.	It recognizes SQL statement and sends them to server.
Does not have a continuation character. Cannot be abbreviated.	Use a dash (-) as continuation character. Can be abbreviated.
Manipulate data and table definition in database.	Does not allow manipulation of values in the database.
It is based on American National Standards Institute (ANSI).	It is Oracle proprietary interface for executing SQL's statement.
Use functions to perform some formatting.	It uses commands to format data.
It uses a termination character to execute command immediately.	It does not require termination characters to execute command immediately.
SQL is a language SQL *Plus is an environment.	SQL is a language SQL *Plus is an environment.

### 2.1 Creating, Altering, and Dropping tables

#### 2.1.1 Create Table:

The CREATE TABLE statement allows you to create and define a table.

##### Syntax:

```
CREATE TABLE table_name  
(  
    Column 1 data type(size),  
    column 2 data type(size),  
    ... Column data type (size)  
);
```

Each column must have a data type. Column name must not contain space or any special character. It can contain underscore (\_).

Example:

```
Create table student  
(  
    studno number(2),  
    studnm varchar2(20),  
    sem number(1),  
    course varchar2(6),  
    join_dt date  
);
```

#### Creating table from existing table:

**Syntax 1:** Copying all columns from another table

```
CREATE TABLE new_table  
AS (SELECT * FROM old_table);
```

Example:

```
CREATE TABLE suppliers  
AS (SELECT * FROM companies);
```

**Syntax 2:** Copying selected columns from another table

```
CREATE TABLE new_table  
AS (SELECT column1, column2, ... columnn FROM old_table);
```

Example:

```
CREATE TABLE suppliers
```

## Unit-2: Managing Tables and data, Data Control And Transaction Control Commands

---

AS (SELECT id, address, city, state, zip FROM companies);

**Syntax 3:** Copying selected columns from multiple tables

CREATE TABLE new\_table

AS (SELECT column1, column2, ... columnn FROM oldtable1, oldtable2, ... oldtablen  
WHERE column=expression);

Example:

CREATE TABLE suppliers AS (SELECT companies.id, companies.address, categories.cat\_type  
FROM companies, Categories WHERE companies.id = categories.id);

### 2.1.2 Alter Table:

**Syntax 1:** Adding new column in the table.

ALTER TABLE table-name

ADD (newcolumnname datatype(size), newcolumnname datatype(size), ...);

Example:

ALTER TABLE student ADD (result varchar2 (20));

**Syntax 2:** Modifying existing column

ALTER TABLE table-name

MODIFY (columnname newdatatype(newsize));

Example:

ALTER TABLE student MODIFY (course varchar2 (7));

### Restriction on ALTER TABLE:

1. One cannot change the name of the table.
2. Once cannot change the name of the column.
3. One cannot drop (remove) the column.

**Note:** Dropping a column is possible in Oracle 8i onwards versions of oracle. Syntax is ALTER TABLE table-name DROP COLUMN column name.

4. One cannot decrease the size of a column if table data exists.
5. One cannot change the data type of a column if table data exists.

### 1.2.3 Drop Table:

Drop Table <Table\_Name>;

Example:

Drop table student;

### 2.2 Data Manipulation Command like Insert, Update, Delete

#### 2.2.1 Insert:

The INSERT INTO statement allows you to insert a single record or multiple records into a table.

**Syntax 1:** Inserting value in all the fields of table

```
INSERT INTO table-name  
VALUES (value-1, value-2, ... value-n);
```

Example:

```
Insert into student values (11,'Palak',1,'PGDCA','01-Jun-2008');
```

**Syntax 2:** Inserting values in all the fields of table using variable.

```
INSERT INTO table-name  
VALUES (&variable1, &variable2, ..., &variableN);
```

Example:

```
INSERT INTO student  
VALUES(&student_no,'&student_nm',&semester,'&course','&Joining_date');
```

#### Rules:

1. Variable name can be same as column name of the table or it can be different.
2. Space is not allowed in variable name.
3. Variable that represents Varchar2, Char and Date type of columns, must be written in single bracket ( ' ' ).

**Syntax 3:** Inserting values in selected fields.

```
INSERT INTO table-name (column name1, column name2, column name3)  
VALUES (value1, value2, value3);
```

Example:

```
INSERT INTO student (studno,studnm,course)  
VALUES (10,'Mishti','BCA');
```

**Syntax 4:** Inserting values in selected fields using variable.

```
INSERT INTO table-name (column name1, column name2, column name3)  
VALUES (&variable1, &variable2, &variable3);
```

Example:

```
INSERT INTO student (studno,studnm,course)  
VALUES (&student_no,;&student_name','&course');
```

**Syntax 5:** Insertion of a data set into a table from another table.

```
INSERT INTO table-name (column name1, column name2, ... , column namen) SELECT  
statement;
```

Example:

```
INSERT INTO suppliers (supplier_id, supplier_name)  
SELECT account_no, name FROM customers;
```

#### 2.2.2 Update:

- ✓ The UPDATE command is used to change or modify data values in a table.
- ✓ UPDATE statement can update all the rows from a table or selected rows from the table.

## Unit-2: Managing Tables and data, Data Control And Transaction Control Commands

---

**Syntax 1:** Update all rows from table.

UPDATE table-name SET column name=expression, column name=expression,...

**Example:**

UPDATE student SET sem=2;

**Syntax 2:** Update selected rows from table.

UPDATE table-name SET =expression, =expression, ... WHERE =expression;

**Example:**

UPDATE student SET studnm='Meeta' WHERE studnm='Mita';

### 2.2.3 Delete:

**Syntax 1:** Remove all rows

DELETE FROM table-name;

**Example:**

DELETE FROM student;

**Syntax 2:** Remove selected rows

DELETE FROM table name WHERE column name=expression;

**Example:**

DELETE FROM student WHERE studno>60;

### 2.2.4 Truncating table:

- TRUNCATE TABLE removes all rows from a table
- TRUNCATE TABLE is functionally identical to DELETE statement with no WHERE clause but
- TRUNCATE TABLE is faster and uses fewer system and transaction log resources than DELETE.

**Syntax:**

TRUNCATE TABLE table-name;

**Example:**

TRUNCATE TABLE student;

## 2.3 Different types of constraints and applying of constration

### 2.4.1 What is Data Constraint?

An Integrity constraint is a mechanism used by Oracle to prevent invalid data entry into the table. It is nothing but enforcing rule for the columns in a table. This technique ensures that the data that is stored in the database will be valid and has integrity. If a single column of the record being entered into the table fails a constraint, the entire record is rejected and not stored in the table. Constraints can be connected to a column or a table by CREATE TABLE or ALTER TABLE command.

Oracle allows defining constraint at

- i. Column Level
- ii. Table Level

#### i. Column Level constraint

When data constraints are defined along with the column definition while creating or altering table, they are known as column level constraints.

### ii. Table Level constraint

When data constraints are defined after defining all the columns of table while creating or altering table, they are known as table level constraints.

### 2.4.2 TYPES OF DATA CONSTRAINTS

There are two types of data constraints

1. I/O Constraints
2. Business Rule Constraints

#### 1. I/O Constraints:

The Input/output (I/O) constraints are further divided into two different constraints.

- a. Primary Key constraint
- b. Foreign Key Constraint

Oracle also has NOT NULL & UNIQUE constraints.

#### 2. Business Rule Constraints:

Business rule can be implemented in Oracle using CHECK constraint. Business Managers determine business rules.

Hence,

Primary Key

Foreign Key

Not Null

Unique

Check ... are all Constraints.

### PRIMARY KEY

- ✓ A primary key is one or more column in a table used to identify each row uniquely in table.
- ✓ Once the column has defined Primary Key constraint, it gets NOT NULL and UNIQUE constraints.
- ✓

#### PRIMARY KEY constraint at column level:

##### Syntax:

Column name data type (size) PRIMARY KEY,

##### Example:

```
CREATE TABLE emp
(
    empno varchar2(4) PRIMARY KEY,
    empnm varchar2(15),
    address1 varchar2 (20),
    address2 varchar2 (20),
    City varchar2 (15)
);
```

- ✓ Now, after creating table, empno column must not be NULL and value entered in empno column must be UNIQUE means one cannot repeat the value of empno column.

#### PRIMARY KEY constraint at table level:



## Unit-2: Managing Tables and data, Data Control And Transaction Control Commands

---

### Syntax:

PRIMARY KEY ( [, , ... ])

### Example:

```
CREATE TABLE empdept  
(  
    empno varchar2(4),  
    empnm varchar2(15),  
    deptno varchar2(3),  
    Salary number (10, 2),  
    PRIMARY KEY (empno, deptno)  
);
```

- ✓ Now, after creating table, empno & deptno must not be NULL and value entered in empno & deptno column must be UNIQUE.
- ✓ Suppose the user is entering following record after creating table  
INSERT INTO empdept values ('e101', 'Kishan', 'd1', 7000);  
Record added successfully. Now user is entering another record.
- ✓ INSERT INTO empdept values('e101', 'K1', 'd11', 3000);
- ✓ This record will not get added and error has been generated.
- ✓ When composite primary key is defined, table will not accept the value if the data of the entire composite column gets repeated.
- ✓ It accepts the value if any one of the composite key column value gets repeated.

### FOREIGN KEY constraint

- ✓ Foreign key represent relationships between tables.
- ✓ A foreign key is a column or group of columns whose values are derived from the primary key of some other table.
- ✓ The table in which foreign key is defined is called a foreign key table or detail table.
- ✓ The table that defines the primary key and is referenced by the foreign key is called primary table or master table.

### Principles of foreign key constraint:

- ✓ Rejects an INSERT or UPDATE of a value, if a corresponding value does not exist in the master key table.
- ✓ If the ON DELETE CASCADE option is set, a DELETE operation in the master table will DELETE corresponding records in the detail table also.
- ✓ Rejects a DELETE for the master table if corresponding records in the detail table exist.
- ✓ Must reference a PRIMARY KEY or UNIQUE column in primary table.
- ✓ Will automatically reference the PRIMARY KEY of the master table if no column or group of columns is specified when creating the foreign key.
- ✓ Foreign key column and primary key column must have same data type and size.
- ✓ Foreign key column name and primary key column name may be different.
- ✓ One column can be primary key column and foreign key column too.

### FOREIGN KEY at column level:

#### Syntax:

Column name data type (size) REFERENCES table name [(column name)]

## Unit-2: Managing Tables and data, Data Control And Transaction Control Commands

---

[ON DELETE CASCADE]

### Example:

```
CREATE TABLE empdetail
(
    empno varchar2(4) REFERENCES emp,
    Salary number (10, 2),
    Design varchar2 (4)
);
```

### FOREIGN KEY at table level:

#### Syntax:

```
FOREIGN KEY (column name [, column name , ...])
REFERENCES table name [(column name) [, column name , ... ]]
```

### Example:

```
CREATE TABLE empdetail
(
    empno varchar2(4),
    Salary number (10, 2),
    Design varchar2 (4),
    FOREIGN KEY empno REFERENCES emp(empno)
);
```

### UNIQUE KEY

- ✓ The purpose of UNIQUE constraint is to ensure that the data in the column is unique and it must not be repeated across the column.
- ✓ A table can have more than one UNIQUE key column.

### UNIQUE constraint at column level:

#### Syntax:

Column name data type (size) UNIQUE,...

### Example:

```
CREATE TABLE emp
(
    empno varchar2(4) UNIQUE,
    empnm varchar2(15)
    address1 varchar2 (20)
    address2 varchar2 (20),
);
```

### UNIQUE constraint at table level:

#### Syntax:

```
UNIQUE (columnname [, columnname , ... ])
```

## Unit-2: Managing Tables and data, Data Control And Transaction Control Commands

---

### Example:

```
CREATE TABLE emp
(
    empno varchar2(4),
    empnm varchar2(15),
    address1 varchar2 (20),
    address2 varchar2 (20),
    city varchar2(15),
    UNIQUE (empno)
);
```

### NOT NULL:

- ✓ A NULL value is different from space or zero.
- ✓ A NULL value can be inserted into the column of any data type.

### NOT NULL constraint at Column Level:

- ✓ When a column is defined as not null, it becomes mandatory column.
- ✓ It means that a value must be entered into the column.

### Syntax:

Columnname datatype(size) NOT NULL

### Example:

```
CREATE TABLE emp
(
    empno varchar2 (4) NOT NULL,
    empnm varchar2 (15) NOT NULL,
    address1 varchar2 (20),
    city varchar2(15) NOT NULL
);
```

**Note:** NOT NULL constraint cannot be applied at table level; it can only be applied at column level.

### CHECK constraint

- ✓ Business Rule validations can be applied to a column using CHECK constraint.
- ✓ CHECK constraint must be specified as logical expression that evaluates either to TRUE or FALSE.

### CHECK constraint at column level:

### Syntax:

Column name data type (size) CHECK (logical expression)

Create a table with following check constraints

- empno must start with 'E'
- data entered in empnm column must be upper case.
- Only allow "Bombay", "Delhi", "Rajkot" and "Pune" as city name.

### Example:

```
CREATE TABLE emp
(
    empno varchar2(4) CHECK(empno like 'E%'),
```

## Unit-2: Managing Tables and data, Data Control And Transaction Control Commands

---

```
empnm varchar2(15) CHECK(empnm=upper(empnm)),  
address1 varchar2 (20) NOT NULL,  
address2 varchar2 (20),  
city varchar2(15) CHECK(city IN ('bombay','delhi','rajkot','pune'))  
);
```

### **CHECK constraint at table level:**

#### **Syntax:**

CHECK (logical expression)

#### **Example:**

```
CREATE TABLE emp  
(  
    empno varchar2(4),  
    empnm varchar2(15),  
    city varchar2(15),  
    CHECK(empno like 'E%'),  
    CHECK(empnm=upper(empnm)),  
    CHECK(city IN ('bombay','delhi','rajkot','pune'))  
);
```

#### **Restrictions on CHECK constraints:**

- ✓ The condition must be a Boolean expression.
- ✓ The condition cannot contain sub query or sequence.
- ✓ The condition cannot include the SYSDATE, UID, USER or USERENV functions.

#### **Adding/Removing Constraint with Alter Table command:**

Alter Table <Table> Add Constraint <name><definition>;

#### **Examples:**

1.Add Primary key in emp table

```
ALTER TABLE emp ADD PRIMARY KEY(empno);
```

2.Add Foreign Key

```
ALTER TABLE empdet ADD CONSTRAINT fk_empno FOREIGN KEY empno REFERENCES  
emp(empno);
```

3.Add NOT NULL

```
ALTER TABLE empdet MODIFY (salary number(10,2) NOT NULL);
```

```
Alter Table<table>Drop Constraint<name>;
```

#### **Example:**

Drop Primary key in emp table.

```
ALTER TABLE emp drop Primary Key;
```

#### **Assign Default value to column**

Column\_name datatype(size) DEFAULT (value)

#### **Example:**

```
create table emp(empno number(4) default (000));
```

## **2.4Select statement**

The SELECT statement allows you to retrieve records from one or more tables

#### **Syntax 1: Selected column and all rows**

## Unit-2: Managing Tables and data, Data Control And Transaction Control Commands

---

SELECT columnname1, columnname2 ... column name n FROM <table name>;

### Example:

SELECT empno, empnm, address1, city FROM EMP;

### Syntax 2: All rows and all columns

SELECT \* FROM <table name>;

### Example:

SELECT \* FROM EMP;

### Selected rows and all columns:

- ✓ To retrieve selected rows from the table, Oracle provides 'where' clause in an SQL statement.
- ✓ When 'where' clause is added to the SQL statement, the Oracle Server compares each record from the table with the condition specified in the 'where' clause and display only those records that satisfy the specified condition.

### Syntax:

SELECT \* FROM <table name> WHERE =expression; Example: SELECT \* FROM EMP WHERE city = 'rajkot';

### Example:

SELECT \* FROM EMP WHERE city = 'rajkot';

### GROUP BY CLAUSE:

- ✓ Group By clause is used to group rows based on distinct (unique) values that exist for specified columns.
- ✓ GROUP BY is used in conjunction with aggregating functions to group the results by the unaggregated columns.

### Syntax:

SELECT column1, column2, column, aggregate function (expression) FROM table  
WHERE =expression GROUP BY column1, column2 ... column;

### EXAMPLE:

Select deptno, sum (sal) from EMP group by deptno;

- ✓ In above example, the common rows in the deptno column are grouped together and the total sum for each department is displayed along with the deptno.

### HAVING CLAUSE:

- ✓ A HAVING clause restricts the results of a GROUP BY clause.
- ✓ It can be used in conjunction with Group By clause.
- ✓ HAVING is used to give condition on group by clause column(s).
- ✓ The HAVING clause is applied to each group of the grouped data.

### Syntax:

SELECT column1, column2 ... column, aggregate function (expression) FROM table  
GROUP BY column1, column2, ... column HAVING groupbycolumn=expression;

### EXAMPLE:

Select deptno, sum (sal) from EMP group by deptno having deptno=10 or deptno=20;

---

## Unit-2: Managing Tables and data, Data Control And Transaction Control Commands

---

- ✓ In above example, the common rows in the deptno column are grouped together and the total salaries for only the deptno specified in HAVING clause are displayed.

### ORDER BY:

- ✓ Oracle allows data from a table to be viewed in a sorted order.
- ✓ The rows retrieved from the table will be sorted in either ascending or descending order.

### Syntax:

SELECT \* FROM tablename ORDER BY columnname, columnname[sort order];

### Examples:

- 1).Retrieve all the rows of emp table in ascending order of employee name

SELECT \* FROM emp ORDER BY empnm;

- 2).Retrieve all the rows of emp table in descending order of employee name

SELECT \* FROM emp ORDER BY empnm DESC;

- 3).Retrieve all the rows of emp table in ascending order of employee name and descending order of city

SELECT \* FROM emp ORDER BY empnm, city DESC;

### LIKE:

- ✓ This operator is used for pattern matching.
- ✓ Pattern matching can be done using % and \_ along with like operator.
- ✓ % is used to match any number of characters.
- ✓ It can be used in any position.
- ✓ When % is used, all characters after % are ignored by Oracle.
- ✓ For example 'A%' represents any name start with A followed by any no of characters.  
**Example:** Select \* from EMP where name like 'C%';
- ✓ These examples give all the detail from employee table whose names are started with C.
- ✓ \_ is used to match any one character.
- ✓ When it is used Oracle will ignore the character only in the position where underscore is put.
- ✓ For ex. '\_ONY' will display the word with 4 characters having first character as any character and remaining 3 characters would be 'ONY'.  
**Example:** Select \* from EMP where name like '\_ca';
- ✓ This example will give the records from EMP table whose name has only 3 characters and last 2 characters are c and a respectively.

### EXISTS:

- ✓ This operator is normally used with sub queries to check whether sub query returns any rows of query result.
- ✓ If value EXISTS (if sub query returns at least one row) it returns TRUE, else returns FALSE.
- ✓ With EXIST operator parent query does not use the sub query result but only checks the existence of rows.

#### Example:

Select ename, job, deptno from EMP e1 where

Exists (select empno from EMP e2 where e1.empno = e2.mgr);

### 2.5Join

- ✓ Some times it is necessary to work with multiple tables as though they were a single entity.
- ✓ Then a single SQL sentence can manipulate data from all the tables.
- ✓ To do this JOIN is used.
- ✓ Tables are joined on columns that have the same data type and data width in the tables.
- ✓ There are three types of join.

1) INNER JOIN

2) OUTER JOIN

a. LEFT OUTER JOIN

b. RIGHT OUTER JOIN

c. FULL OUTER JOIN

3) CROSS JOIN

#### 1) INNER JOIN

- ✓ INNER JOIN is also known as Equi join.
- ✓ It is known as Equi join because the where statement generally compares two columns from two tables with the equivalence operator (=).
- ✓ It returns all rows from both tables where there is match.

#### Syntax (ANSI STYLE):

Select <column-name1>,<column name N> from<table name1> inner join <table name 2>  
On <table name 1>.<column name>=<table name 2>.<column name>[where <condition>]

#### EXAMPLE:

Select empno, bno from emp, branch on emp.bno = branch.bno

#### Syntax (THETA STYLE):

Select <column-name1>,<column name N>from <table name1>,<table name2>  
Where <table name1>.<column name>=<table name2>.<column  
name>[AND<condition>order by<column name1>,<column name N>]

#### EXAMPLE:

Select empno, bno from emp, branch where emp.bno = branch.bno

#### 2) Outer Join:

- ✓ Outer joins are similar to inner joins, but give a bit more flexibility when selecting data from related tables.
- ✓ This type of join can be used in situations where it is desired, to select all rows from the table on the left (or right, or both) regardless of whether the other tables has values in common and (usually) enter NULL where data is missing.

#### Syntax:

Select<column list>  
FROM<left joined table>  
LEFT|RIGHT|FULL[OUTER] JOIN <right joined table>  
ON <join condition>

### Examples:

```
Select empno,ename,dname,loc From emp e left outer join dept d
Where e.deptno=d.deptno;

Select empno,ename,dname,loc From emp e left outer join dept d
on(e.deptno=d.deptno);

Select empno,ename,dname,loc From emp e right outer join dept d
on(e.deptno=d.deptno);

Select empno,ename,dname,loc From emp e full outer join dept d
on(e.deptno=d.deptno);
```

### Self Join:

- ✓ In some situations, it is necessary to join a table to itself, as though joining two separate tables. This is referred to as a self-join.
- ✓ In a self-join, two rows from the same table combine to form a result row.
- ✓ To join a table to itself, two copies of the very same table have to be opened in memory.
- ✓ Hence in the From clause, the table name needs to be mentioned twice.
- ✓ Since the table names are the same, the second table name is translated into a specific memory location.
- ✓ To avoid this, each table is opened using an alias. Now these table aliases will cause two identical table to be opened in different memory locations.
- ✓ This will result in two identical tables to be physically present in the computer's memory.

**Example:** Retrieve the names of the employees and the names of their respective managers from emp table.

```
Select a.ename "Employee",b.ename "Manager"
From emp a, emp b Where a.empno=b.mgr;
```

## 2.6 SubQuery,minus,intersect,union

### Sub-queries

- ✓ A sub-query is a form of an SQL statement that appears inside another SQL statement.
- ✓ It is also termed as nested query. The statement containing a sub-query is called a parent statement.
- ✓ The parent statement uses the rows (called result set) returned by the subquery.
- ✓ It can be used for insert records in a target table, create tables and insert record in the table created, update records in a target table, and to create views.
- ✓ Sub-query in where clause:

Example 1: List the name of all the employee whose manager is "Blake".

```
Select empnm from emp where manager_no = (select empno from emp
where empnm='Blake');
```



## Unit-2: Managing Tables and data, Data Control And Transaction Control Commands

**Example1:** List all employees of department 'Accounting'.

```
Select * from emp
where deptno
in (select deptno from dept where dname='Accounting');
```

**Example2:** List all employees of whose department not located at 'Dallas'.

```
Select * from emp
where deptno not
in (select deptno from dept where loc='Dallas');
```

### Sub-query in from clause:

**Example1:** List first three employees who get highest salary.

```
Select * from (Select * from emp order by sal Desc)
where rownum<4;
```

### MINUS

A minus Query is a query that uses the MINUS operator in SQL to subtract one result set from another result set to evaluate the result set difference. if there is no difference, there is no remaining result set. if there is a difference, the resulting rows will be displayed.

**Table\_1 Name:Yankees1**

f_name	l_name	Position
Babe	Ruth	LF
Louis	Gehrig	1B
Joe	Dimaggio	CF
Mickey	Mantle	CF
Derek	Jeter	SS
Yogi	Berra	C
Willie	Randolph	2B
Graig	Nettles	3B
CC	Sabathia	P

**Table\_2 Name:Yankees2**

firstName	lastName	Positions
Babe	Ruth	LF
Louis	Gehrig	1B
Joe	Dimaggio	CF
Mickey	Mantle	CF
Derek	Jeter	SS
Yogi	Berra	C
Willie	Nettles	3B
Andy	Pettitte	P

**So the first minus query for these tables is Table\_1 Minus Table\_2:**

```
(SELECT f_name,l_name,postion FROM Yankees1
MINUS
SELECT firstName,lastName,Positions FROM Yankees2)
```

The result set should be the rows we have highlighted in RED below:

Graig	Nettles	3B
-------	---------	----

CC	Sabathia	P
----	----------	---

Then you need to subtract Table\_2 MINUS Table\_1:

```
(SELECT firstName,lastName,Positions FROM Yankees2  
MINUS
```

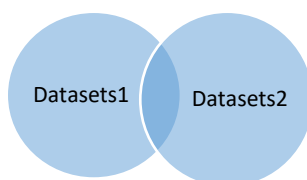
```
SELECT f_Name,l_Name,position FROM Yankees1);
```

The result set should be the rows we have highlighted in GREEN below:

Andy	Pettitte	P
------	----------	---

## INTERSECT

The Oracle INTERSECT operator is used to return the results of 2 or more SELECT statements. However, it only returns the rows selected by all queries or data sets. If a record exists in one query and not in the other, it will be omitted from the INTERSECT results.



**Explanation:** The INTERSECT query will return the records in the blue shaded area. These are the records that exist in both Dataset1 and Dataset2.

Each SELECT statement within the INTERSECT must have the same number of fields in the result sets with similar data types.

### Syntax:

The syntax for the INTERSECT operator in Oracle/PLSQL is:

```
SELECT expression1,expression2,...expression_n
```

```
FROM tables
```

```
[WHERE conditions]
```

```
INTERSECT
```

```
SELECT expression1,expression2,...expression_n
```

```
FROM tables
```

```
[WHERE conditions];
```

Example:

## Unit-2: Managing Tables and data, Data Control And Transaction Control Commands

---

```
select      f_name,l_name,position      from      yankees1      INTERSECT      select  
firstname,lastname,positions from yankees2;
```

Output:

### UNION:

The Oracle UNION operator is used to combine the result sets of 2 or more Oracle SELECT statements. It removes duplicate rows between the various SELECT statements.

Each SELECT statement within the UNION operator must have the same number of fields in the result sets with similar data types.

### Syntax:

The syntax for the UNION operator in Oracle/PLSQL is:

```
SELECT expression1,expression2,...expression_n  
FROM tables  
[WHERE conditions]  
UNION  
SELECT expression1,expression2,...expression_n  
FROM tables  
[WHERE conditions];
```

## Built-in Function

### 2.7 Numeric functions:

#### 1) ABS:

**Syntax:** abs (n)

**Purpose:** This function returns the absolute value of n.

**Example:** Select abs (-3) from dual;

**OUTPUT:** 3

#### 2) POWER:

**Syntax:** Power (m, n)

**Purpose:** This function returns the m raised to the nth power.

N must be an integer, else an error is returned.

**Example:** Select power (2, 3) from dual;

**OUTPUT:** 8

#### 3) ROUND:

**Syntax:** round (n, [m])

**Purpose:**

- Returns n, rounded to m places to the right of a decimal point.
- If m is omitted, n is rounded to 0 places.
- M can be negative to round off digits to the left of the decimal point, m must be an integer.

**Example:** Select round (15.19, 1) from dual;

**OUTPUT: 15.2**

**4) SQRT:**

**Syntax:** SQRT (n)

**Purpose:** Returns square root of n. if n<0,null.

**Example:** Select sqrt(9) from dual;

**Output:** 3

**5) EXP:**

**Syntax:** exp (n)

**Purpose:** Returns e raised to the nth power where e = 2.71828183

**Example:** Select exp (5) from dual;

**OUTPUT:** 148.413159

**6) GREATEST:**

**Syntax:** GREATEST (expr1, expr2 ... expr)

**Purpose:** Returns the greatest value in a list of expression.

**Example:** Select greatest (1, 2, 3, 6) from dual;

**OUTPUT:** 6

**7) LEAST**

**Syntax:** LEAST (expr1, expr2 ... expr)

**Purpose:** Returns the smallest value in a list of expression.

**Example:** Select least (1, 2, 3, 6) from dual;

**OUTPUT:** 1

**8) MOD**

**Syntax:** MOD (m, n)

**Purpose:** Returns the remainder of first number divided by second number passed into the argument.

If the second number is zero, the result is the same as the first number.

**Example:** Select mod (6, 3) from dual;

**OUTPUT:** 0

**9) FLOOR**

**Syntax:** FLOOR (n)

**Purpose:** Returns the largest integer value that is equal to or less than a number.

**Example:** Select floor (24.8) from dual;

**OUTPUT:** 24

**10) CEIL**

**Syntax:** CEIL (n)

**Purpose:** Returns the smallest integer value that is equal to or greater than a number.

**Example:** Select ceil (24.8) from dual;

**OUTPUT:** 25

**11) COS**

**Syntax:** COS(number)

**Purpose:** The Oracle/PLSQL COS function returns the cosine of a number.

**Example:** COS(0.2)

Result:0.980066577841242

### 12) DECODE

**Syntax:** DECODE(expression,search,result[,search,result]...[,default])

**Purpose:** The Oracle/PLSQL DECODE function has the functionality of an IF-THEN-ELSE statement.

**Example:** select job,ename,decode(job,'MANAGER','Excellent','PRESIDENT','Very good','CLERK','Good') from emp;

### 13) LOG

**Syntax:** LOG(m,n)

**Purpose:** The Oracle/PLSQL LOG function returns the logarithm of n base m.

**Example:** select LOG(10,20) from dual;

**OUTPUT:** 1.30102999566398

### 14) LOG10

**Syntax:** LOG10(number)

**Purpose:** The Oracle/PLSQL LOG10 function returns the base e logarithm value for the numeric expression.

**Example:** SELECT LOG10(200) from dual;

**OUTPUT:** 2.3010299566398

### 15) SIGN

**Syntax:** SIGN(number)

**Purpose:** The Oracle/PLSQL SIGN function returns a value indicating the sign of a number.

**Example:** select SIGN(-23) from dual;

**OUTPUT:** -1

### 16) SIN

**Syntax:** SIN(n)

**Purpose:** The Oracle/PLSQL SIN function returns the sine of n.

**Example:** select SIN(3) from dual;

**OUTPUT:** 0.141120008059867

### 17) SINH

**Syntax:** SINH(n)

**Purpose:** The Oracle/PLSQL SINH function returns the hyperbolic sin of n.

**Example:** SINH(3)

**OUTPUT:** 10.0178749274099

### 18) TAN

**Syntax:** TAN(n)

**Purpose:** The Oracle/PLSQL TAN function returns the Tangent of n.

**Example:** TAN(3)

**OUTPUT:** -0.142546543074278

### 19) TRUNC

**Syntax:** TRUNC(number [,decimal\_places])

**Purpose:** The Oracle/PLSQL TRUNC function returns a number truncated to a certain number of decimal places.

**Example:** TRUNC(125.815)

**OUTPUT:** 125

## **2.8 Character Function:**

### **1) Concat:**

**Syntax:** concat(string1,string2)

**Purpose:** it is used to combine the two strings and return combination of both strings.

**Example:** select concat('hi','how are you')

**OUTPUT:** hi how are you

### **2) Initcap:**

**Syntax:** initcap(string)

**Purpose:** it returns the string with the first letter of each word in upper case and all other letters in lower case.

**Example:** Select initcap('this is example') from dual;

**OUTPUT:** This is Example

### **3) Lower:**

**Syntax:** lower(string)

**Purpose:** it returns the string with all letters in lower case.

**Example:** Select lower('ABC') from dual;

**OUTPUT:** abc

### **4) Upper:**

**Syntax:** Upper(string)

**Purpose:** it returns the string with all letters in Uppper case.

**Example:** Select upper('abc') from dual;

**OUTPUT:** ABC

### **5) Lpad:**

**Syntax:** lpad(expr1,n,expr2)

**Purpose:** it returns the Expr1 left padded to the length or n character with the sequence of characters in expr2.

**Example:** Select lpad('abc',10,'\*') from dual;

**OUTPUT:** \*\*\*\*\*abc

### **6) Rpad:**

**Syntax:** rpad(expr1,n,expr2)

**Purpose:** it returns expr1 right padded to length n character with sequence of character in expr2.

This function is useful for formatting output of query.

**Example:** Select rpad('abc',10,'\*') from dual;

**Output:** abc\* \* \* \* \*

### **7) Ltrim:**

**Syntax:** ltrim(string,set)

**Purpose:** It removes from the left end of string all of the character contains in set. If set is not defined it will default as a blank.

**Example:** Select ltrim('xyXxyABC','y')

**Output:** XxyABC

### 7) Rtrim:

**Syntax:** rtrim(string,set)

**Purpose:** It removes from the right end string all of the character contains in set. If set is not defined it will default as blank.

**Example:** Select rtrim('ABC\*\*\*\*\*','\*')

**Output:** ABC

### 8) Replace

**Syntax:** replace (string,search\_string, replacement\_string);

**Purpose:** It returns characters with every occurrence of search string replace with replacement string. If replacement string is omitted then all occurrence of search string are removed.

**Example:** Select replace ('BCA College', 'BCA','PGDCA') from dual;

**Output:** PGDCA College.

### 9) Substr:

**Syntax:** Substr (str,m,n)

**Purpose:** This function returns a part of string beginning at character position define by m to n character long. If m is positive then start from beginning. If m is negative then start from ending.

**Example:** Select substr ('hello', 2, 5) from dual;

**Output:** ello

### 10) length

**Syntax:** length (string)

**Purpose:** It returns the total length or size of the specified string.

**Output:** 5

**Example:** Select length ('hello') from dual;

### 11) CHR:

**Syntax:** CHR(number\_code)

**Purpose:** The Oracle/PLSQL CHR function is the opposite of ASCII function.it returns the character based on the NUMBER code.

**Example:** CHR(116)

**OUTPUT:** 't'

### 12) SOUNDEX:

**Syntax:**SOUNDEX(string1)

**Purpose:** The OraCLE/PLSQL SOUNDEX function returns a phonetic representation (the way it sounds) of a string.

**Example:** SOUNDEX('tech on the net')

**OUTPUT:** 'T253'

## 2.9 DATE FUNCTION:

### 1) ADD\_MONTHS:

**Syntax:** ADD\_MONTHS (date, integer)

**Purpose:**

- This function returns the date plus integer months.
- The date argument can be a date value or any values that can be implicitly converted to DATE.

- The integer argument can be integer value or any values that can be implicitly converted to integer.
- The return type of this function is date.

**Example:** select add\_months('07-JAN-2011',1) from dual;

**Output:** 07-FEB-11

### 2) LAST\_DAY:

**Syntax:** LAST\_DAY (d1)

**Purpose:**

- It returns the last date of specified month.
- The d1 argument can be any date value.

**Example:** select last\_day('07-JAN-2011') from dual;

**Output:** 31-JAN-11

### 3) MONTHS\_BETWEEN:

**Syntax:** MONTHS\_BETWEEN (d1, d2)

**Purpose:**

- It returns the how many months fall between two dates.
- The return type of this function is number value.
- Here user has to give two dates as argument.

**Example:** select months\_between('07-FEB-2011','07-JAN-2011') from dual;

**Output:** 1

### 4) SYSDATE:

**Syntax:** SYSDATE

**Purpose:**

- This function returns the current system date.
- The return type of this function is date.

**Example:** Select sysdate from dual;

**Output:** 06-MAR-11

### 5) NEXT\_DAY:

**Syntax:** NEXT\_DAY(date,weekday)

**Purpose:** The Oracle/PLSQL NEXT\_DAY function returns the first weekday that is greater than a date.

**Example:** NEXT\_DAY('01-Aug-03','TUESDAY')

**OUTPUT:** '05-Aug-03'

### 6) ROUND:

**Syntax:** ROUND(date[,format])

**Purpose:** The Oracle/PLSQL ROUND function returns a date rounded to a specific unit of measure.

**Example:** ROUND(TO\_DATE('22-Aug-03'),'YEAR')

**OUTPUT:** '01-JAN-04'

### 7) SYSTIMESTAMP:

**Syntax:** SYSTIMESTAMP



## Unit-2: Managing Tables and data, Data Control And Transaction Control Commands

---

**Purpose:** The Oracle/PLSQL SYSTIMESTAMP returns the current system date and time(including fractional seconds and time zone)on your loca database.

**Example:** SELECT SYSTIMESTAMP

INTO v\_time  
FROM dual;

**Output:** 2015-07-22 10:35:07.417849-06:00

### 8) TRUNC:

**Syntax:** TRUNC(DATE[,FORMAT])

**Purpose:** The Oracle/PLSQL TRUNC function returns a date truncated to a specific unit of measure.

**Example:** TRUNC(TO\_DATE('22-AUG-03'),'YEAR')

**OUTPUT:** '01-JAN-03'

### 9) TO\_CHAR:

**Syntax:** TO\_CHAR(value[,format\_mask][,nls\_language])

**Purpose:**The Oracle/PLSQL TO\_CHAR function converts a number or date to a string.

**Example:** TO\_CHAR(1210.73,'9999.9')

### 10) TO\_DATE:

**Syntax:** TO\_DATE(string1,format\_mask)[,nls\_language])

**Purpose:** The Oracle/PLSQL TO\_DATE function converts a string to a date.

**Example:** TO\_DATE (2020/10/06','yyyy/mm/dd')

**OUTPUT:** 06-oct-20

## 2.10AGGREGATE FUNCTION:

### 1) AVG:

**Syntax:** AVG ([<distinct>]<n> )

**Purpose:** This function returns the average value of specified column.

**Example:** Select avg (sal) from EMP;

(This statement returns the average value of sal from EMP table.)

### 2) MIN:

**Syntax:** MIN ([<distinct>]<expr> )

**Purpose:** This function returns the minimum value of specified expression or column.

**Example:** Select min (1, 2, 3, 6) from EMP;

**Output:** 1

### 3) MAX:

**Syntax:** MAX ([<distinct>]<expr> )

**Purpose:** This function returns the maximum value of specified expression or column.

**Example:** Select max (1, 2, 3, 4) from dual;

**Output:** 4

**4) COUNT:**

**Syntax:** COUNT ([<distinct>]<expr>)

**Purpose:** This function returns the number of rows where expr is not null.

**Example:** Select count (sal) from EMP;

(This function returns the total number of records into sal column.)

**5) SUM:**

**Syntax:** SUM ([<distinct>]<expr>)

**Purpose:** This function returns the sum of the specified column or n.

**Example:** Select sum (1, 2, 3, 4) from dual;

**OUTPUT:** 10

**2.11 General Functions**

These functions work with any data type and pertain to the use of null values in the expression list. These are all **single row function** i.e. provide one result per row.

**2.11.1 COALESCE() :**

The COALESCE() function examines the first expression, if the first expression is not null, it returns that expression; Otherwise, it does a COALESCE of the remaining expressions. The advantage of the COALESCE() function over the NVL() function is that the COALESCE function can take multiple alternate values. In simple words COALESCE() function returns the first non-null expression in the list.

**Syntax –**

COALESCE (expr\_1, expr\_2, ... expr\_n)

Examples –

select coalesce(null,2) from dual;

output-

2

**2.11.2 DECODE() :**

Facilitates conditional inquiries by doing the work of a CASE or IF-THEN-ELSE statement. The DECODE function decodes an expression in a way similar to the IF-THEN-ELSE logic used in various languages. The DECODE function decodes expression after comparing it to each search value. If the expression is the same as search, result is returned. If the default value is omitted, a null value is returned where a search value does not match any of the result values.

**Syntax –**

DECODE(col|expression, search1, result1

[, search2, result2,...],[, default])

Example –

select ename,job,sal,decode(job,'SALESMAN',sal-100,'CLERK',sal+100,sal) fro

m emp;

## Unit-2: Managing Tables and data, Data Control And Transaction Control Commands

---

output-

ENAME	JOB	SAL	DECODE(JOB,'SALESMAN',SAL-100,'CLERK',SAL+100,SAL)
-------	-----	-----	--

-----

SMITH	CLERK	800	900
-------	-------	-----	-----

ALLEN	SALESMAN	1600	1500
-------	----------	------	------

WARD	SALESMAN	1250	1150
------	----------	------	------

JONES	MANAGER	2975	2975
-------	---------	------	------

### 2.11.3 CASE WHEN:

CASE allows you to perform IF-THEN-ELSE logic in your SQL statements, similar to DECODE.

Syntax

```
CASE [expression]
WHEN condition_1 THEN result_1
WHEN condition_2 THEN result_2...
WHEN condition_n THEN result_n
ELSE result
END case_name
```

The *expression* is used to compare against. Many *conditions* and *results* can be specified, and if a *condition* matches the *expression*, then the *result* is returned. The ELSE keyword specifies what happens if no *condition* is met.

It was introduced into Oracle to replace the DECODE function.

Example:

Table:person

Status

-----

a1

i

t

a2

a3

```
sql> SELECT
      status,
      CASE
        WHEN STATUS IN('a1','a2','a3')
        THEN 'Active'
        WHEN STATUS = 'i'
        THEN 'Inactive'
        WHEN STATUS = 't'
```

```
    THEN 'Terminated'  
END AS STATUSTEXT  
FROM  
    person;
```

**Output:**

STATUS	STATUSTEXT
a1	Active
i	Inactive
t	Terminated
a2	Active
a3	Active

**2.12 Creating users and Role, Grant, Revoke  
(Data Control and Transaction Control Command)**

Database Administrator Security is an often-overlooked aspect of database design. Most computer professionals enter the computer world with some knowledge of computer programming or hardware, and they tend to concentrate on those areas. Many times, little thought or planning goes into the actual production phase of the application. What happens when many users are allowed to use the application across a wide area network (WAN)? With today's powerful personal computer software and with technologies such as Microsoft's Open Database Connectivity (ODBC), any user with access to your network can find a way to get at your database. (We won't even bring up the complexities involved when your company decides to hook your LAN to the Internet or some other wide-ranging computer network!) Are you prepared to face this situation?

Fortunately for you, software manufacturers provide most of the tools you need to handle this security problem. Implementation of these security features varies widely from product to product. Oracle relational database management system supports nearly the full SQL standard. In addition, Oracle has added its own extension to SQL, called PL\*SQL. It contains full security features, including the capability to create roles and assign permissions and privileges on objects in the database.

**How Does a Database Become Secure?**

Has it occurred to you that you might not want other users to come in and tamper with the database information you have so carefully entered? What would your reaction be if you logged on to the server one morning and discovered that the database you had slaved over had been dropped (remember how silent the DROP DATABASE command is)? We examine in some detail how one popular database management system enables you to set up a secure database. Keep the following questions in mind as you plan your security system:

**Who gets the DBA role?**

How many users will need access to the database?

Which users will need which privileges and which roles?

How will you remove users who no longer need access to the database?

Oracle implements security by using three constructs:

- Users
- Roles
- Privileges

### Creating Users

Users are account names that are allowed to log on to the Oracle database. The SQL syntax used to create a new user follows:

```
CREATE USER <user> IDENTIFIED{BY <password>| EXTERNALLY}
[DEFAULT TABLESPACE<tablespace>]
[TEMPORARY TABLESPACE<tablespace>]
[QUOTA{<integer>[k|M UNLIMITED]} ON <tablespace>]
[PROFILE<profile>];
```

If the BY <password> option is chosen, the system prompts the user to enter a password each time he or she logs on.

**Example:** CREATE USER MJKCS IDENTIFIED BY mjk;

If the EXTERNALLY option is chosen, Oracle relies on your computer system login name and password. When you log on to your system, you have essentially logged on to Oracle. There is also an ALTER USER command.

### Alter User

```
ALTER USER<user>[IDENTIFIED{BY <password> | EXTERNALLY}]
[DEFAULT TABLESPACE<tablespace>]
[TEMPORARY TABLESPACE<tablespace>]
[QUOTA{<integer>[K|M | UNLIMITED]} ON <tablespace>]
[PROFILE<profile>];
```

### Drop User

```
DROP USER<user> [CASCADE];
```

## 2.13 Grant, Revoke command

### What is Privilege?

Anything as important as business data has to be well protected. SQL provides protection of data by allocating rights and privileges to authorized users. Their login identification and passwords identify authorized users. Users can be allocated rights and privileges on various database objects. They can have rights like:

- Rights of selecting data (SELECT)
- Rights of Alter object (ALTER)

- Rights of adding data (INSERT)
- Rights of updating data (UPDATE)
- Rights of deleting data (DELETE)
- Rights of index data (INDEX)

These privileges, once granted to a user, can also be revoked. Certain privileges will grant utilities when given to another user and also make him/her eligible to pass on these privileges to other users. Depending upon the nature of work assigned to a user they can be given the required privileges and rights. These commands come in the category of Data Control Language (DCL). Let us discuss of these commands.

### **GRANT Statement:**

The creator of an object automatically becomes the owner of the object. There is no need to be granted any privilege to use the object. In case, you want to share database objects, that you have created, with others, the appropriate privileges can be granted on that particular database object to the other users. Objects privileges can be granted, to others, using the SQL statement GRANT.

GRANT<object privilege(s)>ON<object name>

TO<username>[WITH GRANT OPTION];

The 'WITH GRANT OPTION' allows that user to pass on the privileges on that database objects to other users. By default, users who have been granted a privilege on a database object, can only use the privilege themselves; they cannot grant any privileges on that database object to other users.

**Example:** GRANT SELECT, INSERT ON STUD TO Scott;

### **REVOKE Statement:**

To withdraw the privilege(s) which have been granted to a user, you can use the REVOKE command. The format, similar to that of the GRANT command, is as follows:

REVOKE<object privilege(s)>ON<objectname>FROM<username>;

**Example:** REVOKE UPDATE, DELETE ON STUD FROM Scott;

### **ROLE**

When you log on to the system using the account you created earlier, you have exhausted the limits of your permissions. You can log on, but that is about all you can do. Oracle lets you register as one of three roles: Connect, Resource, DBA (or database administrator). These three roles have varying degrees of privileges. If you have the appropriate privileges, you can create your own role, grant privileges to your role, and then grant your role to a user for further security.

#### **The Connect Role:**

The Connect role can be thought of as the entry-level role. A user who has been granted Connect role access can be granted various privileges that allow him or her to do something with a database. The Connect role enables the user to select, insert, update, and delete records from tables belonging to other users (after the appropriate permissions have been granted). The user can also create tables, views, sequences, clusters, and synonyms.

#### **The Resource Role:**

## Unit-2: Managing Tables and data, Data Control And Transaction Control Commands

---

The Resource role gives the user more access to Oracle databases. In addition to the permissions that can be granted to the Connect role, Resource roles can also be granted permission to create procedures, triggers, and indexes.

### The DBA Role:

The DBA role includes all privileges. Users with this role are able to do essentially anything they want to the database system. You should keep the number of users with this role to a minimum to ensure system integrity

**User Privileges** After you decide which roles to grant your users, your next step is deciding which permissions these users will have on database objects. (Oracle calls these permissions privileges.) The types of privileges vary, depending on what role you have been granted. If you actually create an object, you can grant privileges on that object to other users as long as their role permits access to that privilege. Oracle defines two types of privileges that can be granted to users: system privileges and object privileges. System privileges apply system wide. The syntax used to grant a system privilege is as follows:

```
GRANT system_privilege TO{<user> | role | PUBLIC}
```

```
[WITH ADMIN OPTION];
```

WITH ADMIN OPTION enables the grantee to grant this privilege to someone else

**User Access to Views:** The following command permits all users of the system to have CREATE VIEW access within their own schema.

**Example:** GRANT CREATE VIEW TO PUBLIC;

## 2.14What is transaction?

Transaction control, or transaction management, refers to the capability of a relational database management system to perform database transactions. Transactions are units of work that must be done in a logical order and successfully as a group or not at all. The term unit of work means that a transaction has a beginning and an end. If anything goes wrong during the transaction, the entire unit of work can be canceled if desired. If everything looks good, the entire unit of work can be saved to the database.

## 2.15Starting and Ending of Transaction

### Beginning & Ending a Transaction

Transactions are quite simple to implement. You will examine the syntax used to perform transactions using the Oracle RDBMS SQL syntax. All database systems that support transactions must have a way to explicitly tell the system that a transaction is beginning. (Remember that a transaction is a logical grouping of work that has a beginning and an end.)

```
SET TRANSACTION{READ ONLY | USE ROLLBACK SEGMENT<segment>}
```

Oracle enables the user to specify when the transaction will begin by using the SET TRANSACTION statement. The SET TRANSACTION READ ONLY option enables you to effectively lock a set of records until the transaction ends. You can use the READ ONLY

---

**Unit-2: Managing Tables and data, Data Control And Transaction Control Commands**

---

option with the following commands: SELECT, LOCK TABLE, SET ROLE, ALTER SESSION, and ALTER SYSTEM.

The option USE ROLLBACK SEGMENT tells Oracle which database segment to use for rollback storage space. This option is an Oracle extension to standard SQL syntax. SQL Server's Transact-SQL language implements the BEGIN TRANSACTION command with the following syntax:

Begin {transaction} [transaction\_name]

This allows you to give a transaction a name. We can use SET AUTOCOMMIT to commit transaction automatic using following command:

SET AUTOCOMMIT [ON | OFF]

By default, the SET AUTOCOMMIT ON command is executed at startup. It tells SQL to automatically commit all statements you execute. If you do not want these commands to be automatically executed, set the AUTOCOMMIT option to off.

### **2.16Commit, Rollback, Savepoint**

#### **Finishing a Transaction [COMMIT]**

The Oracle syntax to end a transaction is as follows:

COMMIT [WORK][COMMENT 'text' | FORCE 'text' [, integer]];

The COMMIT command saves all changes made during a transaction. Executing a COMMIT statement before beginning a transaction ensures that no errors were made and no previous transactions are left hanging.

#### **Canceling the Transaction [ROLLBACK]**

While a transaction is in progress, some type of error checking is usually performed to determine whether it is executing successfully. You can undo your transaction even after successful completion by issuing the ROLLBACK statement, but it must be issued before a COMMIT. The ROLLBACK statement must be executed from within a transaction.

ROLLBACK [WORK][TO SAVEPOINT<savepoint> | FORCE 'text'];

The ROLLBACK statement rolls the transaction back to its beginning; in other words, the state of the database is returned to what it was at the transaction's beginning. As you can see, this command makes use of a transaction savepoint.

### **SAVEPOINT**

A SAVEPOINT is like a marker to divide a lengthy transaction into smaller ones. It is used to identify a point in a transaction to which we can later rollback. It is used in conjunction with rollback, to rollback portions of the current transaction.

SAVEPOINT<savepoint name>;

For example, what could happen in the earlier case is, if the user had already entered two records and after entering the third, realizes that the person does not want that third record, then the rollback statement would rollback the entire transaction he just made, including the first two records that are actually required. To overcome this problem you can use savepoint. By creating savepoint after two records make this possible.



## Unit – 3: Oracle Database Objects, Concurrency control using lock

---

### 3.1 View

After a table is created and populated with data, it may become necessary to prevent all users from accessing all columns of a table, for data security reasons.

#### The reasons why views are created:

- ✓ When data security is required
- ✓ When data redundancy is to be kept to the minimum while maintaining data security.

#### Creating views

##### Syntax:

```
Create view viewname as
Select column1, column2 from table name
[Where column name = expression]
[Group by grouping criteria]
[Having predicate]
```

**Example:** Create view vw\_emp as select empno, ename, sal, deptno from emp;

#### Selecting a data set from a view:

Once a view has been created, it can be queried exactly like a base table.

**Syntax:** Select ColumnName1, Column Name2 from ViewName;

**Example:** Select \* from vw\_emp;

#### Updatable Views:

- Views can also be used for data manipulation (The user can perform the Insert, Update and Delete operations).
- Views on which data manipulation can be done are called updatable views.
- When updatable view names is given in an insert update or delete sql statement, modification to data in the view will be immediately passed to the underlying table and also modification in table will affect the view.

#### For view to be updateable, it should meet the following criteria:

- Views defined from single table.
- If the user wants to insert records with the help of a view, then the Primary key column(s) and all the Not Null columns must be included in the view.
- The user can update, delete records with the help of a view even if the Primary key column and not null column(s) are excluded from the view definition.

#### Inserting data in views:

**Example:** Insert into vw\_emp values (500,'Harit', 4000, 20);

#### Updating data in views

**Example:** Update vw\_emp set sal=4500 where empno=101;

#### Deleting data from views

**Example:** Delete from vwemp where empno=500;

## Unit – 3: Oracle Database Objects, Concurrency control using lock

---

### Views defined from multiple tables (Which have been created with a referencing clause)

If a view is created from multiple tables, which created using a referencing clause (Logical linkage exists between the tables), then though the Primary Key Column(s) as well as the NOT NULL columns are included in the View definition, the view's behavior will be as follows:

- An insert operation is not allowed
- The delete or modify operations do not affect the Master Table
- The view can be used to Modify the columns of the detail table included in the view
- If a delete operation is executed on the view the corresponding records from detail table will be deleted.

### Common Restrictions on Updateable Views

The following condition holds true irrespective of the view being created from a single table or multiple tables.

For the view to be updateable the view definition must not include:

1. Aggregate Function
2. Distinct, group by or having clause
3. Sub Query
4. Constants, String or value expressions like `sell_price * 1.05`
5. Union, Intersect or minus clause
6. If a view is defined from another view, the second view should be updateable

### Destroying a view

The drop view command is used to remove a view from the database.

#### Syntax:

Drop view<View Name>

#### Example:

Drop view vw\_emp;

### 3.2 Sequence:

- ✓ Oracle provides an object called a Sequence that can generate numeric values.
- ✓ The value generated can have a maximum of 38 digits.
- ✓ A sequence can be defined to:
  - Generate numbers in ascending or descending order
  - Provide intervals between numbers
  - Caching of sequence numbers in memory to speed up their availability
- ✓ A sequence is an independent object and can be used with any table that requires its output.

## Unit – 3: Oracle Database Objects, Concurrency control using lock

---

### Syntax:

Create sequence Sequence Name  
[Increment By <Integer Value>  
Start With<Integer Value>  
Maxvalue <Integer Value>/ NonMaxvalue  
Minvalue<Integer Value> / Nonminvalue  
Cycle/NoCycle  
Cache <Integer Value>/ No Cache  
Order / Noorder]

### Keywords and Parameters:

#### Increment By:

- Specifies the interval between sequence numbers.
- It can be any positive or negative value but not zero.
- If this clause is omitted, the default value is 1.

#### MinValue:

- Specifies the sequence minimum value.

#### Nominvalue:

- Specifies a minimum value of 1 for an ascending sequence and  $-(10)^{26}$  for a descending sequence.

#### Maxvalue:

- Specifies the maximum value that a sequence can generate

#### Nomaxvalue:

- Specifies a maximum of  $10^{27}$  for an ascending sequence or -1 for a descending sequence. This is the default clause.

#### Startwith:

- Specifies the first sequence number to be generated.
- The default for an ascending sequence is the sequence minimum value (1) and for a descending sequence, it is the maximum value (-1).

#### Cycle:

- Specifies that the sequence continues to generate repeat values after reaching either its maximum value.

#### Nocycle:

- Specifies that a sequence cannot generate more values after reaching the maximum value.

## Unit – 3: Oracle Database Objects, Concurrency control using lock

---

### Cache:

- Specifies how many values of a sequence Oracle pre-allocates and keeps in memory for faster access.
- The minimum value for this parameter is two.

### NoCache:

- Specifies that values of a sequence are not pre-allocated.

**Note:** If the cache/NoCache clause is omitted Oracle caches 20 sequence numbers by default.

### Example:

Create sequence stest1 start with 5 increment by 1 maxvalue 50;

### Inserting data into AddDetail table using sequence

```
Insert into AddDetail(Addno,name,add1,city) values(stest1.nextval,'Dipen','Station Road','Jamnagar');
```

### To reference next value of sequence

#### Syntax:

Select Sequence Name.Nextval from dual;

- ✓ This will display the next value held in the cache.
- ✓ Every time nextval references a sequence its output is automatically incremented from the old value to the new value ready for user.

### Example:

Select stest1.nextval from dual;

### To reference current value of sequence

#### Syntax:

Select<Sequence Name> .Currval From Dual;

### Example

Select stest1.currval from dual;

### Altering a Sequence

#### Syntax:

Alter sequence <<Sequence Name>>

Incremented by interger value

Maxvalue Integer Value/ Nomaxvalue

Minvalue Integer Value/Nominvalue

Cycle/Nocycle

### Example:

Alter sequence stest1 increment by 2;

## Unit – 3: Oracle Database Objects, Concurrency control using lock

---

### Dropping a Sequence:

#### Syntax:

Drop Sequence <<Sequence Name>>;

#### Example:

Drop Sequence stest1;

### 3.3 Synonyms

- ✓ A synonym is an alternative name for objects such as tables, views, sequences, stored procedures, and other database objects.

#### Syntax:

Create Public Synonym **Synonym.Name** for Table Name

#### Example:

Create public synonym employees for scott.emp

- ✓ Now, users of other schemas can reference the table emp, which is now called as employees without having to prefix the table name with the schema named Scott.

#### Example:

Select \* from Employee;

### Dropping Synonyms:

#### Syntax:

Drop Public Synonym **Synonym.Name**;

#### Example:

Drop public Synonym employees;

### 3.4 Database Links:

- ✓ As your database grows in size and number, you will very likely need to share data among them.
- ✓ Sharing data require a method of locating and accessing the data.
- ✓ In Oracle, remote data accesses such as queries and updates are enabled through the use of database links.
- ✓ You will also find information about direct connections to remote database, such as those used in client-server applications.
- ✓ Database links tell oracle how to get from one database to another.
- ✓ If you will frequently use the same connection to a remote database then a database link is appropriate.

#### Database links specify the following connection information:

- 1) The communications protocol (such as TCP/ IP) to use during the connection.
- 2) The host on which the remote database resides.
- 3) The name of the database on the remote host.
- 4) The name of a valid account in the remote database.

## Unit – 3: Oracle Database Objects, Concurrency control using lock

---

5) The password for that account when used, a database link actually logs in as a user in the remote database, and then logs out when the remote data access is complete.

A database link can be private, owned by a single user, or public, in which case all users in the local database can use the link.

### Syntax:

```
CREATE [PUBLIC] DATABASE LINK  
REMOTE_CONNECT CONNECT TO  
[CURRENT USER / IDENTIFIED BY]  
USING 'CONNECTION STRING';  
SELECT * FROM < TABLE NAME> @ REMOTE STRING;
```

### 3.5 Index:

- ✓ Indexes are data structures that have to improve speed in obtaining specific row from table.
- ✓ Indexing a table is an access strategy that is a way to short and search records in the table.
- ✓ An index is an order list of the content of column.
- ✓ Index should be created on a column that is queried frequently.
- ✓ Usually to major types of indexes are used:
  - 1) B-TREE index
  - 2) BITMAP index

### B-TREE index:

Oracle allows the creation of two types of indexes:

- 1) Duplicate index
- 2) Unique index

#### Duplicate index:

Index that allows duplicate value for the indexed column.

#### Unique index:

Index that any duplicate value for the indexed column.

Indexes are further divided into two group based on the number of columns

- 1) **Simple index**
- 2) **Composite index**

#### Duplicate index:

##### 1) Simple duplicate index

An index created on a single column of a table is called simple index.

#### Syntax:

Create index<index name> on<Table name>(<column name>);

## Unit – 3: Oracle Database Objects, Concurrency control using lock

---

### Example:

Create index i1 on emp(empno);

### 2) Composite duplicate index:

An index created on a multiple column of a table is called composite index.

#### Syntax:

Create index <index name> on <table name>(<column name1>,<column name2>...);

#### Example:

Create index i1 on emp(empno,ename);

### Unique index:

#### 1)Unique simple index:

##### Syntax:

Create unique index<index name>on <table name>

##### Example:

Create unique index i1 on emp(empno);

#### 2)Unique Composite index:

##### Syntax:

Createunique index<index name>on<table name>(<column name1>,<column name2>);

##### Example:

Create unique index i1 on emp(empno,ename)

### 3.6 Cluster

Clustering is an important concept for improving Oracle performance. Whenever the database is accessed, any reduction in input / output always helps in improving its throughput and overall performance. The concept of cluster is where member records are stored physically near parent records. Clusters are used to store data from different tables in the same physical data blocks. They are appropriate to use if the records from those tables are frequently queried together. By storing them in the same data blocks, the number of database block reads needed to fulfill such queries decreases thereby improving performance.

The columns within the cluster index are called cluster key that is the set of columns that the tables in the cluster have in common. Since the cluster key column determine the physical placement of rows within the cluster, the cluster key usually the foreign key of one table that references the primary key of another table in the cluster. Following syntax can be used:

#### Syntax:

create cluster <cluster name>(<column name>(<column datatype(size),column datatype(size),... .)

#### Example:

create cluster(workerskill(name varchar2(20));

This creates a cluster with nothing in it.Next,tables are created to be include in this cluster.

## Unit – 3: Oracle Database Objects, Concurrency control using lock

---

```
Create table worker
(
    Name varchar2(20),
    Age number,
    Lodging varchar2(20)
)
cluster workerskill(name);
```

### 3.7 Snapshot

A snapshot is recent copy of a table from database or in some cases, a subset of rows/columns of a table. The SQL statement that creates and subsequently maintains a snapshot normally reads data from database residing on the server. A snapshot is created on the destination system with the create snapshot SQL statement.

The query that create the snapshot closely resembles the code used to create a view. Snapshots are used to dynamically replicate data between distributed databases. The master table will be updatable but the snapshots can be either read-only or updatable. There are two types of snapshots available i.e. **complex snapshots** and **simple snapshots**.

In a simple snapshot, each row is based on a single in a single remote table. A row in complex snapshot may be based on more than one row in remote table, such as via a group by operation or on the result of multi-table join.

#### Creating a snapshot:

```
Create snapshot<snapshotname>
[Tablespace<tablespace>][<storageclause>]
[Refresh[fast/complete/force][start with<date>][Next<date>]]
As <select query>[For update];
```

#### Altering snapshot:

A snapshot is altered using ALTER SNAPSHOT statement. Usually storage and space parameters, types and frequency of refresh are altered using following syntax:

```
Alter snapshot<snapshotname>
[Tablespace<tablespace>][Storage <storageclause>]
[Refresh[fast/complete/force][start with <date>][Next<date>]];
```

#### Dropping snapshot:

```
Drop snapshot<snapshotname>;
```

### 3.8 What Are Locks?

In multi-user systems, many users may update the same information at the same time. Locking allows only one user to update a particular data block while another person cannot modify the same data.



## Unit – 3: Oracle Database Objects, Concurrency control using lock

---

The basic idea of locking is that when a user modifies data, that modified data is locked by that transaction until the transaction is committed or rolled back. The lock is held until the transaction is complete—this is known as **data concurrency**.

The second purpose of locking is to ensure that all processes can always access (read) the original data as they were at the time the query began (uncommitted modification). This is known as **read consistency**.

### 3.9 Locking Issue:

#### 3.9.1 Lost updates:

A lost update is a classic database problem. Actually, it is a problem in all multiuser environments. Simply put, a lost update occurs when the following events occur, such like:

- (1). A transaction in Session1 retrieves (queries) a row of data into local memory and displays it to an end user, User1.
- (2). Another transaction in Session2 retrieves that same row, but displays the data to a different end user, User2.
- (3). User1, using the application, modifies that row and has the application update the database and commit. Session1's transaction is now complete.
- (4). User2 modifies that row also, and has the application update the database and commit. Session2's transaction is now complete.

This process is referred to as a lost update because all of the changes made in Step 3 will be lost.

For example, an employee update screen that allows a user to change an address, city, work number, and so on. The application itself is very simple: a small search screen to generate a list of employees and then the ability to drill down into the details of each employee. This should be a piece of cake. So, we write the application with no locking on our part, just simple SELECT and UPDATE commands.

#### 3.9.2 Pessimistic Locking:

Pessimistic Locking prevents any other application or user from fetching or updating the same record at the same time. Pessimistic locking can be a very powerful mechanism, but often databases can impose limitations. Some databases, such as Oracle, lock a single record; others (for example, Sybase) lock a page or group of records; still others, including Microsoft Access, do not support pessimistic locking at all. Some databases, such as Oracle, prevent anyone else from fetching an object while others, such as Sybase, raise an exception when an attempt is made to update a record that has already been locked and updated.

When a user queries some data and picks a row to change the below statements are used:

#### Example:

```
Select empno, ename, sal from emp
where empno = :empno and ename=:ename and sal=:sal
for update nowait;
```

## Unit – 3: Oracle Database Objects, Concurrency control using lock

---

### 3.9.3 Optimistic Locking:

Optimistic locking is a technique for SQL database applications that does not lock rows for viewing, updating or deleting.

The rows with DML query changes are locked until changes are committed.

Since no locks are taken out during the read, it doesn't matter if the user goes to lunch after starting a transaction, and deadlocks are all but eliminated since users should never have to wait on each other's locks.

The Oracle database uses optimistic locking by default. Any command that begins with UPDATE...SET that is not preceded by a SELECT...FOR UPDATE is an example of optimistic locking. Concurrency control is the mechanism that ensures that the data being written back to the database is consistent with what was read from the database in the first place—that is that no other transaction has updated the data after it was read.

### 3.9.4 Blocking:

Blocking occurs when one session holds a lock on a resource that another session is requesting.

As a result, the requesting session will be blocked—it will hang until the holding session gives up the locked resource.

In almost every case, blocking is avoidable. The five common DML statements that will block in the database are INSERT, UPDATE, DELETE, MERGE and SELECT FOR UPDATE. The solution to a blocked SELECT FOR UPDATE is trivial: simply add the NOWAIT clause and it will no longer block. Instead, your application will report back to the end user that the row is already locked.

### 3.9.5 Deadlocks:

Deadlocks occur when you have two sessions, each of which is holding a resource that the other wants.

For example, if I have two tables, A and B, in my database, and each has a single row in it, I can demonstrate a deadlock easily. All I need to do is open two sessions (e.g., two SQL\*PLUS sessions). In session A, I update table A. In session B, I update table B. Now, if I attempt to update table A in session B, I will become blocked. Session A has this row locked already. This is not a deadlock: it is just blocking. I have not yet deadlocked because there is a chance that session A will commit or rollback and session B will simply continue at that point.

If I go back to session A and then try to update table B, I will cause a deadlock. One of the two sessions will be chosen as a victim and will have its statement rolled back. For example, the attempt by session B to update table A may be rolled back with an error such as the following:

**Update a set x= x+1;**

**ERROR at line 1:**

**ORA-00060: deadlock detected while waiting for resource**

Session A's attempt to update table B will remain blocked—Oracle will not roll back the entire transaction. Only one of the statements that contributed to the deadlock is rolled back. Session B still has the row in table B locked, and session A is patiently waiting for the row to become available. After receiving the deadlock message, session B must decide whether to commit the outstanding work on table B, roll it back, or continue down an alternate path and commit later. As

## Unit – 3: Oracle Database Objects, Concurrency control using lock

soon as this session does commit or roll back, the other blocked session will continue on as if nothing happened.

### 3.9.6 Lock Escalation:

When lock escalation occurs, the system is decreasing the granularity of your locks. An example would be the database system turning your 100 row-level locks against a table into a single table-level lock. Lock escalation is used frequently in databases that consider a lock to be a scarce resource and overhead to be avoided.

Oracle will take a lock at the lowest level possible(i.e., the least restrictive lock possible) and convert that lock to a more restrictive level if necessary. For example, if you select a row from a table with the FOR UPDATE clause, two locks will be created. One lock is placed on the row(s) you selected the other lock,a ROW SHARE TABLE lock, is placed on the table itself. This will prevent other sessions from placing an exclusive lock on the table and thus prevent them from altering the structure of the table, for example. Another session can modify any other row in this table without conflict. As many commands as possible that could execute successfully given there is a locked row in the table will be permitted.

Lock escalation is not a database “feature”. It is not a desired attribute. The fact that a database supports lock escalation implies there is some inherent overhead in its locking mechanism and significant work is performed to manage hundreds of locks.

**Take Note:**oracle will never escalate a lock. Oracle typically refers to the process as Lockconversion.

### 3.10 Lock Types

There a number of different types of locks as listed below:

- **DML Locks-** DML(data manipulation language), in general SELECT ,INSERT,UPDATE and DELETE. DML locs will be locks on a specific row of data, or a lock at the table level, which locks every row in the table.
- **DDL Locks-** DDL(data definition language).in general CREATE,ALTER and so on.DDL locks protect the definition of the structure of objects.
- **Latches-** These are locks that Oracle uses to protect its internal data structure.

#### 3.10.1 DML Locks:

Data manipulation language(DML) lock is placed over the row. This lock prevents other processes from updating(or locking) the row. This lock is released only when the locking process from updating(or locking) the row. This lock is released only when the locking process successfully commits the transaction to the database(i.e., makes the updates to that transaction permanent)or when the process is rolled back.

The complete set of DML locks are

<b>Row Share</b>	Permits concurrent access but prohibits others from locking table for exclusive access
<b>Row Exclusive</b>	Same as row share but also prohibits locking in share mode

## Unit – 3: Oracle Database Objects, Concurrency control using lock

Share	Permits concurrent queries but prohibits updates to the table
Share Row Exclusive	Prevent others from locking in share mode or updating the rows on the whole table
Exclusive	Permits queries but no DML against the table but select ok

**Example:**DML lock using NOWAIT

Select \* from employee for update nowait;

Select \* from employee for update wait 10;

**Take Note:** the above commands will abort if the lock is not release in the specified time period.

### 3.10.1 DDL locks:

Data dictionary language(DDL) lock is placed over the table to prevent structural alternations to the table. For example, this type of lock keeps the DBA from being able to remove a table by issuing DROP statement against the table. This lock is released only when the locking process successfully commits the transaction to the database or when the process is rolled back.

DDL locks are automatically placed against objects during a DDL operation to protect them from changes by other sessions.

There are three types of DDL locks

- **Exclusive DDL Locks:** These prevent other sessions from gaining a DDL lock or TM locks themselves. You can query a table but not modify it. Exclusive locks will normally lock the object until the statement has finished. However in some instances you can use the ONLINE option which only uses a low-level lock this still locks DDL operations but allows DML to occur normally.
- **Share DDL Locks:** This protects the structure of the referenced object against modification by other sessions, But allows modification to the data. Shared DDL Locks allow you to modify the contents of a table but not their structure.
- **Breakable Parse Locks:** This allows an object, such as a query plan cached in the shared pool to register its reliance on some objects. If you perform a DDL against that object, oracle will review the list of objects that have registered their dependence, and invalidate them. Hence these locks are breakable; they do not prevent the DDL from occurring. Breakable parse locks are used when a session parses a statement, a parse lock is taken against every object referenced by that statement. These locks are taken in order to allow the parsed, cached statement to be invalidated(flushed) in the shared pool if a reference is dropped or alter in some way. Use the below SQL to identity any parse locks on views, procedures, grants, etc

## Unit – 3: Oracle Database Objects, Concurrency control using lock

---

### 3.10.1 Latches:

Latches are locks that are held for short period of time, for example the time it takes to modify an in-memory data structure. They are used to protect certain memory structures such as the database block buffer cache or the library cache in the shared pool.

They are lightweight low-level serialization mechanism to protect the in-memory data structure of the SGA. They do not support queuing and do not protect database objects such as tables or data files.

Oracle uses atomic instructions like “test and set” and “compare and swap” for operating on latches. Since the instructions to set and free latches are atomic, the operating system itself guarantees that only one process gets to test and set the latch even though many processes may be going for it simultaneously. Since the instruction is only one instruction, it can be quite fast (but the overall latching algorithm itself is many CPU instructions!). Latches are held for short periods of time and provide a mechanism for cleanup in case a latch holder dies abnormally while holding it. This cleanup process would be performed by PMON.

It is possible to use manual locking using the FOR UPDATE statement or LOCK TABLE statement, or you can create your own locks by using the DBMS\_LOCK package.

### 3.10.1 Manual Locking and User-Defined Locks:

When we update a table, Oracle places a TM lock on it to prevent other session from dropping that table (or performing most DDL). We have TX locks that are left on the various blocks we modify so others can tell what data we own. The database employs DDL locks to protect objects from change while we ourselves are changing them. It uses latches and locks internally to protect its own structure.

There are two options for locking table

Manually lock data via a SQL statement.

Create our own locks via the DBMS\_LOCK package.

#### 3.10.4.1 Manual Locking:

The SELECT...FOR UPDATE statement is the predominant method of manually locking data. We can also manually lock data using the LOCK TABLE statement. This statement is used rarely, because of the coarseness of the lock. It simply locks the table, not the rows in the table. If you start modifying the rows, they will be locked as normal. So, this is not a method to save on resources (as it might be in other RDBMSs). You might use the LOCK TABLE IN EXCLUSIVE MODE statement if you were writing a large batch update that would affect most of the rows in a given table and you wanted to be sure that no one would block you. By locking the table in this manner, you can be assured that your update will be able to do all of its work without getting blocked by other transactions. It would be the rare application; however, that has a LOCK TABLE statement in it.

### 3.10.4.2 Creating Your Own Locks:

Oracle actually expose to developers the enqueue lock mechanism that it uses internally, via the DBMS\_LOCK package. You might use this package to serialize access to some resource external to Oracle. The DBMS\_LOCK package allows you to manually release a lock when you are done with it, or even to keep it as long as you are logged in.

MIKAS

## Unit – 4:Introduction to PL/SQL, Advanced PL/SQL

### 4.1 Introduction of PL/SQL:

- PL/SQL stands for Procedural Language/SQL. PL/SQL is the Oracle's procedural extension to SQL with design features of programming languages. It is a 'Block Structured' Language i.e., the basic units are logical blocks, which can contain any number of nested sub-blocks.
- The PL/SQL block may have variables, constants, and control structures, loop structures, error handling mechanism and so on.
- PL/SQL is processed by PL/SQL engine inside the Oracle server. The SQL statement executor processes the individual SQL statements, and the PL/SQL engine handles the PL/SQL program as a single unit.

### SQL Vs PL/SQL:

SQL	PL/SQL
Structured Query Language.	Programming language integrated with SQL.
SQL doesn't include all the things that normal programming languages have, such as loops, if etc.	PL/SQL is a normal programming language that includes all the features of most other programming languages.
SQL is executed one statement at a time.	PL/SQL is executed as a block of code.
SQL tells the database WHAT TO DO(declarative) not HOW TO DO.	PL/SQL tell the database HOW TO DO things(procedural).
SQL is used to code Queries,DML and DDL statements.	PL/SQL is used to code program blocks, triggers, functions, procedures and package.
You can embed SQL in a PL/SQL program.	But you cannot embed PL/SQL within a SQL statement.

### 4.2 PL/SQL Block Structure:

PL/SQL is a block structure language. A PL/SQL program may contain one or more blocks(called sub-blocks). Each block can be divided into three sections.

- **Declaration Section**(Optional)
- **Execution Section**
- **Exception Section**(Optional)

**Syntax:**

**[DECLARE**

**<Declaration of variables/constants/cursors and exception declaration>**

**[BEGIN**

**<Executable PL/SQL and SQL statements>**

### [EXCEPTION

<Actions for error conditions>]

END;

#### ➤ **DECLARATION SECTION:**

- Declaration section contains memory variable, constants and other objects declaration and initialization.
- This section begins with the keyword DECLARE.
- This section is optional if there is no variable, constant inside the block.
- Cursors and user defined exceptions are also declared in this section.

#### ➤ **EXECUTABLE SECTION:**

- The executable section is mandatory in PL/SQL block.
- Section begins with keyword BEGIN.
- This section can have other PL/SQL blocks inside it.
- Execution section contains SQL-PL/SQL executable statements.

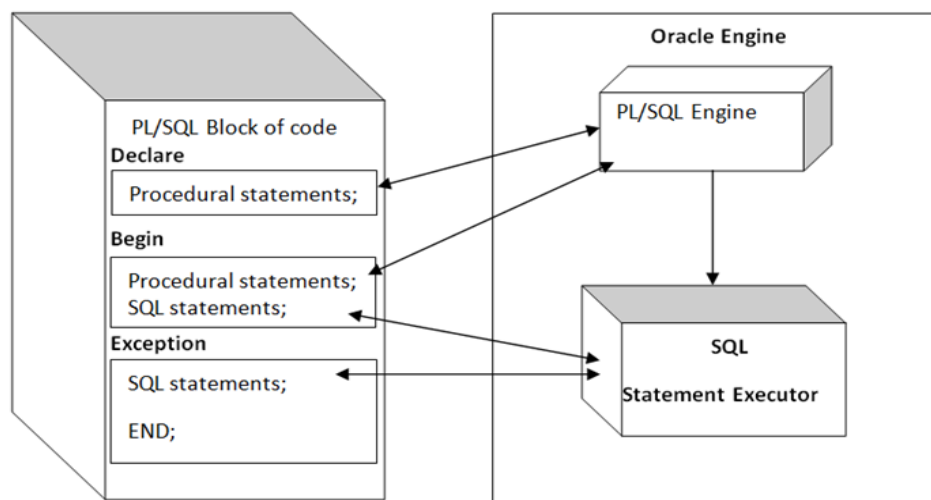
#### ➤ **EXCEPTION SECTION:**

- This is an Optional Section.
- Exception section contains code to handle errors that may arise in execution section.

### **Execution of PL/SQL Block**

The entire PL/SQL block is sent to Oracle engine for execution. Oracle engine sent procedural statements to PL/SQL engine and SQL statements are sent to the SQL executor in the Oracle engine. The PL/SQL engine and SQL Executor resides in the Oracle engine.

Following diagram gives an idea of how these statements are executed and how environment it is to bundle SQL code within a PL/SQL block. Since the Oracle engine is called only once for each block, the speed of SQL statement execution is very much improved, when compared to the Oracle engine being called once for each SQL sentence.





### Advantages of PL/SQL

**Block Structures:** PL SQL consists of blocks of code, which can be nested within each other. Each block forms a unit of a task or a logical module. PL/SQL Blocks can be stored in the database and reused.

**Procedural Language Capability:** PL SQL consists of procedural language constructs such as conditional statements (if else statements) and loops like (FOR loops).

**Better Performance:** PL SQL engine processes multiple SQL statements simultaneously as a single block, thereby reducing network traffic.

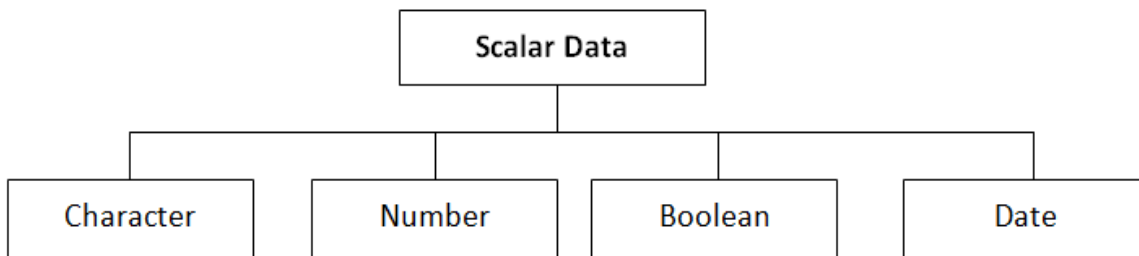
**Error Handling:** PL/SQL handles errors or exceptions effectively during the execution of a PL/SQL program. Once an exception is caught, specific actions can be taken depending upon the type of the exception or it can be displayed to the user with a message.

### 4.3 Language Construct Of PL/SQL:

PL/SQL has been categorizing into three sections of data types:

1. Scalar
2. LOB
3. Composite

A **Scalar** data type is not made up of a group of elements. It is atomic in nature. There are four major categories of scalar data types:



➤ **Character:**

Characters Data type can store text. The text includes letters, numbers, and special characters. Character data types include CHAR and VARCHAR2.

**CHAR:** The CHAR data type is used for fixed-length string values. The allowable length is between 1 and 32,767. If you do not specify a length for the variable, the default length is one.

if you specify a length of 10 and the assigned value is only five characters long, the value is padded with trailing spaces because of the fixed-length nature of this data type. The CHAR type is storage efficient, but it is performance efficient.

**VARCHAR2:** The VARCHAR2 is used for variable-length string values. Again, the allowable length is between 1 and 32,767. Varchar2 column can take 4000 characters, which is smaller than the allowable length for the variable.

## Unit – 4: Introduction to PL/SQL, Advanced PL/SQL

---

suppose you have two variables with data type of CHAR(20) and VARCHAR(20), and both are assigned the same value, 'Oracle Database'. The string value is only 15 characters long. The first variable, with CHAR(20), is assigned a value padded with five spaces.

Other character data types are LONG(32,760 bytes, shorter than VARCHAR2), RAW(32,767 bytes), and LONG RAW(32,767 bytes, shorter than RAW). The RAW data values are neither interpreted nor converted by oracle. VARCHAR2 is the most recommended character data type.

➤ **Number:**

NUMBER type can be used for fixed-point or floating point decimal numbers. It provides an accuracy of up to 38 decimal places.

**NUMBER(p,s);**

Here p is the precision of a number is the total number of significant digits in that number, and the s- scale is the number of significant decimal places.

If scale has a value that is negative, positive, or zero, it specifies rounding of the number to the left of the decimal place, to the right of the decimal place, or to the nearest whole number, respectively. If a scale value is not used, no rounding occurs.

➤ **Boolean:**

PL/SQL has a logical data types, Boolean, that is not available in SQL. It is used for Boolean data TRUE, FALSE, or NULL only.

➤ **Date:**

Date type stores data and time information. A user can enter a date in many different formats with the TO\_DATE() function, but a date is always stored in standard 7-byte format. The valid date range is from January 1,4712 B.C., to December 31,9999 A.D. The time is stored as the number of seconds past midnight. If the user leaves out the time portion of the data, it defaults to midnight(12:00:00 A.M.).

Varoious DATE functions are available for date calculations. For example, the SYSDATE function is used to return the system's current date.

**NLS:**

The National Language Support(NLS) is for character sets in which multiple bytes are used for character representation. NCHAR and NVARCHAR2 are examples of NLS data types.

**LOB:**

The LOB type store large values of character, raw, or binary data. It allow up to 4 gigabytes of data. LOB variables can be given one of the following data types:

- The BLOB type contain a pointer to the large binary object inside the database.
- The CLOB type contains a pointer to a large block of single-byte character data of fixed width.
- The NCLOB type contains a pointer to a large block of multibyte character data of fixed width.
- The BFILE type contains a pointer to large binary objects in an external operating system file. It would contain the directory name and the filename.

## Unit – 4: Introduction to PL/SQL, Advanced PL/SQL

A **COMPOSITE DATA TYPE** is made up of elements or components. PL/SQL supports three composite data types:- Records, Tables, and Arrays.

### Variables

Variables in PL/SQL blocks are named variable. A variable name must begin with a character and can be followed by a maximum of 29 other characters. Reserve words cannot be used as a variable name. Case is insignificant when declaring variable names. The assigning of a value to a variable can be done either using the assignment operator (:=) or selecting or fetching table data values into variables.

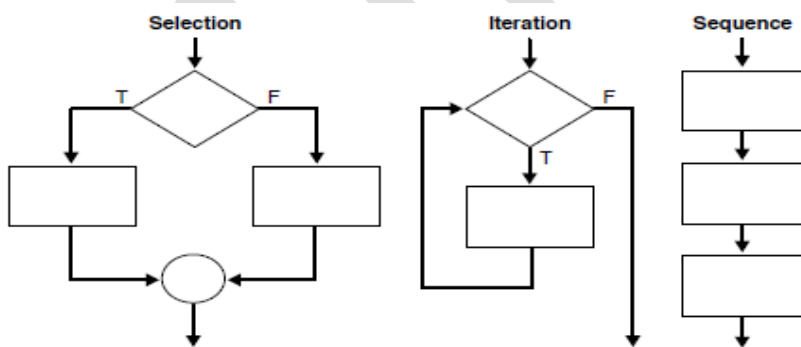
### RULES FOR VARIABLES IN PL/SQL:

- A variable must begin with a character and can be followed by a maximum of 29 other characters.
  - Reserved words cannot be used as variable names unless enclosed within double quotes.
  - Case (Capital or Small Letter) is not important when declaring variable names.
  - A space cannot be used in a variable name.
  - The assigning of a value to a variable can be done in two ways:
    - Using the assignment operator := (i.e. a colon followed by an equal to sign).
- For Ex. a:=10; or a:=&a;

### 4.4 CONTROL STRUCTURE:

Control structures are required to control the flow of PL/SQL program. It can be classified into the following categories:

- [1] Conditional Control
- [2] Iterative Control
- [3] Sequential Control



### [1] CONDITIONAL CONTROL:

PL/SQL allows the use of an IF statement to control the execution of a block of code.

PL/SQL has five conditional or selection statements available for decision making:

- (1) IF...THEN...END IF
- (2) IF...THEN...ELSE...END IF
- (3) IF...THEN...ELSE IF..END IF
- (4) CASE..END CASE

#### 1) IF...THEN...END IF:

## Unit – 4:Introduction to PL/SQL, Advanced PL/SQL

A simple IF statement performs action statements if the result of the condition is TRUE. If the condition is FALSE, no action is performed , and the program continues with the next statement in the block.

### Syntax:

```
IF <condition> THEN
<Action>;
END If;
```

### Example:

```
DECLARE n_sales NUMBER := 2000000;
BEGIN
  IF n_sales > 100000 THEN
    DBMS_OUTPUT.PUT_LINE( 'Sales revenue is greater than 100K ' );
  END IF;
END;
/
```

## IF THEN ELSEIF:

It is an extension of the simple IF statement. It provides action statements for the TRUE outcome as well as for the FALSE outcome.

### Syntax:

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

END IF;
```

### Example;

```
IF n_sales > 200000 THEN
  n_commission := n_sales * 0.1;
ELSIF n_sales <= 200000 AND n_sales > 100000 THEN
  n_commission := n_sales * 0.05;
ELSIF n_sales <= 100000 AND n_sales > 50000 THEN
  n_commission := n_sales * 0.03;
ELSE
  n_commission := n_sales * 0.02;
END IF;

END;
/
```

## 2) IF...THEN...ELSE IF...END IF:

It is an extension to the previous statemet. When you have many alternatives/options, you can use previously explained statements, but the ELSEIF alternative is more efficient than the other two.

### Syntax:

```
IF(boolean_expression 1)THEN
  S1; -- Executes when the boolean expression 1 is true
ELSIF( boolean_expression 2) THEN
```

## Unit – 4:Introduction to PL/SQL, Advanced PL/SQL

---

```
S2; -- Executes when the boolean expression 2 is true
ELSIF( boolean_expression 3) THEN
S3; -- Executes when the boolean expression 3 is true
ELSE
S4; -- executes when the none of the above condition is true
END IF;
```

**Example:**

```
DECLARE
a number(3);
BEGIN
a:=&a;
IF ( a = 10 ) THEN
dbms_output.put_line('Value of a is 10' );
ELSIF ( a = 20 ) THEN
dbms_output.put_line('Value of a is 20' );
ELSIF ( a = 30 ) THEN
dbms_output.put_line('Value of a is 30' );
ELSE
dbms_output.put_line('None of the values is matching');
END IF;
dbms_output.put_line('Exact value of a is: ' || a );
END;
/
```

### 3) CASE...END CASE:

The case statement is an alternative to the IF...THEN...ELSE IF...END IF statement. The CASE statement begins with keyword CASE and ends with the keywords END CASE. The body of the CASE statement contains WHEN clauses, with values or conditions, and action statements. When a WHEN clause's value/condition evaluates to TRUE, its action statements are executed.

**Syntax:**

```
CASE [variable_name]
WHEN value1/condition1 THEN action_statement1;
WHEN value2/condition2 THEN action_statement2;
...
WHEN valueN/conditionN THEN action_statementN;
ELSE action_statemet;
END CASE;
```

**Example:**

```
DECLARE
c_grade CHAR( 1 );
c_rank VARCHAR2( 20 );
BEGIN
c_grade := 'B';
CASE c_grade
WHEN 'A' THEN
c_rank := 'Excellent' ;
WHEN 'B' THEN
c_rank := 'Very Good' ;
WHEN 'C' THEN
```

## Unit – 4: Introduction to PL/SQL, Advanced PL/SQL

---

```
c_rank := 'Good' ;  
WHEN 'D' THEN  
  c_rank := 'Fair' ;  
WHEN 'F' THEN  
  c_rank := 'Poor' ;  
ELSE  
  c_rank := 'No such grade' ;  
END CASE;  
DBMS_OUTPUT.PUT_LINE( c_rank );  
  
END;
```

**Take note:** CASE program executed only in the newer versions. This topic will not work on Oracle 8i.

### [2] ITERATIVE CONTROL/LOOPING STRUCTURE:

Iterative control statements perform one or more statements repeatedly, either a certain number of times or until a condition is met. There are three forms of iterative structures:

- (1) BASIC LOOP
- (2) WHILE...LOOP
- (3) FOR...LOOP

#### (1) BASIC LOOP:

A basic loop is a loop that is performed repeatedly. Once a loop is entered, all statements in the loop are performed. When the bottom of the loop is reached, control shifts back to the top of the loop. The loop will continue infinitely. The only way to terminate a loop is by adding an EXIT statement inside the loop.

**Syntax:**

```
LOOP  
  <statements>;  
  EXIT [WHEN <condition>];  
  Increment statement;  
END LOOP;
```

**Example:** Generate a PL/SQL block to display 10 numbers.

```
DECLARE  
  i number(3) :=1;  
BEGIN  
  DECLARE  
    i number(3) :=1;  
  BEGIN  
    Loop  
    Exit when(i>5);  
    dbms_output.put_line(i);  
    i :=i+1;  
  end loop;  
END;  
/
```

**Output:**

1  
2  
3  
4  
5

**PL/SQL procedure successfully completed.**

### **(2) WHILE\_LOOP:**

While loop has a condition associated with the loop. The condition is evaluated and, if the condition is TRUE, the statements inside the loop are executed. If the condition is FALSE, execution continues from the next statement to END LOOP.

If the condition in the WHILE\_LOOP is evaluated to FALSE the very first time, then loop will be bypassed and will not be executed. Control will be transferred directly to the 'Next Statement'. The WHILE loop does not need an EXIT statement to terminate.

**Syntax:**

```
WHILE <condition>
LOOP
  <Loop body statement>
  Increment statement;
END LOOP;
```

**Example: generate a PL/SQL block to display numbers using WHILE LOOP**

```
DECLARE
  a number(2) := 10;
BEGIN
  WHILE a < 20 LOOP
    dbms_output.put_line('value of a: ' || a);
    a := a + 1;
  END LOOP;
END;
/
```

**Output:**

value of a: 10  
value of a: 11  
value of a: 12  
value of a: 13  
value of a: 14  
value of a: 15  
value of a: 16  
value of a: 17  
value of a: 18  
value of a: 19

PL/SQL procedure successfully completed.

Example 2:using exit in while loop

DECLARE

    n\_counter NUMBER := 1;

BEGIN

    WHILE n\_counter <= 5

    LOOP

        DBMS\_OUTPUT.PUT\_LINE( 'Counter : ' || n\_counter );

        n\_counter := n\_counter + 1;

        EXIT WHEN n\_counter = 3;

    END LOOP;

END;

/

Output:

Counter : 1

Counter : 2

PL/SQL procedure successfully completed.

➤ **Difference: Basic Loop and While loop**

Basic Loop	While Loop
It is performed as long as the condition is false.	Performed as long as the condition is true.
It is a post-test loop because it tests the condition inside the loop.	It checks condition before entering the loop so it is pretest loop.
it is performed at least one time.	It is performed zero or more time.
It needs the EXIT statement to terminate.	There is no need for an EXIT statement.

**(3) FOR...LOOP:**

We use FOR...LOOP if we want the iterations to occur a fixed number of times. The FOR...LOOP is executed for a range of values.

It provides Auto-Increment and Auto declaration of variables.

**Syntax:**



## Unit – 4:Introduction to PL/SQL, Advanced PL/SQL

---

**FOR<variable> IN [REVERSE] <start Range>...<End Range>**

**LOOP**

**<loop body statement>**

**END LOOP;**

**Take note:** Two dots(..) between Start and End range serves as a 'RANGE OPERATOR'. If the lower bound is larger than the upper bound in a FOR...LOOP, without using REVERSE keyword, the loop will not be executed at all.

**Example:**

```
                DECLARE

a number(2);

BEGIN

  FOR a in 10 .. 20 LOOP

    dbms_output.put_line('value of a: ' || a);

  END LOOP;

END;

/
```

**Output:**

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
value of a: 20
```

**PL/SQL procedure successfully completed.**

**[3] SEQUENTIAL CONTROL:**

**(1) The GOTO Statement:**

## Unit – 4: Introduction to PL/SQL, Advanced PL/SQL

---

The GOTO statement changes the flow of control within a PL/SQL block. A GOTO statement with a label may be used to pass control to another part of the program.

**Syntax:** GOTO label;

..

..

<< label >>

statement;

A GOTO statement can branch out of an IF statement, a loop, or a sub-block but it cannot branch into an IF statement, a loop, or a sub-block.

**Example:**

```

                                DECLARE

    a number(2) := 10;

BEGIN
    <<loopstart>>
    -- while loop execution
    WHILE a < 20 LOOP
        dbms_output.put_line ('value of a: ' || a);
        a := a + 1;
        IF a = 15 THEN
            a := a + 1;
            GOTO loopstart;
        END IF;
    END LOOP;
END;
/
```

**Output:**

value of a: 10

value of a: 11

value of a: 12

value of a: 13

value of a: 14

value of a: 16

value of a: 17

value of a: 18

value of a: 19

**PL/SQL procedure successfully completed.**

### 4.4 %Type and %Rowtype

#### **%type attribute**

- PL/SQL uses the %TYPE attribute to declare variables based on definitions of columns in a table so user doesn't need to remember data type and size of column of table.
- If a column's attributes change, the variable's attributes will change as well.
- This provides for data independence, reduces maintenance costs, and allows program to adapt to changes made to the table.

**Syntax:** Tablename.columnname%TYPE;

#### **%rowtype attribute**

- PL/SQL uses the %ROWTYPE attribute to declare variables based on definitions of entire column in a table so user doesn't need to remember data type and size of columns of table.
- If a column's attributes change, the variable's attributes will change as well.
- This provides for data independence, reduces maintenance costs, and allows program to adapt to changes made to the table.

**Syntax:** Variable name tablename%ROWTYPE;

### 4.5Using Cursor

- The Oracle Engine uses a work area for its internal processing in order to execute an SQL statement.
- This work area is called a Cursor. A cursor is opened at the client end.
- The data is stored in the cursor is called the Active Data Set.
- The size of the cursor in memory is the size required to hold the number of rows in the Active Data Set.

#### **Types of Cursors**

Cursors are classified into two types.

1. Implicit Cursor(SQL cursor:Opened and Managed by Oracle).
2. Explicit Cursor(User Defined Cursor: Opened and Managed by User).

#### **(1).Implicit Cursor:**

- ✓ A cursor that is created, opened and managed by Oracle Engine for its internal processing is known as Implicit Cursor.
- ✓ Implicit Cursor using SELECT statement returning one row of data. You cannot use implicit cursor for retrieving more than one row.

## Unit – 4: Introduction to PL/SQL, Advanced PL/SQL

- ✓ You can access information about the most recently executed SQL statement through SQL cursor attributes:-
- ✓ (Both Implicit and Explicit cursors have four attributes.)

Attribute Name:	Description
%ISOPEN	Returns TRUE if cursor is open, FALSE otherwise.
%FOUND	Returns TRUE if record was fetched successfully, FALSE otherwise.
%NOTFOUND	Returns TRUE if record was not fetched successfully, FALSE otherwise.
%ROWCOUNT	Returns number of records processed from the cursor.

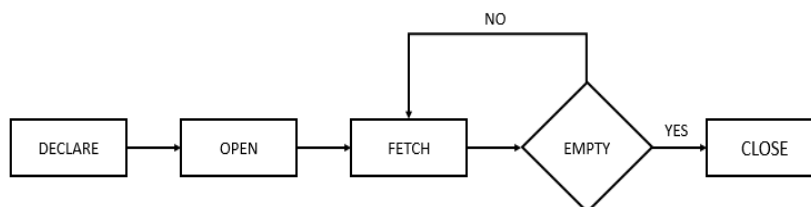
### (2). Explicit Cursor:

- For Queries that returns multiple rows, you have to explicitly create a Cursor.
- You have total control of when to open the cursor, when to fetch a row from it, and when to close it.

Four actions can be performed on an Explicit Cursor:

- Declare the Cursor
- Open the Cursor
- Fetch the data from the cursor one row at a time into PL/SQL memory variables.  
Process the data held in the memory variables using loop  
Exit from loop after processing is complete.
- Close the cursor.

#### ➤ Controlling Explicit Cursors:



#### ❖ Declaring a Cursor:

**Syntax:** `CURSOR <cursor_name> IS <select statement>;`

A Cursor is defined in the declarative part of a PL/SQL block. This is done by naming the cursor and mapping it to a query. When a cursor is declared, the oracle engine is informed that a cursor of the said name needs to be opened. There is no memory allocation at this point in time.

#### ❖ Opening a Cursor:

**Syntax:** `OPEN cursorname;`

Opening the cursor executes the query and creates the information active that contains all rows, which meet the query search criteria. An open statement retrieves records from a database table and places the records in the cursor.

### ❖ Fetching a Record from the Cursor:

**Syntax:** `FETCH cursor_name INTO var1,var2,...;`

The fetch statement retrieves the rows from the active data set opened in the server into memory variables declared in PL/SQL code block on the client one row at a time.

Each time a Fetch is executed, the cursor pointer is advanced to the next row in the Active Data Set. The Fetch statement is places inside a Loop...End Loop which causes the data to be fetched and processed until all the rows in Active Data Set are processed.

### ❖ Closing a Cursor:

**Syntax:** `CLOSE cursorname;`

#### ➤ Difference between Implicit cursor and explicit cursor

Implicit Cursor	Explicit Cursor
Maintained internally by PL/SQL Opened and Closed automatically when the query is executed.	Defined,opened and closed explicitly in the program. The cursor has a name.
The cursor attributes are prefixed with SQL(for e.g.,SQL%FOUND)	The cursor attributes are prefixed with the cursor name(for e.g., C1%FOUND)
The cursor attribute %ISOPEN is always FALSE, because the implicit cursor is closed immediately when the query completes.	The %ISOPEN attribute will have a valid depending upon the status of the cursor.
Only one row can be processed;the SELECT statement with the INTO clause is used.	Any number of rows can be processed. Iterative routines should be set up in the program, and each row should be fetched explicitly(except for a cursor FOR loop).

### 4.6Exception Handling:

- When user-defined or system related exception(errors) are encountered, the control of the PL/SQL block shifts to the Exception Handling section.
- An Exception Handler is nothing but a logical code block inside the memory to resolve the Exception condition is known as Exception Handler.

#### Types Of Exceptions:

(1) PREDEFINED EXCEPTION

(2) USER-NAMED EXCEPTION or INTERNAL EXCEPTION

(3) USER-DEFINED EXCEPTION or BUSINESS-LEVEL EXCEPTION

(4) CUSTOMIZE EXCEPTION

**(1).PREDEFINED EXCEPTION:**

The Oracle Engine has a set of pre-defined oracle error handlers called Named exceptions. These error handlers are referenced by their names.

The following are some of the pre defined oracle named exception handlers:

Exception Name	Oracle Error	Value
<b>(1)Login_denied</b> Raised when an invalid username/Password was used to log onto oracle	<b>ORA-01017</b>	-1017
<b>(2)No_DATA_FOUND</b> Raised when a select statement returns zero rows.	ORA-1403	+100
<b>(3)TOO_MANY_ROWS</b> Raised when a select statement returns more than one row.	ORA-01422	-1422
<b>(4)VALUE_ERROR</b> Raised when the datatype or data size is invalid.	ORA-06502	-6502
<b>(5)ZERO_DIVIDE</b> Raised when divided by zero.	ORA-01476	-1476
<b>(6)OTHERS</b> Stands for all other exceptions not Explicitly named.		

**Syntax:**

```
EXCEPTION
WHEN EXCEPTION NAME then
USER_DEFINED message;
```

**Example:**

```
DECLARE
c_id customer.id%type := 8;
c_name customer.Name%type;
c_sal customer.salary%type;
BEGIN
SELECT name, salary INTO c_name, c_sal
FROM customer
WHERE id = c_id;
DBMS_OUTPUT.PUT_LINE ('Name: ' || c_name);
DBMS_OUTPUT.PUT_LINE ('salary: ' || c_sal);
```

### EXCEPTION

```
WHEN no_data_found THEN
    dbms_output.put_line('No such customer!');
WHEN others THEN
    dbms_output.put_line('Error!');
END;
/
```

### (2).USER-Named EXCEPTION or INTERNAL EXCEPTION:

- User-named Exception is used to give user-defined name to the reserved exception number. For that pragma Exception\_init() function can be used.
- PRAGMA action word is a call to a Pre-compiler, which immediately binds the numbered Exception handler to a name when encountered.
- The Function EXCEPTION\_INIT() takes two parameters, the first is the user defined Exception name, the second is the oracle engine's Exception number. These lines will be included in the Declare section of the PL/SQL block.

#### Syntax:

```
DECLARE
    <Exception_name> EXCEPTION;
    Pragma exception_init(<Exception_name, <error code>);
BEGIN
    ....
EXCEPTION
    WHEN<Exception_name> then
        <action>;
END;
```

#### Example:

```
DECLARE
    cno customer.id%type;
    cname customer.name%type;
    csalary customer.salary%type;

    -- Exception name declared below
    already_exist EXCEPTION;
    -- pragma statement to provide name to numbered exception
    pragma exception_init(already_exist, -1);

BEGIN

    cno:=&cno;
    cname:=&cname;
    csalary:=&csalary;
    INSERT into customer values(cno, cname, csalary);
    dbms_output.put_line('Record inserted');
EXCEPTION
```

```
        WHEN already_exist THEN
            dbms_output.put_line('Record already exist');

END;
/
```

### (3).USER-DEFINED EXCEPTION or BUSINESS LEVEL EXCEPTION:

- User-defined error conditions must be declared in the declarative part of any PL/SQL block. If the condition exists, the call to the user-defined Exception is made using a RAISE statement.
- The Exception once raised is then handled in the Exception Handling section of the PL/SQL code block.

Three sections to handle/display Exceptions:

- I. Exception Declaration
- II. Raising the Exception
- III. Invoking the Exception

#### Syntax:

```
DECLARE
    <Exception_name> EXCEPTION;
BEGIN
    ....
    If<condition> then
        RAISE<Exception_name>;
    EXCEPTION
        When<Exception_name> then
            User defined action to be taken;
END;
```

#### Example:

```
DECLARE
    c_salary customer.salary%type;
    c_id customer.id%type;
    Sal_is_null Exception;
BEGIN
    c_id:=&c_id;
    select salary into c_salary from customer
```



```
        where id=c_id;
        if c_salary is null then
            raise sal_is_null;
        else
            update customer set salary=salary+100 where id=c_id;
        end if;
    EXCEPTION
        When sal_is_null then
            Dbms_output.put_line('salary is NULL');
    END;
/
```

### (4).CUSTOMIZE EXCEPTION HANDLING:

- It allows the user to handle the excetion as per user defined Exception.
- Comparison with the user defined exception is that in that previous type user has to follow the given tasks in three section i.e.,
  - (1) Declaration of Exp.
  - (2) Raising
  - (3) Invoking.

All these transaction will take more processing time than customize exception handling because in this type only user has to bind the exception in between the range.

-20001 to -20999 with user defined error messages.

**Syntax:** raise\_application\_error(-ve no,'error msg');

#### Example:

```
declare
eno emp.empno%type;
com emp.comm%type;

begin
eno:=&eno;
select comm into com from emp where empno=eno;
if com is null then
raise_application_error(-20995,'commision is null');
else
update emp set comm=comm+1000 where empno=eno;
end if;
end;
/
```

### 4.7 Creating and using Procedure,Functions:

#### Procedures/Functions:

- Set of SQL or PL/SQL statements which are logically grouped together to perform a specific task is known as Procedure or Function.

## Unit – 4: Introduction to PL/SQL, Advanced PL/SQL

---

- Procedures and functions are named PL/SQL programs that are stored in a compiled form in the database. At the time of creation, oracle compiles the procedures or function.
- Functions take zero or more parameters and return a value. Procedures take zero or more parameters and return no values. Both functions and procedures can receive or return zero or more values through their parameter lists.
- The primary difference between procedures and functions, other than the return value, is how they are called. Procedures are called as stand-alone executable statements.

➤ **Advantages/Benefits of using Procedure and Function:**

**(1) Security:** Procedure and function is stored in the form of objects, so who are granted to access those functions is permit only.

**(2).Performance:**

- Amount of information sent over a network is less.
- No need to compile each and every time whenever it is called.
- Reduction in disk I/O.

**(3).Memory Allocation/Requirement:** it provides reusability of code, so memory requirement is less. Only one copy of procedure needs to be loaded for execution by multiple users. Once a copy of a procedure or function is opened in oracle engine's memory. Other users who have appropriate permission may access it when required.

**(4).Productivity:** Redundant coding can be avoided, increasing productivity.

**CREATING PROCEDURE:**

**Syntax:**

**CREATE [OR REPLACE] PROCEDURE <Procedure Name>  
(<parameter List>) {is/as}**

**<declaration section>**

**BEGIN**

**<PL/SQL code>**

**EXCEPTION**

**<PL/SQL code>**

**END;**

**CALLING PROCEDURE:**

The procedure is called by specifying its name along with the list of parameters (if any) in parenthesis.

**Procedurename (parameter1,... . .)];**

**For example: p\_sum(20,30);**

Also , procedure can be called using EXECUTE (EXEC) command:

**For example: EXEC p\_sum(20,30);**

**Alter Procedure:**

**Alter Procedure<procedure\_Name> COMPILE;**

**Drop Procedure:**

## Unit – 4:Introduction to PL/SQL, Advanced PL/SQL

---

Drop Procedure<procedure\_Name> ;

### Keywords And Parameters:

The keywords and the parameters used for creating database procedures are explained below:

REPLACE	Recreates the procedure if it already exists. This option is used to change the definition of an existing procedure without dropping, recreating and re-granting object privileges previously granted on it. If a procedure is redefined the Oracle engine recompiles it.
Procedure	Is the name of the procedure to be created.
Argument	Is the name of an argument to the procedure. Parentheses can be omitted if no arguments are present.
IN	Indicates that the parameter will accept a value from the user.
OUT	Indicates that the parameter will return a value to the user.
IN OUT	Indicates that the parameter will either accept a value from the user or return a value to the user.
Data type	Is the data type of an argument. It supports any data type supported by PL/SQL
PL/SQL body	Is the definition of procedure consisting of PL/SQL statements.

### **CREATING FUNCTION:**

#### **Syntax:**

```
CREATE OR REPLACE FUNCTION <Fun Name>
(<parameter List>) RETURN <Data Type> {is/as}
<declaration section>
BEGIN
    <PL/SQL Code>
    RETURN <expression>
EXCEPTION
    <PL/SQL Code>
END;
```

## Unit – 4:Introduction to PL/SQL, Advanced PL/SQL

---

The main difference between a function and a procedure is that a function always returns a value to the calling block.

Primary use of a Function is to return a value with an explicit RETURN statement. The body can contain more than one RETURN statement, but only one is executed with each function call.

- **Function Header:**

The function header comes before the reserved word IS/AS. The header contains the name of the function, the list of parameters(if any), and the RETURN data type. For example:  
create or replace function emp\_chk(eno in number) return varchar2 is  
enm emp.ename%type

- **Function Body:**

The body must contain at least one executable statement. If there is no declaration, the reserved word BEGIN follows IS/AS. There can be more than one return statement, but only one RETURN is executed in a function call.

- **Calling a function:**

You call a function by mentioning its name along with its parameters(if any). A procedure does not have an explicit RETURN statement, so a procedure call can be an independent statement on a separate line. A function does return a value, so the function call is made via an executable statement, such as an assignment, selection, or output statement. For example:

**Total:=f\_sum(&n1,&n2);**

**Dropping Function:**

**DROP FUNCTION<Function\_name>;**

### 4.8 PACKAGE:

- A package is an Oracle object, which holds other objects within it.
- Objects commonly held within a package are procedures, functions, variables, constants, cursors and exceptions.
- The tool used to create a package is SQL\* Plus.
- It is a way of creating generic, encapsulated, re-useable code.
- A package once written and debugged is compiled and stored in an Oracle Database.
- All users who have executed permissions on the Oracle Database can then use the package.

### Components of an Oracle Package

A package has usually two components:

1. Package specification
2. Package body.

### [1]. PACKAGE SPECIFICATION:

## Unit – 4: Introduction to PL/SQL, Advanced PL/SQL

---

- Contains Name of the package.
- Contains Declaration of procedure, Function, Variables, Cursors, and Exception etc.
- It does not contain any code for procedure/function.
- Objects must be declared in this section before it is referenced.

**Syntax:**

**CREATE OR REPLACE PACKAGE Pack\_Name**

**{IS/AS}**

**PL/SQL Object Declaration;**

**END Pack\_Name;**

### [2]. PACKAGE BODY:

- Package body contains the definition of public objects that are declared in the specification.
- If package header does not contain any procedure/functions then package body is optional.

**Syntax:**

**CREATE OR REPLACE PACKAGE BODY Package\_Name**

**{IS/AS}**

**PL/SQL Object Body Part;**

**END Package\_Name;**

**Packages offer the following advantages:**

- Packages enable the organization of commercial applications into efficient modules.
- Each package is easily understood and the interfaces between packages are simple, clear and well defined.
- Packages allow granting of privileges efficiently.
- A package's public variables and cursors persist for the duration of the session.
- Therefore all cursors and procedures that execute in this environment can share them.
- Packages enable the overloading of procedures and functions when required.
- Packages improve performance by loading multiple objects into memory at once.
- Therefore, subsequent calls to related subprograms in the package require no I/O.
- Packages promote code reuse through the use of libraries that contain stored procedures and functions, thereby reducing redundant coding.

### **Execution of Package :**

When a package is invoked, the Oracle engine performs three steps to execute it:

- 1. Verify user access:** Confirms that the user has EXECUTE system privilege granted for the subprogram
- 2. Verify procedure validity:** Checks with the data dictionary to determine whether the procedure and functions used in package are valid or not.
- 3. Execute:** The package subprograms are executed.

### Syntax of Package Specification:

```
CREATE OR REPLACE PACKAGE package name IS  
Declaration of function;  
Declaration of procedure;  
Variable declaration;  
END packagename;
```

### Syntax of Package Body:

```
CREATE OR REPLACE PACKAGE BODY packagebodyname IS  
Definition of function;  
Definition of procedure;  
Variable declaration;  
END packagebodyname;
```

### **Example:**

```
Create or replace package pck_employee is  
Function get_deptnm (vempno in number) return number is;  
Procedure search_employee (vempno in number, vename out  
varchar2, vjob out varchar2) is;  
End pck_employee;
```

## **4.9 Triggers**

Database triggers are database objects created via the SQL\* Plus tool on the client and stored on the Server in the Oracle engine's system table. These database objects consist of the following distinct sections:

1. A named database event
2. A PL/SQL block that will execute when the event occurs

The Oracle engine allows the procedures that are implicitly executed by the user when an insert, update or delete is issued against a table. These procedures are called database triggers.

The major issues that make these triggers standalone are that, they are fired implicitly (i.e. internally) by the Oracle engine itself and not explicitly called by the user.

### **Use of Trigger:**

- A trigger can permit DML statements against a table only if they are issued, during regular business hours or on predetermined weekdays
- A trigger can also be used to keep an audit trail of a table (i.e. to store the modified and deleted records of the table) along with the operation performed and the time on which the operation was performed
- It can be used to prevent invalid transactions.
- Enforce complex security authorizations.

## Unit – 4:Introduction to PL/SQL, Advanced PL/SQL

**Note:** The PL/SQL block cannot contain transaction control SQL statements like COMMIT, ROLLBACK, and SAVEPOINT in trigger.

### Create Trigger:

```
Create[or Replace] trigger <trigger name>{Before | After}
{Delete , insert,update[of column]} on <table>
[referencing {Old as old, New as new}]
[for each row [when<condition>]]
Declare
    [Variable declaration]
    [Constant declaration]

Begin
    <PL/SQL Body>

Exception
    <Exception node>
END;
```

### Keywords and Parameters

The keywords and the parameters used for creating database triggers are explained below:

OR REPLACE	Recreates the trigger if it already exists. This option can be used to change the definition of an existing trigger without requiring the user to drop the trigger first.
Trigger Name	Is the name of the trigger to be created.
BEFORE	Indicates that the Oracle engine fires the trigger before executing the triggering statement.
AFTER	Indicates that the Oracle engine fires the trigger after executing the triggering statement
DELETE	Indicates that the Oracle engine fires the trigger whenever a DELETE statement deletes any row of table.
INSERT	Indicates that the Oracle engine fires the trigger whenever an INSERT statement inserts any row in table.
UPDATE[OF column]	Indicates that the Oracle engine fires the trigger whenever an UPDATE statement changes any value of table. "OF column" is optional part of trigger and specially used with UPDATE trigger. If the OF clause is added, the Oracle engine fires the trigger when the column mentioned with OF clause is affected by UPDATE Statement.
ON	Specifies the name of the table, which the trigger is to be created. A trigger cannot be created on a table in the schema SYS.
FOR EACH ROW	Designates the trigger to be a row trigger. The Oracle engine fires a row trigger once for each row that is affected by the triggering statement and meets the optional trigger constraint

## Unit – 4: Introduction to PL/SQL, Advanced PL/SQL

	defined in the WHEN clause. If this clause is omitted the trigger is a statement trigger.
REFERENCING	Specifies correlation names. Correlation names can be used in the PL/SQL block and WHEN clause of a row trigger to refer specifically to old and new values of the current row. The default correlation names are OLD and NEW. If the row trigger is associated with a table named OLD or NEW, this clause can be used to specify different correlation names to avoid confusion between table name and the correlation name.
WHEN	Specifies the trigger restriction. The trigger restriction contains a SQL condition that must be satisfied for the Oracle engine to fire the trigger. This condition must contain correlation names and cannot contain a query. Trigger restriction can be specified only for the row triggers. The Oracle engine evaluates this condition for each row affected by the triggering statement

### Drop Trigger

Drop Trigger <trigger\_name>;

### 4.10 Creating Objects

Object-oriented programming is especially suited for building reusable components and complex applications. In PL/SQL, object-oriented programming is based on object types. They let you model real-world objects, separate interfaces and implementation details, and store object-oriented data persistently in the database.

#### Declaring and Initializing Objects in PL/SQL

An object type can represent any real-world entity. For example, an object type can represent a student, bank account, computer screen, rational number, or data structure such as a queue, stack, or list.

#### Example:

```
CREATE TYPE address_typ AS OBJECT (  
    street VARCHAR2(30),  
    city VARCHAR2(20),  
    state CHAR(2),  
    postal_code VARCHAR2(6) );
```

### 4.11 Collections

Collections are similar to Arrays in other languages. A collection is an ordered group of elements. Collections must be defined with the TYPE statement, and then variables of that type can be created and used.

#### Types of Collections:

- (1) PL/SQL Tables or Index-by tables
- (2) Nested Tables
- (3) VARRAYs



## Unit – 4: Introduction to PL/SQL, Advanced PL/SQL

---

### (1) PL/SQL Tables or Index-by tables:

- A PL/SQL is a one-dimensional, indexed by integers.
- It looks like an ARRAY but it is not exactly the same. There is a difference between ARRAY and PL/SQL table.
- Key difference between arrays and PL/SQL tables is that a row in a PL/SQL table does not exist (Memory is not allocated) until you assign a value to that row.
- If you reference a row which does not a part of SQL . we cannot issue commands like INSERT/UPDATE/DELETE etc. on it.
- PL/SQL tables cannot be stored in the database.

Syntax:

Step:1:

```
TYPE <table_type_name> IS TABLE OF <data type> INDEX BY  
BINARY_INTEGER;
```

Step:2:

```
<table_name><table_type>;
```

#### ➤ Characteristics of PL/SQL TABLE:

- (1) **One-Dimensional:** A PL/SQL table can have only one column. Hence it is similar to a one-dimensional array.
- (2) **Unbounded/Unconstrained:** There is no predefined limit to the number of rows in a PL/SQL table. The PL/SQL table grows dynamically as you add more rows to the table.
- (3) **Sparse:** in a PL/SQL table, a row exists in the table only when a value is assigned to that row. Rows do not have to be defined sequentially. That is why it is called sparse.
- (4) **Homogeneous Elements:** All rows in a PL/SQL table contain values of the same data type.
- (5) **Indexed By BINARY\_INTEGER:** the memory of BINARY INTEGER is 4 bytes. Hence the range is very large.

Example:

```
DECLARE
```

```
    Type t_char is table of varchar2(10) index by binary_integer;
```

```
    V_char t_char;
```

```
    --v binary_integer;
```

```
BEGIN
```

```
    V_char(0):='SCOTT';
```

```
    V_char(1):='SMITH';
```

```
    Dbms_output.put_line(v_char(0));
```

```
    Dbms_output.put_line(v_char(1));
```

```
    Dbms_output.put_line(v_char.first);
```

```
    Dbms_output.put_line(v_char.last);
```

```
    Dbms_output.put_line(v_char.next(0));
```

```
    V_char.delete(1);
```

```
    Dbms_output.put_line('deleted');
```

```
END;
```

### (2) Nested Table:

- Varray have a limited number of entries, a second type of collector is "Nested Table".

## Unit – 4: Introduction to PL/SQL, Advanced PL/SQL

---

- Nested Table has no limit on the number of entries per row.
- A nested table is, as its name implies, a table within a table.
- It is a table that is represented as a column within another table.
- You can have multiple rows in the nested table for each row in the main table.

### Syntax :

```
Create or replace type <table name> as table of <user-defined type>
Create table <table name>
(
    Columnname datatype(size),
    columnname nested table
)
nested table <column name contain nested table data type>
store as <table name>;
```

### Example:

```
Create or replace type stud_ty as object
(
    Name varchar2(15),
    Add1 varchar(30),
    City varchar2(15)
);
Create type stud_nt as table of stud_ty;
Create table studrec
(
    Stream varchar2(15),
    Studdet stud_nt
)
Nested table studdet store as stud_tab;
```

### Inserting record into nested table:

```
Insert into studrec values ('BCA4', stud_nt(stud_ty('xyz', 'jagnath plot', 'rajkot'),
stud_ty('abc', 'kalawad road', 'rajkot'),
stud_ty('pqr', 'university road', 'rajkot')
));
```

- Varray cannot be indexed, while nested table can be indexed.
- In varray, the data in the array is stored with the rest of the data in the table.
- While in nested table, the data may be stored out-of-line.

### SELECTING ROW FROM NESTED TABLE

- To support queries of the columns and rows of a nested table, oracle provides a special keyword, THE.

### EXAMPLE:

```
Select name from the (select studdet from studrec where stream like 'BCA4');
```

### (3) Varrays:

- Varray is known as collector means sets of elements that are treated as part of a single row.
- A varying array allows you to store repeating attributes of a record in a single row.
- For example, suppose Dona wants to track which of her tools were borrowed by which of her worker.
- You should create a table to keep track of this.

#### Example:

```
Create table borrower
(
  name varchar2(25),
  tool varchar2(25)
);
```

- Thus, if a single worker borrowed three tools, the worker's name would be repeated in each of the three records.
- Collection such as varying arrays allows you to repeat only those column values that change, potentially saving storage space.

#### Creating a Varray:

- You can create varray based on either an abstract datatype or one of the oracle standard data type.
- To create varray use the 'as varray ( )' clause of the create type command.

#### Syntax:

Create or replace type as varray (size) of data type (size)

#### Example:

Create or replace type tools\_va as varray (5) of varchar2 (20);

#### Inserting records into Varray ( ):

- When a data type is created, the database automatically creates a method called a constructor method for the data type.
- Since a varying array is an abstract data type, we need to use constructor method to insert records into tables that use varying arrays.

**Example:** Insert into borrower values ('Neel',tools\_va('hammer','ax','sledge',null,null));

- This insert into first specifies the value of the name column.
- Because the name column is not a part of abstract data type.

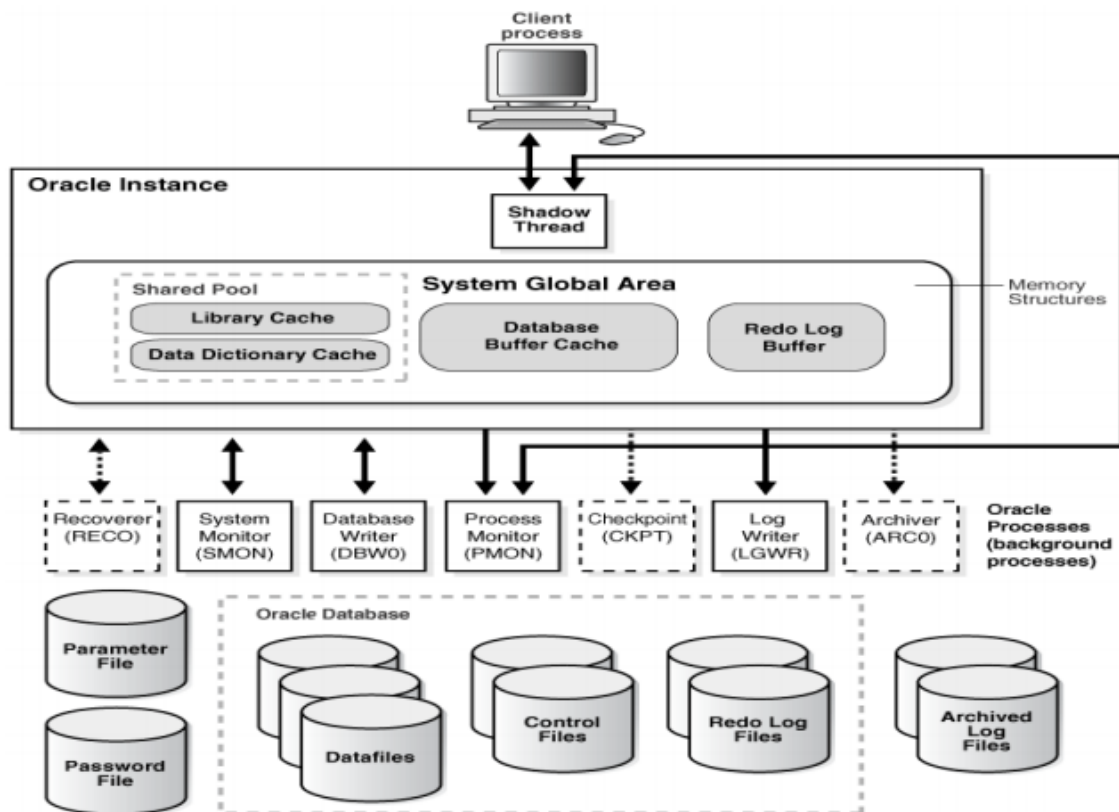
#### Selecting data from varray ( )

- The maximum number of entries per row, called its LIMIT, and the current number of entries per row, called its COUNT.
- This query cannot be performed directly via a SQL select statement.
- To retrieve the COUNT AND LIMIT from varray you need to use PL/SQL.

MIKACC

### 5.1 Instance Architecture (Database processes, Memory Structure, Data files)

The instance is the method used to access the data and consists of process and system memory. It consists of oracle process and shared memory necessary to access information in the database. It is made up of the user processes, the oracle background processes and the shared memory used by these processes, shown in below figure:



**The Oracle memory structure:** Oracle uses shared memory for caching of data and indexes, storing shared procedures. The shared memory is broken into various pieces (i.e. System Global Area (SGA) and the Program Global Area (PGA))

**SGA:** It is a shared memory region, where the data and control information for one oracle instance will be shared. The SGA is allocated when the Oracle instance starts and de allocate when the Oracle instance is shutdown. For each instance, there is one separate SGA. SGA consists the following elements:

**Database buffer cache:** It is used to store the most recently used data blocks. This blocks can contains modified data that has not yet been written to the disk (called dirty block), block that have been written to disk since modification (called clean box) or block that have not been modified. When user update a row in a table, a foreground server process reads the data block that contains the row from a data file on disk into the buffer cache. Then, the server process can modify the data block in server memory.

**Redo log buffer:** It stores a log of changes made to the databases. The redo log buffers are written to the redo log file as quickly as possible.

**The shared pool:** It uses the library cache and data dictionary which stores the shared SQL areas and internal information respectively.

**Library Cache:** It is used to store shared SQL. Here the parse tree and the execution plan for every unique SQL statement are cached. It stores and shares parsed representation of the most recently executed SQL statements and PL/SQL programs. Suppose when user issues a SQL statement, Oracle parses the statement and determines the most efficient execution plan for the statement, then Oracle caches the statement in the shared pool. If another user issues the same statement, Oracle can share the statement already in memory rather than perform identical steps necessary to execute the statement again by caching the statements in the memory, Oracle's shared SQL reduces the server overhead.

**Data dictionary cache:** It contains a set of tables and views that Oracle uses as a reference to the database. Physical and logical structure of database is stored here. It also contains

- User information, such as user privileges.
- Integrity constraints defined for tables in the database.
- Names and data types of all columns in the database.
- Information on space allocated and used for schema objects.
- Oracle frequently accesses the data dictionary for the parsing of SQL.

**Processes:** The Oracle RDBMS uses two types of processes i.e. user process and Oracle process (called background process)

**[1] User processes:** It is the user's connection to the RDBMS system. It communicates user input with the Oracle server process by using Oracle program interface. It is also used to display the information requested by the user and also compiles this information into a more useful form.

**[2] Oracle processes:** It performs the function for users. It is divided into two processes:

[a] Server processes (Shadow processes)

[b] Background processes

**[a] Server processes:** Which perform functions for invoking process? It communicates with user and interacts with Oracle to carry out the user's request. If the user process needs a data which is not available in SGA, the shadow process is responsible for reading the data blocks from the data files into the SGA. There is one-to-one correspondence between user processes and shadow processes. Although one shadow process can connect to multiple user processes, which reduces the utilization of system resource.

**[b] Background processes:** These processes are used to perform the various tasks within the RDBMS system. Following are the different Oracle background processes:

**DBWR (Database Writer):** It is responsible for writing dirty blocks from the database block buffer to disk. When a transaction changes data in a data block, the effect will not immediately be reflected to the disk, but DBWR writes this data in a more efficient way to the disk. The DBWR usually writes only when the database block buffers are needed for data to be read. If a system contains asynchronous I/O (AIO), then there is one DBWR.

**LGWR (Log writer):** The LGWR process is responsible for writing data from the log buffer to the redo log file. The log writer process records information about the changes made by all transactions that commit. Oracle performs transaction logging as follows:

A user carry out a transaction, oracle creates small search called redo entries that contain just enough Information necessary to regenerate the changes made by the transactions.

- Oracle temporarily stores the transactions redo entries in the server's redo log buffer.
- When the transaction is committed, LGWR reads the corresponding redo entries in the redo log buffer and writes them to the database transaction log. The database's transaction log is a set of files dedicated to logging the redo entries created by all system transaction.

**CKPT (Checkpoint):**A check point is an event in which all modified database buffers are written to the data files by the DBWR. It is responsible for signaling DBWR process to perform a checkpoint and to update all the data file and control file. The CKPT is optional. If the CKPT process is not present, the LGWR assumes these responsibilities. The purpose of check point is to establish mileposts of transaction consistency on disk. After performing a checkpoint, the changes made by all the committed transactions have been written to the database's data files. Therefore, a checkpoint indicates how much of the transaction log's redo entries oracle must apply if a server crash occurs and database recovery is necessary. During the checkpoint, oracle must also update the headers in all of the database's data files to indicate the checkpoint. The checkpoint process has two methods i.e. the normal checkpoint and the fast checkpoint.

- a) **Normal check point:**In normal checkpoint, the DBWR writes a few more buffers every time it is active. This type of check point takes much longer but affects the system less.
- b) **Fast check point:**In the fast checkpoint, the DBWR writes a large number of buffers at the request of the check point each time it is active. It is much faster and more efficient in terms of I/O, but it has great effect on system performance at the time of the checkpoint.

**PMON (Process monitor):** It is responsible for keeping track of database processes and cleans up the cache and frees resources that might still be allocated. It is also responsible for restarting any dispatcher processes that might have failed. For example a user connection do not end gracefully, means a network error might unexpectedly disconnect the user database session before he can disconnected from oracle. An instance's process monitors process notice when user connections have been broken. PMON cleans up after orphaned connections by rolling back a dead session's resources that might otherwise block other users from performing database work.

**SMON (System Monitor):** It performs instance recovery at instance startup. This includes cleaning temporary. It also performs many internal operations. It periodically coalesces the free extents in a table space's data files to create large free extents.

**RECO(Recovery):**It is used to clean transaction that were pending in a distributed database. RECO is responsible for committing or roiling back the local portion of the disputed transactions.

**ARCH (Achiever):** It is responsible for copying the online redo log files to archival storage when they become full. The archive process automatically back up the transaction log files after LGWR fills them with redo entries. The sequential set of archive transaction log files that ARCH creates is collectively called the database's archived transaction log.

**LCKn (Parallel server lock):** Up to 10 LCK processes are used for inter instance locking when the Oracle parallel server option is used.

**Dnnn (Dispatcher):** It is used for every communications protocol in use.

## 5.2 Creating and Altering Database

create database test

logfile group 1 ('/path/to/redo1.log') size 100M,

group 2 ('/path/to/redo2.log') size 100M,

group 3 ('/path/to/redo3.log') size 100M

character set WE8ISO8859P1

national character set utf8

datafile '/path/to/system.dbf' size 500M autoextend on next 10M maxsize unlimited  
extent management local

sysaux datafile '/path/to/sysaux.dbf' size 100M autoextend on next 10M maxsize unlimited

undo tablespace undotbs1 datafile '/path/to/undotbs1.dbf' size 100M

default temporary tablespace temp tempfile '/path/to/temp01.dbf' size 100M;

## 5.3 Opening and Shutdown Database

### Startup:

The three steps to starting an Oracle database and making it available for systemwide use are: Start an instance, Mount the database, Open the database. A database administrator can perform these steps using the SQL\*Plus STARTUP statement or Enterprise Manager.

### Shutdown:

The three steps to shutting down a database and its associated instance are: Close the database, Unmount the database, Shut down the instance. A database administrator can perform these steps using the SQL \*Plus SHUTDOWN statement or Enterprise Manager. Oracle automatically performs all three steps whenever an instance is shut down.

## 5.4 Initialization Parameter

The parameter file (sometimes called init.ora) contains configuration information for the database to use at startup time. The parameter file you configure how much RAM the database is going to use, where to find the control files, where to write trace files, and a whole host of other information. In most cases the database will not start without a parameter file. Oracle allows you to have a manual parameter file (called a PFILE) or a server-side parameter file (called a SPFILE). It is necessary to create a new parameter file for each new database. The parameter file contains important information concerning the structure of the database of the oracle tuning parameters is described as follow:

**DB\_NAME:** This parameter specifies the name of the database. It is a string of eight or fewer characters. It is similar to SID. The default database is built with DB\_NAME=oracle.



**DB\_DOMAIN:** This parameter specifies the network domain. With the help of DB\_NAME parameters, this parameter is used to identify the database over network. The default database was built with DB\_DOMAIN=WORLD.

**CONTROL\_FILES:** This parameter specifies one or more control files to be used for this database.

**DB\_BLOCK\_SIZE:** This parameter specifies the size of the oracle data block. The data block is the smallest unit of space with the data files or in memory. It affects the performance. The default size 2KB. After the database is built the block size cannot be changed.

**DB\_BLOCK\_BUFFERS:** This parameter specifies the number of blocks to be allocated in memory for database caching. It also affects the performance.

**PROCESSES:** This parameter specifies the number of processes or threads that can be connected to the oracle. It must include five extra processes to account for the background processes.

**ROLLBACK\_SEGMENT:** This parameter specifies the list of rollback segments.

## **5.5 Control Files and Redo Log Files**

### **5.5.1 CONTROL FILE:**

- This is a very important file that is required for the oracle database to function.
- If any one of the control files is unavailable, the database is shutdown.
- Hence it is recommended that multiple copies of the control files are maintained in the database or separate disks.
- The control file keeps a record of the names, size and locations of different physical files of the database.
- It contains the information used to start an instance, such as the location of the data file and redo log files.
  - Oracle needs this information to start the database instance.
  - Control files must be protected.
    1. The entries maintained in the control file are:
    2. The database identifier and name.
    3. Time of database creation.
    4. Table space name.
    5. Name and location of data files and online redo log files.
    6. Current online redo log file sequence number.

### **5.5.2 REDO LOG FILE:**

- Redo log files hold information used for recovery in the event of a system failure.
- Redo log files store a log of all changes made to the database.
- This information is used in the event of a system failure.

- It redo log information is lost, you cannot recover the system.
- Since the redo log files are very important for recovery purpose, oracle recommends that redo logs be multiplexed.
- Multiplexing is maintaining multiple copies.
- Each copy of redo log files should be of the same size and placed on separate disks.
- This will prevent loss of data in redo file in the event of the loss of a disk.
- You can transfer the redo entries to another media before overwriting them; this process is known as archiving.
- Archiving can be done automatically every time a redo log file becomes full and LGWR starts writing to another file.
- The event when LGWR stops writing to one file and starts writing to another file is called a Log Switch.

### **5.6 Tablespace (Create, Alter, Drop)**

- The database is divided into one or more logical pieces known as tablespaces.
- A tablespaces is used to logically group data together.
- For the different application one can create a separate tablespace.
- At the time of database creation it will automatically create SYSTEM tablespace.
- Which is used to store important internal structures such as data dictionary, the system stored procedures.
- The SYSTEM tablespace is used as the default for all database users, which is not desirable.
- The Oracle tablespace is the lowest logical layer of the oracle data structure.
- The tablespace consists of one or more data files.
- Tablespace is made read-write by default, but it can be altered to become read-only.
- A tablespace can consist of 1022 data files.

A tablespace can create with the following options

**Online:** This option specifies that the tablespace be brought online after creation it can be used immediately.

**Offline:** It specifies that the tablespace is left offline after creation.

**Read only:** It specifies that tablespace is read only. It is of no meaning to make the tablespace read only at the time of creation. Create the tablespace and populate them, then if desire makes the tablespace readOnly.

**Permanent:** This specifies that the tablespace is for permanent objects. This is the default parameter. This option is used for all schema objects except for temporary tablespaces.

**Temporary:** This specifies that the tablespace is for temporary objects. Create Tablespace command is used to create tablespace. One has to specify the name of the data file(s) that the tablespace has.

**EXAMPLE:** CREATE TABLESPACE msc DATAFILE '\disk03\mymscfile1.dbs' SIZE 10M permanent;

This command creates the tablespace and makes it immediately available. After this command is issued, one will find a file at the specified location and with the specified size.

### **5.7 Rollback Segments (Create, Alter)**

Use the CREATE ROLLBACK SEGMENT statement to create a rollback segment, which is an object that Oracle Database uses to store data necessary to reverse, or undo, changes made by transactions. The information in this section assumes that your database is not running in automatic undo mode (the UNDO\_MANAGEMENT initialization parameter is set to MANUAL or not set at all). If your database is running in automatic undo mode (the UNDO\_MANAGEMENT initialization parameter is set to AUTO), then user-created rollback segments are irrelevant. Further, if your database has a locally managed SYSTEM tablespace, then you cannot create rollback segments in any dictionary-managed tablespace. Instead, you must either use the automatic undo management feature or create locally managed tablespaces to hold the rollback segments. The following syntax is used to create Rollback Segments:

```
Create [Public] Rollback Segment<name>
{Tablespace<tablespace> | storage};
```

#### **Example1:**

```
CREATE TABLESPACE rbs_ts
DATAFILE 'rbs01.dbf' SIZE 10M
EXTENT MANAGEMENT LOCAL UNIFORM SIZE 100K;
/* this example and the next will fail if your database is in automatic undo mode.*/
CREATE ROLLBACK SEGMENT rbs_one TABLESPACE rbs_ts;
```

#### **Example2:**

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

#### **Alter Rollback Segments:**

```
Alter Rollback segment<name>
{Online | Offline | Storage | Shrink to<size> };
```

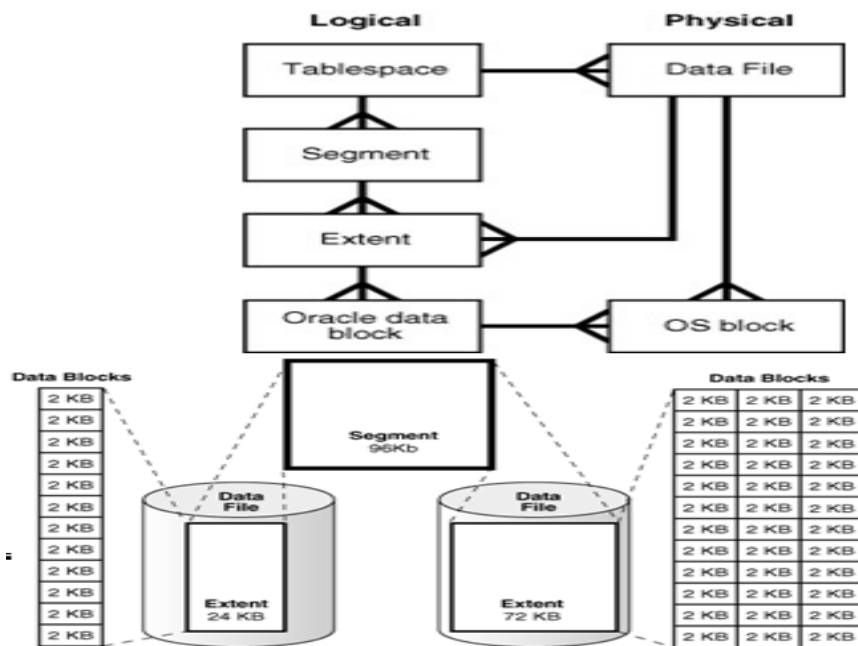
#### **Example1:**

```
ALTER ROLLBACK SEGMENT rbs_one ONLINE;
```

#### **Example2:**

```
ALTER ROLLBACK SEGMENT rbs_one SHRINK TO 100 M;
```

## 5.8 Oracle Blocks



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. Following figure shows 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 above 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. 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. Use the CREATE TABLESPACE statement to create a tablespace, which is an allocation of space in the database that can contain schema objects.

## 5.9 Import

- The import utility allows you to restore the database information held in previously created Export files.
- It is the complement utility to Export. Import loads data in the following order:

- i. Table definitions
- ii. Table Data
- iii. Table Indexes
- iv. Triggers/Constraints/Bitmap Index

- First, new tables are created.
- Then data is imported and indexes are built.
- Then triggers are imported, integrity constraints are enabled on the new tables, and any bitmap and functional indexes are built.
- There are three modes of imports

Mode	Description
User	Imports all objects owned by a user.
Table	Imports all or specific tables owned by a user along with index, constraints and triggers.
Database	Imports all objects of the database

- By default, import commits only after loading each table, and it performs a rollback when an error occurs, before continuing with the next object.

**Syntax:**

```
Imp<username/password>file=<file name>from user=<user name>  
table=<table name>
```

**5.10 Export**

Export is logical backup of database.

- This utility copy the data and database to a binary OS file in special format.
- Export files store information about schema objects created for database.
- Using this utility we can backup database while it is open and avail for use.
- There are three modes of export.

Mode	Description
User	Export all objects owned by a user
Table	Export all or specific tables owned by a user along with index, constraints and triggers
Database	Exports all objects of the database except the one owned by SYS.

- There are three types of export:

Incremental Export	Only database data that has changed is exported. For example if tables A, B and C exist, and only table A's information has been modified since the last incremental export, only table A is exported.
--------------------	--

Cumulative Export	Only database data that has been changed since the last cumulative or complete export is exported. For example if tables A, B and C exist and only table A's and table B's information has been modified since the last cumulative export, only the changes to table A and B are exported.
Completed Export	All database data is exported. (usually once a month)

**Syntax:**

Exp username/password inctype=<incremental/cumulative/complete>  
File=<export\_file name>

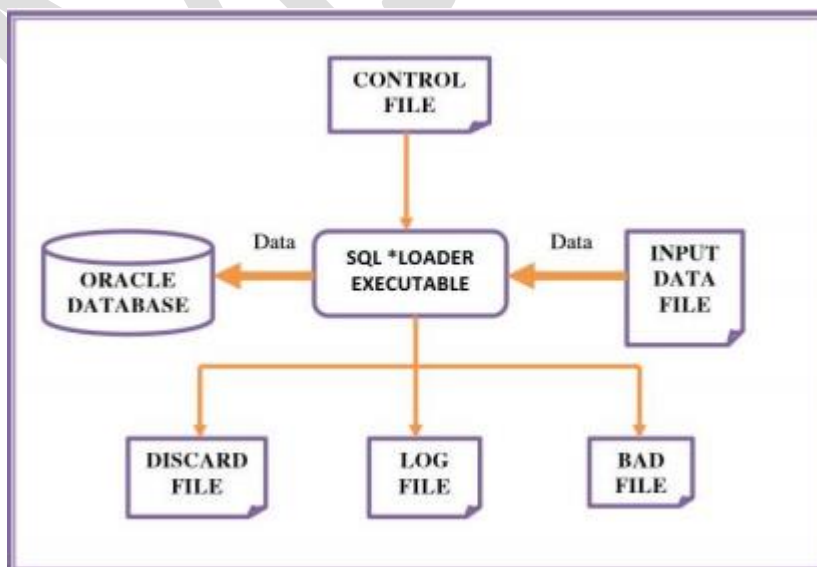
**5.11 SQL \*Loader:**

SQL \*LOADER is high speed data loading utility that loads data from external files into tables in an oracle database.

- It is the only way you can load data created in another DBMS into oracle.
- SQL \*LOADER loads the table using a control file.
- The Data may be in a separate file or may be included in the control file.
- The data may be in the fixed length fields.

**Features:**

1. Loads data into multiple files.
2. Loads fixed/variable length data.
3. Converts data to oracle data types.
4. Combine more than one physical record into one logical record.
5. Break single physical record into multiple logical records.
6. Generates unique keys via sequence generator.
7. Use SQL function before data insert.



**SQL \*LOADER FILES:**

- 1) Control File
- 2) Log File
- 3) Bad File
- 4) Discard File
- 5) Data File

**Control file:**

- The control file is a text file written in a language that SQL \*LOADER understands.
- The control file tell the SQL \*LOADER where to find the data, how to parse and interpret

the data, and where to insert the data.

The control file provides the following information to the SQL \*LOADER:

- The name and location of the input data file.
- The format of the records in the input data file.
- The name of the table or tables to be loaded.
- The name and the location of the bad file and discard file.

**Log file:**

- When SQL \*LOADER begins execution it creates a LOG FILE.
- If it cannot create a log file, execution terminates.
- The log file contains detailed summary of the load, including a description of any errors.

Log file contains following information:

- The names of the control file, log file, bad file, discard file and data file.
- The values of several command line parameters.
- Error messages for records that cause errors.

**Bad File:**

- The bad file contains the records rejected, either by SQL \*LOADER or by ORACLE.
- Lack of free space in a table space, can also cause insert operation to fail.
- Whenever SQL \*LOADER encounters a database error while trying to load a record, it writes that record to a file known as the BAD file.

**Discard File:**

- The discard file contains records that were filtered out of the load because they did not match any record selection criteria specified in the control file.

**5.12 Managing Automated Database Maintenance Task**

Oracle Database has automated a number of 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.

### **About Automated Maintenance Tasks**

Automated maintenance tasks are tasks that are started automatically at regular intervals to perform maintenance operations on the database. An example is a task that gathers statistics on schema objects for the query optimizer. Automated maintenance tasks run in maintenance windows, which are predefined time intervals that are intended to occur during a period of low system load. You can customize maintenance windows based on the resource usage patterns of your database, or disable certain default windows from running. You can also create your own maintenance windows.

**Oracle Database has three predefined automated maintenance tasks:**

**Automatic Optimizer Statistics Collection:-** collects optimizer statistics for all schema objects in the database for which there are no statistics or only stale statistics. The statistics gathered by this task are used by the SQL query optimizer to improve the performance of SQL execution.

**Automatic segment Advisor:-** identifies segments that have space available for reclamation, and makes recommendations on how to defragment those segments.

you can also run the Segment Advisor manually to obtain more up-to-the-minute recommendations or to obtain recommendations on segments that the Automatic Segment Advisor did not examine for possible space reclamation.

**Automatic SQL Tuning Advisor:-** Examines the performance of high-load SQL statements, and makes recommendations on how to tune those statements. You can configure this advisor to automatically implement SQL profile recommendations.

### **5.13 Managing Resources with Oracle Database Resource Manager**

Oracle Database provides database resource management capability through its Database Resource Manager.

#### **What is the Database Resource Manager?**

The main goal of the Database Resource Manager is to give the Oracle Database server more control over resource management decisions, thus circumventing problems resulting from inefficient operating system management.

#### **What problems Does the Database Resource Manager Address?**

When database resource allocation decisions are left to the operating system, you may encounter the following problems:

- **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 is unable to prioritize one task over another.
- **Inability to manage database-specific resources**, such as parallel execution servers and active sessions.



**How does the Database Resource Manager Address These problems?**

The Oracle Database Resource Manager helps to overcome these problems by allowing the Database more control over how machine resources are allocated.

Specifically, using the Database Resource Manager, you can:

- Guarantee certain users a minimum amount of processing resources regardless of the load on the system and the number of users.
- Distribute available processing resources by allocating percentages of CPU time to different users and applications. In a data warehouse, a higher percentage may be given to ROLAP (relational on-line analytical processing) applications than to batch jobs.
- Limit the degree of parallelism of any operation performed by members of a group of users.
- Create an **active session pool**. This pool consists of a specified maximum number of user sessions allowed to be concurrently active within a group of users. Additional sessions beyond the maximum are queued for execution, but you can specify a timeout period, after which queued jobs will terminate.
- Allow **automatic switching** of users from one group to another group based on administrator defined criteria. If a member of a particular group of users creates session that executes for longer than a specified amount of time, that session can be automatically switched to another group of users with different resource requirements.
- Prevent the execution of operations that the optimizer estimates will run for a longer time than a specified limit.
- Create an **undo pool**. This pool consists of the amount of undo space that can be consumed in by a group of users.
- Limit the amount of time that a session can be idle. This can be further defined to mean only sessions that are blocking other sessions.
- Configure an instance to use a particular method of allocating resources. You can dynamically change the method, for example, from a daytime setup to a nighttime setup, without having to shut down and restart the instance.
- Allow the cancellation of long-running SQL statements and the termination of long running sessions.

**5.14 Oracle Scheduler Concepts**

Oracle Database provides advanced job scheduling capabilities through Oracle Scheduler (the scheduler).

Organizations have too many tasks, and manually dealing with each one can be daunting. To help you simplify these management tasks, as well as offering a rich set of functionality for complex scheduling needs, Oracle provides a collection of functions and procedures in the DBMS\_SCHEDULER package. Collectively, these functions are called the scheduler, and they are callable from any PL/SQL program.

**Basic Scheduler Concepts**

The scheduler offers a modular approach for managing tasks within the Oracle environment. Advantage of modularity include easier management of your database environment and reusability of scheduler objects when creating new tasks that are similar to existing tasks.

In the scheduler, most components are database objects like a table, which enables you to use normal Oracle privileges.

The basic elements of the scheduler are:

- Programs
- Schedules
- Jobs
- Events
- Chains

### **Programs**

A scheduler program object is a collection of metadata about what will be run by the scheduler. It includes information such as the name of the program object, program action(for example, a procedure or executable name), program type(for example, PL/SQL and java stored procedures or PL/SQL anonymous blocks) and the number of arguments required for the program.

### **Schedules**

A schedule specifies when and how many times a job is executed. Jobs can be scheduled for processing at a later time or immediately. For jobs to be executed at a later time, the user can specify a date and time when the job should start. For jobs that repeat over a period of time, an end date and time can be specified. Which indicates when the schedule expires.

### **Jobs**

A job is a user defined task that is scheduled to run one or more times. It is a combination of what needs to be executed(the action) and when (the schedule). Users with the right privileges can create jobs either by:

- Specifying as job attributes both the action to perform(for example, an inline PL/SQL anonymous block) and the schedule by which to perform the action(for example, every day at noon, or when a certain event occurs)
- Specifying as job attributes the names of an existing program object and an existing schedule object.

Like programs and schedules, jobs are objects that can be named and saved in the database.

### **Events**

An event is a message sent by one application or system process to another to indicate that some action or occurrence has been detected. An event is raised (sent) by one application or process, and consumed(received) by one or more applications or processes.

There are two kinds of events in the scheduler:

- Events raised by the Scheduler  
The Scheduler can raise an event to indicate state changes that occur within the Scheduler itself.
- Events raised by an application  
An application can raise an event to be consumed by the Scheduler. The scheduler reacts to the event by starting a job. You can create a schedule that references an event instead of containing date, time , and recurrence information. If a job is assigned to such a schedule (an event schedule),the job runs when the event is raised. You can also create a job that has no schedule assigned and that directly references an event as the means to start the job.

## Chains

A chain is a grouping of programs that are linked together for a single, combined objective. An example of a chain might be “run program A and then program B, but only run program C if programs A and B complete successfully, otherwise run program D.” A scheduler job can point to a chain instead of pointing to a single program object.

### 5.15 Scheduling Jobs with Oracle Scheduler

#### Create A Schedule:

Schedules are created using the DBMS\_SCHEDULER.CREATE\_SCHEDULE procedure.

```
DBMS_SCHEDULER.CREATE_SCHEDULE
(
    Schedule_Name VARCHAR2
    Start_date      TIMESTAMP
    Repeat_interval VARCHAR2
    End_date        TIMESTAMP
    Comments        VARCHAR2
);
```

The predefined frequencies: YEARLY, MONTHLY, DAILY, HOURLY, MINUTELY, SECONDLY.

#### Example:

```
Dbms_schedule.create_schedule
(schedule_name=>'SAVE_THE_WORLD',
Repeat_interval=>'FREQ=MINUTELY;INTERVAL=98',
Comments=>'Execute this task every 98 minutes because I am trapped in an improbable
TV show.');
```

At the very minimum, every schedule you create must have a name(a good practice would be to give it a descriptive name such as Sch\_Monday\_6am) and a repeat interval telling it when to execute.

#### Create a Program:

```
DBMS_SCHEDULER.CREATE_PROGRAM
(
    Program_name VARCHAR2      A unique name for the program
    Program_type  VARCHAR2      the type of program being created. Valid options are
                                'STORED PROCEDURE','PLSQL BLOCK' and "EXECUTABLE".
    Program_action VARCHAR2      if the type is STORED PROCEDURE.
                                The action should be the name of the stored procedure. For PLSQL BLOCK, it must be the
                                full anonymous block and it must end with a semi-colon. For EXECUTABLE it must be the
                                full path to the executable.
    Number_of_arguments  NUMBER      Number of parameters the action takes. This is
                                ignored for PLSQL BLOCK.
    Enabled               BOOLEAN      Should this program be created as enabled or not? The
                                default is FALSE.
```

```
Comments      VARCHAR2      Comments about the program
);
```

**Example:**

```
Dbms_scheduler.create_program
(
    Program_name=>'EXAMPLE_PROGRAM',
    Program_type=>'STORED_PROCEDURE',
    Program_action=>'example_pkg.example_procedure'
    Enabled=>TRUE,
    Comments=>'This creates a program.'
);
```

**Create a Job:**

```
DBMS_SCHEDULER.CREATE_JOB
(
    Job_name          VARCHAR2
    Program_name      VARCHAR2
    Schedule_name     VARCHAR2
    Enabled           BOOLEAN
    Comments          VARCHAR2
);
```

**Example:**

```
Dbms_scheduler.create_job
(
    Job_name=>'Example_job',
    Program_name=>'Example_program',
    Schedule_name=>'Sch_last_day_of_month',
    Enabled=>TRUE,
    Comments=>'this job will run the example_program program
according to the sch_last_day_of_month schedule.'
);
```

**5.16 Administering Oracle Scheduler**

It contains the following sections:

- Configuring the Scheduler
- Monitoring and Managing the Scheduler
- Import/Export and the Scheduler

**(1) Configuring the scheduler**

Following tasks are necessary when configuring oracle scheduler.

**Task 1:** Setting Scheduler Privileges

**Task 2:** Configuring the Scheduler Environment

**Task 1: Setting Scheduler Privileges**

You should have the SCHEDULER\_ADMIN ROLE TO administer the Scheduler. Typically, database administrators will already have this role with the ADMIN option as part of the DBA(or equivalent) 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 grant to execute code. As any user, you should consider granting individual scheduler system privileges instead.

**Task 2: Configuring the Scheduler Environment**

This includes following things:

- ✓ Creating Job Classes
- ✓ Creating Windows
- ✓ Creating Resource plans
- ✓ Creating Window Groups
- ✓ Setting Scheduler Attributes

**Creating Job Classes:**

To create job classes, use the CREATE\_JOB\_CLASS procedure.

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB_CLASS(
    Job_class_name=>'my_jobclass1',
    Resource_consumer_group=>'my_res_group1',
    Comments=>'This is my first job class:');
END;
/
```

**Creating Windows:**

To create windows use the CREATE\_WINDOW procedure.

```
BEGIN
  DBMS_SCHEDULER.CREATE_WINDOW(
    window_name=>'my_window1',
    resource_plan=>'my_resourceplan1',
    start_date=>'15-APR-03 01.00.00 AM',
    repeat_interval=>'FREQ=DAILY',
    end_date=>'15-SEP-04 01.00.00 AM Europe/Lisbon',
    duration=>interval '50' minute,
    window_priority=>'HIGH',
    comments=>'This is my first window.');
```

```
END;
/
```

**Creating Resource Plan**

To create resource plans, use the CREATE\_SIMPLE\_PLAN procedure.

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_SIMPLE_PLAN(
    simple_plan=>'my_simple_plan1',
```

```
        consumer_group1=>'my_group1',
        group1_cpu=>80,
        consumer_group2=>'my_group2',
        group2_cpu=>20);

END;
/
```

### Creating Windows Group

To create window groups, use the `CREATE_WINDOW_GROUP` and `ADD_WINDOW_GROUP_MEMBER` procedures.

```
BEGIN
DBMS_SCHEDULER.CREATE_WINDOW_GROUP(
    group_name=>'my_window_group1',
    comments=>'this is my first window group');
DBMS_SCHEDULER.ADD_WINDOW_GROUP_MEMBER(
    group_name=>'my_window_group1',
    window_list=>'my_window1', 'my_window2');
DBMS_SCHEDULER.ADD_WINDOW_GROUP_MEMBER(
    group_name=>'my_window_group1',
    window_list=>'my_window3');

END;
/
```

### Setting Scheduler Attributes

There are several Scheduler attributes that control the behavior of the Scheduler. They are `default_timezone`, `log_history`, `max_job_slave_processes`, and `event_expiry_time`. The values of these attributes can be set by using the `SET_SCHEDULER_ATTRIBUTE` procedure.

#### (2) Monitoring and Managing Scheduler

This includes:

- ✓ Viewing the currently Active Window and Resource plan
- ✓ Finding information About Currently Running jobs
- ✓ Monitoring and Managing Window and Job logs
- ✓ Changing Job priorities
- ✓ Managing Scheduler Security

#### Viewing the currently Active Window and Resource plan

You can view the currently active window and the plan associated with it by issuing the following statement:

```
SELECT WINDOW_NAME, RESOURCE_PLAN FROM DBA_SCHEDULER_WINDOWS WHERE
ACTIVE='TRUE';
```

WINDOW_NAME	RESOURCE_PLAN
MY_WINDOW10	MY_RESOURCEPLAN1

### **Finding Information About currently Running Jobs**

You can check a job's state by issuing the following statement:

```
SELECT JOB_NAME, STATE FROM DBA_SCHEDULER_JOBS
WHERE JOB_NAME='MY_EMP_JOB1';
```

JOB_NAME	STATE
-----	-----
MY_EMP_JOB1	DISABLED

### **Monitoring and Managing window and job Logs**

#### JOB LOGS

You can view information about job runs, job state changes, and job failures in the job log. The job log is implemented as the following two data dictionary views:

- \*\_SCHEDULER\_JOB\_LOG
- \*\_SCHEDULER\_JOB\_RUN\_DETAILS

#### WINDOW LOG

The Scheduler makes an entry in the window log each time that:

- ✓ You create or drop a window
- ✓ A window opens
- ✓ A window closes
- ✓ Windows overlap
- ✓ You enable or disable a window

### **Changing Job Priorities**

you can change job priorities by using the SET\_ATTRIBUTE procedure. Job priorities must be in the range 1-5 with 1 being the highest priority.

```
BEGIN
DBMS_SCHEDULER.SET_ATTRIBUTE(
    name=>'my_emp_job1',
    attribute=>'job_priority',
    value=>1);
END;
/
```

### **Managing Scheduler Security**

You should grant the CREATE JOB system privilege to regular users who need to be able to use the Scheduler to schedule and run jobs. You should grant MANAGE SCHEDULER to any database administrator who needs to be able to manage system resources. Granting any other Scheduler system privilege or role should not be done without great caution. In particular, the CREATE ANY JOB system privilege and the SCHEDULER\_\_ADMIN role, which includes it, are very powerful because they allow execution of code as any user. They should only be granted to very powerful roles or users.

### **(3) Import/Export and the Scheduler**

You must use the Data pump utilities(impdp and expdp) to export Scheduler objects. You cannot use the earlier import/export utilities (IMP and EXP) with the Scheduler. Also, Scheduler objects cannot be exported while the database is in read-only mode.

An export generates the DDL that was used to create the Scheduler objects. All attributes are exported. When an import is done, all the database objects are recreated in the new database.

All schedules are stored with their time zones, which are maintained in the new database.

MIKAS