

# **MODERN DATABASE MANAGEMENT SYSTEM (MDBMS)**

**B.Sc III Year (For All Streams) of  
All Universities**

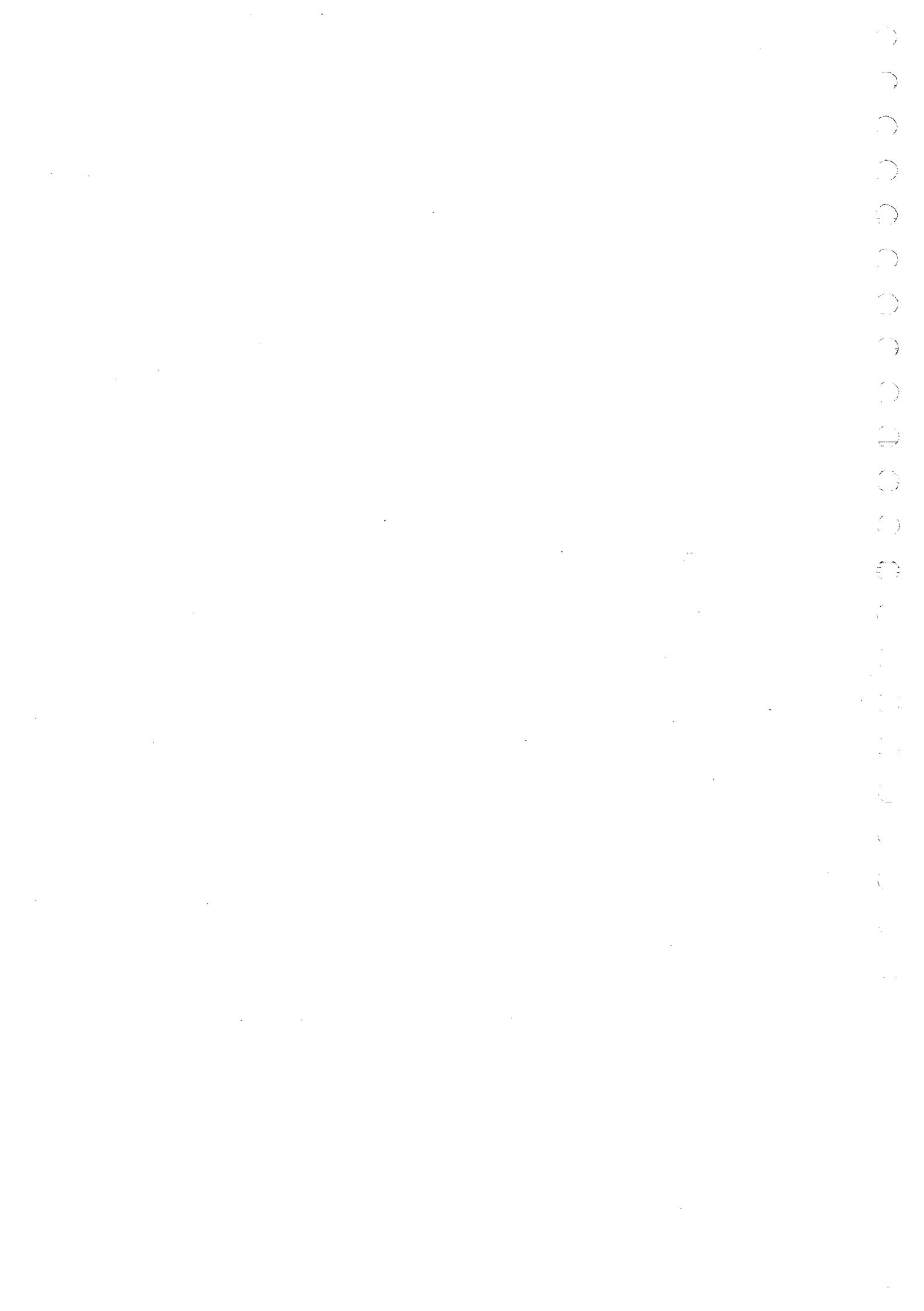
**Written By:  
Mr. G. GANESH**

**MCA**

Department of Computer Science  
Sri Sarada Degree College for women  
Yellareddy Guda, Ameerpet  
Hyderabad-73.

**PRESENTED BY  
GS SOLUTIONS  
Nirmal, Adilabad.**

Like us on: [mylovelyinfo4u@gmail.com](mailto:mylovelyinfo4u@gmail.com)



## UNIT- III

### INTERACTION WITH DATABASES AND CONSTRUCTION OF INFORMATION SYSTEM

#### \* Introduction to SQL :-

- There are numerous versions of SQL. The original version was developed at IBM's San Jose Research Laboratory
- This language, originally called Sequel was implemented as part of the "System-R" project in the early 1970's
- The Sequel language has evolved and its name has changed to SQL (structured Query language)
- In 1986, the American National Standards (ANSI) published an SQL Standard
- IBM has published its own 'corporate' SQL Standard, the Systems Application Architecture Database Interface (SAA-SQL)
- SQL has clearly established itself as the standard relational database language

SQL allows users to access data in relational database Management Systems, Such as oracle, Informix, Sybase, Access etc..

SQL allows users to manipulate and define the data in a database.

SQL is an ANSI Standard computer Language

→ SQL is more than a query tool, although that was its original purpose and retrieving data is still one of its most important functions

→ SQL is used to control all of the functions of DBMS, which provides for its users, including

#### \* Data Definition Language:-

In DDL user define the Structure and Organization of the stored data and relationships among the stored data items.

#### \* Data Retrieval Language:-

An application program to retrieve stored data from the database and use it

#### \* Data manipulation language:-

DML allows a user or an application program to update the database by adding new data, removing old data, and modifying previously stored data

#### \* Data control language:-

DCL can be used to restrict a user's ability to retrieve, add and modify data, protecting stored data against unauthorized access

#### \* Data Sharing:-

It is used to co-ordinate data sharing by concurrent

Users, Ensuring that they do not interfere with one another

### \* Data Integrity :-

It defines integrity constraints in the database, Protecting it from corruption due to inconsistent updates or system failures

### \* Role of SQL in a Database Architecture

→ An SQL-based relational database application

involves a user-interface, a set of tables in the database and a relational database

Management System (RDBMS) with an SQL Capability

→ within the RDBMS, SQL will be used to create

the tables, translate user requests maintain the data dictionary and system catalog update and maintain the tables, establish security and carry out back up and recovery procedures

→ A relational DBMS (RDBMS) is a data management system that implements a relational data

model, one where data are stored in a collection of tables, and the data relationships are represented by common values, not links.

→ The original purposes of the SQL Standard are as follows:-

- To Specify the Syntax and Semantics of SQL data definition and manipulation languages
- To define the data structures and basic operations for designing, accessing, maintaining, controlling and protecting an SQL database
- To provide a vehicle for portability of database definition and application modules between conforming DBMS
- To specify both minimal and complete standards, which permit different degrees of adoption in products
- To provide an initial standard, to include specifications for handling such topics as referential integrity, transaction management, user-defined functions, join operators and character sets

\* Benefits of SQL :-

1. Reduced training costs :-

- Training an organization can concentrate on one language. A large labor pool of IT professionals trained in a common language reduces retaining when hiring new employees.

## 2. Productivity :-

→ Is professionals can learn SQL thoroughly and become proficient with it from continued use and since they are familiar with the language in which programs are written, programmers can more quickly maintain existing programs.

## 3. Application portability :-

→ Applications can be moved from machine to machine when each machine uses SQL. further, it is economical for the computer software industry to develop off - the - shelf application software when there is a standard language.

## 4. Application longevity :-

→ A standard language tends to remain so for a long time; hence there will be little pressure to replace old applications. Rather, applications will simply be updated as the standard language is enhanced or new versions of DBMS are introduced.

## 5. Reduced dependence on a single vendor :-

- when a non-proprietary language is used, it is easier to use different vendors for the DBMS training, and Educational Services, application

Software and Consulting assistance

## 6. Cross - System communication

→ Different DBMSs and application programs can more easily communicate and cooperate in managing data and processing user programs.

## \* SQL Environment

→ An SQL environment includes an instance of an SQL database management system along with the databases accessible by that DBMS and the users and programs that may use that DBMS to access the databases.

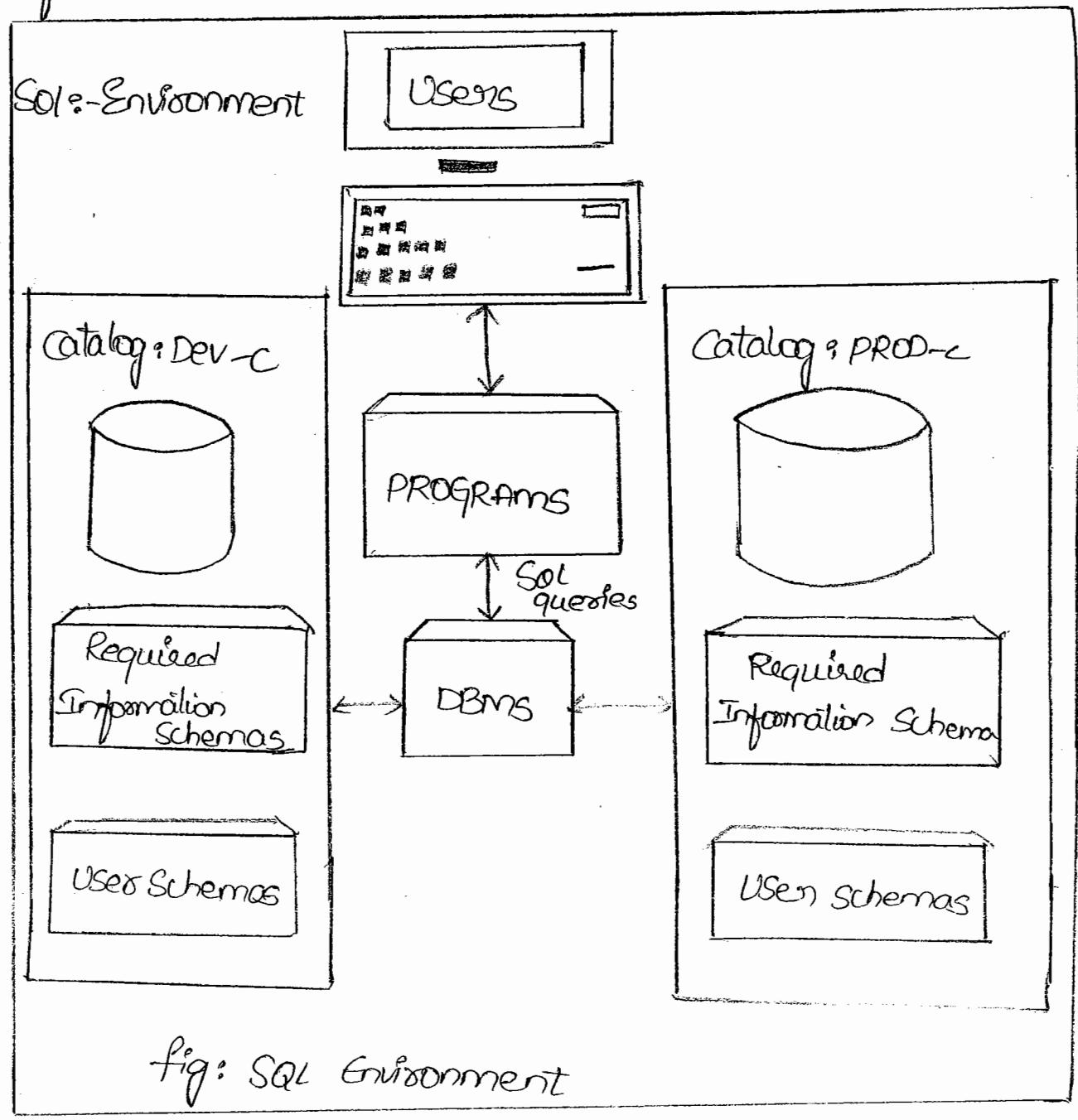
→ Each database is contained in a catalog, which describes any object that is a part of the database, regardless of which one created that object

→ The term schema is not clearly defined in the standard it may be taken in general to contain description of the objects, such as the base tables, views, constraints and so on

→ If more than one user has created objects in the database, combining information about all users schemas will yield information for the entire database.

→ Each catalog must contain an information Schema, which contains descriptions of all schemas in the catalog, tables, views, attributes, privileges, constraints and domains.

→ The figure below is a simplified schematic of an SQL Environment;



- The information contained in the catalog is maintained by the DBMS as a result of the SQL Commands issued by the users and does not require conscious action by the user to build it.
- Users can browse the catalog contents by using SQL Select Statements

→ The SQL language has several parts:

i) Data definition language:

→ The SQL DDL provides commands for defining relation schemes, deleting relations, creating indices and modifying relation schemes

ii) Data manipulation language:-

→ The SQL DML includes a query language based both the relational algebra and the tuple relational calculus

→ It includes commands to insert, delete and modify tuples in the database

iii) Data control language:-

→ These commands help the DBA to control the database and include commands to grant or revoke privileges to access the database or particular objects within the database and to start or remove transactions that would effect

the database

iv) Embedded data manipulation languages:-

→ The embedded form of SQL is designed for use within general-purpose programming language such as PL/I, COBOL, PASCAL, FORTRAN and C.

v) View definition:

→ The SQL DDL includes commands for defining views

vi) Authorization:-

→ The SQL DDL includes commands for specifying access rights to relations and views

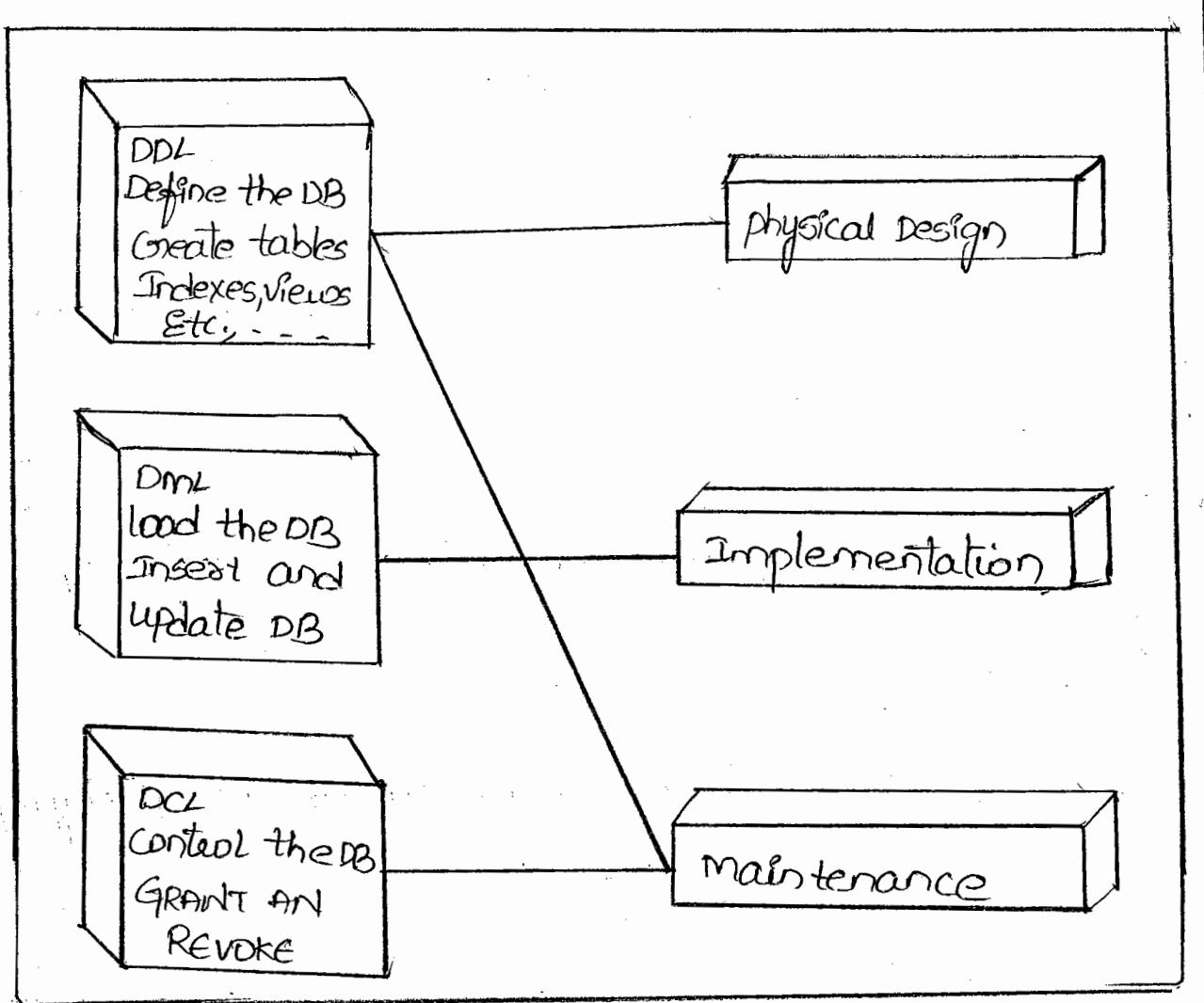
vii) Integrity:-

→ The original System-R Sequel language includes commands for specifying complex integrity constraints

→ Newer versions of SQL, including the ANSI Standard, provide only a limited form of integrity checking

viii) Transaction control:-

→ SQL includes commands for specifying the beginning and ending of transactions. Several implementations including IBM(SAA-SQL), allow explicit locking of data for concurrency control



"Various types of Commands used in Database Development process"

\* Schema and table definition

\* Database Schema:-

→ A Database Schema is described in a formal language supported by the database management system (DBMS)

→ In a relational Database, the schema defines the tables, the fields in each table, and the

relationships between fields and tables

→ Schemas are generally stored in Data Dictionary

Although a Schema is defined in text database

Language, the term is often used to refer to a graphical depiction of the database structure

→ SQL defines a catalog as a named collection of schemas but does not indicate how a catalog should be defined

→ Defining a database schema in SQL is straight forward. It is only necessary to identify the start of a schema definition, with a create schema statement

→ A sample schema definition statement is as follows

CREATE SCHEMA MATHINDRA-CONSTRUCTION

AUTHORIZATION TONY-MELTON

domain definitions

table definitions

view definitions

Etc.

### Levels of Database Schema

1. Conceptual Schema, a map of concepts and their relationships
2. Logical Schema ; a map of Entities and their

attributes and relations

3. physical Schema, a particular implementation of a logical Schema
4. Schema object, oracle database object
5. Schema is the over all structure of the database

Each Dataschema includes

1. A list of variables, each with an associated description definition and format
2. A list of domains each with an associated definition
3. A list of themes and modules with their respective definitions

SQL Tables

→ In relational databases and flat file databases, a table of data is a set of data elements (values) that is organized using a model of vertical columns (which are identified by their name) and horizontal rows, the cell being the unit where a row and column intersect

→ A table has a specified number of columns but can have any number of rows

→ Each row is identified by the values appearing in a particular column subset which has been identified as a unique key index

- In database terms, a table is responsible for storing data in the database. Database tables consist of rows and columns.
- Row contains each record in the table, and the column is responsible for defining the type of data that goes into each cell.
- Therefore, if we need to add a new person to our table, we would create a new row with the person's details.

### \* The Oracle Table 'DUAL'

- Dual is an predefined virtual table which contains one row and one column.
  - By default dual table column datatype is Varchar.
  - Generally this table is used to test functionality means to test predefined (or) user defined functions [functionality testing].
  - And also dual table is used to generate the sequence numbers from sequence object.
  - In dual table we can't perform DML operations because dual table is pre-defined virtual table.
- SQL>desc dual;

| Name  | Null? | Type        |
|-------|-------|-------------|
| Dummy |       | Varchar2(1) |

SQl>Select \* from dual;

-D-  
X

→ To facilitate such calculations via a select, Oracle provides a dummy table called DUAL.

Example: SQL>Select 2\*2 from dual;

Output:  $\frac{2 \times 2}{4}$

The current date can be obtained from the table dual in the required format as shown below

SYSDATE:- Sysdate is a pseudo column that contains the current date and time. It requires no arguments when selected from the table. DUAL and returns the current date

Ex:- SQl> Select Sysdate from dual;

Output:- SYSDATE  
06 - FEB - 98

## \* NULL VALUE CONCEPTS

- It is an undefined, unknown, unavailable value,
- Any arithmetic operations are performed on null values then again it will become null only
- If a column in a table is optional, we can insert a new record or update an existing record without adding a value to this column. This means that the field will be saved with a NULL value.
- NULL values are treated differently from other values
- NULL is used as a placeholder for unknown or inapplicable values

NOTE:- It is not possible to compare NULL and 0; they are not equivalent

- If a column in a row has no value, then the column is said to be null, or to contain a null.
- Any arithmetic expression containing a null always evaluates to null.

Example:- NULL added to 10 is null, in fact, all operators

## Principals of null values:-

- i) Setting a NULL value is appropriate when the actual value is unknown
- ii) A NULL value is not equivalent to a value of zero if the datatype is number and spaces if the data type is character
- iii) A NULL value will evaluate to NULL in any expression E.g.- Null multiplied by 10 is NULL
- iv) NULL value can be inserted into columns of any datatype
- v) If the column has a NULL value, Oracle ignores the UNIQUE, FOREIGN KEY, CHECK constraints that may be attached to the column.

\* What is SQL? Explain briefly

SQL :-

→ SQL (structured query language) is a database language that is compatible with the RDBMS and it does not need to relearn the basics when migrating from one language to another language

→ Thus, it is referred to as a portable language

whose Syntax and Command Structure (that consists of less than 100 words) are easy to learn.

→ Though, SQL have several different forms, there are very minute differences among their functionalities i.e., they can be operated in a similar way

→ Basically, SQL performs the following functions:

- i) Create table and database Structures
  - ii) Perform addition, deletion and modification of database Structures
  - iii) Perform difficult / complex queries so that raw data gets transformed into information, which can be useful
- These functions are performed without much human effort and they can be categorized into the following two languages.

a) Data Definition language

b) Data Manipulation language

c) Data Definition Language (DDL) :-

→ DDL refers to a language that consist of SQL Commands for the creation of database objects like Tables, Views, Indexes, etc.

→ It also contains commands that are used to specify access rights to those database objects

→ The following are the different data definition commands

in SQL:

1. CREATE SCHEMA AUTHORIZATION:- This Command is used for creating a database Schema
2. CREATE TABLE:- This command is used for creating a new table in the users database Schema. It includes the following constraints
  - i) NOT NULL:- This constraint is used for ensuring that null values do not exist in any column
  - ii) UNIQUE:- This constraint is used for ensuring that duplicate values do not exist in any column
  - iii) PRIMARY KEY:- This constraint is used for specifying a primary key for the table
  - iv) FOREIGN KEY:- This constraint is used for specifying a foreign key (for a column) in table
  - v) DEFAULT:- This constraint is used for specifying a default value in table
  - vi) CHECK:- This constraint is used for validating data in an attribute
3. CREATE INDEX:- This command is used for creating an index for a table
4. CREATE VIEW:- This command is used for creating a logical subset of data (rows/columns) derived from one or more base tables

5. ALTER TABLE :- This command is used for changing a table's definition by adding, deleting or updating the attributes / constraints
6. CREATE TABLE AS :- This command is used for creating a table by using a query that exists in the user's database schema
7. DROP TABLE :- This command is used for permanently deleting a table
8. DROP INDEX :- This command is used for permanently deleting an index
9. DROP VIEW :- This command is used for permanently deleting a view

#### b) Data manipulation language (DML) :-

→ DML refers to a language that consists of SQL commands so as to perform data operations like insert, delete, update and retrieve in the database tables

→ The following are the different data manipulation commands in SQL :

1. INSERT :- This command is used for inserting a row in a table
2. SELECT :- This command is used for selecting some specific attributes from the tables
3. WHERE :- This command is used for specifying conditions on rows by using conditional expressions.

4. Group By :- This command is used for grouping some specific data / rows based on the specified grouping condition.

5. HAVING :- This command is used for specifying conditions on groups by using conditional expressions.

6. ORDER By :- This command is used for ordering some specific rows based on the specified attributes.

7. UPDATE :- This command is used for modifying values of attributes in tables.

8. DELETE :- This command is used for deleting rows from a table.

9. COMMIT :- This command is used for permanently saving the changes in the database.

10. ROLL BACK :- This command is used for storing the data back to the original data.

The DML also includes some operators like,

i) Comparison operators :- These operators are used for comparing between two different expressions. Such operators include =, <, >, <=, >=, <>

ii) Logical operators :- These operators are used for comparing two different expressions. Such operators include AND, OR and NOT.

3. Special operators :-

→ These operators are also used for comparing two expressions. Such operators include the following:

- i) BETWEEN :- This operator is used for checking whether an attribute value lies within the range or not
- ii) IN :- This operator is used for checking whether an attribute value matches any of the values in the Specified list
- iii) IS NULL :- This operator is used for checking whether an attribute value returns a null value
- iv) LIKE :- This operator is used for checking whether an attribute value matches a specified String pattern
- v) EXISTS :- This operator is used for checking whether a Set of rows satisfying the conditions return a value
- vi) DISTINCT :- This operator is used for retrieving unique values from the table i.e., removing duplicate rows from the final result set.

#### Aggregate Operators :-

→ These operators are used along with the Select Statement and generate a result set that mathematically Summarizes the Columns in the table

→ The following are the aggregate operators Supported by SQL:

- i) COUNT :- This operator is used for retrieving total number of tuples consisting of non null values in the Specified Column
- ii) MIN :- This operator is used for retrieving a minimum value from a set of values present in the Specified Column

- iii) MAXs - This operator is used for retrieving a maximum value from a set of values present in the specified column
- iv) SUMs - This operator is used for retrieving a sum of all the values present in the specified column
- v) AVGs - This operator is used for retrieving an average value of all the values present in the specified column

- SQL is referred to as a non-procedural language that specifies only the things that are to be performed without specifying the way of performing them.
- The current version of SQL is SQL3 or SQL-99
- A standard SQL called ANSI SQL standards is specified by the American National Standard Institute (ANSI)
- This standard is also used by the International Organization for Standardization (ISO)
- Though government and commercial database specifications require the conformed ANSI/ISO SQL Standard, changes are made to the standard by most RDBMS vendors
- Thus an SQL based applications can be migrated very easily from one RDBMS to another without performing any changes in the standard.

## \* DATATYPES (SQL)

- Datatype identifies type of data in a column.
- Data stored in a relational database can be stored using a variety of data types
- The primary ORACLE data types are NUMBER, VARCHAR, and CHAR for storing numbers and text data
- Every constant, variable, and parameter has a data type (also called a type) that determines its storage format, constraints, valid range of values, and operations that can be performed on it
- SQL provides many predefined data types and subtypes.
- The data types supported by SQL are,
  - i) Numeric :- This datatype is used for storing data values, which are of type 'number'. It has several different formats like,
  - ii) Number(L,D) :- In this, 'L' denotes the number of digits that can be included and 'D' denotes the decimal place that is to be specified
- For example, Number(6,3) specifies that the numbers that can be stored with three decimal places and may have upto six digits and also a sign before the decimal
- Such type of data include the following examples,

812579.111

+2231.001

ii) Integer :- It stores only the integer values i.e., it doesn't include decimal values. It is abbreviated as INT

iii) Small INT :- It also stores only the integer values but only upto a limited value i.e., upto six digits

iv) Decimal(L,D) :- Its specifications is similar to Number(L,D). But here, the storage lengths must be greater like ; Decimal(10,3), Decimal(8,2), etc.,

2. Character :- This data type is used for storing data values that are of type characters. It has several different formats like

i) char(L) :- It can store a fixed-length of characters i.e., from 1 to 255 characters. for instance, varchar(18) (store upto 18 characters. But in this, the data types doesn't leave any unused space. consequently, the RDBMS i.e., oracle automatically converts varchar to varchar<sup>x</sup>) specifies that it can store upto 18 characters. And, if the strings used only certain characters i.e., 9 characters then the remaining spaces are left unused

→ It is used to store fixed length alpha numeric

data in bytes

→ maximum limit is upto 2000 bytes

→ By default char datatype having one byte

Syntax :- columnname char(maxsize)

Ex:-

T<sub>1</sub> → char(10)

| Name   |
|--------|
| Soumya |

|   |   |   |   |   |   |  |  |  |
|---|---|---|---|---|---|--|--|--|
| S | O | U | M | Y | A |  |  |  |
|---|---|---|---|---|---|--|--|--|

{

memory  
wastage area

i) Varchar(L) or Varchar2(L) :- It can store a variable length character data. for instance, Varchar(16) store upto 16 characters. But in this, the data types doesn't leave any unused space. Consequently, the RDBMs i.e., oracle automatically converts Varchar to Varchar2.

→ It is used to store variable length alphanumeric data in bytes

→ maximum limit is 4000 bytes

→ here remaining spaces are automatically deallocated

Syntax :- columnname Varchar2(maxsize)

Ex:-

T<sub>1</sub> → Varchar2(10)

| name   |
|--------|
| Soumya |

|   |   |   |   |   |   |  |  |  |
|---|---|---|---|---|---|--|--|--|
| S | O | U | M | Y | A |  |  |  |
|---|---|---|---|---|---|--|--|--|

no wastage of memory

3. Date:- This data type is used for storing the date values in the Julian format. It can include Year, month and day values.

→ It is used to store dates in Oracle date format

Syntax:- columnname date

→ In Oracle by default date format is

DD-MON-YY

Here mon is monthname

Ex :- 31-DEC-12

4. Time:- This data type is used for storing the data that contains hour, minute and second values.

5. Timestamp:- This datatype is used for storing the data that contains year, month, day, hour, minute and second values.

6. float:- This data type is used for storing the data that contains decimal values

7. Real:- This data type is used for storing the data that contains a single-precision floating point number.

8. Double precision:- This datatype is used for storing the data that contains a double-precision floating point number.

9. Intervals- This data type is used for storing the data that specifies a particular period of time. It has the following two formats,

i) year-month intervals- It stores the data that contains a year, month or both values

ii) day-time intervals- It stores the data that contains a day, minute or second values.

10. Logicals- This data type is used for storing the data values that specify either a true or false value

11. Currency- This data type is used for storing data values that are of type currency format

12. Autonumber- This datatype is used for storing data values that automatically generates a number. It is basically supported by MS. Access

13. Sequence- This data type is used for storing the data values that are in a sequence order. It is basically supported by Oracle.

14. LONG- Character data of variable size up to 2GB in length. Only one LONG column may be defined per table. LONG columns may not be used in Subqueries, functions, Expressions, Where clauses, or indexes. A table containing long data may not be clustered

15. LONG RAW:- Raw binary data; otherwise the same as long (used for Images)

16. LONG VARCHAR:- Same as long

17. SMALLINT:- Same as Number

18. RAW(size):- Raw binary data, size bytes long, maximum size = 255 bytes

19. ROWID:- A value that uniquely identifies a row in an Oracle database. It is returned by the pseudo-column ROWID. Table columns may not be assigned this type.

#### \* Difference Between CHAR, VARCHAR AND VARCHAR2

##### CHAR

1. fixed length memory storage
2. CHAR takes up 1 byte per character
3. use char when the data entries in a column are expected to be the same size
4. EX:

Declare test char(100);  
test = "Test";

Then "test" occupies 100 bytes first four bytes with values and rest with blank data

## VARCHAR

1. Variable length memory storage (changeable)
2. VARCHAR takes up 1 byte per character, +2 bytes to hold length information
3. Varchar when the data entries in a column are expected to vary considerably in size

4. Ex:-

```
Declare test Varchar(100);
test = "Test" -
```

Then "test" occupies only  $4+2=6$  bytes . first four bytes for value and other two bytes for variable length information

## VARCHAR2 :-

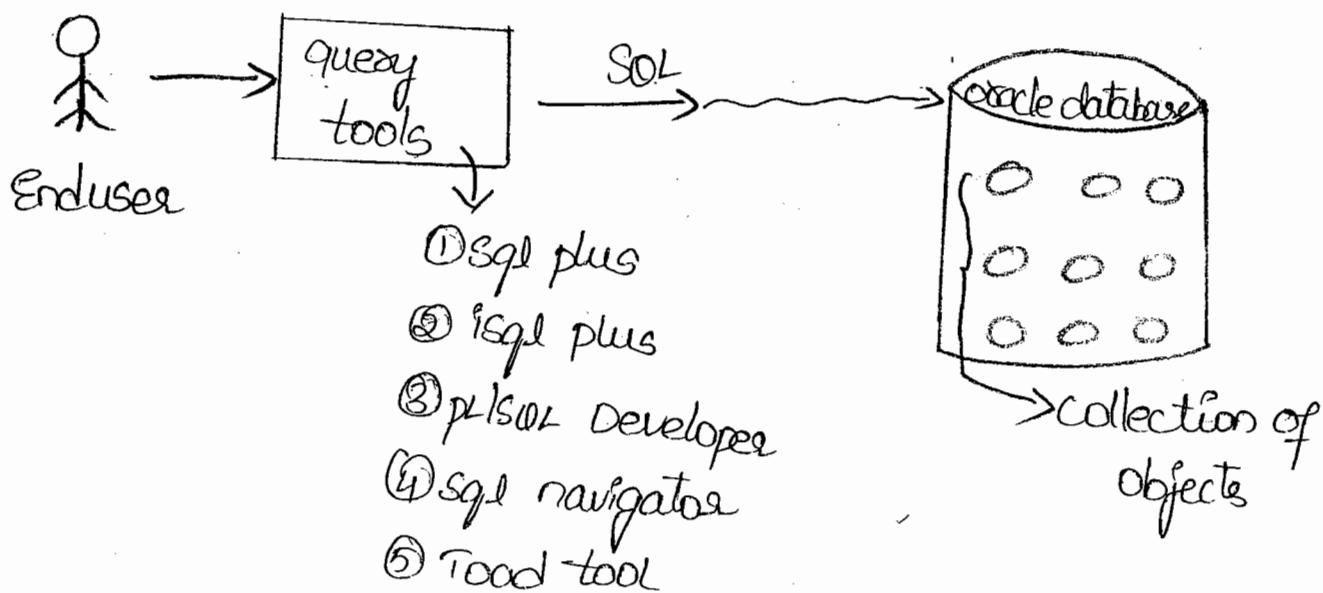
1. A replacement for varchar in new versions of oracle
2. VARCHAR can store up to 2000 bytes of characters while VARCHAR2 can store up to 4000 bytes of characters
3. If VARCHAR is used in declaration then it will occupy space for NULL values, In case of VARCHAR2 data type it will not occupy any space.

## \* SQL COMMANDS

SQL (Structured query language) :-

→ Any end user does not interact with oracle database directly. But enduser can interact with oracle database by using some query tools. Those are

1. Sql plus
2. Isql plus
3. PL/SQL Developer
4. Sql navigator
5. Toad tool.



- Enterprise Edition is better to load oracle
- Scott is the username which is available in Enterprise Edition only but not in other
- from oracle 10g onwards

Username Scott  
 Password Tiger

Error: Account locked

### \* unlocking a User:-

Username 1sys as Sysdba  
 Password Sys

SQL> Alter user Scott account unlock; ↴

SQL> conn Scott / Tiger ↴

Password : tiger

Confirm password : tiger

→ Here Conn is Connect

→ If we want to Show the User, then

SQL> show user; ↴

SQL> user is "SCOTT"

→ If we want to View all tables

SQL> Select \* from tab; ↴

table is displayed

→ If we want to clear the Screen

SQL> cl scr; ↴

or

Shift + delete

The Screen is Cleared

\* SQL can be divided into 5 major languages:-

① DDL (Data Definition Language):-

- Create
- Alter
- Drop
- Truncate
- Rename (Oracle 9i)

② DML (Data Manipulation Language):-

- Insert
- Update
- Delete
- Merge (Oracle 9i)

③ DQL (Data Retrieval Language):-

- Select

④ Transaction Control Language (TCL)

- Commit (Save)
- Roll back (Undo)
- Save

⑤ Data control language (DCL) :-

- Grant (Giving permissions)
- Revoke (Cancel permissions)

1. Data Definition Language (DDL) Commands:-

- DDL refers to a language that consists of SQL commands for the creation of database objects

like tables, views, indexes, etc.,

→ It also contains commands that are used to specify access rights to those database objects. The following are the different data definition commands in SQL:

→ DDL command are used to describe Structure (columns) of the table

\* Create :-

→ It is used to create database objects like tables, views, procedures etc.,

\* Creating a table

Syntax:- Create table tablename (col1 datatype(size),  
col2 datatype(size), ---);

here col is column

Ex:- SQL> Create table newyear (sno number(10), name  
varchar2(10));



Table created

⇒ To view structure of the table :-

SQL> desc tablename;

⇒ The structure of the newyear table is :-

SQL> desc newyear;



| Name | NULL | TYPE        |
|------|------|-------------|
| Sno  |      | Number(10)  |
| Name |      | varchar(10) |

## Syntax for create Command

```
CREATE TABLE table-name(Col1 datatype1 Constraint1,
                        Col2 datatype2 Constraint2,
                        Col3 datatype3 Constraint3,
                        ...
                        Coln datatypen Constraintn,
                        PRIMARY KEY(Col1, Col2 ...),
                        FOREIGN KEY(Col1, Col2 ...) REFERENCES
                        table-name ON CONSTRAINT
                        Constraint-name);
```

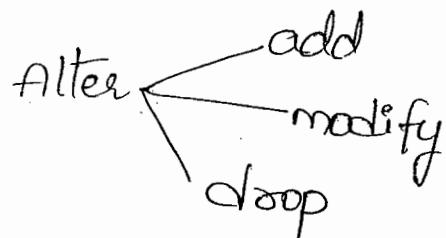
## \*ALTER :-

→ The ALTER TABLE command makes all changes in the structure of a table. It is followed by one of the following three keywords.

ADD:- To add a column to a table

MODIFY:- To change column characteristics of a table

DROP:- To delete a column from an existing table.



Syntax:-

ALTER TABLE tablename {ADD / MODIFY} column name datatype;

Adding a column to a table:

→ An Existing table can be altered by adding one or more columns.

→ In the above Example, user can add new column, named Emp-Email to the Employee table as follows

Syntax:- alter table tablename add (col1 datatype(size),  
col2 datatype(size), ---);

Ex:- If we want to add only one column into the Existing table. Then

Sol>alter table newyear add sal number(10);

Table altered

Sol>desc newyear;

| Name | Null? | Type         |
|------|-------|--------------|
| Sno  | ---   | Number(10)   |
| Name |       | Varchar2(10) |
| Sal  |       | Number(10)   |

(08)

→ If we want to add only one column into the Existing table. Then

SQL> alter table newyear add(sal1 number(10));

table altered

SQL> desc newyear;

| Name | NULL? | Type         |
|------|-------|--------------|
| Sno  |       | NUMBER(10)   |
| Name |       | VARCHAR2(10) |
| SAL  |       | NUMBER(10)   |
| SAL1 |       | NUMBER(10)   |

Ex:-

SQL> alter table newyear add Ssname Varchar2(10)  
Ssal number(10);

Error: invalid alter table option

Reasons- In the above Example we are adding  
two columns

→ So, if we want to add two (or) more than  
two columns into the Existing table we Should  
use parenthesis

SQL> alter table newyear add(ssname Varchar2(10),  
Ssal number(10));

Table altered.

SQL> desc newyear;

| Name   | NULL? | Type         |
|--------|-------|--------------|
| SNo    |       | number(10)   |
| Name   |       | VARCHAR2(10) |
| SAL    |       | Number(10)   |
| SAL1   |       | number(10)   |
| SsName |       | VARCHAR2(10) |
| SSAL   |       | Number2(10)  |

Notes- It is possible to add number of columns into the table at last position only but not in middle of the table

Exe- If the table contains the columns SNo, Sname. If we want to add Sal column into that table then

| Sno | Sname | Sal |
|-----|-------|-----|
|     |       |     |

✓

(True)

| Sno | Sal | Sname |
|-----|-----|-------|
|     |     |       |

✗

(False)

\* modifying a column's data type:

→ ALTER TABLE Command can also be used to change Column's data type. (or) column datatype size only.

Syntax- alter table tablename modify (col1 datatype(size), col2 datatype(size), ...);

Ex- If we want to modify only one column datatype

```
SQL>alter table newyear modify Snb varchar(10);  
table altered
```

SOL>desc neuropar;

| Name   | NULL? | Type         |
|--------|-------|--------------|
| SnNo   |       | VARCHAR2(10) |
| NAME   |       | VARCHAR2(10) |
| SAL    |       | NUMBER(10)   |
| SSNNo  |       | NUMBER(10)   |
| SSNAME |       | VARCHAR2(10) |
| SSAL   |       | NUMBER(10)   |
| SAL1   |       | NUMBER(10)   |

Ex- If we want to modify two (or) more than two column datatypes then we should use parenthesis otherwise it gives error as "invalid alter table option".

SOL>alter table newyear modify(name number(10));

Sal varchar(10));

table altered

Sol>desc newyear;

| Name   | NULL? | Type         |
|--------|-------|--------------|
| SNo    | NULL? | VARCHAR2(10) |
| Name   |       | NUMBER(10)   |
| SAL    |       | VARCHAR2(10) |
| SSNO   |       | NUMBER(10)   |
| SSNAME |       | VARCHAR2(10) |
| SSAL   |       | NUMBER(10)   |
| SAL1   |       | NUMBER(10)   |

Exe- SQL>alter table newyear modify SNo VARCHAR2(20);  
 table altered

SQL>desc newyear;

| Name   | NULL? | Type         |
|--------|-------|--------------|
| SNo    |       | VARCHAR2(20) |
| Name   |       | NUMBER(10)   |
| SAL    |       | VARCHAR2(10) |
| SSNO   |       | NUMBER(10)   |
| SSNAME |       | VARCHAR2(10) |
| SSAL   |       | NUMBER(10)   |
| SAL1   |       | NUMBER(10)   |

\* In the above Example SNo having datatype size is changed from VARCHAR2(10) to VARCHAR2(20);

Note :- We can change, from one datatype to another datatype, when the related column is Empty (or) when the related column is Empty then only we can change from one datatype to another datatype.

Ex :-

| Sno | Name |
|-----|------|
| 1   | SS   |

The above table contains number datatype to Sno & Varchar datatype to Name column

→ If we want to modify Sno of number datatype to varchar (or) any other datatype other than number (OR)

If we want to modify name of varchar datatype to any other datatype other than varchar, then it gives an error as

Error :- Column to be modified must be Empty to change datatype.

\* Dropping a column for a table :-

→ It is used to drop number of columns into the existing table

Method :- If we want to drop a "single column" at a time without using parenthesis we can use following

## Syntax

Syntax :- alter table tablename drop column columnname;

In the above Syntax column is keyword

Ex :- SQL>alter table newyear drop column Sal;  
table altered

SQL>desc newyear;

| Name   | NULL? | Type         |
|--------|-------|--------------|
| Sno    |       | VARCHAR2(20) |
| Name   |       | NUMBER(10)   |
| SSNO   |       | NUMBER(10)   |
| SSNAME |       | VARCHAR2(10) |
| SSAL   |       | NUMBER(10)   |
| Sal    |       | NUMBER(10)   |

## method @:-

If we want to drop a single (or) multiple columns with using parenthesis we are using following Syntax

Syntax :- alter table tablename drop (col1, col2, ---);

Ex :- SQL>alter table newyear drop (SSNO, SSNAME, SSAL);  
table altered

SQL>desc newyear;

| Name | NULL? | Type        |
|------|-------|-------------|
| SNo  |       | VARCHAR(20) |
| NAME |       | NUMBER(10)  |
| SAL1 |       | NUMBER(10)  |

Ex: SQL>alter table newyear drop(SAL1);  
Table altered

SQL>desc newyear;

| Name | NULL? | Type        |
|------|-------|-------------|
| SNo  |       | VARCHAR(20) |
| NAME |       | NUMBER(10)  |

Note:-

1. we cannot drop all columns in a table
2. If our table contains only one column then we cannot drop that column.

\*DROP:-

→ It is used to remove database objects from database

Syntax:- drop objecttype objectname;

Ex:- drop table tablename;

Ex:- drop procedure procedurerename;

Ex:- drop view viewname;

Note:- upto oracle log if we want to delete table, then table is deleted permanently

diagrammatic way

\* before oracle log:-

`drop table tablename;`

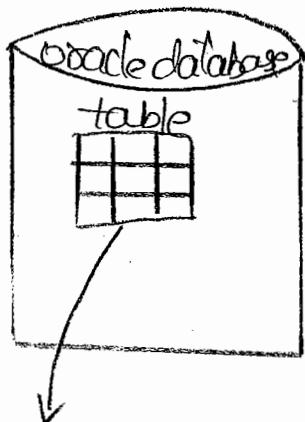


table is permanently dropped

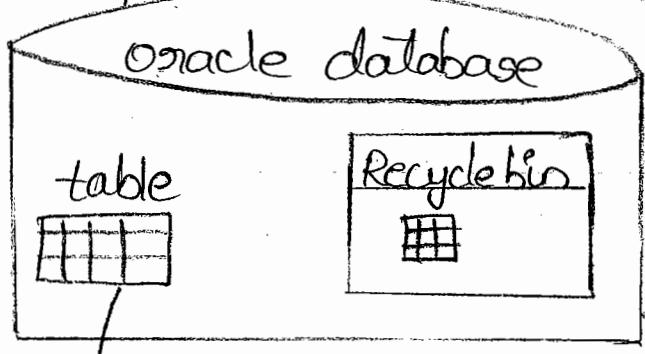
Sol> `drop table newyear;`

table dropped

\* from oracle log in

Enterprise Editions:-

`drop table tablename;`



here table is temporarily dropped  
(this table is moved to Recycle Bin)

\* If we want to get it back from Recycle Bin:-

Syntax:- `flashback table tablename to before drop;`

here to, before are keywords.

\* drop table permanently:-

`drop table tablename purge;`

Exe- for from oracle(log)

SOL>drop table newyear;  
table dropped

SOL>desc newyear;

Error: object newyear does not Exist

\* SOL> flashback table newyear to before drop;  
flashback complete

SOL>desc newyear;

| Name | NULL? | Type       |
|------|-------|------------|
| Sno  |       | NUMBER(10) |

\* SOL>drop table newyear purge;  
Table dropped

SOL>desc newyear;

Error: object newyear does not Exist

Note- If the table is permanently dropped then we can't get it from recycle bin (08) from any where so if we can complete the command

SOL>drop table newyear purge;

After this we can't use

SQL> flashback table newyear to before drop;

→ because already the table is permanently dropped

### \* Truncate :-

→ This command is used to delete "total data permanently" from the table but columns are not deleted

Syntax - truncate table tablename;

Ex :- SQL> create table newyear as select \* from emp;

The meaning of the above SQL Command is the data which is available in Emp, is also available in newyear

table created

SQL> Select \* from newyear;

| EmpNo | ENAME  | JOB       | MGR  | HIREDATE  | Sal  | COMM | DeptNo |
|-------|--------|-----------|------|-----------|------|------|--------|
| 7369  | SMITH  | CLERK     | 7902 | 17-Dec-80 | 800  |      | 20     |
| 7409  | ALLEN  | SALESMAN  | 7698 | 20-feb-81 | 1600 | 300  | 30     |
| 7521  | WARD   | SALESMAN  | 7698 | 22-feb-81 | 1250 | 500  | 30     |
| 7568  | JONES  | MANAGER   | 7839 | 02-Apr-81 | 2975 |      | 20     |
| 7654  | MARTIN | SALESMAN  | 7698 | 28-Sep-81 | 1250 | 1400 | 30     |
| 7698  | BLAKE  | MANAGER   | 7829 | 01-may-81 | 2850 |      | 30     |
| 7782  | CLARK  | MANAGER   | 7629 | 09-jun-81 | 2450 |      | 10     |
| 7788  | SCOTT  | ANALYST   | 7566 | 19-Apr-87 | 3000 |      | 20     |
| 7839  | KING   | PRESIDENT |      | 17-Nov-81 | 5000 |      | 10     |

|      |        |          |      |           |      |   |    |
|------|--------|----------|------|-----------|------|---|----|
| 7844 | TURNER | SALESMAN | 7698 | 08-Sep-81 | 1500 | 6 | 30 |
| 7876 | ADAMS  | CLERK    | 7788 | 23-May-81 | 1100 |   | 20 |
| 7900 | JAMES  | CLERK    | 7698 | 03-Dec-81 | 950  |   | 30 |
| 7802 | FORD   | ANALYST  | 7566 | 03-Dec-81 | 3000 |   | 20 |
| 7934 | MILLER | CLERK    | 7782 | 23-Jan-82 | 1300 |   | 10 |

14 rows Selected

SQL>truncate table newyear;

Table truncated

SQL>Select \* from newyear;

no rows Selected

SQL>desc newyear;

| Name     | NULL? | Type         |
|----------|-------|--------------|
| EmpNo    |       | NUMBER(11)   |
| ENAME    |       | VARCHAR2(10) |
| JOB      |       | VARCHAR2(9)  |
| MGR      |       | NUMBER(4)    |
| HIREDATE |       | DATE         |
| SAL      |       | NUMBER(7,2)  |
| Comm     |       | NUMBER(7,2)  |
| DeptNo   |       | NUMBER(2)    |

\*Rename:-

→ It is used to rename a table and rename a

table and rename a column also  
rename a table-

Syntax :- rename oldtablename to newtablename;

Ex :- SQL> rename newyear to abc; ↵

Table renamed

SQL> desc abc;

| Name     | NULL? | Type         |
|----------|-------|--------------|
| EmplNo   |       | NUMBER(4)    |
| ENAME    |       | VARCHAR2(10) |
| JOB      |       | VARCHAR2(9)  |
| MGR      |       | NUMBER(4)    |
| HIREDATE |       | DATE         |
| SAL      |       | NUMBER(7,2)  |
| COMM     |       | NUMBER(7,2)  |
| DeptNo   |       | NUMBER(2)    |

If :- SQL> desc newyear;

Error :- Object newyear does not exist Because it is already renamed to abc

\* Imp renaming a column (oracle 9.2) :-

Syntax :- alter table tablename rename column oldcolumnname to newcolumnname;

Ex:- SQL> Select \* from pg8;

| <u>SNo</u> | <u>SNAME</u> |
|------------|--------------|
| 1          | xx           |
| 2          | tt           |

SQL> Alter table pg8 rename column SNo to SNo1;

Table altered

SQL> desc pg8;

| <u>Name</u> | <u>NULL?</u> | <u>Type</u>  |
|-------------|--------------|--------------|
| SNo1        |              | NUMBER(10)   |
| SName       |              | VARCHAR2(10) |

SQL> Select \* from pg8;

| <u>SNo1</u> | <u>SNAME</u> |
|-------------|--------------|
| 1           | xx           |
| 2           | tt           |

Note :- By default all DDL commands are automatically committed (Saved)

→ DML (Data Manipulation Language) :-

→ DML refers to a language that consists of SQL Commands So as to perform data operations like insert, delete, update and retrieve in the database

## tables

→ The following are the different data manipulation Commands in SQL:

### \* Insert :-

→ This command is used to add information or column values to a row in a table. Its Syntax is as follows.

#### Method 1 :-

Syntax :- `insert into tablename values(value1, value2, ...);`

Here values is keyword

Ex :- `SQL> insert into newyear values(1, 'abc');`  
1 row created

`SQL> select * from newyear;`

| <u>Sno</u> | <u>NAME</u> |
|------------|-------------|
| 1          | abc         |

`SQL> insert into newyear values(2, 'pqrs');`  
1 row created

`SQL> select * from newyear;`

| <u>Sno</u> | <u>NAME</u> |
|------------|-------------|
| 1          | abc         |
| 2          | Pqrs        |

### Disadvantages :-

→ It takes more time because for one name we have to type insert command for another name

We have to type 2nd time insert command.

→ And for n names we have to type n times  
insert command

→ for this time is wasted

method ② :- (using Substitutional operator (&))

Syntax :- Insert into tablename values(&c01, &c02, ...);

here & means Enter value for

Ex :- SOL>Insert into newyear values(&sno, &sname);

Enter value for sno: 3

Enter value for sname: xyz

1 row created

SOL>/

Enter value for Sno: 4

Enter value for Sname: mno

1 row created

SOL>Select \* from newyear;

| SNO | SNAME |
|-----|-------|
| 1   | abc   |
| 2   | pqr   |
| 3   | xyz   |
| 4   | mno   |

### \* Method 3:- (Skipping Columns)

Syntax:-

```
insert into tablename (Col1, Col2, ....) Values (Value
1, Value 2, ...);
```

Note: The above Syntax is used for Only adding last row.

Ex: SQL > insert into newyear (name) Values ('ijk');

1 row created.

SQL > Select \* from newyear;

| <u>SNO</u> | Name |
|------------|------|
| 1          | abc  |
| 2          | pqr  |
| 3          | xyz  |
| 4          | mno  |
|            | ijk  |

Ex: SQL > insert into newyear(sno, name) values (6, 'rama');

1 row created

SQL > Select \* from newyear;

| <u>Sno</u> | Name |
|------------|------|
| 1          | abc  |
| 2          | pqr  |

|   |      |
|---|------|
| 3 | xyz  |
| 4 | mno  |
| 5 | ijk  |
| 6 | Rama |

6 rows Selected

SQL> alter table newyear add address Varchar(10);  
 Table altered

SQL> Select \* from newyear;

| <u>sno</u> | <u>Name</u> | <u>ADDRESS</u> |
|------------|-------------|----------------|
| 1          | abc         |                |
| 2          | pqr         |                |
| 3          | xyz         |                |
| 4          | mno         |                |
| 5          | ijk         |                |
| 6          | Rama        |                |

Note: If we want to insert address of abc the above syntax i.e., method ③ is not working, because abc is a record which is located in middle of the table.

## \*update

→ This Command is Used to change or modify the Existing Column Values of a Row in a table.

→ The Syntax is as follows,

### Syntax:

```
update tablename set columnname=newvalue where
columnname = oldvalue;
```

### Ex:-

```
SQL> update emp set sal=1000 where ename =
      'SMITH';
```

## \*Where clause:-

→ This clause is Used to Specify a Condition. All the Rows/Tuples which Satisfy the Condition are "Satisfied" Selected.

### Ex:-

```
SQL> update newyear set address='Hyd' where
      name = 'mno';
```

1 row updated

```
SQL> select * from employe newyear;
```

| SNo | Name | Address |
|-----|------|---------|
| 1   | abc  |         |
| 2   | pqr  |         |
| 3   | xyz  |         |
| 4   | mno  | Hyd     |
|     | ijk  |         |
| 6   | Rama |         |

6 rows Selected.

- The rows that satisfy the condition of the WHERE clause are updated based on the assignments given in the SET clause.
- All the rows in the table get updated whenever WHERE clause is not specified.

\* Delete :-

- It is used to delete all rows (or) particular rows from a table.

Syntax :

Delete from tablename;

( ∵ all rows are deleted)

Syntax:

Delete from tablename where Columnname=Value;

(∴ particular rows are deleted)

Ex:-1

SQL > delete from newyear;  
6 rows deleted.

SQL > Select \* from newyear;  
no rows selected.

SQL > rollback;  
rollback complete.

SQL > Select \* from newyear;

| <u>SNO</u> | <u>Name</u> | <u>ADDRESS</u> |
|------------|-------------|----------------|
| 1          | abc         |                |
| 2          | pqr         |                |
| 3          | xyz         |                |
| 4          | mno         |                |
|            | ijk         |                |

Note:

→ Whenever we are using "delete from tablename;" and "truncate table tablename;" then database sever delete all rows from a table.

→ But whenever we are using "Delete from tablename;" deleted Data temporarily stored in

buffer, that data we can get it back by Using rollback Command.

→ Whereas when we are Using "Truncate table tablename;" total data permanently deleted.

→ That's why we can't get it back this Data Using rollback Command.

\* DQL/DRL :- (Data Retrieval Language) :-

Select:

→ This Command is mostly Used in SQL queries It is Considered as the basic SQL Command which is Used to retrieve information or to Select Specific Columns from the tables.

→ Every SQL query Starts with the SELECT keyword and is followed by the list of Columns that form the resulting relation.

→ The Syntax for SELECT Command is as follows,

Syntax: Select col1, col2, .... from tablename  
where Condition  
group by Columnname  
having Condition  
Order by Columnname [asc / desc];

Note:-

→ In place of col1, col2, ...., '\*' is used if all columns are selected.

\* 4 types for Select

\* 4 types for Select

(1) Select all columns and all rows.

(2) Select all columns and particular rows

(3) Select all particular columns and all rows.

(4) Select particular columns and particular rows.

Note:-

→ particular columns (or) particular rows means we have to use "where"

→ For selecting all columns (or) all rows means we have to use "\*"

\* Transaction Control Language (TCL) :-

\* Transaction:-

→ A logical unit of work between two points is called transaction

→ All database Systems provides two Transaction commands

i) Commit

ii) Save point

### i) Commit-

→ This Command is used to Save the Transaction permanently into database

### ii) Savepoint-

→ A logical mark between transaction is called Save point

Syntax- Savepoint Savepointname;

### \* Rollback-

→ It is Used to Undo transaction from memory.

Ex:-

SQL > insert -----

SQL > delete -----

SQL > Save point S1;

SQL > update ---

SQL > delete ---

SQL > Savepoint S2;

SQL > insert -----

SQL > update ---

SQL > rollback to Savepoint S2;

### \*Note:

→ Both Commit and Rollback Commands Ensure the update integrity requirements in a Database and they are used only with the Data manipulation Commands, which can perform addition, deletion or updation of a particular row.

### \*Data Control Language (DCL):-

→ This language Commands are of two types.

- (i) Grant
- (ii) Revoke

#### (i) Grant:

→ The Grant Command gives users privileges to base tables and Views.

→ The Syntax of this Command is as follows;

#### Syntax:

Grant privileges ON object TO Users [WITH GRANT OPTION];

→ An object is either a base table or a View. Several privileges can be specified.

→ The following are some of them.

(a) SELECT:

→ The right to access all columns of the table specified as object including the columns which are added later using ALTER - TABLE Command.

(b) INSERT:

→ The right to insert rows with values in the named column of the table named as object including the columns which are added later.

(c) DELETE:

→ The right to delete the rows from the table named as object.

(d) REFERENCES:

→ The right to define foreign keys which refer the specific column of the table-object.

→ If the user has a privilege with the grant option, then he can pass it to other users (with or without grant option) by Using GRANT Command.

→ A User who creates a View has specified privileges which are used to define the View. If the view is updatable in the underlying table then the User holds the same privileges on the View.

→ A User who creates a base table automatically has all the privileges along with the rights to grant the privileges to other Users.

### (ii) Revoke:

→ A 'Revoke' Command takes away the privileges provided with Grant Command. It is the reverse of GRANT Command.

### Syntax:

---

Revoke [GRANT OPTION FOR] privileges ON object FROM users { RESTRICT / CASCADE };

---

→ The Revoke Command can withdraw either the GRANT OPTION on a privilege or the privilege itself.

→ When only the GRANT OPTION on a privilege is to be withdrawn, the 'GRANT OPTION FOR' clause is included in the Revoke Command, otherwise it is optional.

→ If the RESTRICT keyword is specified in the Revoke Command, then this command will be rejected if the revocation of a privilege from the specified user will result in revocation of the same privilege from other users as well.

→ If the 'CASCADE' keyword is specified in the Revoke Command then the privileges or grant option specified in the command will be revoked from all users who obtained this privilege through GRANT command and from the same user.

Examples of GRANT and Revoke Commands:-

→ Boss: GRANT SELECT ON Books TO Emp17 - with GRANT OPTION.

→ Emp17: GRANT SELECT ON Books TO Emp14 with GRANT OPTION.

→ Boss: Revoke SELECT ON Books FROM Emp17 RESTRICT.

## \* NULL Value Concepts:-

- It is an undefined, unknown, unavailable-value.
- Any Arithmetic operations are performed on Null Values then again it will become null only.
- If a column in a table is optional, we can insert a new record or update an existing record without adding a value to this column. This means that the field will be saved with a Null value.
- Null values are treated differently from other values.
- Null is used as a placeholder for unknown or inapplicable values.

### Note :-

- It is not possible to compare Null and 0; they are not equivalent.
- If a column in a row has no value, then the column is said to be Null, or to contain a Null.
- Any arithmetic expression containing a

Null always evaluates to Null.

Example:-

→ Null added to 10 is null, infact, all -  
- operators.

\* Principles of Null Values:-

(i) Setting a Null Value is appropriate

when the actual value is unknown.

(ii) If, Null Value is not equivalent to a

Value of zero if the Data type is number  
and spaces if the Datatype is character.

(iii) A Null Value will ("Equivalent") evaluate

to Null if any Expression. e.g.: Null multiplied  
by 10 is Null.

(iv) Null Value can be inserted into Columns

of any Data type.

(v) If the Column has a Null Value,

Oracle ignores the UNIQUE, FOREIGN KEY,

CHECK Constraints that may be attached

to the Column.

## \* Advanced Select Queries:-

### Select Statements with Operators:-

- An operator manipulates individual data items and returns a result. The Data items are called operands or arguments.
- Operators are represented by Special characters or by keywords.
- For Example, the multiplication operator is represented by an asterisk (\*) and the operator that tests for nulls is represented by the keyword IS NULL.
- There are two general classes of operators : Unary and Binary.

### \* UNARY OPERATORS:-

- A Unary operator uses Only one operand. A Unary operator typically appears with its operand in the following format:

Syntax:-

operator operand

### \* BINARY OPERATOR:-

- A Binary operator Uses two operands. A binary operator appears with its operands

in the following format:

Syntax :-

Operand1    operator    Operand2

\* CHARACTER OPERATORS :-

→ character operators are used in expressions to manipulate character strings.

Syntax:-

character operators

operator : "

Description: Concatenates character strings

Example:-

SELECT 'The Name of the employee is: '||ENAME FROM  
EMP;

\* OTHER OPERATORS :-

→ An operator is a reserved word or a character used primarily in an SQL statement's WHERE clause to perform operation(s), such as;

(i) Arithmetic operators [+,-,\*,/, mode] ]  
↳ (for finding remainder)

(ii) Comparison operators [Relational operators =, <,  
<=, >, >=, <> (or) != ]

(iii) Logical operators [AND, OR, NOT]

## (ir) Special Operators

Note:

→ Comparison operators, Logical operators and Special operators are used in "WHERE" clause.

Note:-

→ For SELECT after columns we are Using Arithmetic operator.

Note:

→ Arithmetic operators are used in number, Data Datatype columns.

### (i) SQL ARITHMETIC OPERATORS:

→ SQL mathematical operations are performed using mathematical operators (+, -, \*, /, and %).  
 → We can use SQL like a calculator to get a feel for how these operators work.  
 → These are also combined with query for performing calculation for the selected Data in the table.

→ These operations are combined within update command in order to perform calculation for the Data.



## SQL Mathematical Operators:

| Operator  | Description                     | Example                    |
|-----------|---------------------------------|----------------------------|
| + (Unary) | Makes operand positive          | SELECT +3 FROM DUAL;       |
| - (Unary) | Negates operand                 | SELECT -4 FROM DUAL;       |
| /         | Division (numbers & Dates)      | SELECT SAL/10 FROM EMP;    |
| *         | Multiplication                  | SELECT SAL * 5 FROM EMP;   |
| +         | Addition (numbers and - Dates)  | SELECT SAL + 100 FROM EMP; |
| -         | Subtraction (numbers and Dates) | SELECT SAL - 100 FROM EMP; |

## (ii) SQL COMPARISON OPERATORS:-

→ Comparison operators are used to compare the column Data with Specific Values in a Condition.

→ Comparison operators are also used along with the SELECT Statement to filter Data based on Specific Conditions.

→ These operators return with a Boolean - Value in Order to handle the Expression

these are combined with where clause of the SQL query which are used to retrieve, update, delete selected content of the table based on the condition given using these operators in where clause.

→ Some of the examples are shown below for the customer table. And the operators are explained with example in the below table:

→ The below table describes each comparison operator.

| Operator   | Description   | Example   |
|------------|---|---|
| =          | Equality test   | SELECT ENAME "Employee"<br>FROM EMP WHERE SAL = 1500;                 |
| !=, <>, <> | Inequality test   | SELECT ENAME FROM EMP<br>WHERE SAL != 5000;                           |
| >          | Greater than test   | SELECT ENAME "Employee",<br>JOB "Title" FROM EMP WHERE<br>SAL > 3000; |
| <          | Less than test  | SELECT * FROM PRICE WHERE<br>MINPRICE < 30;                           |
| >=         | Greater than or<br>equal to test                                | SELECT * FROM PRICE WHERE<br>MINPRICE >= 20;                          |
| <=         | Less than or equal<br>to test.                                  | SELECT ENAME FROM EMP<br>WHERE SAL <= 1500;                           |
| IN         | "Equivalent to any<br>member of" test.<br>Equivalent to "= ANY" | SELECT * FROM EMP WHERE<br>ENAME IN ('SMITH', 'WARD');                |

|                                |   |   |
|--------------------------------|---|---|
| ANY /<br>SOME                  | Compares a Value to Each<br>Value in a list of 9etwo-<br>ned by a query. Must be<br>preceded by =, !=, >, <, <=,<br>or >=. Evaluates to FALSE<br>if the query returns no<br>rows. | SELECT * FROM DEPT<br>WHERE LOC=SOME('NEW<br>YORK', 'DALLAS');                        |
| NOT IN                         | Equivalent to "!= ANY".<br>Evaluates to FALSE if any<br>member of the set is<br>NULL.   | SELECT * FROM DEPT<br>WHERE LOC NOT IN<br>('NEW YORK', 'DALLAS');                     |
| ALL                            | Compares a Value with<br>Every Value in a list of<br>returned by a query. Must<br>be preceded by =, !=, >, <,<br><=, or >=. Evaluates to<br>TRUE if the query returns<br>no rows. | SELECT * FROM EMP<br>WHERE SAL >= ALL<br>(1400, 3000);                                |
| [NOT]<br>BETWEEN<br>x and y    | [NOT] greater than or Equal<br>to x and less than or<br>Equal to y.   | SELECT ENAME, JOB<br>FROM EMP WHERE<br>SAL BETWEEN 3000<br>AND 5000;                  |
| EXISTS                         | TRUE if a Sub-query<br>returns at least one row.  | SELECT * FROM EMP<br>WHERE EXISTS<br>(SELECT ENAME FROM<br>EMP WHERE MGR IS<br>NULL). |
| X[NOT]<br>like y<br>[escape z] | TRUE if x does [not] match<br>the pattern y. Within y, the<br>character "%" matches any   | SELECT * FROM EMP<br>WHERE ENAME LIKE<br>'% E %'                                      |

String of zero or more characters except null.  
 The character "-" matches any single character. Any character following ESCAPE is interpreted literally, useful when y contains a percent (%) or underscore (-)

IS (NOT) NULL Tests for nulls. This is the only operator that should be used to test for nulls.

SELECT \* FROM EMP WHERE COMM IS NOT NULL AND SAL > 1500;

### (iii) SQL LOGICAL OPERATORS:-

→ There are three logical operators namely, AND, OR, and NOT.

→ These operators compare two conditions at a time to determine whether a row can be selected from for the output.

→ When retrieving data using a SELECT statement you can use logical operators in the WHERE clause, which allows you to combine more than one condition.

### Logical Operators Description

- OR for the row to be selected at least one of the conditions must be true.
- AND for a row to be selected all the specified

Conditions must be true.

→ NOT for a row to be selected the Specified Condition must be false.

|     |  |  |
|-----|--|--|
| NOT | Returns TRUE if the following Condition is FALSE.<br>Returns FALSE if it is TRUE. If it is UNKNOWN it remains UNKNOWN. | SELECT * FROM EMP WHERE NOT(job IS NULL)<br>SELECT * FROM EMP WHERE NOT(sal BETWEEN 1000 AND 2000) |
| AND | Returns TRUE if both Component Conditions are TRUE. Returns FALSE if either is FALSE; otherwise returns UNKNOWN.       | SELECT * FROM EMP WHERE job = 'clerk' AND deptno = 10  |
| OR  | Returns TRUE if either Component Condition is TRUE. Returns FALSE if both are FALSE. Otherwise, returns UNKNOWN.       | SELECT * FROM EMP WHERE job = 'CLERK' OR deptno = 10   |

(iv) Special Operators:-

|             |             |
|-------------|-------------|
| (a) in      | not in      |
| (b) between | not between |
| (c) is null | is not null |
| (d) like    | not like    |

(a) in:

- This operator is used to pick the values one by one from list of values.
- Generally in place of OR operator we are using in operator.
- Because in operator performance is very high compare to or operator.
- And also in operator is used in "multiple row" Subqueries

Syntax:-

```
SELECT * from tablename where particular
columnname in(list of Values);
```

Note:-

SQL> Select \* from Emp where deptno in  
(10, 20);

SQL> Select \* from Emp where deptno in(10,  
20, Null);

Both SQL Commands displayed same result.

Note:-

→ Not in operator does not work with Null values.

Ex: SQL> Select \* from Emp where deptno  
not in(10,20,null)

No rows Selected.

Ex: SQL>Select \* from Emp where deptno is  
(null);

No rows Selected.

Ex: SQL>Select \* from Emp where deptno  
not in(null);

No rows Selected.

Ex: SQL>Select \* from Emp where Ename in  
('KING', 'SMITH');

SQL>Select \* from Emp where Ename in  
('king', 'smith');

No rows Selected.

→ Because in our table the names are in  
Capital letters.

(b) Between Operator:-

→ This operator is used to retrieve range  
of values. This operator is also called as  
"between .... and " operator.

Syntax :-

Select \* from tablename where Columnname  
between lowvalue AND highvalue;

Ex:-

SQL> Select \* from Emp where Sal between  
2000 and 5000;

Note:- In Oracle between 2000 and 5000  
means it displays

|      |           |
|------|-----------|
| 2000 | 3001      |
| ;    | but not ; |
| 5000 | 4999      |

Note:-

→ In "between-operator" always the values  
are from low value to high value.

Ex:- 3000 to 5000 etc.

\* Null:-

→ It is an undefined, unknown, unavailable,  
→ Any Arithmetic operations are performed  
on null Values then again it will become  
null only.

Ex:- Null + 70 = Null (or) Null \* 10 = Null etc.

\* Display ename, Sal, Comm, Sal+Comm of the Emp SMITH from the Emp table.

( $\because$  Given that, in our table Comm for SMITH is null)

SQL> Select ename, Sal, Comm, Sal+Comm from Emp where ename = 'SMITH';

Output:

| <u>ENAME</u> | <u>SAL</u> | <u>COMM</u> | <u>SAL+COMM</u> |
|--------------|------------|-------------|-----------------|
| SMITH        | 800        |             |                 |

→ To Overcome this problem Oracle introduced NVL() function.

\* NVL :-

→ NVL() is a predefined function which is used to substitute (or) replace Userdefined values in the place of Null.

Ex: null + 70 = null

$$0 + 70 = 70$$



(User defined Value)

Syntax:

$\boxed{\text{NVL}(\text{Exp1}, \text{Exp2})}$

→ Here these 2 expressions i.e., Exp 1 and Exp 2 must belongs to same datatype.

Ex:

NVL(number, number)  
NVL(10, 20)

NVL(char, char)  
NVL(a, b).

Note:

→ If Exp1 is null then it returns Exp2  
otherwise it returns Exp1.

Ex:1: NVL(null, 50)

$$\begin{aligned} &\text{null} + 70 \\ &50 + 70 = 120 \end{aligned}$$

$$\begin{aligned} &\textcircled{2} \quad \text{NVL}(20, 50) \\ &\text{null} + 70 \\ &20 + 70 = 90 \end{aligned}$$

∴ The previous SQL> becomes

SQL> Select ename, sal, comm, sal+NVL(comm, 0) from emp where ename = 'SMITH';

Output:

| ENAME | SAL | COMM | SAL+NVL(COMM, 0) |
|-------|-----|------|------------------|
| SMITH | 800 |      | 800              |

(Or)

SQL> Select ename, sal, comm, sal+NVL(0, 0) from emp where ename = 'SMITH';

Output:

| <u>ENAME</u> | <u>SAL</u> | <u>COMM</u> | <u>SAL+NVL(0,0)</u> |
|--------------|------------|-------------|---------------------|
| SMITH        | 800        |             | 800                 |

(or)

SQL> Select ename, sal, comm, sal+NVL(null,0) from emp where ename = 'SMITH';

Output:

| <u>ENAME</u> | <u>SAL</u> | <u>COMM</u> | <u>SAL+NVL(null,0)</u> |
|--------------|------------|-------------|------------------------|
| SMITH        | 800        |             | 800                    |

$$\begin{aligned}
 \text{Here: } &= \text{Sal} + \text{NVL}(\text{Comm}, 0) \\
 &= 800 + \text{NVL}(\text{null}, 0) \\
 &= 800 + 0 \\
 &= \underline{\underline{800}}
 \end{aligned}$$

(C) IS Null, is Not Null:-

These operators are used in where clause only.

These operators are used to test whether a column having null values (or) not.

Syntax:

Select \* from tablename where Columnname is null;

(or)

Syntax:

Select \* from tablename where Columnname  
is not null;

\* Display the Employees who are not getting Comm from Emp table?

SQL> Select \* from Emp where Comm is null;

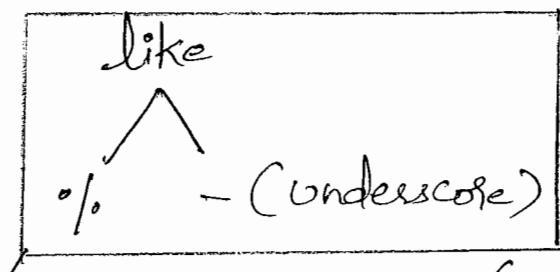
\* Display the Employees who are getting Comm from Emp table?

SQL> Select \* from Emp where Comm is not null;

(d) Like Operator:-

→ This operator is used to search Data based on character pattern.

→ Along with like operator we are using 2 "wild Card" characters, those are



(∴ for Single character)

(%) Used for  
group of characters  
(or) string)

Syntax:

Select \* from tablename where Columnname  
like 'characterpattern';

\* Display the employees whose ename Starts  
with M. from Emp table?

SQL> Select \* from Emp where ename like  
'M%';

\* Display the employees whose ename Second  
letter would be L from Emp table?

SQL> Select \* from Emp where ename like  
'\_L%';

\* CONCATINATION OPERATOR (||= double pipes):-

→ This operator is used to Concatinate Column  
values with literal strings and also used to  
concatinate number of columns.

Ex:

Select ename || ' ' || sal from Emp;

Output:

ENAME || ' ' || SAL

## CONSTRAINTS

44

- Constraints are used to prevent invalid data entry into our tables.
- Constraints are created on table columns.
- Oracle Server supports the following types of constraints.
  1. NOT NULL
  2. Unique
  3. Primary key
  4. Foreign key
  5. Check
- The above are also called as constraints (or) constraint types.
- All above constraints are defined in 2 levels.
  - i) Column level constraints
  - ii) Table level constraints.
- i) Column level constraints:
  - In this method we are defining constraints on individual columns i.e., whenever we are defining the column then only we are using constraint type.

Syntax:

```
create table tablename(col1 datatype (size) constraint  
type, col2 datatype (size) constraint type---);
```

→ ex: create table bank

```
(acctno number(10) not null  
name varchar2(10) primary key  
balance number(10));
```

## ii) Table level constraints:-

→ In this method we are defining constraints on group of columns i.e., first we have defining all columns and at last position we are specifying constraint type along with group of columns.

Syntax:-

```
create tablename(col1 datatype (size), col2 datatype  
(size), ---, constraint type (col1, col2));
```

\*NOTE:-

|             | At column level | At table level |
|-------------|-----------------|----------------|
| NOTNULL     | Possible        | Impossible     |
| unique      | possible        | possible       |
| primary key | possible        | possible       |
| foreign key | possible        | possible       |
| check       | possible        | possible       |

## ① NOT null:-

- NOT null constraint does not support table level.
- This constraint does not accept null values but it will accept duplicate values.

### i) column level:-

Ex: SQL> create table t1(sno number(10) not null  
name varchar2(10));

table created

SQL> desc t1;

| Name | NULL?    | Type         |
|------|----------|--------------|
| sno  | not null | number(10)   |
| name | -        | varchar2(10) |

Ex: SQL> create table t1(sno number(10) not null,  
name varchar2(10) not null);

Table created

SQL> desc t1;

| Name | NULL?    | Type         |
|------|----------|--------------|
| sno  | not null | number(10)   |
| name | not null | varchar2(10) |

SQL> insert into t<sub>1</sub> values (null, 'xyz');

ORA-1200: cannot insert null into sno.

## 2. unique:

→ This constraint does not accept duplicate values but it will accept null values.

→ And also oracle server automatically creates "btree indexes" on unique constraints columns.

### i) column Level:

SQL> create table t<sub>2</sub>(sno number(10) unique, name varchar2(10));

### ii) Table level:

SQL> create table t<sub>3</sub>(sno number(10), name varchar2(10), unique(sno, name));

table created

SQL> desc t<sub>3</sub>;

| Name | null? | Type         |
|------|-------|--------------|
| sno  |       | number(10)   |
| name |       | varchar2(10) |

### NOTE:-

column level:

| sno | name |
|-----|------|
| ✓ 1 | abc  |
| * 1 | xyz  |

Table level

| sno | name     |   |
|-----|----------|---|
| 1   | sowjanya | ✓ |
| 1   | sowmya   | ✓ |
| 1   | sowjanya | ✗ |

→ Because in table level record level checking is available

Ex:- SQL> select \* from t3;

| sno | name     |
|-----|----------|
| 1   | sowjanya |
| 1   | sowmya   |

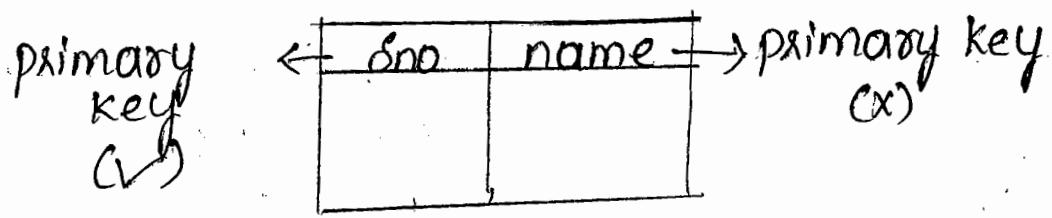
(3) primary key:-

→ primary key uniquely identifies a record (or) a row in a table, there can be only one "primary key" in a table.

→ And also primary key does not accept duplicate values and null values..

→ And also oracle server automatically creates "btree indexes" on those columns.

i) column level:-



i) create table t1(sno number (10) primary key, name varchar(10));

ii) Table level:-

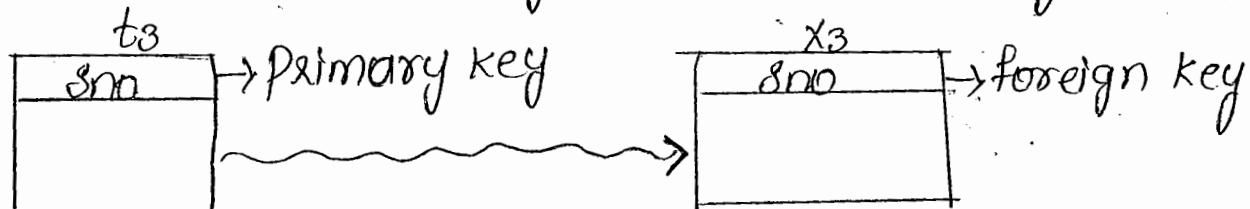
SQL> create table z1(sno number (10), name varchar (10), primary key (sno, name));

→ This table level primary key is also called as composite primary key i.e., This is a combination of columns in a single primary key.

4) Foreign key:-

→ If we want to establishes relationship between tables then we are using "Referential integrity constraint" foreign key.

→ One table foreign key must belongs to another table primary key And also these two columns must belongs to same datatype.



- Always foreign values based on primary key values only.
- Generally primary key does not accept duplicate, null values whereas foreign key accepts duplicate, null values.

| t <sub>3</sub> |
|----------------|
| Sno            |
| 1              |
| 2              |
| 3              |

| x <sub>3</sub> |
|----------------|
| Sno            |
| 1✓             |
| 2✓             |
| 3✓             |
| 4✗             |
| 1✓             |
| 2✓             |
| 3✓             |
| 2✓             |
| 3✓             |

Foreign  
key

SQL> create table t<sub>1</sub> (Sno number(10) primary key);

Table created

SQL desc t<sub>1</sub>;

| Name | NULL?   | Type   |
|------|---------|--------|
| Sno  | NOTNULL | NUMBER |

\* Column level:- References)

Syntax:-

- create table tablename(foreign key column name datatype (size) references master table name(primary key column name));
- The above syntax is applicable if foreign key column name and primary key column names are different (or) foreign key column name,

primary key column names are same.

Ex:-

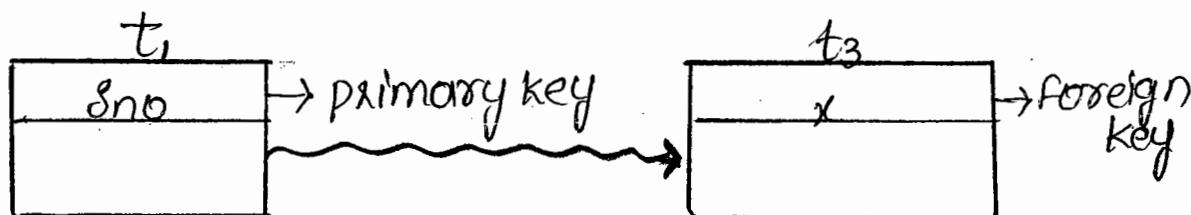
SQL> create table t<sub>1</sub>(Sno number(10) primary key);

SQL> create table t<sub>2</sub>(Sno number(10) references t<sub>1</sub>);

SQL> create table t<sub>1</sub>(Sno number(10) primary key);

SQL> create table t<sub>2</sub>(Sno number(10) references t<sub>1</sub>(Sno));

Ex:-



→ Sno, x columns must have same datatypes.

SQL> create table t<sub>1</sub>(Sno number(10) primary key);

SQL> create table t<sub>3</sub>(x number(10) references t<sub>1</sub>(Sno));

\* Table level (foreign key, references):-

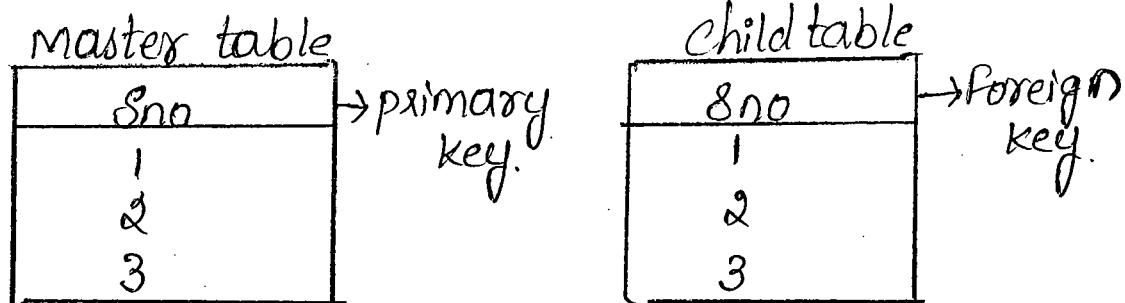
Syntax:-

→ create tablename (col<sub>1</sub>, datatype(size), col<sub>2</sub>, datatype(size), ..., foreign key (col<sub>1</sub>, col<sub>2</sub>, ...)) references master table name (primary key column names);

Ex:- SQL> create table t<sub>5</sub>(Sno number(10), name varchar2(10), primary key (Sno, name));

SOL) create table k5(Sno number (10), name varchar(20), foreign key(Sno, name) references t5);

Ex:-



Default:-

- This constraint provides a value to an attribute when a new data attribute value is inserted into a table.
- It triggers only when the end user does not make any entry for a particular attribute that is specified as default.

Example:-

EMP-SAL CHAR(6) DEFAULT '20,000' NOT NULL.

- This specifies that a default value '20,000' is assigned to the attribute EMP-SAL.
- When a new row is inserted in the EMPLOYEE table and when the end user does not make any entry for that attribute i.e., EMP-SAL.

CHECK:-

- This constraint is used for data validation when

a new value is inserted into a table.

- (consequently, it checks whether the specified condition is satisfied) then the data is accepted
- consequently, it checks whether the specified condition is satisfied or not.
- If the condition is satisfied then the data is accepted by the RDBMS; otherwise the data is not accepted an error message is produced.

Example:-

```
EMP_SAL : INTEGER CHECK (EMP_SAL IN  
                           ('20,000', '17000', '25,000'))
```

- This code checks whether the EMP\_SAL lies within the given range of values 20000, 17000 and 25000.

## \* SQL INDEXES:-

- Indexes are Special look up tables that the Database Search Engine can use to Speed up Data retrieval.
- An index is a pointer to Data in a table.
- An index in a Database is Very Similar to an index in the back of a book.
  - for Example, if you want to reference all Pages in a book that discuss a certain topic, you first refer to the index, which lists all topics alphabetically, and are then referred to one or more specific page-numbers.
- An index helps Speed up Select Queries and Where clause, but it slows down Data input with UPDATE and INSERT statements.
- Indexes can be created or dropped with no effect on the Data.
- Creating an index involves the CREATE

INDEX Statement, which allows you to name the index to specify the table and which column or columns to index, and to indicate whether the index is in ascending or descending order.

### CREATE INDEX COMMANDS:-

→ The Basic Syntax of CREATE INDEX is as follows:

`CREATE INDEX index-name ON Table-name;`

### Single-Column indexes:-

→ A Single-Column index is one that is created based on only one table column. The basic Syntax is as follows:

#### Syntax:

`CREATE INDEX index-name ON Table-name(Col-name)`

### UNIQUE indexes:-

→ Unique indexes are used not only for performance, but also for Data integrity.  
→ A Unique index does not allow any

u  
duplicate values to be inserted into the table.  
→ The basic syntax is as follows:

Syntax:-

CREATE INDEX index-name ON table-name (Col-name);

Composite indexes:-

- A Composite indexes is an index on two or more columns of a table.  
→ The basic Syntax is as follows:

Syntax:-

CREATE INDEX index-name ON table-name (Col1, Col2);

Implicit indexes:-

- An index can be dropped Using SQL Drop Command.  
→ The basic Syntax is as follows:

Syntax:-

DROP INDEX index-name;

## When Should indexes be avoided?

→ The following guidelines indicates when the use of an index because performance may be slowed or improved.

- Indexes should not be used on small tables.
- Tables that have frequent, large batch update or insert operations.
- Indexes should not be used on columns that contain a high number of Null-values.
- Columns that are frequently manipulated should not be indexed.

## \* VIEWS :-

→ View is a database object, which is used to provide authority level of security.

→ Generally Views are created by Database administrator.

→ Generally Views does not stores data that's why Views are also called as Virtual tables.

- View is also called as window of a table
- A View is actually a composition of a table in the form of a predefined SQL-query.
- A View can contain all rows of a table or select rows from a table.
- Generally Views are created from Base-tables, Based on the base table '2' types of Views supported by all Database Systems.

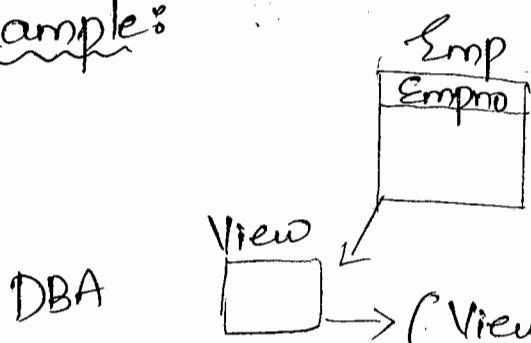
(i) Simple View

(ii) Complex View

(i) Simple View:-

→ A View which is created from single table those type of Views are called "Simple Views"

Example:



(View is created by Using -  
- Only one table)

Syntax:-

```
Create View View-name As  
SELECT Column1, Column2.....  
FROM table-name  
WHERE [Condition];
```

Example:-

SQL > Create or replace View V1  
as

Select \* from emp where deptno=10;

→ We can also perform DML operations through Simple Views to base table based on the restrictions.

WITH CHECK OPTION:-

→ If we want to create constraints on Views then we are writing with check option clause.

→ When we are writing with check option clause we are inserting where Condition visibility Values only.

## Syntax:-

Create or replace View Viewname  
as

Select \* from tablename where Condition  
with check option;

→ The check option is a CREATE VIEW  
statement option.

→ The purpose of the with check option is  
to ensure that all update and insert  
satisfy the conditions in the view definition.  
If they do not satisfy the condition(s),  
the UPDATE (OR) INSERT returns an error.

## Example:

CREATE VIEW Customers\_View AS

SELECT name, age

FROM CUSTOMERS

WHERE age is NOT NULL

WITH CHECK OPTION;

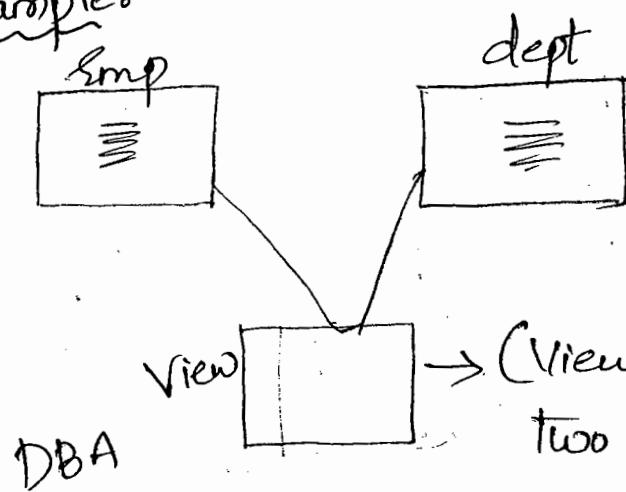
→ The with check option in this case should  
deny the entry of any null values in the  
view's AGE column, because the view is

defined by Data that does not have a Null Value in the AGE Column.

### (iii) Complex Views:-

→ A View which is created from multiple tables those type of Views are called Complex Views.

#### Example:-



→ (View is Created by Using two (or) more tables)

#### Ex:-

SQL> create or replace View V1

as

Select ename, sal, dname, loc from emp,dept  
where emp.deptno = dept.deptno;

→ Whenever we are trying to perform DML operations through Complex view to base table some table columns are effected by some other table columns are not effected.

### Updateable Views:-

→ Views can also be used for Data manipulation

Views on which Data manipulation can be done are called Updateable Views.

→ For a View to be updateable, it should meet the following criteria:

Views defined from Single table :-

- (a) If the User wants to INSERT records with the help of a View, then the PRIMARY key column(s) and all the NOTNULL columns must be included in the View.
- (b) The User can update, Delete records with the help of a View even if the primary key column and NOTNULL columns are excluded from the View definition.

A View can be updated under certain conditions.

→ The Select clause may not contain the keyword Distinct.

→ The Select clause may not contain Summary functions.

→ The Select clause may not contain Set operators.

→ The Select clause may not contain an ORDER

By clause.

→ The FROM clause may not contain as Multiple tables.

→ The Where clause may not contain Subqueries.

→ The Query may not contain Subqueries.

- The Query may not contain Group by or Having.
- Calculated Columns may not be updated.
- All NOT Null Columns from the base table must be included in the View in order for the Insert Query to function.

Inserting Rows into a View:-

- Rows of Data can be inserted into a View.
- The Same rules that apply to the update Command also apply to the INSERT Command.

Dropping Views:-

- Obviously, where you have a View, you need a way to drop the View if it is no longer needed.
- The Syntax is Very Simple as given below:

Drop VIEW View-name;

Deleting Rows into a View:-

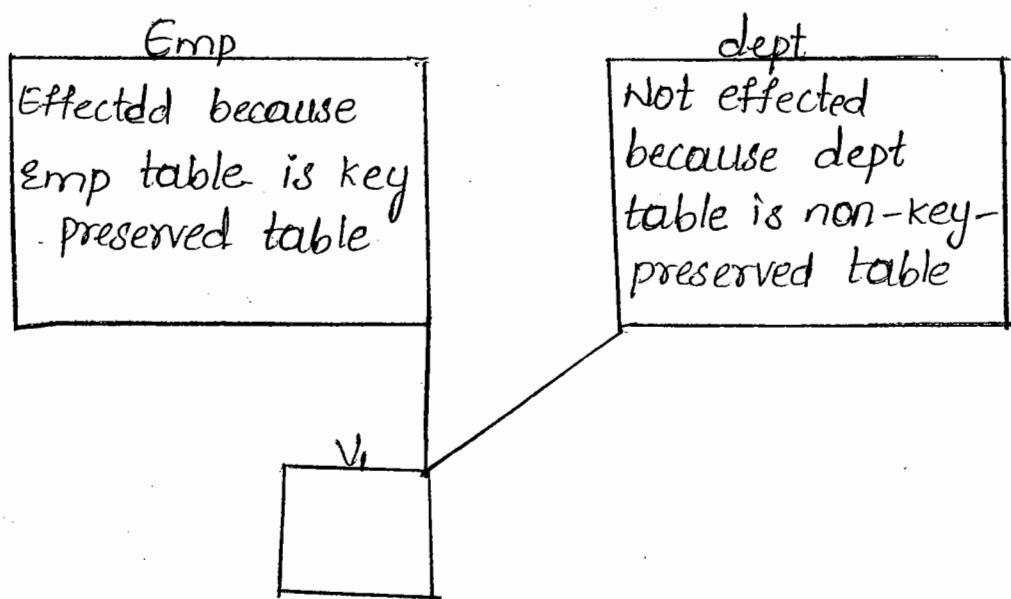
- Rows of Data can be deleted from a View.

- The Same rules that apply to the update and insert Commands apply to the Delete - Command.

→ Following is an example to delete a record having AGE = 32.

SQL > DELETE FROM CUSTOMERS-VIEW

where age = 32;



→ DBA creates view V1 based on emp & dept.

→ Data entry operator performs DML operations on that view.

→ In that DML operations, DML operations are effected to some tables and not effected to some other tables. Effected tables are key preserved and not effected tables are non-key preserved tables

→ Data entry operator informed this situation

to project manager. project manager searches the PL/SQL developer.

→ PL/SQL developer solves this problem efficiently successfully.

→ To overcome this problem oracle introduced "instead of triggers in PL/SQL".

→ "Instead of triggers" are created on views. By default "instead of triggers" are row level triggers.

\* Triggers (PL/SQL):-

\* Triggers = automatically performed operations.

Triggers:-

→ It is also same as "stored procedure" and also it will automatically invoked, whenever DML operations performed on table.

→ There are 2 types of triggers supported by Oracle.

- i) Statement level trigger.
- ii) Row level trigger.

→ In statement level triggers, trigger body is executed only once only once for a DML statement whereas in row-level triggers, trigger

body is executed for each row for DML statement.

Syntax:-

→ create or replace trigger triggername  
(insert/update/delete) on tablename [for each row]  
→ Trigger specification.

Begin  
===== } Trigger body.  
end;

Trigger event

Trigger timing

before/After

used when we are using row level trigger

\* Difference between statement and row level triggers:

SQL> create table test(col1 date);

\* statement level triggers:

SQL> create or replace trigger tcl  
after update on emp

begin  
insert into test values(sysdate);  
end;  
/

Execution:-

SQL> update emp set sal=sal+100 where deptno=10;

3 rows updated

SQL> select \* from test;

COL1

31-Jan-13

∴ 3 rows updated but only on time sysdate (31-Jan-13) is displayed because it is statement level triggers.

SQL> drop trigger tc1;

\* Row level trigger:-

SQL> create or replace trigger tc1

after update on emp

for each row

begin

insert into test values(sysdate);

end;

/

Trigger created

execution:

SQL> update emp set sal=sal+100 where deptno=10;  
3 rows updated.

SQL> select \* from test;

COL1

31-JAN-13 → Stmt level trigger

31-JAN-13

31-JAN-13

31-JAN-13

3 rows updated &  
3 times sysdate (31-Jan-  
13) is displayed  
because it is row  
level trigger.

\* Row level triggers:-

→ In row level triggers, trigger body is executed for each row for DML statement. That's why we are using "for each row" clause in triggers specification. And also data automatically stored in

2 rollback segment qualifiers.

→ These are (i)old (ii)New

→ When we are using those qualifiers in trigger body we must use colon(:) in front of the qualifier name.

Syntax: :old.columnname and

Syntax: :new.columnname

\*

|      | insert | update | delete |
|------|--------|--------|--------|
| :new | ✓      | ✓      | X      |
| :old | X      | ✓      | ✓      |

Q: Write a PL/SQL row level trigger on employee table whenever user deletes data in some rows those rows are automatically stored in another table.

SQL> create table backup as select \* from emp  
where 1=2;

SQL> create or replace trigger tc1  
after delete on emp  
for each row

begin  
insert into backup values (:old.empno,  
:old.ename, :old.job, :old.mgr, :old.hiredate,

:old.sal, :old.comm, :old.deptno);

end;

/

### Execution:-

SQL> delete from emp where sal>2000;

SQL> Select \* from emp;

SQL> select \* from backup;

→ Statement level triggers are fast because these executed each row at a time. But row level triggers are slow because these executed for each row.

→ Whenever user deletes data in some rows Those rows are automatically stored in another table by using row level triggers only but not by using statement level triggers.

### \*Instead of triggers\*

→ Oracle 8i. introduced instead of triggers instead of triggers are created on views. By default instead of triggers are row level triggers.

### Syntax:-

Create or replace trigger triggername

instead of insert/ update / delete on viewname

for each row.

Begin

end;

\* Before instead of triggers:

SQL> update v, set dname='xyz'  
where dname='SALES';

error:

→ cannot modify a column which maps to  
non-key preserved table.

\* instead of triggers:

SQL> create or replace trigger tcq  
instead of update on v,  
for each row

begin

update dept set dname=:new.dname where  
dname=:old.dname;

update dept set loc=:new.loc where loc=:old.loc;  
end;

/

execution:

SQL> update v1 set dname='xyz'  
where dname='SALES';  
rows updated.

\*By using the above solution dept table is also effected.

## \* TRIGGERS:-

- Triggers and procedures are very powerful database objects because they are stored in the database and controlled by the DBMS.
- Both triggers and procedure that initiates an action when an event occurs.
- They are stored and managed by DBMS.
- Triggers cannot be called or executed; the DBMS automatically fires the trigger as a result of a data modification to the associated table.
- Triggers are used to maintain the referential integrity of data by changing the data in a systematic fashion.
- Each trigger is attached to a single, specific table in the database.
- Database triggers are database objects created via the SQL\* PLUS tool on the client and stored on the server in the oracle engines system table.
- These database objects consists of the following distinct sections.

- A named database event.
- A PL/SQL block that will execute when the event occurs.

### use of database Triggers:-

→ oracle engine supports database triggers it provides a highly customizable database management system.

→ To customize management information by the oracle engine are as follows.

- A trigger can permit DML statements against a table.
- A trigger can also be used to keep an audit trail of a table.
- It can be used to prevent invalid.
- Enforce complex security authorizations.

### How to apply database Triggers:-

→ A trigger has three basic parts.

1. A triggering event or statement.
2. A trigger Restriction.
3. A trigger action.

## 1.Triggering Event or statement:-

- It is a SQL statement that causes a trigger to be fired.
- It can be INSERT, UPDATE OR DELETE statement for a specific table.

## 2.Trigger Restriction:-

- A trigger restriction specifies a boolean(logical) expression that must be TRUE for the trigger to fire.
- A trigger restriction is specified using a WHEN clause.

## 3.Trigger Action:-

- A trigger action is the PL/SQL code to be executed when a triggering statement is encountered and any trigger restriction evaluates to TRUE.

## Types Of Triggers:-

- Depending upon when a trigger is fired, it may be classified as:
  - Statement-level triggers.
  - Row level triggers.

- Before triggers

- After triggers.

### Statement level triggers:-

→ A statement trigger is fixed only for once for a DML statement irrespective of the number of rows affected by the statement.

→ For example, if you execute the following UPDATE command STUDENTS table, statement trigger for update is executed by only for once.

→ Update student set bcode='b3'  
where bcode='b2';

→ However, statement triggers cannot be used to access the data that is being inserted, updated or deleted. In other words, they do not have access to keywords new and old, which are used to access data.

→ Statement level triggers are typically used to enforce rules that are not related to data for example; it is possible to implement a rule that says "no body can modify BATCHES table after 9 P.M.

→ statement level trigger is that default type of trigger.

### Row level Triggers:-

→ A row trigger is fired once for each row that is affected by DML command for example, if an update command updates 100 rows then row-level trigger is fired 100 times whereas a statement level trigger is fired only once.

→ Row level triggers are used to check for the validity of the data. they are typically used to implement rules that cannot be implemented by integrity constraints.

→ Row level triggers are implemented by using the option FOR EACH ROW in CREATE TRIGGER statement.

### Before triggers:

→ While defining a trigger, you can specify whether the trigger is to be fired before the command is executed.

→ Before triggers are commonly used to check the validity of the data before the action is

performed.

→ For instance, you can use before trigger to prevent deletion of row if deletion should not be allowed in the given case.

After triggers:-

→ After triggers are fired the triggering action is completed.

→ For example, if after trigger is associated with insert command then it is fired after the row is inserted into the table.

Syntax for creating a trigger:-

```
CREATE OR REPLACE TRIGGER [schema] triggername
{Before, After}
{Delete, INSERT, UPDATE [OF column, ...]}
ON [schema] table name
[REFERENCING {OLD AS old, NEW AS new}]
[FOR EACH ROW [WHEN condition]]
DECLARE
    Variable declarations;
    Constant declarations;
BEGIN
    PL/SQL subprogram body;
EXCEPTION
    exception pl/sql block;
END;
```

## \*JOINS\*

20

- Joins are used to retrieve data from multiple tables
- If they are joining 'n' tables We are using 'n-1' joining conditions
- Oracle Server Supports 4 types of joins
  - 1. Equi join (or) inner join
  - 2. Non Equi join
  - 3. Self join
  - 4. Outer join

→ The above joins are also called as 8<sup>i</sup>-joins, because those are used upto oracle 8<sup>i</sup>

### \*9i joins (or) ANSI Joins -

- i) inner join
- ii) left outer join
- iii) right outer join
- iv) full outer join
- v) natural join

1. Exe - Select \* from Emp;  
14 rows are displayed

2. Exe - SQL>select \* from dept;  
4 rows are displayed

## Notes-

- We can also retrieve data from multiple tables without using "joining" conditions. In this case Oracle Server uses Crossjoin. This cross-join implemented based on the "Cartision product"
- In SQL joins default join is "Cross-join"

## ③ Exs-

SQL> select ename, sal, deptno, loc from Emp, dept;  
56 rows are displayed

→ The above example is without joining concept, which is displayed from Emp and dept tables

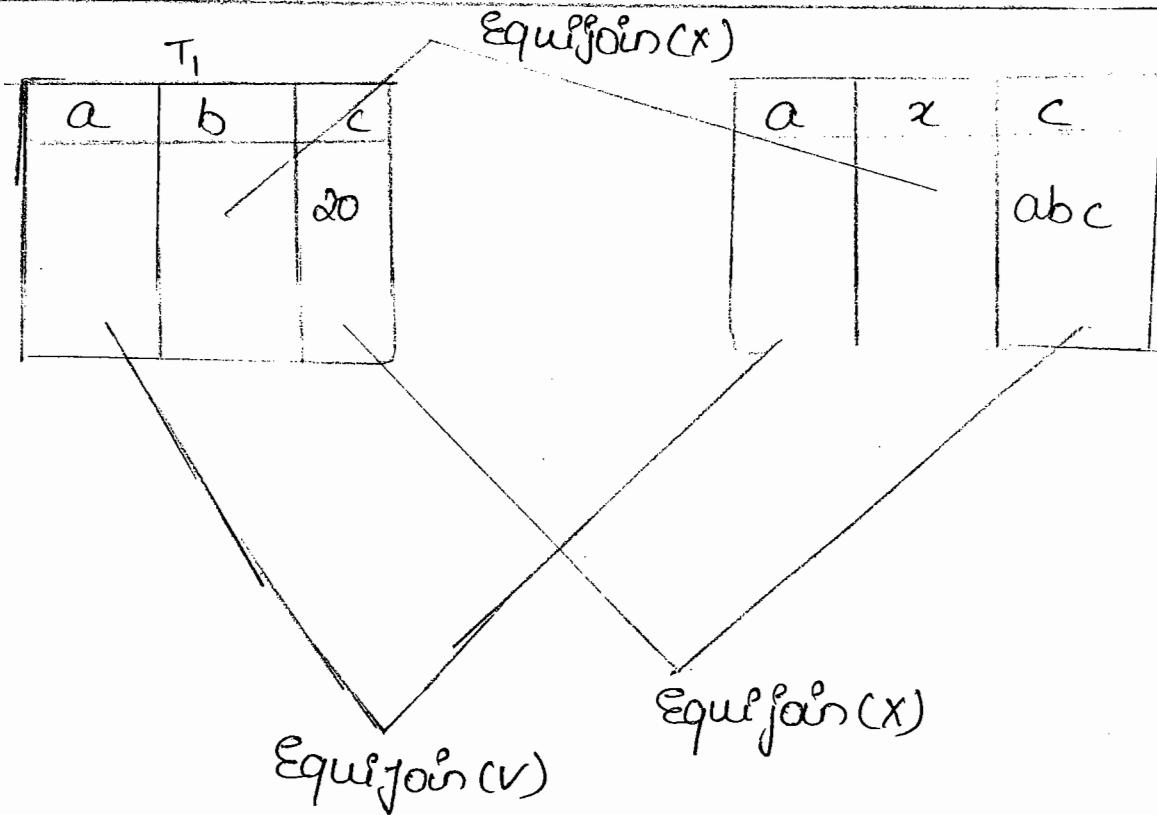
→ There is a Cross-join i.e.  $14 \times 4 = 56$  rows

## 1. Equi Joins-

→ Based on equality condition we are retrieving data from multiple tables. Here joining conditional columns must belongs to same datatype.

→ When tables contains common columnnames then only we can use this Equi join  
(∴ Even one columnname is similar in 2 tables then only we can use this Equi join)

## Exs



- In the above Example  $a$  of  $T_1$  and  $a$  of  $T_2$  can join because  $T_1$  contains the columnname  $a$  &  $T_2$  is also contains that Similar columnname  $a$
- $T_1$  of  $c$  contains number datatype and  $T_2$  of  $c$  contains Varchar datatype so we can't join
- Syntax of Equijoin-

Select col1, col2, ---  
from table1, table2 .

Where table1.common columnname = table2.common  
columnname

Ex:-

SOL>select ename, sal, deptno, dname, loc from Emp.  
dept where Emp.deptno = dept.deptno;

error :- column ambiguously defined

Explanation :-

SQl>select<sup>③</sup> ename, sal, deptno, dname, loc

① from Emp, dept

② where Emp.deptno = dept.deptno;

→ first control goes to "from clause" and it will check whether Emp, dept tables are available (or) not, if those tables are available the control goes to "where clause"

→ In "where clause" it checks the condition if Condition is correct then control return true otherwise false

→ After that it will goes to "Select clause" and it will checks whether there is common column of Emp and dept are available (or) not, if Common column names are available then it will gives an error as "column ambiguously defined"

→ for solving the above problem we should write

SQl>select ename, sal, dept.deptno, dname, loc

from Emp, dept where Emp.deptno = dept.deptno;  
(or)

SQl>Select ename, Sal, Emp. deptno, dname, loc from Emp, dept Where Emp. deptno = dept. deptno;

Notes - Generally to avoid future ambiguity we must specify every columnname along with tablename

Exe - SQl>Select Emp.ename, Emp.sal, Emp.deptno, dept.dname, dept.loc from Emp, dept Where Emp.deptno = dept.deptno;

→ The above Example is very complex so we should use "aliasnames"

\* Using aliasnames-

SQl>Select e.ename, e.sal, d.deptno, d.dname, d.loc from Emp e, dept d  
Where e.deptno = d.deptno;  
(or)

SQl>Select ename, sal, d.deptno, dname, d.loc  
from Emp e, dept d  
Where e.deptno = d.deptno;

→ But here deptno 40 does not displayed Even we are using d.deptno also

Notes - By using Equijoins We are retrieving matching rows only

Exe [dept table]

[Emp table]

deptno

deptno

|          |                     |
|----------|---------------------|
| 10 ----- | 10 (matching)       |
| 20 ----- | 20 (matching)       |
| 30 ----- | 30 (matching)       |
| 40 ----- | NULL (Not-matching) |

So equijoin does not display '40' deptno

Q8- Display the Employees who are working in the location Chicago from emp, dept tables using Equijoin?

SOL>Select ename, loc

from Emp, dept

Where Emp.deptno = dept.deptno

AND loc = 'CHICAGO';

Note- In 8i joins we can't use another where clause after first where clause. In that place we can use "AND" operator

→ If we want to use conditional clauses after joining condition then we must use "AND" operator in 8i joins

→ Inhere as in 9i joins we can use either "and" (or) "where" in the place of "and" operator

Q6- Write a join to display dname, Sum(sal) from Emp, dept tables?

SOL> Select dname, Sum(sal)

from Emp e, dept d

where e.deptno = d.deptno

group by dname;

Notes- In question the group function i.e Sum(sal) is given so we should use "group by" clause

Output:-

| DName      | Sum(SAL) |
|------------|----------|
| Accounting | 15201.25 |
| Research   | 14619.38 |
| Sales      | 8571.75  |

Ex :- SOL> Select d.deptno, dname, Sum(sal)

from Emp e, dept d

where e.deptno = d.deptno

groupby d.deptno, dname;

| O/P:- | deptno | dname      | Sum(sal) |
|-------|--------|------------|----------|
|       | 10     | Accounting | 15201.25 |
|       | 20     | Research   | 14619.38 |
|       | 30     | Sales      | 8571.75  |

Ex :- SOL> Select dname, Sum(sal)

from Emp e , dept d  
Where e.deptno = d.deptno  
group by dname  
having sum(sal) > 10000

| <u>O/P:-</u> | <u>dname</u> | <u>Sum(sal)</u> |
|--------------|--------------|-----------------|
|              | Accounting   | 15201.25        |
|              | Research     | 14619.38        |

Exe- SQL> select dname, sum(sal)

from Emp e, dept d  
Where e.deptno = d.deptno  
group by rollup(dname);

↳ (to display grand total value)

## 2. Non-Equi joins -

→ Based on other than Equality condition (<>, <, <=, >, >=, between, in) We are retrieving data from multiple tables . This join is also called as "between and" join

Exe- SQL> select \* from Emp;

Emp table displayed.

SQL> select \* from dept;

Dept table displayed.

SQL> select \* from salgrade;

Salgrade table displayed

Notes- deptno is the common columnname in Emp and dept tables So we can use "Equi join" to those two tables

→ But there is no similar column in Salgrade with Emp and dept. In this case we should use "Non-Equi join"

Ex :- SQL> Select ename, sal, losal, hisal

from Emp, Salgrade

Where Sal >= losal and Sal <= hisal

(09)

SQL> Select ename, sal, losal, hisal

from Emp, Salgrade

Where sal between losal and hisal;

③ Self join :-

→ Joining a table to itself is called Self join

→ Here joining conditional columns must belongs to Same datatype

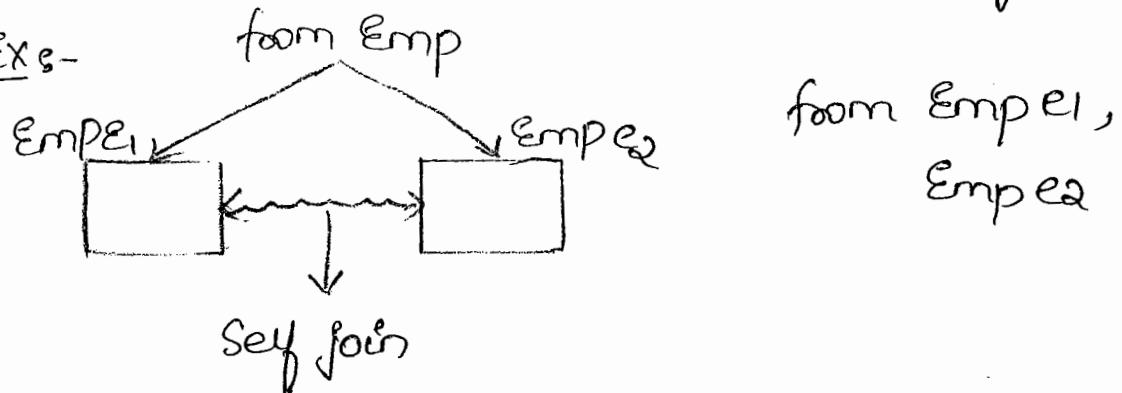
→ Generally self join is used to compare number of columns with one column in a Singletable (09)

→ It is used to compare one value with all other values in a Single column

→ In self-joins we must create alias-names for the table in from-clause

→ These alias-names must be different names otherwise database server returns ambiguity

→ Ex-



from Empl,  
Empr

Syntax - from tablename aliasname1, tablename  
aliasname2;

Impl.v)

Q1:- Display Emprname, their manager names from Emp table using Self join ?

SQL>Select el.ename "Employees",  
      e2.ename "manager"

from Emp el, Emp e2

Where el.mgr=e2.empno;

(or)

SQL>Select el.ename, el.mgr, e2.empno, e2.ename  
from Emp el, Emp e2

### Ex- Explanations-

| Emp E <sub>1</sub> |       |      |
|--------------------|-------|------|
| Empno              | ename | mgr  |
| 7369               | Smith | 7902 |
| 7499               | Allen | 7698 |
| 7902               | ford  | 7560 |
| 7698               | Blake | 7890 |

| Emp E <sub>2</sub> |        |      |
|--------------------|--------|------|
| Empno              | ename  | mgr  |
| 7369               | Smith  | 7902 |
| 7499               | Allen  | 7696 |
| 7902               | (ford) | 7566 |
| 7698               | Blake  | 7899 |

(Smith - ford)

Q6- Display the employees who are joining in the same month from Emp table using Self join?

SOL> Select e1.ename, e1.hiredate, e2.hiredate, e2.ename

from Emp e1, Emp e2

where to\_char(e1.hiredate,'mon')=to\_char(e2.hiredate,'mon') and e1.Empno > e2.Empno,

(To avoid redundant data)

Q8- Display the employees who are getting same salary in different departments from Emp table using self join

SOL> Select e1.ename, e1.sal, e1.deptno, e2.ename, e2.sal, e2.deptno

from Emp e<sub>1</sub>, Emp e<sub>2</sub>

where e<sub>1</sub>.sal = e<sub>2</sub>.sal

and e<sub>1</sub>.deptno <> e<sub>2</sub>.deptno

and e<sub>1</sub>.Empno <> e<sub>2</sub>.Empno

Exe- deptno(10)

→ 20000

50000

deptno(20)

→ 20000

60000

④ Outer joins-

→ This join is used to retrieve all rows from one table and matching rows from another tables

Exe- dept table

- deptno -

10 - - - - - - - 10

20 - - - - - - - 20

30 - - - - - - - 30

40 - - - - - - - Null

↓

all rows from  
dept table

Emptable

- Empno -

10

20

30

Null

↓

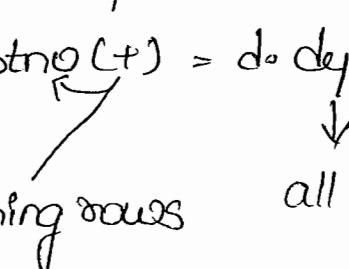
matching rows from  
Emp table

→ Generally by using Equi join we are retrieving matching rows only. If we want to retrieve non matching rows then we are using "join operator" in Equi join, this is called as outer join

→ This join operator can be used only one side at a time in joining condition

Notes- Which side we want to display all rows that side we can't use "+" operator, but we can use otherside

→ Here dept table contains all rows i.e., 10, 20, 30, 40 deptnos but in Emp table 10, 20, 30 so we can use "+" operator at the side of Emp table

Ex :- SQL > select ename, sal, d.deptno, dname, loc  
from emp e, dept d  
where e.deptno (+) = d.deptno  
  
 matching rows      all rows

→ for the above command 40<sup>th</sup> deptno is also displayed along with 10, 20, 30

In :- SQL > select ename, sal, d.deptno, dname, loc from emp e, dept d

where e.deptno = d.deptno  
(or)

SQL> select ename, sal, d.deptno, dname, loc  
from Emp e, dept d  
where e.deptno = d.deptno

→ for the above two statements output is same

\* q1 joins (or) ANSI Joins -

→ q1 joins are also called as ANSI joins, because these joins are applicable in all databases

i) inner join

ii) left outer join

iii) right outer join

iv) full outer join

v) natural join

i) inner joins -

→ It is also same as 8i-Equi join but Syntax is different and this q1-inner join performance is very high than 8i-Equi join

→ This join, also returns matching rows only when tables contains common column name then only we can use this join

$\rightarrow$  SQL > select ename, sal, d.deptno, dname, loc  
 from Emp e join dept d  
 on e.deptno = d.deptno;

$\rightarrow$  Notes - The difference between 8i joins and 9i joins in syntax is, in place of where we can use on and in place of comma(,) to separate number of tables we can use "join" in 9i

Q8 - Display the Employees who are Working in the location of Chicago from Emp, dept tables using 9i inner join

SQL > select ename, loc  
 from Emp e join dept d  
 on e.deptno = d.deptno  
 where loc = 'Chicago';

here we can also use and

Ex:-

SQL > select \* from t1,

| A | B | C |
|---|---|---|
| X | Y | Z |

SQL > select \* from t2;

| A | B |
|---|---|
| X | Y |

SQL> select \* from t<sub>1</sub>,t<sub>2</sub>  
 where t<sub>1</sub>.a = t<sub>2</sub>.a and t<sub>1</sub>.b = t<sub>2</sub>.b; } 8i joins

SQL> select \* from t<sub>1</sub> join t<sub>2</sub>  
 on t<sub>1</sub>.a = t<sub>2</sub>.a and t<sub>1</sub>.b = t<sub>2</sub>.b; } 9i joins

### \* "using" clause:-

→ In 9i joins We can also use using clauses in place of on clauses ; these type of joins performance is very high compare to 9i joins. And also these joins returns common columns only once

### Ex:- i) In 8i inner join:-

SQL> select \* from t<sub>1</sub>,t<sub>2</sub>  
 where t<sub>1</sub>.a = t<sub>2</sub>.a and t<sub>1</sub>.b = t<sub>2</sub>.b;

O/P:-    A    B    C    A    B  
 ---    ---    ---    ---    ---  
 X      Y      Z      X      Y

### ii) In 9i inner join:-

SQL> select \* from t<sub>1</sub> join t<sub>2</sub>  
 on t<sub>1</sub>.a = t<sub>2</sub>.a and t<sub>1</sub>.b = t<sub>2</sub>.b;

O/P:-    A    B    C    A    B  
 ---    ---    ---    ---    ---  
 X      Y      Z      X      Y

66  
iii) Extension of join by "using" clause:-

SOL> select \* from t<sub>1</sub> join t<sub>2</sub>

Using(a,b);

O/P:-

| A | B | C |
|---|---|---|
| X | Y | Z |

\* Rules in 'using' clauses:-

→ When we are using 'using clauses' We can not use alias names on joining conditional columns

→ Ex:- SOL> select ename, sal, d.deptno, dname, loc  
from Emp e join dept d  
using(deptno);

O/P:- error (so we can write above query as)

SOL> Select ename, sal, deptno, dname, loc  
from Emp e join dept d  
using(deptno);

Notes- outer joins are used to display non-matching rows also

→ left outer join = left side all rows and at right side matching rows

right outer join = right side all rows and at left side matching rows

ii) left outer join :-

→ This join returns all rows from left side table and matching rows from right side table and also returns null values in place of non-matching rows in another table.

Ex :- SQL > Select \* from t<sub>1</sub>;

| A | B | C |
|---|---|---|
| X | Y | Z |
| P | Q | R |

SQL > Select \* from t<sub>2</sub>;

| A | B |
|---|---|
| X | Y |
| S | T |

SQL > Select \* from t<sub>1</sub> left outer join t<sub>2</sub> .

on t<sub>1</sub>.a = t<sub>2</sub>.a and t<sub>1</sub>.b = t<sub>2</sub>.b;

O/P :-

| A | B | C | A | B |
|---|---|---|---|---|
| X | Y | Z | X | Y |
| P | Q | R | - | - |

→ Null values

iii) Right outer joins -

SOL > Select \* from  $t_1$  right outer join  $t_2$   
on  $t_1.a = t_2.a$  and  $t_1.b = t_2.b$

| <u>O/P<sub>5</sub></u> - |   |   | A      B |   |
|--------------------------|---|---|----------|---|
| A                        | B | C | A        | B |
| -                        | - | - | -        | - |
| x                        | y | z | x        | y |
| -                        | - | - | s        | t |

null values

→ This join returns all rows from right side table and matching rows from left side table. And also returns null values in the place of non-matching rows.

iv) full outer joins -

→ This join returns all rows from all tables and also returns null values in place of non-matching rows.

Ex:- SOL > Select \* from  $t_1$  full outer join  $t_2$   
on  $t_1.a = t_2.a$  and  $t_1.b = t_2.b$

| <u>O/P<sub>6</sub></u> - |   |   | A      B |   |
|--------------------------|---|---|----------|---|
| A                        | B | C | A        | B |
| -                        | - | - | -        | - |
| x                        | y | z | x        | y |
| p                        | q | r | -        | - |
| -                        | - | - | s        | t |

## v) Natural joins-

- When tables contains common columns then only we can use this natural join
- This join also behaves like inner join and also here we are not allowed to use joining condition
- This join also returns common columns one time only
- Notes- When we are using this join we are not allowed to use alias name on joining condition - all column (common column)
- Natural join is working upto 2 tables only if more than 2 tables means it is not working
- Ex 1 :- SQL > Select ename, sal, deptno, dname, loc from emp  $\text{e}$  natural join dept de;

Ex 2 :- SQL > Select \* from t<sub>1</sub> natural join t<sub>2</sub> ;

\* Imp Q :- In 8<sup>i</sup>

SQL > Select ename, sal, dname, loc  
from emp , dept ;  
56 rows Selected

by default cross join is used here

Q :- In 9<sup>i</sup>

SQL > Select ename, sal, dname, loc  
from emp cross join dept ;

56 rows Selected

### \*Gross join:-

- By using this join we can retrieve data from multiple tables or tables without using joining condition
- This join internally uses cartesian product
- It is also same as oracle's default join
- Oracle's default join is gross join only
- Ex :- SQL> Select ename, sal, dname, loc  
from Emp Gross join dept;

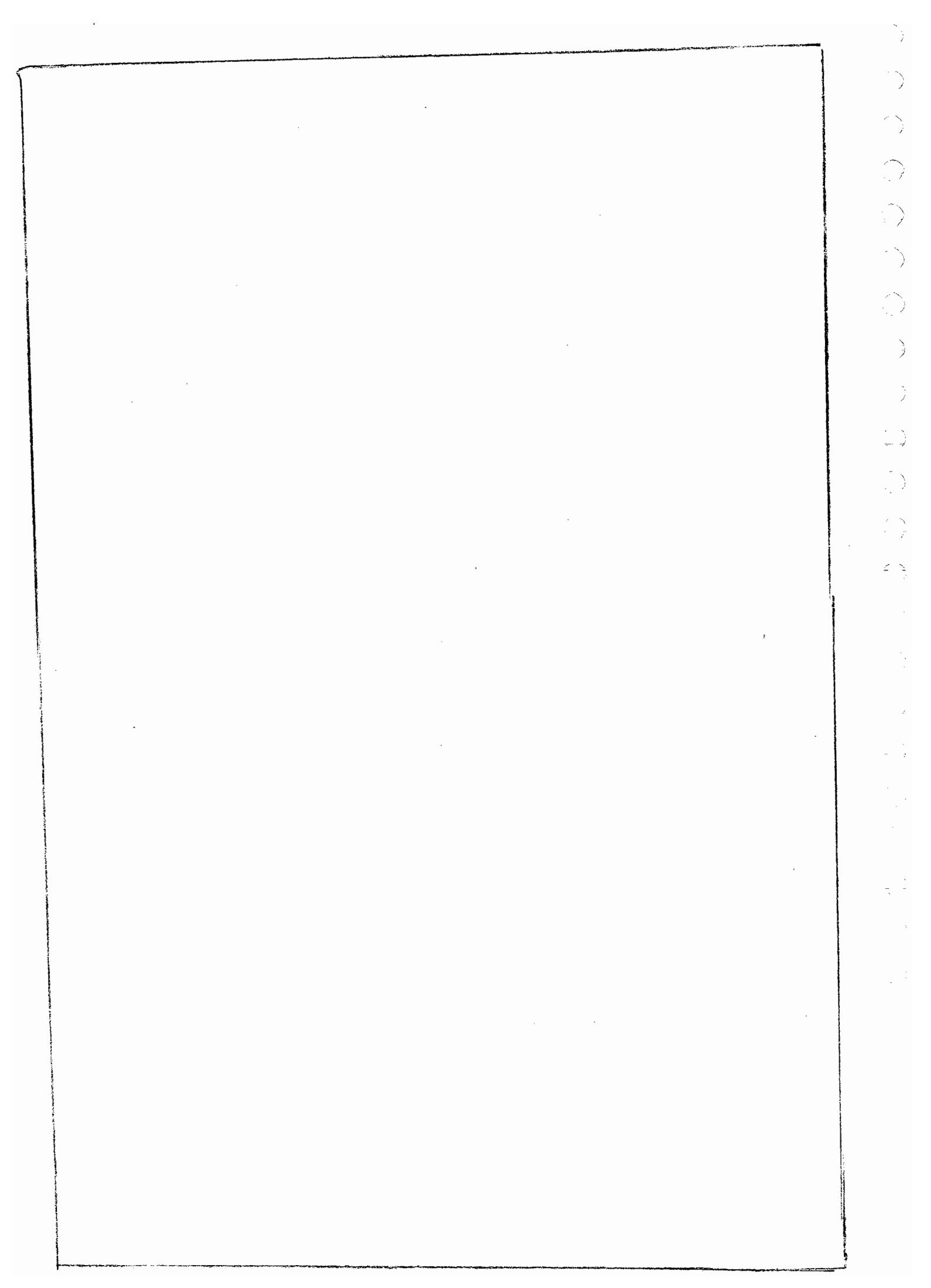
### \*joining more than two tables (3 tables)

| SI joins  | ANSI joins / Qi joins   |
|---|---|
| <u>Syntax :-</u>  | <u>Syntax :-</u>  |
| Select col1, col2, col3, ---<br>from table1, table2, table3<br>where table1.commoncol =<br>table2.commoncol<br><u>AND</u><br>table2.commoncol = table3.<br>Commoncol; | Select col1, col2, ---<br>from table1 join table2<br>on table1.commoncol =<br>table2.commoncol join<br>table3<br>on table2.commoncol =<br>table3.commoncol; |

Note :- mostly used joins in each database is

- i) inner join
- ii) outer join
- iii) self join

→ Because these joins are based on normalized concept



## \*ADVANCED SQL\*

### \*Relational set operators:-

- Set operations are used to retrieve data from single (or) multiple tables
- These operators are also called as "vertical joins"

1. UNION
2. UNION ALL
3. INTERSECT
4. MINUS

#### 1. UNION:-

- This operator is used to combine two or more tables so as to produce a single table
- That is, it combines the output of two queries to produce a single result
- Moreover, it generates unique values excluding the duplicate values
- Though, UNION operator in SQL has same functionality as JOIN operators in relational algebra
- The Syntax for UNION is as follows

Select col1, col2, ...  
from table1

UNION

Select col1, col2, ...  
from table2, ...

Exe-

SOL>Select job from Emp where deptno=10  
Union

Select job from Emp where deptno=20;

2. UNION ALL operators-

→ This operator Combines two or more tables,  
So as to generate a single table. It returns  
all the rows from the given two or more  
tables

→ In this, the duplicate rows are also selected  
because the UNION ALL operator retrieves the rows  
without separating the unique values

→ The Syntax for UNION ALL is as follows,

Select col1, col2 ---  
from table1

UNION ALL

Select col1, col2 ---  
from table2

Exe- SOL>Select job from Emp where deptno=10  
Union all

Select job from Emp where deptno=20;

3. INTERSECT operators-

→ This operator Combines two or more table  
to retrieve only the common rows i.e., the

rows that appear in both tables

→ The Syntax for INTERSECT is as follows

Select col1, col2 ---  
from table1

INTERSECT

Select col1, col2 ---  
from table2;

Ex:-

SOL>Select job from Emp where deptno=10  
INTERSECT

Select job from Emp where deptno=20;

#### 4. MINUS operator

→ This operator Combines two or more tables to eliminate all the duplicate rows (existing in the tables) and returns only those rows that exist in the first table

→ The Syntax for MINUS is as follows

Select col1, col2 ---  
from table1

MINUS

Select col1, col2 ---  
from table2;

Exs-

SQL>Select job from Emp Where deptno=10  
minus

Select job from Emp Where deptno=20<sub>g</sub>

→ Whenever we are using Set of data always  
Corresponding Expressions must belongs to Same  
datatype and also database Server returns  
first query columns as column heading

Exs- SQL>Select dname from dept  
Union

Select ename from Emp<sub>g</sub>

Notes- We can also retrieve data from  
multiple queries in Corresponding Expressions  
does not belongs to Same datatype also  
In this case we are using an appropriate  
type conversion function

## \*SQL JOIN OPERATORS

### \*Subqueries and Correlated queries:-

#### \* Sub-Queries :-

- A Query within another query is called nested Query (or) Sub-query.
- Generally Subqueries are used to retrieve data from single (or) multiple tables.  
"based on more than one step process" indirect process.
- Subqueries can be used with the Select insert, update, and delete statements along with the operations like =, <, >, !=, In, Between etc.

There are few rules that subqueries must follow :-

- Subqueries must be enclosed within parenthesis.
- A Subquery can have only one column in the SELECT Clause, unless multiple columns are in the main Query for the subquery to compare its selected columns.
- An ORDER by cannot be used in a subquery although the main query can use an ORDER

by the group by can be used to perform the same function as the ORDER BY in a subquery.

→ Subqueries that return more than one row can only be used with multiple value operators, such as the IN operator.

→ The SELECT list cannot include any references to values that evaluate to a BLOB, ARRAY, CLOB, or NCLOB.

→ A sub Query cannot be immediately enclosed in a set function.

→ The Between operator cannot be used with a subquery, however, the Between can be used with in a subquery.

#### 1. Single Row Subqueries :-

Display the Employees who are getting more than the average salary from Emp Table?

SQL > Select \* from Emp where Sal > Avg(Sal);

Error :- Group functions i.e avg(Sal) is not allowed in where clause.

SQL > Select \* from Emp where Sal > (Select avg(Sal) from Emp);

→ The above Query is non-correlated Query because child Query is executed first after

that only parent Query is executed.

→ The above Query is "single row subquery".  
because here child Query returns single value.

Ex: SQL > Select avg (sal) from Emp:

Avg (sal).

2073.21429.

→ In a single row subqueries we can use  
 $=, <, <=, >, >=$  operators.

Multiple row Subqueries:

SubQuery :-

q) Display the Employees whose salary more than the highest paid Employee in dept no 20 from Emp table?

SQL > Select \* from emp where sal > (Select max(sal) from emp where dept no = 20);

q) Display the Employees who are getting more than the lowest paid Employee in dept no 10 from Emp table?

SQL > Select \* from Emp where sal > (Select min(sal) from Emp where dept no = 10);

NOTE :- When a child Query contains max & min functions and also resource table (Ex: Emp) having large amount of data and also we are comparing values. Then those type of Queries degrades performance of the application.

→ To overcome this problem all database systems provides subquery special operator "any", "All". These operators are used along with the relational operators.

→ Performance of the above two Queries is very less. So the above Queries becomes, SQL > Select \* from Emp where SQL > all (Select sal from Emp dept no = 20);  
SQL > select \* from Emp where :sal > any (Select sal from Emp where dept no = 10);

\* Multiple Column Subqueries :

→ We can also compare multiple columns of the Child Query with the multiple columns of the Parent Query. These type of Subqueries are also called as multiple column Subqueries.

→ In this case we must specify parent Query where conditional columns within parenthesis.

Syntax :-

Select \* from table name where (col<sub>1</sub>, col<sub>2</sub>, ...)

In (Select col<sub>1</sub>, col<sub>2</sub>, ... from table name where condition);

q) Display the Employees whose job, mgr, match with the job, mgr of the Employee Scott.

SQL > Select \* from Emp where (job, mgr) in (Select job, mgr from Emp where ename = "SCOTT");

Ex:- SQL > Select dept no, Sal, ename, from Emp where (Sal, dept no) in (Select max(Sal), dept no from Emp group by dept no);

O/P :-

| dept no | Sal  | ename |
|---------|------|-------|
| 10      | 5000 | King  |
| 20      | 3100 | Scott |
| 30      | 2850 | Blake |

### \*Co-related Sub-Querys !\*

→ Generally in non-related Subquery Child Query is executed first then only parent query Executed.

→ Where as in co-related Sub-queries parent Query executed first then only child Query Executed.

→ In correlated sub-queries we must create an alias name for the parent Query table

& pass this alias name into child query where condition. Then only these two queries are co-related.

→ Ex: SQL > Select \* from Empe where sal = (Select \* from --- where e---);

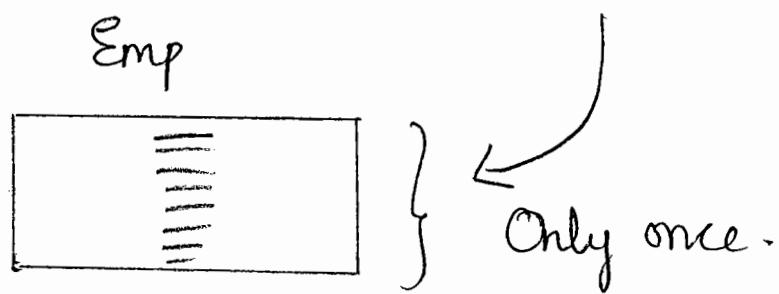
→ Whenever we are using these Queries database server uses 3 steps execution process i.e; database servers get a candidate row from the parent Query table then the control passed into child Query where condition. Based on the evaluation values it compares each value with parent table.

→ Generally, these Queries are used in the "denormalization process".

NOTE :- Generally in non-correlated subqueries child queries are executed only once for a parent table, whereas in co-related subqueries child Query is executed for each row in parent table.

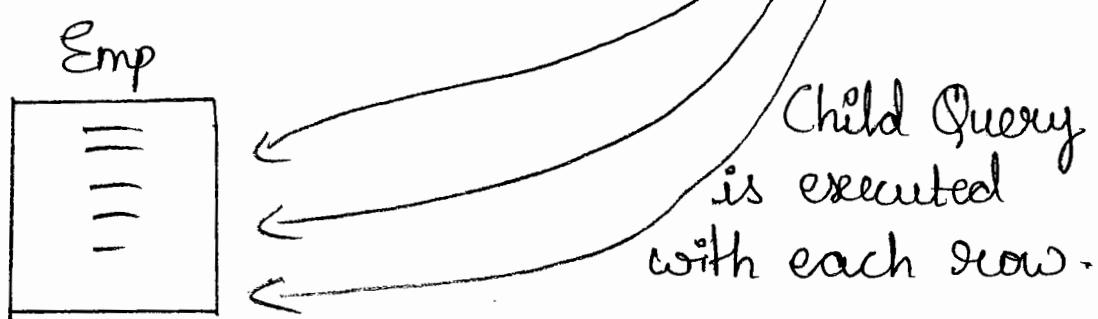
Explanation :- 1) non-co-related -

SQL > Select \* from Emp where sal = (Select sal.---);



2) Co-related:

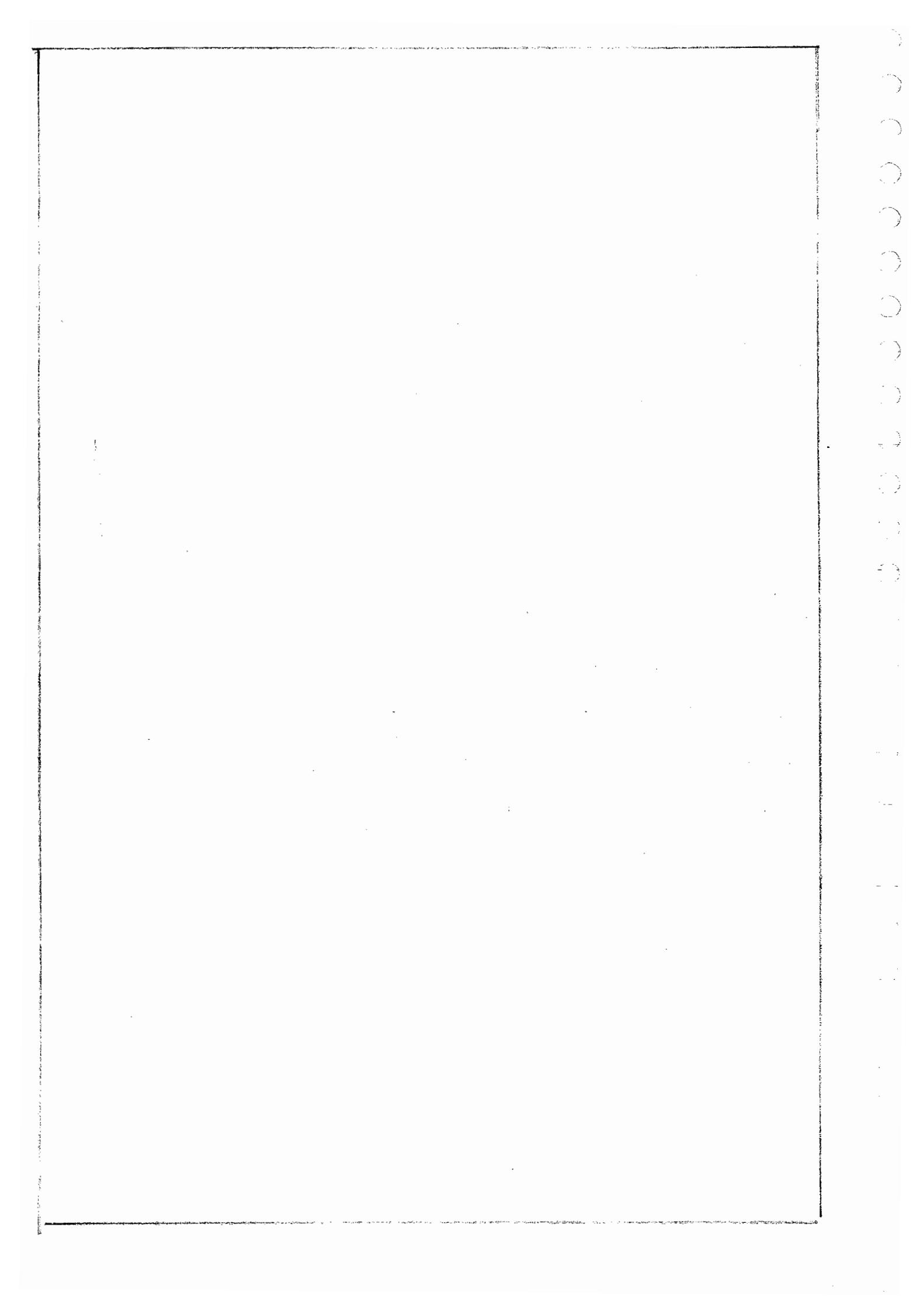
SQL > Select \* from Emp where Sal = (Select Sal ->)



9) Display the Employees whose salary more than avg sal of their jobs from Emp table using co-related subquery.

SQL > Select \* from Emp e where Sal > (Select avg (Sal) from Emp where job = e.job);

(3 Step Execution process).



## \*functions in SQL

→ functions are used to solve some particular task  
and also functions must return a value

→ oracle supports 2 types of functions

- I. Predefined functions
- II. Userdefined functions

### I. Predefined functions:-

1. Number functions

2. Character functions

3. Date functions

4. Group functions (or) aggregate functions

### 1. Number functions:-

These functions operate over number data

a) abs(): (absolute function)

→ This function always returns positive values

Ex: SQL> Select abs(-20) from dual;

Output:-

abs(-20)

20

### Notes-

dual: dual is an predefined virtual table which contains one row and one column.

→ By default dual table column datatype is varchar2

→ Generally this table is used to test functionality

means to test predefined (or) user defined function  
(functionality testing)

→ And also dual table is used to generate the sequence numbers from Sequence object

→ In dual table we can't perform DML operations because dual table is predefined virtual table

SOL> desc dual;

| Name  | NULL? | Type        |
|-------|-------|-------------|
| DUMMY | ---   | VARCHAR2(1) |

SOL> select \* from dual;

| D |
|---|
| X |

Ex:

SOL> select ename, comm, sal, abs(comm-sal) from Emp where comm is not null;

Explanation:- We are giving comm is not null in Where clause, because

→ In our Emp table Some Employees are not getting comm nothing but Some Employees are having comm is null. So the above query displays the Employees who are getting the comm

→  $\text{abs}(\text{comm}-\text{sal})$  means, in our table towner is an Emp he have comm '0' & sal 1500 So

$$\text{Comm-Sal} = 0 - 1500 = -1500 \quad X$$

$$\text{abs}(\text{comm}-\text{sal}) = \text{abs}(0 - 1500) = \text{abs}(-1500) = 1500 \quad \checkmark$$

ii) mod(m,n) :-

It returns remainder after m divided by n

Ex: SQL> Select mod(10,5) from dual;

mod(10,5)  
0

Ex: SQL> Select mod(2,5) from dual;

mod(2,5)  
2

c) Round(m,n) :-

It rounds given floatted value number m based on n

Ex: SQL> Select round(1.8) from dual;

O/P:- round(1.8)  
2  
(or)

SQL> Select round(1.5) from dual;

O/P:- round(1.5)  
2  
(or)

SQL> Select round(1.4) from dual;

O/P:- round(1.4)  
1

SQL> Select round(1.23456,3) from dual;

O/P:- round(1.23456,3)  
1.235

round:

(1.234 56,3)

123

1.234 but 56

→ 56 is more than 50% out of 100 50

1.235 is the answer

Ex:- SOL> Select round(1.234 36,3) from dual;

O/P:- round(1.234 36,3)

1.234

Notes:-

Round always checks remaining number if that remaining number is above 50%. Then 1 is added to the rounded number during that roundation

Ex:- SOL> Select ename, Sal/22 , round(Sal/22), round(Sal/22,1) from Emp;

d) tunc(m,n) :-

It truncates given floatted value number m based on n

Ex:- SOL> Select tunc(1.9) from dual;

O/P:- tunc(1.9)

1

\* Because after decimal point all values are truncated

Ex:- SOL> Select tunc(1.23456,3) from dual;

O/Ps- round(1.23456,3)

1.234

e) greatest(Exp1, Exp2, ...), least(Exp1, Exp2, ...) :-

→ greatest returns maximum value among given Expressions whereas least returns minimum value among given Expressions

Exs- SOL> select greatest(2,5,9) from dual;

O/Ps- GREATEST(2,5,9)  
9

Exs- SOL> select least(2,5,9) from dual;

O/Ps- LEAST(2,5,9)  
2

Imp Notes- greatest, least are used to compare two values only.

f) max(columnname), min(columnname) :-

→ these are used to compare Column values

Exs- SOL> select max(sal) from Emp;

MAX(SAL)  
5000

Exs- SOL> select ename, comm, sal, greatest(comm,sal)

from Emp where comm is not null;

→ In the above query greatest(comm,Sal) means it compares comm and Sal. And it displays greatest numbers

## a) ceil(), floor() :-

ceil return nearest greatest integer value whereas  
as floor return nearest lowest integer value.

Ex:- SQL> select ceil(1.3) from dual;

O/P:-  $\frac{\text{ceil}(1.3)}{2}$

Ex:- SQL> select floor(1.9) from dual;

O/P:-  $\frac{\text{floor}(1.9)}{1}$

## 2. character functions :-

→ These functions operate over character data

### a) upper() :-

→ It is used to convert a string (or) column values into uppercase

Ex:- SQL> select upper('welcome') from dual;

O/P:-  $\frac{\text{upper}('welcome')}$   
WELCOME

→ [This string is must be in sing quotes because it is a single word]

SQL> Select upper(ename) from Emp;

→ (This is the columnname in Emp table so for column name we cannot use single quotes)

b) lower() :-

→ It is used to convert a string (or) column values into lowercase

SOL>select lower(ename) from Emp;

Notes:

This upper and lower are not effected to Emp table

→ SOL> update Emp Set ename = lower(ename);

→ SOL> update Emp Set Ename = upper(ename);

c) initcap() :-

→ It return initial letter as capital letter and all remaining letters are small

SOL>select initcap(ename) from Emp;

SOL>Select initcap('ab cd ef') from dual;

QPS:- INIT CAPC

Ab cd ef

Notes:

If Spaces are available in between any 2 (or) more strings then that strings are treated as newstring

d) length() :-

→ It returns total length of the string "including spaces", and also it returns number datatype

Ex :- SOL> select length('AB CD') from dual;

O/P: length('AB\_CD')  
-----  
5

SOL>select length('ABCD') from dual;

O/P: length('ABCD')  
-----  
4

Imp  
e) Substr():

→ It will Extract position of the String with in given String based on last two parameters

Ex: - Select substr('ABCDEFGH', 2, 3) from dual;

Sub  
---  
BCD

→ Because 2 means in ABCDEF GH = B  
1 2 3 4 5 6 7 8

→ 3 means in B C D E F G H = B C D  
1 2 3

SOL>select substr('ABCDEFGH', 2, 3) from dual;

Ans: BCD

SOL> Select substr('ABCDEFGH', 2, 3) from dual;

Ans: GH

↓  
(-ve means search from right to left)

(ABCDEF|GH -2,3)

Ans: GH

SOL> Select Substr('A B C D E F G H', -6) from dual;

Ans :- -6

SOL> Select Substr('A B C D E F G H', -6, 3) from dual;

-6 => C D E f g h  
-6 -5 -4 -3 -2 -1

3 => C D E f g h

Ans :- C D E

Syntax - Substr(columnname (or) 'stringname',

Starting position, number of characters from  
that position)

(Should be numbers)

\* Display the Employees whose ename Second  
letter would be L A from Emp table using Substring  
function

SOL> Select Substr(ename, 2, 2) from Emp ex)

Notes -

Conditionally retrieve data means we should use

"where" Clause

→ for the above answer, all Employee names are  
displayed along with LA also

Exs like

SMITH = MI

MI  
←

ALLEN = LL

SOL> Select \* from Emp Where Substr(ename, 2, 2) = 'LA'

### Imp Notes

→ We are not allowed to use group functions (or) aggregate functions in Where clause but we can use number, character, date functions in Where clause.

### f) instr() :-

→ It returns position of the character (or) position of the delimiter (or) position of the String with in given String

→ Always this function returns number datatype

Ex :- SOL> Select instr('ABCDEF', 'Z') from dual;

Sols:- 0'

Reason :- Because Z is not available in our string  
i.e., 'ABCDEF'

Ex :- SOL> Select instr('ABCDEF', 'B') from dual;

Sols:- 2

Notes :- [delimiter is \*, -, %, +, '' Etc]

Ex :- SOL> select instr('ABCDEF', 'CD') from dual;

Sols:- 3  
↓  
∴ This total is consider as 1 letter)

Ex-Sol> Select insta('ABCDEFGHIJCDKHI', '(D') from dual;

Sol:- 3

Imp(Ex)-Sol> Select insta('ABCDEFGHICDIJCDKHI', '(D') , -5, 2) from dual;

Sol:- 3

('ABCDEF<sub>123</sub>GHI<sub>456789</sub>CDIJKCDKH' , '(D') , -5, 2)  
↓                  ↓  
1st occurrence  
2nd occurrence

Counting is starts from left to right

(Ex)-Sol> Select insta('ABCDEF<sub>123</sub>GHI<sub>456789</sub>CDIJKCDKH' , '(D', -4, 2) from dual;

Sol:- 9

('ABCDEF<sub>123</sub>GHI<sub>456789</sub>CDIJKCDKH' , '(D', -4, 2)  
↓                  ↓  
3rd occurrence    2nd occurrence    1st occurrence

(because cursor is here only)

Syntax - `instr(columnname (or) 'Stringname', 'str',  
Searching position, number of occurrences);`

numbers only

### Notes

Always in String, returns position of the String(or) delimiter (or) character based on last 2 parameters

→ But Oracle Server counts characters from left side first position onwards

Exs - SQL> Create table movie (title VARCHAR(10));  
table created

SQL> Insert into movie values ('&title');

SQL> Select \* from movie;

|              |      |
|--------------|------|
| <u>title</u> |      |
| ABCD         | EFGH |
| XXX          | XXX  |
| SSS          | 999  |
| ASDEFQ       | HIK  |

{ } → are movie names

SQL> Select Substr(title, 1, instr(title, ' ') - 1) from movie;

Substr(tit) :-

ABCD

XXXX

SSS

ASDEFG

i) ABCD EfG#

Substr(title, 1, instr(title, ' ') - 1)  $\rightarrow \textcircled{1}$

= instr(title, ' ') - 1

= 5 - 1

= 4

$\textcircled{1} \Rightarrow \text{substr(title, 1, 4)}$

ABCD EfG#  
1 2 3 4

Ans:- ABCD

ii) SSS GGGG

Substr(title, 1, instr(title, ' ') - 1)  $\rightarrow \textcircled{1}$

= instr(title, ' ') - 1

= 4 - 1

= 3

$\textcircled{1} \Rightarrow \text{substr(title, 1, 3)}$

SSS GGGG

Ans: SSS

### \* lpad : (left pad)

- It will fill remaining places with specified characters on left side
- Here always second parameter returns total length of the string

Ex: SQL> select lpad('ABCD', 10, 'B') from dual

Sol: - BBBBBBACD

### Notes:

We can take any delimiter, any character, any number etc. at the position of third parameter

### \* rpad : (right pad)

- It will fills remaining places with the specified characters on right side.

Ex: SQL> select rpad('ABCD', 10, '#') from dual;

Sol: ABCD#####

Ex: SQL> select rpad(ename, 10, '\_') || sal from emp;

Sol: RPAD(ename, 10, '\_') || SAL

SMITH ————— 800

ALLEN ————— 1600

;

;

;

MILLER ————— 1300

ltrim() :- (t<sub>o</sub>rim = remove) (l<sub>o</sub>trim = left trim)

→ It is used to remove Specified characters on left side

SOL> select ltrim('ssmissths', 's') from dual;

SQL:- missths

rtrim() :-

→ It is used to remove Specified characters on right side

SOL> select rtrim('ssmissths', 's') from dual;

SQL:- ssmissth

trim() :-

Oracle also 8.0 introduced trim function. It is used to remove left and right side of the Specified characters and also used to remove first and last Spaces

SOL> select trim('s' from 'ssmissths') from dual;

SQL:- missth

Ex:- SOL> select trim ('Smith') from dual;

SQL:- Smith

→ But end user does not recognized whether the Space is removed (or) not So the above SOL Command can be written as

SOL> select length(trim ('smith')) from dual;

SQL:- 5

\* translate(), replace() :-

→ translate() is used to replaces character by character whereas replace() is used to replaces character by string (or) String by String

Ex: SQL> Select translate('ABCDEF', 'fedcba', 123456)  
from dual;

Sol: - 654321

SQL> Select translate('JOHN', 'H', 'N') from dual;

Sol: JONN

SQL> Select translate('JOHN', 'H', 'NSBCD') from dual;

Sol: JONN

SQL> Select replace('JOHN', 'H', 'N') from dual;

Sol: JONN

SQL> Select replace('JOHN', 'H', 'NSBCD') from dual;

Sol: JONNSBCDN

SQL> Select job, replace(job, 'SALESMAN', 'MARKETING')  
from dual;

SQL> Select replace('A B C', ' ', 'ORACLE') from dual;

Sol: AORACLEBORACLECORACLE

SQL> Select replace('A B C', ' ', 'hi') from dual;

SQl :- ABC

SOL> Select replace ('ssmissthss', 's') from dual;

SQl :- mith

SOL> Select replace ('A B c', '') from dual;

SQl :- ABC

### \* Date functions

→ In oracle by default date format is DD-MON-YY

→ Oracle Server supports following date functions

- a) Sysdate
- b) add-months( )
- c) last-day( )
- d) next-day( )
- e) months\_between( )

#### a) Sysdate :-

→ It returns current date of the System in oracle date format

→ SOL> Select sysdate from dual;

SQl :- 07-JAN-13

#### b) add-months( ) :-

→ It is used to add (or) subtract number of months to the specified date based on Second parameter

SOL>select add\_months(sysdate, 1) from dual;

Sol: 07-FEB-13

SOL>select add\_months(sysdate, -1) from dual;

Sol: 07-DEC-12

c) last\_day():

→ It returns last day of the specified month

SOL>select last\_day(sysdate) from dual;

Sol: 31-JAN-13

(∴ last day in January is 31<sup>st</sup>)

d) next\_day():

→ It returns next occurrence day from the specified date based on second parameter

SOL>select next\_day(sysdate, 'sat') from dual;

Sol: 12-JAN-13

SOL>select next\_day(sysdate, 'monday') from dual;

Sol: 14-JAN-13

SOL>select next\_day(sysdate, 'saturday') from dual;

Sol: 12-JAN-13

e) months\_between():

Syntax - months\_between(date1, date2)

→ This function returns number of months between 2 dates

Imp

→ Always this function returns "number" datatype

SQL> Select months\_between(sysdate, hiredate) from Emp;

Note :-

To removing floating points which are occurring in output by using above SQL command we have

SQL> Select round(months\_between(sysdate, hiredate)) from Emp;

Note :-

If date1 is less than date2 then it returns negative sign. So using abs() function to get positive sign

X(Date + Arithmetic) :-

- a) Date + Number      ✓ possible
- b) Date - Number      ✓ possible
- c) Date1 + Date2      X impossible
- d) Date1 - Date2      ✓ possible

SQL> Select sysdate + 1 from dual;

Sol: 08-JAN-13 (Tomorrow's date)

SQL> Select Sysdate - 1 from dual;

Sol: - 06-JAN-13 (Yesterday's date)

SQL> Select Sysdate - Sysdate from dual;

Sol: - 0

Q) Display first date of the Current month using predefined date function

SQL> Select add\_months(last\_day(sysdate), -1) + 1  
from dual;

Sol: 01-JAN-13

Rough:-

→ last-day(Sysdate)

31-JAN-13

⇒ (31-JAN-13) + 1

01-FEB-13

⇒ add\_months(last-day(sysdate), -1) + 1

→ add-months(01-FEB-13) - 1

01-JAN-13

SQL> Select add\_months(last-day(sysdate), -1) + 5  
from dual;

Sol: 05-JAN-13

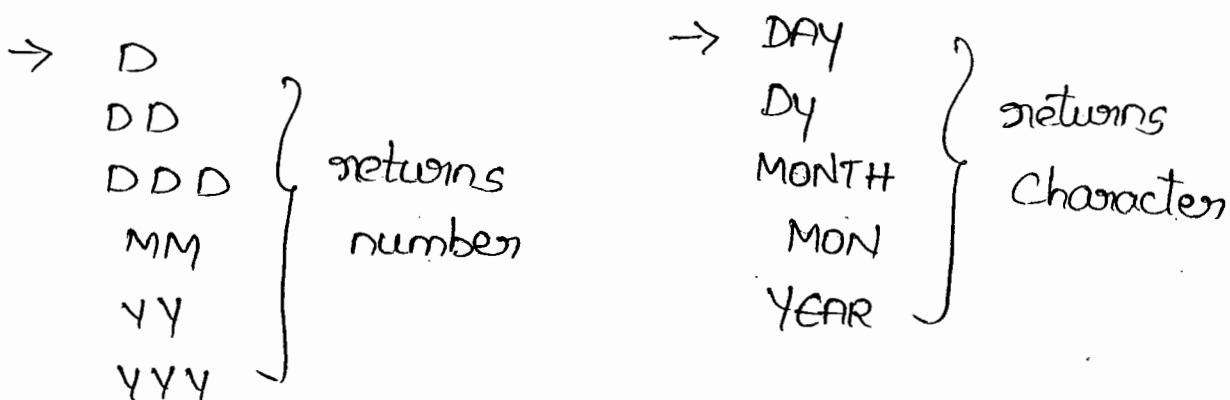
## VVV Imp:- Date Conversion functions:-

1. To\_char()
2. To\_date()
1. To\_char() :-

It is used to convert date type into character type  
i.e it is used to convert datatype into datestring

SOL>Select to\_char(sysdate,'DD/MM/YYYY') from dual;

Ans: 07/11/2013



SOL>Select to\_char(sysdate,'DAY') from dual;

Ans:- MONDAY

↳(today day)

↓  
(Capital so ans should  
be in capital letters)

SOL>Select to\_char(sysdate,'day') from dual;

Ans:- monday

↳(in small letters)

## Notes

to\_char() is case Sensitive function

SOL> Select to\_char(sysdate, 'DY') from dual;

↳ (Capital letters)

SOL :- MON

↳ (monday)

SOL> Select to\_char(sysdate, 'dy') from dual;

↳ (small letters)

SOL :- mon

SOL> Select to\_char(sysdate, 'DD') from dual;

SOL :- 07

SOL> Select to\_char(sysdate, 'D') from dual;

SOL :- 2 → (monday)

'D' ⇒ Sunday = 1

Monday = 2

Tuesday = 3

Wednesday = 4

Thursday = 5

Friday = 6

Saturday = 7

D = day of the week

DD = day of the month

DDD = Day of the year

SOL> Select to\_char(sysdate, 'DDD') from dual;

SOL :- 007

SQl>select to\_char(sysdate, 'DDth') from dual;

Sq1: 07th

SQl>select to\_char(sysdate, 'DDsp') from dual;

Sq1: - SEVEN

SQl>select to\_char(sysdate, 'DDspth') from dual;

Sq1: - SEVENTH

Notes Here Sp = Spellout

SQl>select to\_char(sysdate, 'month') from dual;

Sq1: january

SQl>select to\_char(sysdate, 'mon') from dual;

Sq1: JAN

SQl>select to\_char(sysdate, 'HH : MI : SS') from dual;

Sq1: - 09 : 06 : 41

(Q) :- 15-JUN-04 to 15/JUNE/04

|              |
|--------------|
| HH = Hours   |
| MI = Minutes |
| SS = Seconds |

SQl>Select to\_char('15-JUN-04', 'DD/MONTH/yy')  
from dual;

Output's enm09

Notes always to-char first parameter must be  
datatype like Sysdate etc.

## \* to\_date() :-

→ It is used to convert date String into date type  
(oracle date format)

SOL>select to\_date('17/june/03') from dual;

Sol: - 17-JUN-03

### Imp

SOL>select to\_date('17/06/03') from dual;

Sol: - error; not a valid month

### Imp

SOL>select to\_date('17/06/03', 'DD/MM/YY') from dual;

Sol: 17-JUN-03

### Notes

→ whenever we are using to\_date we must use format mask i.e to\_date passing parameters return values match with the default date format return values, otherwise use a second parameter as same as first parameter format.

SOL>select to\_char(to\_date('15-JUN-04'), 'DD/MM/YYYY')  
from dual;

Sol: 15/06/2004

Imp

⑨ Display the Employees who are joining in the month December from Emp table using to-char conversion function

SOL>select \* from Emp where to\_char(hiredate,'mm')  
= 12 ;

SOL>select \* from Emp where to\_char(hiredate,'mon')  
= 'DEC' ;

⑩ Display the Employees who are joining in the year 81 from Emp table using to-char conversion function

SOL>select \* from Emp where to\_char(hiredate,'yy')  
= 81 ;

SOL>select ename, hiredate from Emp;

SOL>select ename, hiredate, to\_char(hiredate,'yyy\y')  
from Emp;

SOL>select '09-DEC-03'+5 from dual;

Output :- Error, because we can't add String type i.e '09-DEC-03' to the number i.e 5

SOL>select to\_date('09-FEB-03')+5 from dual;

SOL> 14-FEB-03

## 4\* Group functions (or) aggregate functions

→ Group functions are built-in SQL functions that operate on groups of rows and return one value for the entire group.

→ These functions are :

- a) max()
- b) min()
- c) avg()
- d) sum()
- e) count(\*)
- f) count(columnname)

a) max() :-

→ This function is used to get the maximum value from a column.

Syntax: SQL> Select MAX(columnname) from Tablename;

Ex:- To get the maximum Salary drawn by an Employee, the query would be ;

SQL> Select max(Salary) from Employee;

b) Min() :-

→ This function is used to get the minimum value from a Column.

Syntax- SQL> Select min(columnname) from Tablename;

Ex- To get the minimum salary drawn by an Employee, the query would be;

SQL> Select min(sal) from Emp;

Avg( )e-

→ This function is used to get the average value of a numeric column

Syntax- SQL> Select avg(columnname) from Tablename;

Ex- To get the average salary, the query would be-

SQL> Select avg(sal) from Emp;

Note- All group functions returns single value

SQL> Select avg(comm) from Emp;

Output: 550

$$\frac{300 + 500 + 1400 + 0}{4} = 550$$

Note- By default all group functions ignores null values except count(\*) →

| Comm     |
|----------|
| 300      |
| 500      |
| 1400     |
| 11101111 |

Q8- If null values are also added means the query becomes

SOL>select avg(null(comm,0)) from Emp

Output- 157.142857

$$\frac{300+500+1400+0+-+--+-+--}{14} = 157.142857$$

d) Sum()-

→ This function is used to get the Sum of a numeric column

Syntax- SOL>select sum(columnname) from Tablename;

Ex- To get the total Salary given out to the Employees

SOL>select sum(sal) from Emps;

e) Count(\*)-

→ It returns number datatype. It returns number of rows in a table

SOL>select count(\*) from Emps;

Op- 14

SOL>select count(\*) from dual;

Op- 1

\* Count(\*) returns number of rows

f) Count(columnname) :-

→ This function counts number of not-null values in a column

SOL > Select count(comm) from Emp;

O/P :- 4

SOL > Select count(ename) from Emp;

O/P :- 14

→ But Count(\*) includes null values also

Ex :- Select count(\*) from Emp;

O/P :- 14

| <u>Comm</u> |
|-------------|
| - 300 n①    |
| - 500 n②    |
| - 1400 n③   |
| =           |
| =           |
| 0 n④        |
| =           |

\* SOL DISTINCT :-

→ This function is used to Select the distinct rows. For example:- If you want to Select all distinct department names from Employee table, the query would be:

SOL > Select Distinct dept from Employee;

To get the count of Employees with unique name, the query would be:

Select count(Distinct name) from Employee;

## \*SQL first() functions-

The `first()` function returns the first value of the selected column

Syntax:- SQL> Select `first(column-name)` from tablename;

Example:-

We have the following "orders" table:

| O_Id | OrderDate  | OrderPrice | Customer |
|------|------------|------------|----------|
| 1    | 2008/11/12 | 1000       | Hansen   |
| 2    | 2008/10/23 | 1600       | Nilsen   |
| 3    | 2008/09/02 | 700        | Hansen   |
| 4    | 2008/09/03 | 300        | Hansen   |
| 5    | 2008/08/30 | 2000       | Jensen   |
| 6    | 2008/10/04 | 100        | Nilsen   |

Now we want to find the first value of the "orderprice" column

We use the following SQL Statement:

Select `first(OrderPrice)` As `firstOrderPrice` from orders

Tip:- Lookaround if `first()` function is not supported

Select OrderPrice from orders order by O\_Id  
Limit 1

The result-set will look like this:

firstorder price

1000

### \* SOL LAST() functions -

The last() function returns the last value of the Selected column

Syntax-

Select last(orderprice) As lastorderprice from orders;

Output:- 100

### \* SOL Where clause

→ The Where clause is used when you want to retrieve specific information from a table excluding other irrelevant data

→ So SOL offers a feature called Where clause, which we can use to restrict the data that is retrieved

→ The condition is provide in the Where clause filters the rows retrieved from the table and gives you only those rows which you expected to see

→ Where clause can be used along with Select, Delete, update statements

### Syntax of SQL Where Clauses -

Where {column or expression} {Comparison-operator} value

Syntax for a where clause with select statement is:

Select column-list from table-name Where Conditions,

- column or Expression - Is the Column of a table or a Expression
- Comparison - operators - operators like = < > etc.
- value - Any user value or a Column name for Comparison.

### Note:-

→ The conditions are specified Using comparison or logical operators like >, <, =, like, NOT etc.

### Example:-

SQL>Select \* from Emp Where Sal > 5000;

→ Aliases defined for the Columns in the Select Statement cannot be used in the Where clause to Set conditions. only aliases created for tables can be used to reference the columns

in the table.

## \*How to use Expressions in the Where clause?

Expressions can also be used in the Where clause of the Select Statement

for Example - lets Consider the Employee table. If you want to display Employee name, Current Salary, and a 20% increase in the Salary for only those products where the percentage increase in salary is greater than 30000, the Select Statement can be written as shown below

Select name, Salary, Salary $\times 1.2$  AS new-Salary  
from Employee where Salary $\times 1.2 > 30000$

Outputs -

| name     | Salary | new-Salary |
|----------|--------|------------|
| Harithik | 35000  | 37000      |
| Harsha   | 35000  | 37000      |
| Disha    | 30000  | 36000      |

## \*SQL Order By Clause -

→ The SQL order by clause is used to Sort the data in ascending or descending order, based on one or more columns. Some database sorts query results in ascending order by default

## Syntax:-

The basic Syntax of order by clause is as follows:

Select column-list

from table-name

[where condition]

[order By Column1, Column2, ---ColumnN] [Asc]  
[Desc],

→ You can use more than one column in the order by clause. Make sure whatever column you are using to sort, that column should be in column list

## Example:-

→ Consider customers table is having following records:

| ID | Name    | AGE | ADDRESS   | SALARY   |
|----|---------|-----|-----------|----------|
| 1  | Ramesh  | 32  | Ahmedabad | 20000.00 |
| 2  | Khilan  | 25  | Delhi     | 1500.00  |
| 3  | Kaushik | 23  | Kota      | 2000.00  |

→ Following is an example which would sort the result in ascending order by Name and SALARY

SQL> Select \* from customers

order by Name, Salary.

## \* SQL Group By clause -

- This clause is used to derive similar data items into Set of logical groups.
- This clause is defined within the Select Statement and is used in conjunction with aggregate functions.
- However, if aggregate functions are not used then execution of Group By clause generates "not a Group By Expression" Error.
- In other words, it can be said that a Group By clause is valid only if used in combination with aggregate functions.

### Syntax:-

Select Col1, Col2, ---

from Table-name(s)

[Where Condition1, Condition2, ---]

[GroupBy Col1, Col2, ---]

[Order By Col1, Col2 [ASC/DESC]];

Notes- The Group By clause follows the Where clause in a Select Statement and precedes the Order by clause.

Ex:- Deptno in Emp table

Deptno

|    |   |    |   |                 |  |
|----|---|----|---|-----------------|--|
| 20 | } | 10 | } | This is Similar |  |
| 30 |   | 10 |   | 10              |  |
| 30 |   | 10 |   | 10              |  |
| 20 |   |    |   |                 |  |
| 30 | } | 20 | } |                 |  |
| 30 |   | 20 |   | 20              |  |
| 30 |   | 20 |   | 20              |  |
| 10 |   | 20 |   | 20              |  |
| 20 |   | 20 |   | 20              |  |
| 10 |   |    |   |                 |  |
| 30 | } | 30 | } |                 |  |
| 20 |   | 30 |   | 30              |  |
| 30 |   | 30 |   | 30              |  |
| 20 |   | 30 |   | 30              |  |
| 20 |   | 30 |   | 30              |  |
| 10 |   |    |   |                 |  |

Q8:- Display number of Employees department wise from Emp table using group by

SOL>Select deptno, count(\*) from Emp group by  
deptno

Output:- deptno      Count(x)

|    |   |
|----|---|
| 10 | 3 |
| 20 | 5 |
| 30 | 6 |

Q8- Display number of Employees job wise from Emp table

SOL>Select job, Count(\*) from Emp group by job;  
Output:-

| <u>Job</u> | <u>Count(*)</u> |
|------------|-----------------|
| ANALYST    | 2               |
| CLERK      | 4               |
| MANAGER    | 3               |
| PRESIDENT  | 1               |
| SALESMAN   | 4               |

Ex8

SOL>Select deptno, max(sal), min(sal) from Emp  
 group by deptno;

Output:-

| <u>Deptno</u> | <u>max(sal)</u> | <u>min(sal)</u> |
|---------------|-----------------|-----------------|
| 10            | 7450            | 2810.25         |
| 20            | 4150            | 1919.38         |
| 30            | 2400            | 1200            |

Note:-

We can also use group by clauses without using group functions. for example

SOL>Select deptno from Emp group by deptnos;

O/Ps

| deptno |
|--------|
| 10     |
| 20     |
| 30     |

Imp Notes

\* Rule ①- Other than group function columns, specified after Select those columns must use after group by otherwise oracle Server returns an error as "not a group by expression"

Ex :- SQL>Select deptno, ename , max(sal) from Emp  
group by deptno, ename;

Ex :- SQL>Select deptno, job from Emp group by  
deptno, job;

\* Rule ②- Whenever we are try to display group functions along with columns oracle Server returns an Error "not a single group - group - function". To overcome this problem we must use "group by"

Ex :- SQL>Select deptno, Sum(sal) from Emp group by  
deptno;

outputs

| <u>deptno</u> | <u>Sum(Sal)</u> |
|---------------|-----------------|
| 10            | 8750            |
| 20            | 10875           |
| 30            | 9400            |

Q8- Display those departments having more than 5 employees in them from Emp table using group by

SOL>select deptno , count(\*) from Emp group by deptno where count(\*)>5;

Errors: After groupby we can't use where clause

SOL>Select deptno, count(\*) from Emp group by deptno having count(\*)>5;

Q9- deptno    count(\*)

\*SQL Having clause-

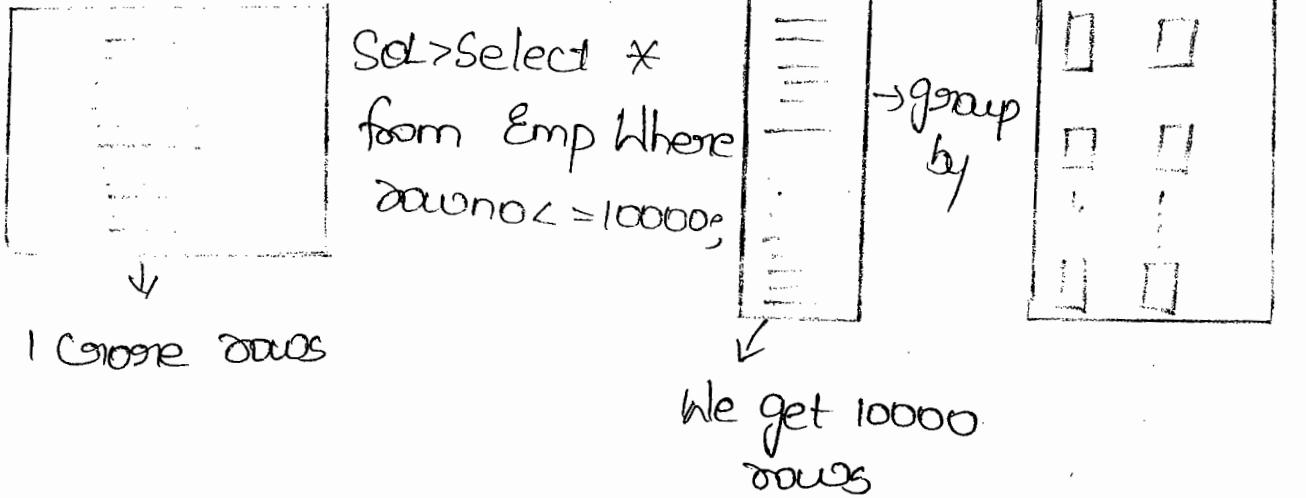
→ "Having" clause is an extension of Group by clause and is similar to "where" clause

→ After group by clause we are not allowed to use where clause, in place of where clause

ANSI/ISO SQL introduced having clause

Ex

## Emp table



- Generally if we want to restrict rows in a table then we are using where clause
- where as if we want to restrict groups after group by then we must use having clauses and also in where clauses we are not allowed to use group functions

Note:- We can also use where clause before group by but this where clause is used to retrieve data conditionally

SQL > select Deptno, Count(\*) from Emp Where  
Sal > 1000 group by Deptno having Count(\*) > 3;

| O/p:- | Deptno | Count(*) |
|-------|--------|----------|
|       | 20     | 5        |
|       | 30     | 6        |

SQL > select Deptno, Sum(Sal) from Emp group by

deptno having sum(sal)>10000

| <u>O/Ps</u> | <u>Deptno</u> | <u>Sum(sal)</u> |
|-------------|---------------|-----------------|
|             | 10            | 14701.25        |
|             | 20            | 13819.38        |

### \* Oracle Sequences \*

- Sequence is a database object which is used to generate sequence numbers automatically
- Generally sequences are used to generates primary key values automatically
- Sequences are created by database administrators in real time.
- Sequence is an independent database object Once the Sequence has been created number of users simultaneously access that Sequence
- Syntax:-

|                              |
|------------------------------|
| Create Sequence Sequencename |
| Start with n                 |
| increment by n               |
| min value n                  |
| max value n                  |
| Cycle/no cycle               |
| Cache/no cache;              |

→ Ex:- SQL> Create Sequence s,

Start with 5

Increment by 2

Maxvalue 100;

→ If we want to access Sequence numbers Oracle Server provides 2 pseudo columns

i) currval

ii) nextval

Notes here currval means current value

Syntax- Sequencename . currval

Syntax- Sequencename . nextval

→ If we want to generate Sequence numbers by using Select Statements then we must use "dual" table

→ Syntax- SQL> select Sequencename . currval from dual;

Syntax- SQL> select Sequencename . nextval from dual;

Ex- SQL> select si.currval from dual;

Error- Sequence si.currval is not yet defined in this Session

Notes- Always currval returns current Sequence number if Session (buffer) having a value,

9

That's why if we want to generate first Sequence number, then we must use nextval Pseudo column otherwise it gives error as "Sequence s1.nextval is not yet defined in this session"

→ So

SOL>Select s1.nextval from dual;

nextval

5

SOL>Select s1.nextval from dual;

nextval

+

Notes - We can also change all Sequence properties by using alter, but we can not change starting sequence number.

Syntax - alter Sequence Sequencename  
Property name newvalue;

Ex SOL>alter Sequence s1

increment by -3;

SOL>Select s1.nextval from dual;

SOL>alter Sequence s1

start with 2;

errors - Cannot alter Starting Sequence numbers

→ Start with cannot be less than minvalue

Exe- SQL> Create Sequence sq

Start with 7

Increment by 2

minvalue 10

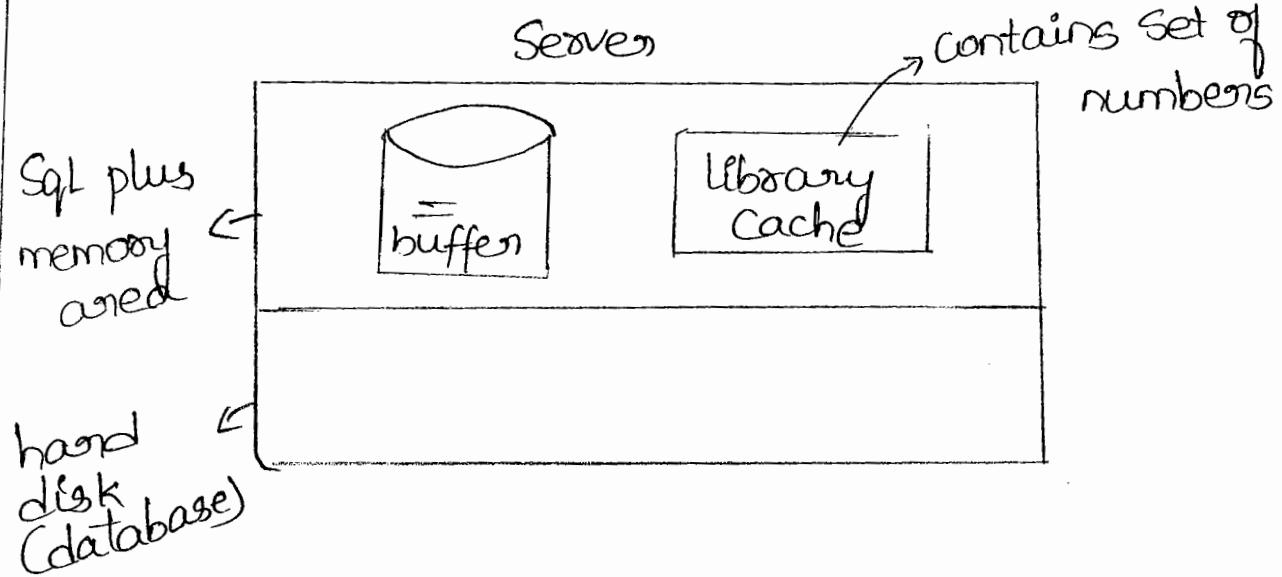
error- START WITH can not be less than MINVALUE

\* Caches-

→ Cache is an option used by DBA Database Administrator to improves performance of the application, that is if we want to generate sequence values very fastly. Then only database administrator used this cache option.

→ Generally, when we are generating a sequence values those values are automatically created from disk. If we want to improve performance database administrator assigns some set of sequence numbers into cache memory area.

→ Whenever users uses next sequences number database servers generate sequence numbers from cache memory area. But when system crashes automatically cache values are losted, because this cache is located at RAM memory area.



~~impl~~  
By default cache value is 20 And also cache minimum value is 2

Exe- SQL>Create Sequence S1

Start with 1

increment by 1;

SQL>select s1.nextval from dual;

Nextval

1

SQL>desc V\$Seq-Sequences;

SQL>Select Sequence-name, last-number from V\$Seq-Sequence where Sequence-name = 'S1'

Sequence-name

last-number

S1

21

Exe- SQL>Create Sequence S1,

Start with 1

increment by 1

Cache 4;

engine-The number of values to cache must be greater than 1

→ In oracle these sequence objects are used in "autoincrement" concept

→ In oracle if we want to generate primary key values then we are using autoincrement concept . That is autoincrement mechanism we are using sequences within triggers, i.e Sequence are created through sequence generation within SQL Engine

→ Ex- SQL>create table test(sno number(10) Primary key, name varchar2(10));

SQL>create sequence s5

Start with 1

increment by 1;

SQL>insert into test(sno, name) values(s5.nextval,  
'dname');

Enter value for name ; abc

Enter value for name ; xyz

SQL>select \* from test;

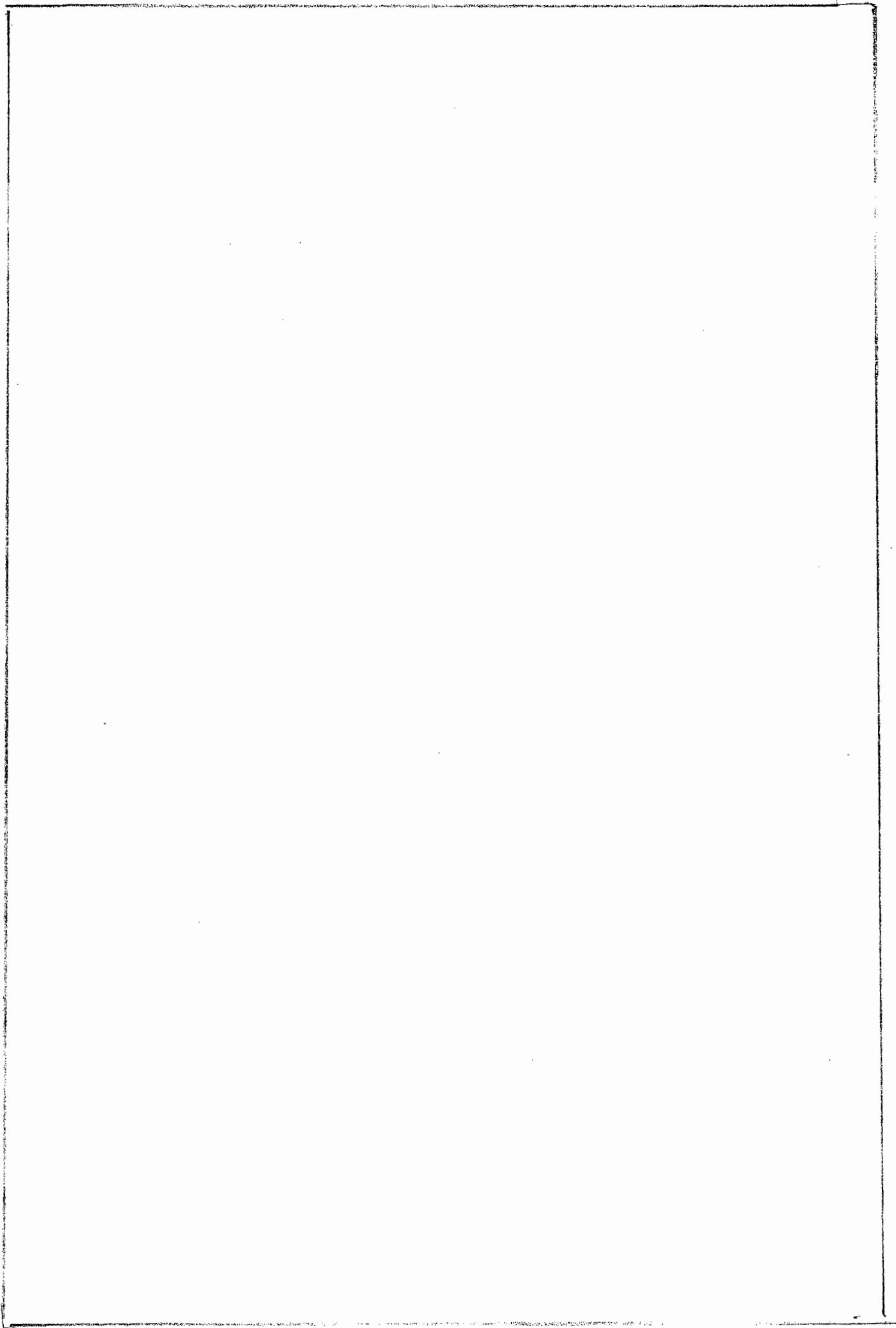
| Sno | Name |
|-----|------|
| 1   | abc  |
| 2   | xyz  |

qB

→ All Sequences information is stored in  
User-Sequences data dictionary

Exe- SQL>desc User-Sequences;

→ we can also drop Sequence by using  
"drop Sequence Sequencename"



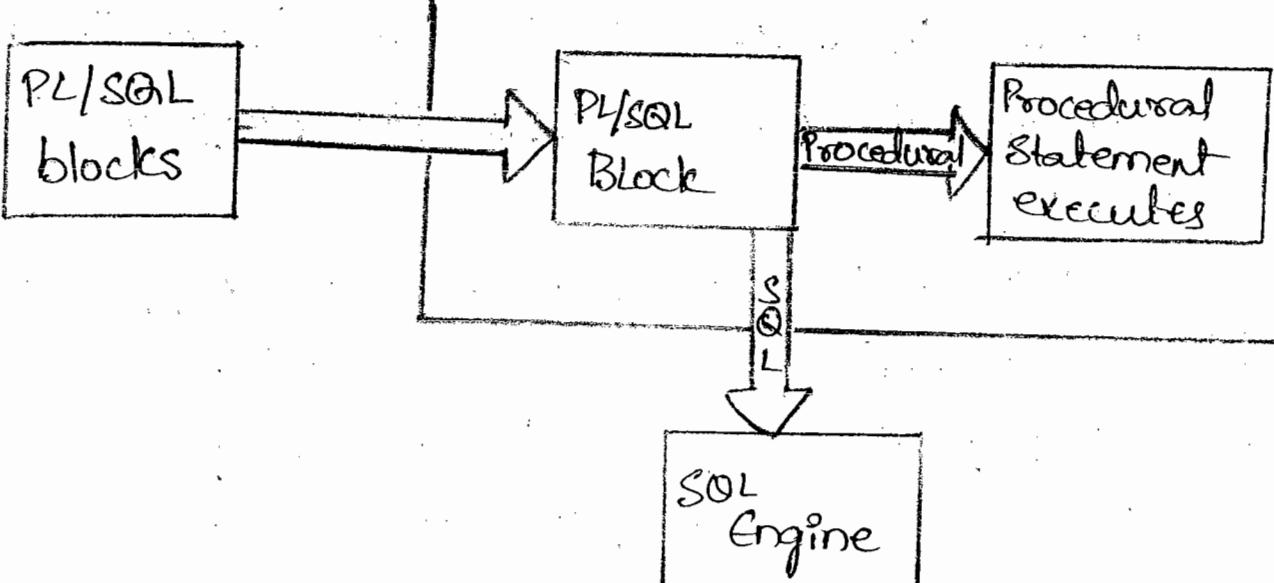
## \* PL / SQL

94

- Oracle 6.0 introduced PL/SQL. PL/SQL is a procedural language extension for SQL i.e. it is the combination of SQL and procedural language concepts.
- Basically PL/SQL is a block structured programming language.
- whenever we are submitting PL/SQL block into the Oracle server all SQL statements are executed separately using SQL engine & also procedural statements are executed separately using PL/SQL engine

### PL/SQL Architecture

Oracle Server



## \* Difference between SQL and PL/SQL.

| SQL   | PL/SQL  |
|---|---|
| 1) It is a structured query language or a data-oriented language        | 1) It is a combination of both procedural language and structured query language.     |
| 2) It provides only the things that are to be performed in the database | 2) It provides the way of performing the things in the database.                      |
| 3) It is used by many databases like MS Access, Oracle, Sybase etc.     | 3) It is used specifically by the Oracle database.                                    |
| 4) It consists of simple and easy statements                            | 4) It consists of a group of complex statements.                                      |
| 5) Only one statement can be executed at a time.                        | 5) Multiple statements can be executed at a time.                                     |
| 6) It is used to create and manipulate the data objects                 | 6) It is used to create the application programs.                                     |
| 7) It includes DDL and DML statements for building queries              | 7) It includes triggers, procedures, functions and package for building program code. |

8. It cannot be used as a programming language.

9) The SQL code can be embedded in a PL/SQL program

10) It doesn't provide any of the software engineering features

11) The execution speed of SQL statements is slow.

8. It can be used as a programming language like C++, Java, Ada, PHP, etc.

9) The PL/SQL code cannot be embedded in a SQL program.

10) It provides software engineering features like exception handling, encapsulation, overloading, information hiding etc

11) The execution speed of PL/SQL statements is fast.

### \* Procedural SQL:

→ The PL/SQL programming language was developed by Oracle Corporation in the late 1980s as procedural extension language for SQL and the Oracle relational database.

following are notable facts about PL/SQL:

→ PL/SQL is a Completely portable, high-performance

transaction-processing language.

- PL/SQL provides a built-in interpreted and OS independent programming environment.
- PL/SQL can also directly be called from the Command-line SQL\*plus interface.
- Direct call can also be made from external programming language calls to database.
- PL/SQL's general syntax is based on that of ADA and pascal programming language.
- Apart from Oracle, PL/SQL is available in TimesTen in-memory database and IBM DB2.

Features of PL/SQL:-

PL/SQL has the following features:

- PL/SQL is tightly integrated with SQL.
- It offers extensive error checking.
- It offers numerous data type.
- It offers a variety of programming structure.
- It supports structured programming through function and procedures.

- 10
- The support object-oriented programming.
  - It supports developing web applications and sever pages.

### Basic syntax

Structure of a PL/SQL block:

```
DECLARE  
<declarations section>  
BEGIN  
<executable command(s)>  
EXCEPTION  
<exception handling>  
END;
```

The 'Hello World' Example:

```
DECLARE  
message varchar2(20) := 'Hello, world!';  
BEGIN  
dbms_output.put_line(message);  
END;  
/
```

(OR)

DECLARE [optional]

→ Variable declarations, cursors, user defined  
Exceptions;

BEGIN [Mandatory]

→ DML, TCL

→ Select --- -into ---,

→ conditional Statement, control statements;

Exception [optional]

→ handling runtime errors;

END ; [mandatory]

→ There are 2 types of blocks supported by PL/SQL

① Anonymous blocks

② Named blocks

① Anonymous blocks:-

→ The PL/SQL block of code that does not have any specific name is called as Anonymous PL/SQL blocks.

→ A PL/SQL block of code is enclosed between BEGIN and END clauses. If a forward

slash is entered after the END clause that block will be executed immediately.

Example for anonymous blocks:

```
declare
  =
Begin
  =
end;
```

### ⑨ Named blocks :-

→ The PL/SQL block of code that does have any specific name is called named blocks

Example for Named blocks:

Ex:- procedures, Functions, Triggers, packages

### \* The PL/SOL Identifiers :

→ PL/SOL identifiers are constants, variables, exceptions, procedures, cursors, and reserved words

→ The identifiers consist of a letter optionally followed by more letters, numerals, dollar signs, underscores, and number signs and should not exceed 30 characters.

→ By default, identifiers are not case-sensitive. so you can use integer or INTEGER to represent a numeric value. You cannot use a reserved keyword as an identifier.

### \* The PL/SQL Delimiters :-

→ A delimiter is a symbol with a special meaning.

→ Following is the list of delimiters in PL/SQL.

| Delimiter | Description  |
|-----------|--|
| +,-,*,/   | Addition, subtraction/negation, multiplication division. |
| %         | Attribute indicator                                      |
| '         | Character string delimiter                               |
| .         | Component selector                                       |
| (,)       | Expression or list delimiter                             |
| :         | Host variable indicator                                  |
| ,         | Item separator   |
| "         | Quoted identifier delimiter                              |
| =         | Relational operator                                      |
| @         | Remote access indicator                                  |
| ;         | Statement terminator                                     |
| :=        | Assignment operator                                      |
| =>        | Association operator                                     |
|           | Concatenation operator                                   |

|                |  |
|----------------|--|
| **             | Exponentiation Operator                  |
| <<, >>         | label delimiter (begin and end)          |
| /*, */         | multi-line comment delimiter (begin/end) |
| --             | single-line comment indicator            |
| ..             | Range Operator                           |
| <,>,<=,>=      | Relational Operator                      |
| <>, !=, ne, ne | Different Versions of NOT EQUAL          |

### \* The PL/SQL COMMENTS :-

- Program comments are explanatory statements that you can include in the PL/SQL code that you write and helps anyone reading its source code
- All programming language allow for some form of comments
- The PL/SQL Support single and multi-line comments.
- All characters available inside any comment are ignored by PL/SQL Compiler
- The multi-line comments are enclosed by /\* and \*/.

## DATA TYPES IN PL/SQL :-

- PL/SQL provides supports with wide range of datatypes and subtypes
- A subtype is used in the PL/SQL program to make the data types compatible with data types in other programs while embedding PL/SQL code in another program, such as Java program.

|                      |  |
|----------------------|--|
| BINARY-INTEGER       | Signed integer in range -2,147,483,648 through 2,147,483,647, represented in 32 bits.  |
| BINARY-FLOAT         | Single-precision IEEE 754-format floating-point number.  |
| BINARY-DOUBLE        | Double-precision IEEE 754-format floating-point number.  |
| NUMBER(prec,scale)   | Fixed-point or floating point number with absolute value in range 1E-130 to (but not including) 1.0E126. A NUMBER variable can also represent 0. |
| DEC(prec,scale)      | ANSI specific fixed-point type with maximum precision of 38 decimal digits.  |
| DECIMAL(prec, scale) | IBM specific fixed-point type with maximum precision of 38 decimal digits  |

|                      |  |
|----------------------|--|
| NUMERIC (pre, scale) | Floating type with maximum precision of 38 decimal digits.   |
| DOUBLE PRECISION     | ANSI specific floating-point type with maximum precision of 126 binary digits (approximately 38 decimal digits)          |
| FLOAT                | ANSI and IBM specific floating-point type with maximum precision of 126 binary digits (approximately 38 decimal digits). |
| INT                  | ANSI specific integer type with maximum precision of 38 decimal digits.  |
| INTEGER              | ANSI and IBM specific integer type with maximum precision of 38 decimal digits   |
| SMALLINT             | ANSI and IBM specific integer type with maximum precision of 38 decimal digits   |
| REAL                 | Floating-point type with maximum precision of 63 binary digits (approximately 18 decimal digits)                         |

\*NULLS in PL/SQL:-

- PL/SQL NULL values represent missing or unknown data and they are not an integer, a character, or any other specific data types.
- Note that NULL is not the same as an empty data string or the null character value '\0'

→ A null can be assigned but it cannot be equated with anything, including itself.

### \* Variable :-

- A variable is nothing but a name given to a storage area that our programs can manipulate.
- Each variable in PL/SQL has a specific data type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory and the set of operations that can be applied to the variable.
- The name of a PL/SQL Variable consists of a letter optionally followed by more letters, numerals, dollar signs, underscores, and number signs and should not exceed 30 characters.
- By default, variable names are not case-sensitive.
- PL/SQL Reserved words are not used as Variable Name.

## \* Variable Declaration in PL/SQL :-

- PL/SQL Variable must be declared in the declaration section or in a package as a global variable.
- When a variable is declared, PL/SQL allocates memory of the variable's value and the storage location is identified by the variable name.

Syntax :-

|              |                 |
|--------------|-----------------|
| Variablename | datatype(size); |
|--------------|-----------------|

Ex:- declare

```
a    number(10);
b    varchar2(10);
```

- When variable\_name is a valid identifier in PL/SQL, datatype must be a valid PL/SQL datatype or any user defined datatype.
- When a size, scale or precision limit is provided with the data type, it is called a Constrained declaration. Constrained declarations require less memory than unconstrained declarations. for Example:

```
sales number(10, 2);
name varchar2(25);
address varchar2(100);
```

→ After declaring a variable, PL/SQL assigns it a default value of NULL.

\* Storing a value into Variable :-

→ By using assignment operator ( $\coloneqq$ ) we can store the value into the variable

Syntax:-

variablename  $\coloneqq$  Value;

Ex:- SQL> declare  
      a number(10);  
      begin  
         a:= 50;  
         end;  
      /

The 'Hello world' Example :

DECLARE  
      message varchar2(20):= 'Hello, world!';

BEGIN  
      dbms\_output.put\_line(message);

END;

/

→ THE END; line signals the end of the PL/SQL block.

→ To run the code from SQL Command line, you may need to type / at the beginning of the first blank line after the last line of the code.

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

106

Hello world

PL/SQL procedure successfully completed.

\* Scope of the variable :-

- PL/SQL allows the nesting of blocks, i.e., each program block may contain another inner block.
- If a variable is declared and ~~located~~ in an inner block, it is not accessible to the outer block.
- However, if a variable is declared and ~~accessible~~ to an outer block, it is also accessible to all nested inner blocks.

→ There are two types of variable scope:

→ Local variables :-

Variables declared in an inner block and not accessible to outer blocks.

→ Global variables :-

Variables declared in the outermost block or a package.

→ Decision-making structures require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statement to be executed if the condition is determined to be true, and optionally other statements to be executed if the condition is determined

to be false

→ following is the general form of a typical Conditional (i.e., decision making) structure found in most of the programming language.

\* Display a message (or) Variable values:-

Syntax-

dbms-output.put-line ('message');  
(or)

dbms-output.put-line (variablename);

Packagename method

Notes- To know the version

SQL>select \* from version;

Ex:- SQL>set serveroutput on;

SQL>begin

dbms-output.put-line('welcome');

end;

/

O/Ps- Welcome

→ If we are typing dbms-output.put-line('msg'); first that msg is stored in buffer. To get that msg from buffer we should use SQL>set serverout put on; before our program

Ex :- declare

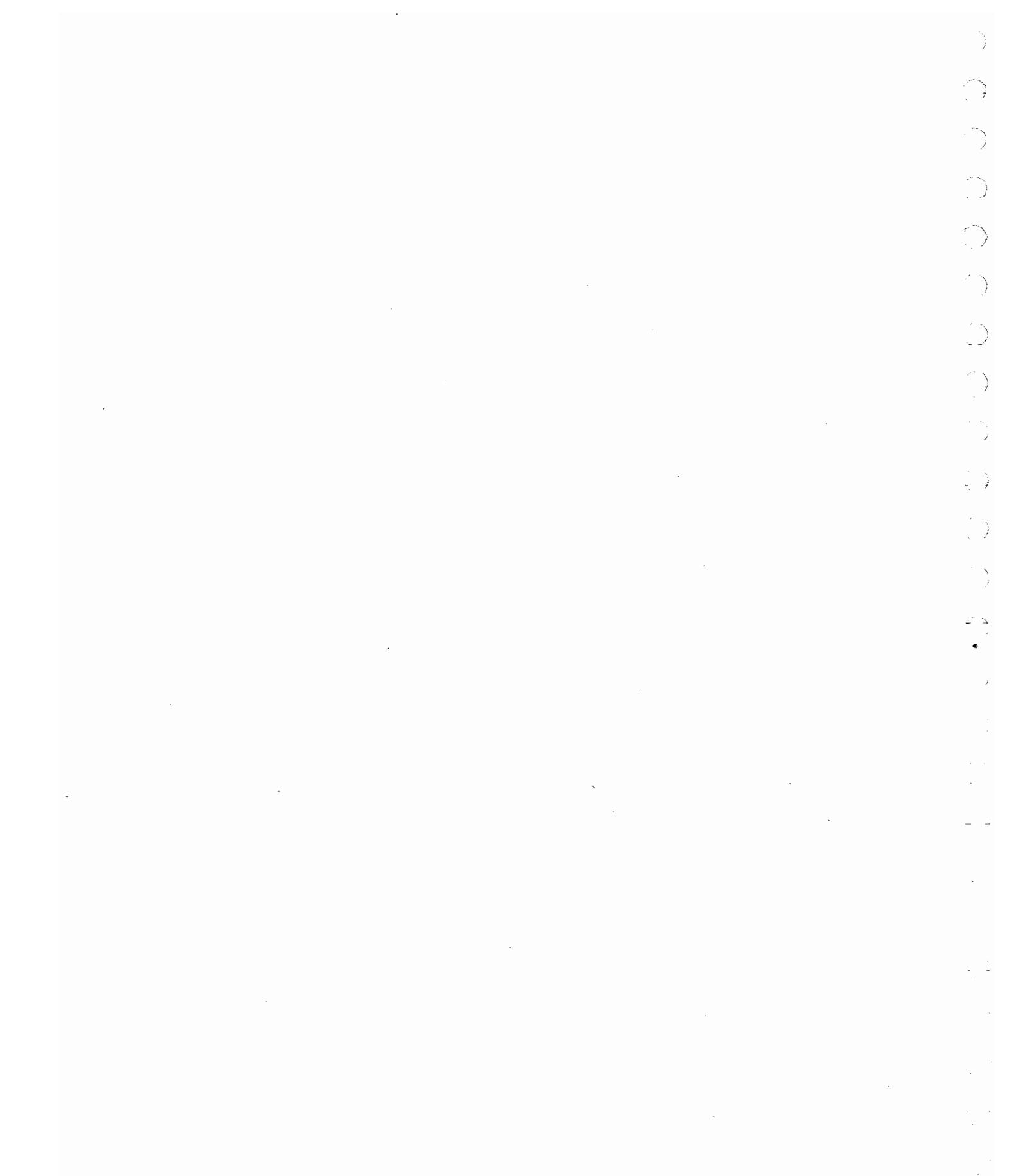
```
a number(10);  
begin  
a:=50;  
dbms-output.put-line(a);  
end;  
/  
Outputs - 50
```

Ex :- declare

```
a number(10):=a;  
begin  
dbms-output.put-line(a);  
end;  
/  
outputs:-
```

Ex :- declare

```
a number(10):=fa;  
begin  
dbms-output.put-line(a);  
end;  
/  
outputs:- Enter value for a
```



## \* Select --- into --- clause :-

Select --- into --- clause is used to retrieve data from table and store it into PL/SOL variable.

Basically select --- into --- clause always returns "Single record (or) Single value" at a time.

Ex:-

| Empno | Ename | Sal  |
|-------|-------|------|
| 7902  | FORD  | 3000 |
| 7566  | JONES | 4000 |
| 7639  | KING  | 5000 |
| !     | !     | !    |

→ Single record

Syntax:-

Select col1, col2 --- into Variable1, Variable2 --- from tablename Where condition;

→ This Statement used in Executable Section i.e., in [Begin Section] of the PL/SOL block

Write a pl/sol program for user Entered Empno, display name of the Employee & his Salary from Emp table ?

declare

v\_ename Varchar2(10);

v\_sal number(10);

```
begin
```

```
Select ename, Sal 'into V-ename, V-Sal
```

```
from Emp where Empno = &no;
```

```
dbms-output.put-line(V-ename || ' ' || V-Sal);
```

```
end;
```

```
/
```

Output:-

```
Enter Value for number:7902
```

```
FORD 3000
```

Note:-

Whenever we are using not null, constant keywords in declare section we must assign the value at the time of declaring the variable in declare section only

Ex:- declare

```
a number(10) not null := 30;
```

```
b constant number(10); := 9;
```

```
begin
```

```
dbms-output.put-line(a);
```

```
dbms-output.put-line(b);
```

```
end;
```

```
/
```

Note:- declare

a number(10) not null;

begin

a := 30; → Error

dbms-output.put-line(a);

end;

/

Error:

Note:- We can also use default clause in place of assignment operator when we are declaring the variables in declare section, but not in executable section

Ex:- declare

a number(10) default 10;

begin

dbms-output.put-line(a);

end;

/

Ex:- declare

a number(10);

begin

a default 30;

dbms-output.put-line(a);

end;

/

Error:

\* Write a PL/SQL program to retrieve max sal from Emp table & display that sal?

declare

v-Sal number(10);

begin

Select max(sal) into v-Sal from Emp;

dbms-output.put-line(v-Sal);

end;

/

6000

Note:-

In PL/SQL Expressions, we are not allowed to use group functions, decode() conversion function.

Ex: declare

a number(10);

b number(10);

c number(10);

begin

a:=90;

b:=20;

c:=greatest(a,b);

dbms-output.put-line(c);

end;

/

→ it is number function but  
not a group function

So there is no  
error.

Ex:

```

declare
  a number(10);
  b number(10);
  c number(10);
begin
  a:=90;
  b:=20;           → It is group function, So
  c:=max(a,b);   error is occurred
  dbms_output.put_line(c);
end;
/

```

### \* Variable attributes:-

Variable attributes are used in place of datatypes in variable declaration  
 → Whenever we are using these attributes Oracle Server automatically allocates memory for the variable corresponding column datatypes in a table  
 → There are 2 types of Variable attributes

Supported by PL/SQL

- i) Column level attributes
- ii) Row level attributes

## i) Column level attributes:-

In this method we are defining attributes for individual columns these attributes are represented by using %type

Syntax:- Variablename tablename.columnname % type;

Ex:- declare

V-ename Emp.ename% type; → Varchar2(20)

V-Sal Emp. sal% type; → number(10)

V-hiredate Emp. hiredate% type; → date

begin

Select ename, sal, hiredate into V-ename,  
V-Sal, V-hiredate, from Emp where Empno=8no;  
dbms-output.put-line(v-ename||' '||v-sal||'  
' || v-hiredate);

end;

/

## ii) Row level attributes:

Here a single variable can be represented all different datatypes in a table (or) in a entire record. This variable is also called record type variable. It is also same as structures in C-language. This attribute represented by %.Rowtype

Syntax: Variablename : tablename %> datatype;

Ex: declare

```
i Emp%> datatype;
```

begin

Select ename, sal, hiredate into i.ename, i.sal,  
i.hiredate from Emp where Empno = &Sno;

```
dbms_output.put_line(i.ename || ' ' || i.sal  
|| ' ' || i.hiredate);
```

end;

/

Output:

Enter Value for no: 7902

FORD 3600.03-DEC-81

\* declare

```
i Emp%> datatype;
```

:  
□

| Empno | ename | job   | Mgr | Hiredate | sal | comm | deptno |
|-------|-------|-------|-----|----------|-----|------|--------|
| 7902  | FORD  | CLERK | --- | ---      | --- | ---  | ---    |

→ In row level attribute we can also use \* in place of columns; at that time we should not use variables, in that place we should use Variablename which is declared in declare section.

Exe declare

? Emp%rowtype;

begin

Select \* into i from Emp where Empno=6no;  
dbms\_output.put\_line(i.ename||' '||i.sal||' '||  
i.hiredate||' '||i.deptno);

end;

/

Enter Value for no: 7902

FORD 3600 03-DEC-81 20

### \* Conditional Statements :-

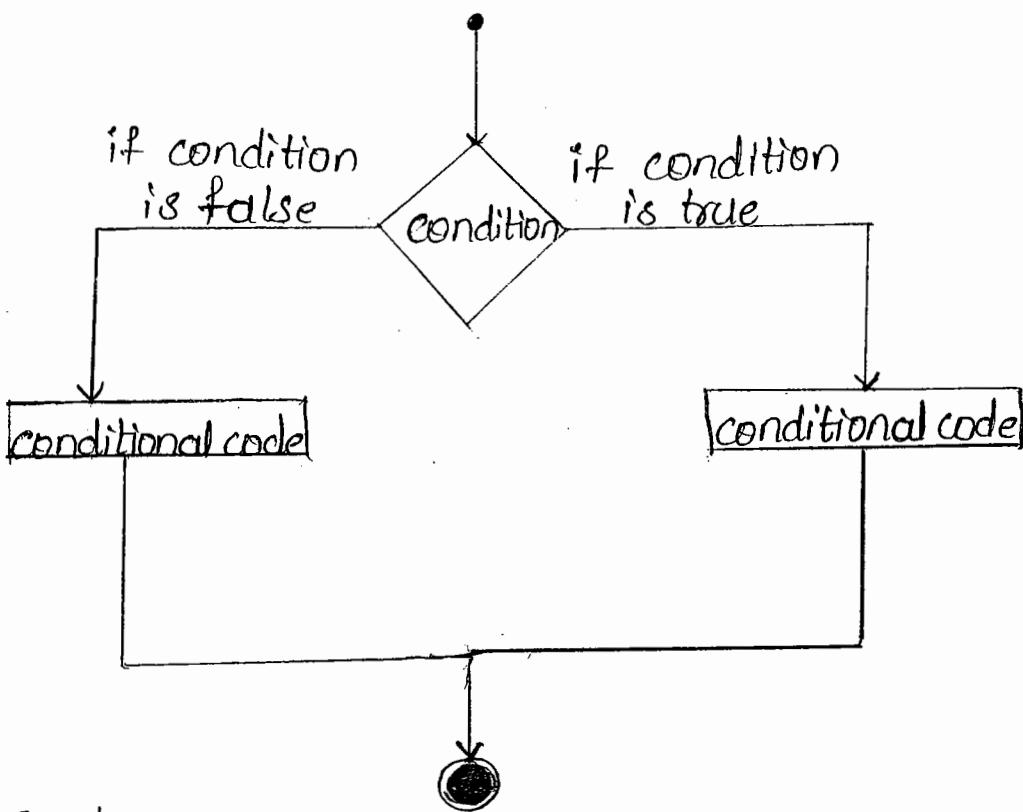
→ PL/SQL programming language provides following types of decision-making statements.

#### 1. If then (Simple if) :-

→ It is Simplest form of If control Statement frequently used in decision making and changing the control flow of the program execution

→ The If Statement associates a condition with a sequence of statements enclosed by the keywords THEN and END IF

→ If the condition is true, the statements get executed, and if the condition is FALSE or NULL, then



Syntax:-

```

if (condition) then
  stmts;
else
  stmts;
end if;
  
```

→ IF statement adds the keyword ELSE followed by an alternative sequences of statement. if the condition is false or null, then only the alternative sequence of statements get executed. it ensures that either of the sequences of statements is executed.

### 3. IF-THEN-ELSEIF:-

→ If we want to check more number of conditions then we should use "elseif"

Syntax:-

```
if condition1 then  
stmts;  
elseif condition2 then  
stmts;  
elseif condition3 then  
stmts;  
-----  
else  
stmts;  
end if;
```

Ex :-

```
declare  
v_deptno number(10);  
begin  
select deptno into v_deptno from dept where  
deptno=&n0;  
if v_deptno=10 then  
dbms_output.put_line ('ten');  
elseif v_deptno=20 then  
dbms_output.put_line ('twenty');
```

114

```
elseif v-deptno=30 then  
dbms_output.put_line('thirty');  
else  
dbms_output.put_line('others');  
end if;  
end;  
/  
/
```

Output:-

Enter value for no:10  
ten

Enter value for no:20  
twenty

Enter value for no:30  
thirty

Enter value for no:40  
Others

Enter value for no:50

Error:-

→ ORA-01403: no data found.

Rule①:-

→ If PL/SQL block contains select----- into ----- clause and also if requested data is not available in table then oracle server returns

an error as

ORA-01403: no data found.

Rule②:-

→ If PL/SQL block contains DML statements and also if requested (also) data is not available in a table then oracle server does not return any error in this case. To handle these type of blocks we are using "Implicit cursor attributes".

Ex:-

SQL> begin

    delete from emp where ename='ameerpet';  
end;

/

Output:-

→ PL/SQL procedure successfully completed.

Note:-

→ In emp table there is no name like ameerpet, but also it does not display error because we are not wrote select---into--- clause in begin section.

Rule③:-

→ If select---into--- clause try to return

more than one value then oracle server returns an error.

**ORA-1402:** exact fetch returns more than requested number of rows.

→ For Rule③ Ex:-

→ Is the example, in that example in place of dept we can use emp then rule③ is the explanation.

### \*Select case statement:

→ Like the IF statement, the CASE Statement selects one sequence of statements to execute.

→ However, to select the sequence, the CASE Statement uses a selector rather than multiple Boolean expressions.

→ A selector is an expression, whose value is used to select one of several alternatives.

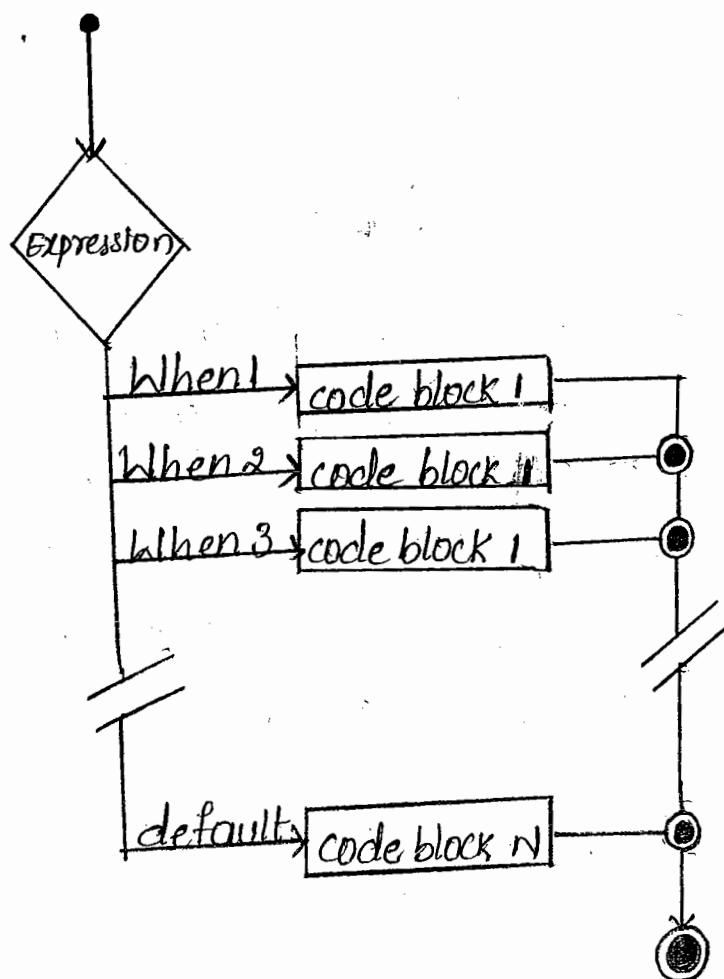
### Syntax:-

→ The syntax for case statement in PL/SQL is:

## CASE Selector

```
WHEN 'Value1' THEN S1;  
WHEN 'values' THEN S2;  
WHEN 'Value3' THEN S3;  
----  
ELSE Sn;---- default case  
END CASE;
```

## Flowchart:



## Example:-

```
DECLARE  
grade char(1)='A';
```

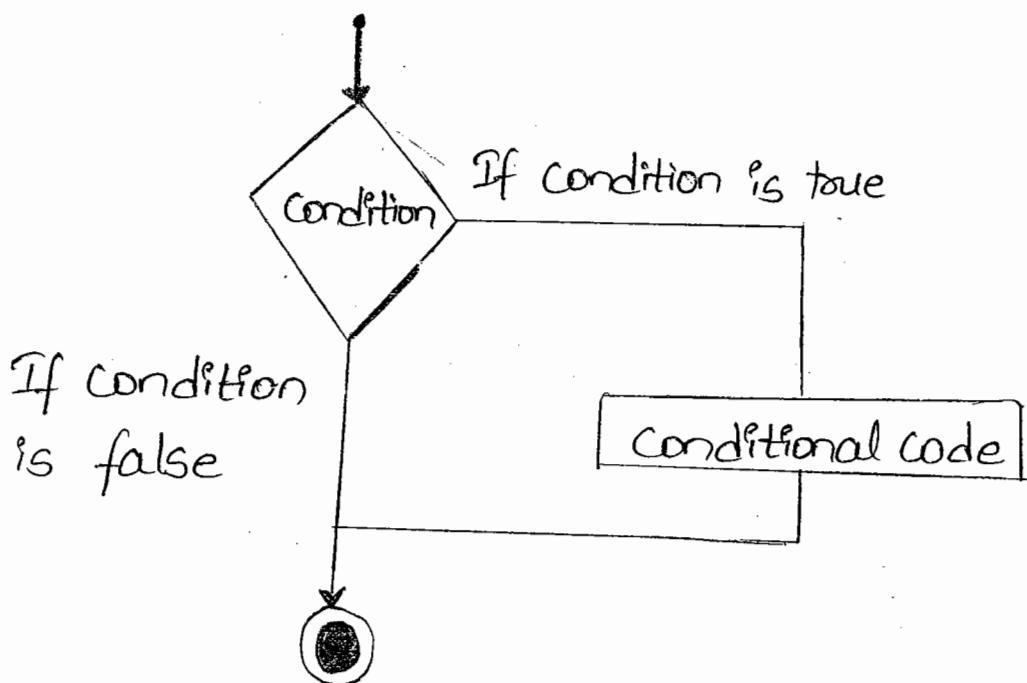
the If Statement does nothing.

Syntax:

```
If Condition THEN  
Statement;  
END If;
```

→ Where Condition is a Boolean or relational condition, and Statement is a Simple or Compound Statement

Flowchart:



Example:-

DECLARE

a number(2):=10;

BEGIN

a:=10;

-- check the boolean condition using if Statement

If( $a < 20$ ) THEN

--- If condition is true then print the following  
dbms-output.put-line('a is less than 20');

END If;

dbms-output.put-line('Value of a is:' || a);

END;

/

- If the Boolean Expression condition Evaluates to true then the block of code inside the if Statement will be Executed
- If Boolean Expression Evaluates to false then the if Statement will not be Executed

## 2. If then else:-

- The IF-THEN-ELSE statement allows the user to choose between several alternatives.
- The IF statement associates a condition with a sequences of statements enclosed by the keywords THEN and ENDIF.
- If the condition is TRUE, the statements get executed , and if the condition is FALSE or null then the else statement is executed.

```

BEGIN
    CASE grade
        When 'A' then dbms_output.put_line('Excellent');
        When 'B' then dbms_output.put_line('Very good');
        When 'C' then dbms_output.put_line('Well done');
        When 'D' then dbms_output.put_line('You passed');
        When 'F' then dbms_output.put_line('Better try
                                         again');
        else dbms_output.put_line('No such grade');
    END CASE;
END;

```

### \*Loops in PLSQL:-

- In order to execute a block of code several number of times.
- In general, to execute the statements sequentially, loops are implemented.
- The first statement in a function is executed first, followed by the second, and so on.
- programming languages provide various control structures that allow for more complicated execution paths.
- A loop statement allows to execute a statement and group of statements multiple times.

→ PL/SQL provides the following types of loop to handle the looping requirements.

① Simple loop

② While loop

③ For loop

1. Simple Loop:

→ In this loop structure, sequence of statements is enclosed between the LOOP and ENDLOOP statements.

→ At each iteration, the sequence of statements is executed and then control resumes at the top of the loop.

→ This loop is also called as infinite loop.

Syntax:

```
begin
loop
dbms_output.put_line('welcome');
end loop;
end;
```

NOTE:

→ It displays welcome msg infinite time.

→ To exit from infinite loop we are following

two methods.

Method 1:-

Syntax:-

Exit when trueCondition;

→ If condition is true then we can exit from infinite loop.

Ex:-

```

declare
  n number(10):=1;
begin
loop
dbms_output.put_line(n);
exit when n>=10;
n:=n+1;
end loop;
end;
/

```

Output:-

```

1
2
3
4
5
6
7
8
9
10

```

Ex:-

```
declare
n number(10):=1;
begin
loop
dbms_output.put_line('Welcome');
exit when n>=10;
n:=n+1;
end loop;
end;
/
```

Output:-

Welcome }  
| } 10 times  
Welcome }

Method ② Using if:-

Syntax:-

```
if condition then
exit;
end if;
```

} → This performance is very low, so in project it is used very less.

Ex:-

```
declare
n number(10):=1;
begin
loop
dbms_output.put_line(n);
if n>=10 then
```

```

exit;
end if;
n:=n+1;
end loop;
end;
/

```

## 2. While loop:-

→ Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.

Syntax:-

|                   |                                  |
|-------------------|----------------------------------|
| while (condition) | → This parenthesis are optional. |
| loop              |                                  |
| stmts;            |                                  |
| end loop;         |                                  |

Ex:-

```

declare
  n number(10):=1;
begin
  while (n<=10)
    loop
      dbms_output.put_line(n);
      n:=n+1;
    end loop;
  end;
/

```

### 3. For loop:-

→ Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.

#### Syntax:-

```
for indexvariablename in lowerbound..upperbound  
loop  
    Statements;  
end loop;
```

#### Ex:-

```
declare  
n number(10);  
begin  
for n in 1..10  
loop  
dbms_output.put_line(n);  
end loop;  
end;  
/
```

#### Output:-

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10
```

Ex:-

```

declare
n number(10);
begin
for n in 1..10
loop
dbms_output.put_line(n);

```

n:=n+1; → error: We are not allowed to use increment/decrement expressions in for loop.

end;

/

Ex:-

```

declare
n number(10);
begin
for n in reverse 1..10
loop
dbms_output.put_line(n);
end loop;
end;

```

/

Output:-

10  
9  
8  
7  
6  
5  
4  
3  
2  
1

Note:-

→ Whenever we are using for loop index variable internally oracle server defines an "integer datatype" to that index variable.

Ex:-

```
begin
for n in 1...10
loop
dbms_output.put_line(n);
end loop;
end;
/
```

→ In the above example we are not declare variable even though it is correct because we are using here "for" loop.

\* Bind Variable (or) non PLSQL variable:-

→ These are session variables created at host environment. That's why these variables.

(Here host is dos, SQL) are also called as variables.

→ We can also use these variables in PLSQL to execute a subprogram like procedures, functions etc) contains out, inout parameters.

Step①:-

→ Creating a bind variable:-

Syntax:-

SQL> variable variablename datatype;

Step②:-

→ Using bind variable:-

Syntax:-

SQL> variablename;

Step③:-

→ Display value from bind variable:-

Syntax:-

SQL> print variablename;

Ex:-

SQL> variable of number;

SQL> declare

    a number(10) := 500;

    begin

        :g := a/2;

    end;

    /

SQL> print g;

250.

C C C C C C C C C C

C C C C C C C C C C

C C C C C C C C C C

C C C C C C C C C C

C C C C C C C C C C

## \*. Cursor :-

- Cursor is a mechanism that enables the user to retrieve the output of a query from a relation by processing one row at a time.
- Cursors are defined in SQL procedures, so as to perform complex logic computation by following row-by-row approach.
- It is basically, a construct used in procedural SQL so as to store the data row returned after evaluating an SQL query.
- The cursors are stored in a private memory area, which is used to process multiple records and also this is a "record-by-record" process.

## Types of Cursors:

- There are two types of Static Cursors Supported by Oracle.

  - (i) Implicit Cursor
  - (ii) Explicit Cursor.

## (i). Implicit Cursor :-

- An implicit cursor is a cursor, which is automatically created in procedural SQL whenever the execution of SQL statement results in only one value.

- When a PL/SQL block contains Select...into... clause (or) DML statements then automatically Oracle Server creates a "memory area". This memory area is also called as SQL area (or) Content area (or) "implicit cursor".
- This memory area returns single record when we are using Select...into... clause in PL/SQL.
- This memory area also returns multiple records when we are using DML statements in PL/SQL blocks and also internally these records are processed at a time by using SQL engine, that's why explicitly developer does not control Oracle Server.
- The following steps are performed when an implicit cursor is opened;
  - i). The position of pointer is changed such that it points to first row in the cursor.
  - ii). SQL statement is fetched and executed by SQL Engine.
  - iii). After the complete execution, PL/SQL automatically closes the implicit cursor.

→ These activities are performed outside the user programmatic control i.e; user does not have any control over an implicit cursor.

→ PL/SQL creates implicit cursor for insert, update and delete statement that are executed in a program.

### Drawbacks of Implicit Cursor:

- i). It is less efficient when compared to explicit cursor.
- ii). It is highly susceptible to data error.
- iii). It provides less programmatic control to users.

### Example :-

declare

v\_ename varchar2(10);

v\_sal number(10);

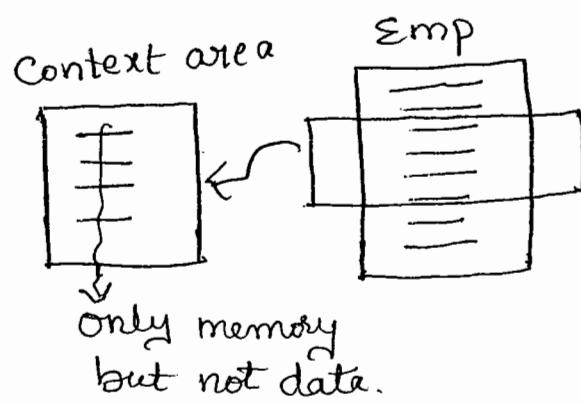
begin

select ename, sal into v\_ename, v\_sal from emp where empno = &nno;

dbms\_output.put\_line(v\_ename || ' ' || v\_sal);

end;

/



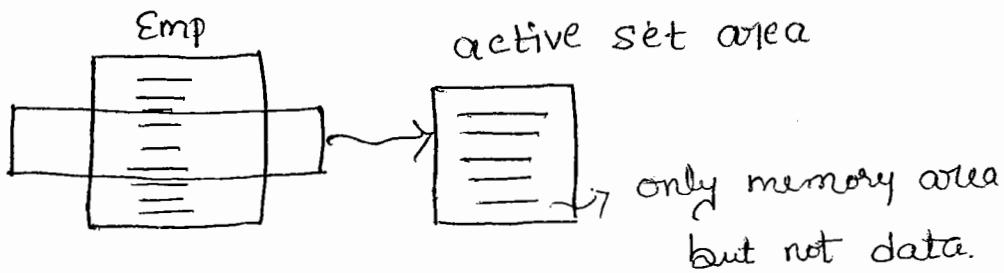
### Output :-

Enter value for no : 7566

JONES 2000

### ii) Explicit Cursor :-

- Explicit cursor refers to select statement that is explicitly declared within the declaration block of the program code.
- This cursor is created so as to store the output (i.e; two or more rows) generated after the execution of SQL statement.
- Moreover, if more than one value is to be returned by a select statement, then explicit cursor is used.
- In contrast to implicit cursor, explicit cursor allows the users to have complete control over the information present in the database.
- By default explicit cursor is a "record-by-record process".
- Explicit cursor memory area is also called as "active set area".



### \*. Explicit Cursor Life Cycle :-

→ The user needs to initially create an explicit cursor by using the following statements;

- i. Declare
- ii. Open
- iii. fetch
- iv. close.

#### i. Declare :

→ In declare section of the PL/SQL block we are defining the cursor memory area by using the following Syntax.

Syntax:

cursor cursorname is Select statement;

Example:

```
declare
  cursor c1 is select * from emp where
    job = 'CLERK';
```

### ②. Open:

→ Whenever we are opening the cursor then only Oracle Server fetches data from table into cursor memory area.

→ Because when we are using open statement then only cursor Select statements are executed.

Syntax: Open cursorname;

Example: Open C1;

→ This statement used in executable section of the PL/SQL block.

→ whenever we are opening the cursor, implicitly cursor pointer points to the first record in the cursor.

### ③. Fetch: (fetching from cursor)

→ Using fetch statement we are fetching data from cursor memory area into PL/SQL variables.

Syntax:-

fetch cursorname into variable1, variable2, ...;

④ Close:-

→ Whenever we are using close statement all the resources allocated from cursor memory area automatically released.

Syntax:-

~~~~~ close Cursorname;

Example:-

Sql> declare

```

Cursor c1 Select ename,sal from emp;
v_ename Varchar2(10);
v_sal number(10);
begin
open c1;
fetch c1 into v_ename,v_sal;
dbms_output.put_line(v_ename||' '||v_sal);
fetch c1 into v_ename,v_sal;
dbms_output.put_line(v_ename||' '||v_sal);
fetch c1 into v_ename,v_sal;
dbms_output.put_line(v_ename||' '||v_sal);
close c1;
end;
/
```

O/P:-

|       |      |
|-------|------|
| SMITH | 2850 |
| ALLEN | 1650 |
| MARD  | 1200 |

→ From the above example we observe that if we write fetch 3 times means the output is displayed for 3 records.

→ We cannot do like this so, we have to use loops but the loops output is infinite.

→ To exit from infinite loop in the cursor we have to write the following statement after the fetch statement.

"exit when C1%notfound"

#### \*. Explicit Cursor Attributes:

→ Cursor attributes are defined in order to retrieve the information regarding the state of cursor.

→ These attributes are used with % delimiter.

Syntax: name\_of\_cursor % attribute\_name

→ Every explicit cursor having following four attributes.

①. %notfound

③. %isopen

②. %found

④. %rowcount

→ Among these attributes all these attributes are used cursor name only.

Note:-

- Except "%rowcount" all other cursor attributes returns boolean value either true or false.
- "%rowcount" attribute returns number type.

①. %notfound :-

- This cursor attribute returns boolean value either true or false.
- This attribute returns any of the following values;

True: If no row is returned.

False: If a row is successfully returned.

Invalid\_Cursor: If the cursor has not been opened or has been closed.

Null: If no fetch is performed on the opened cursor.

Q:- Display all employee names and their salaries from emp table using Explicit cursor.

```
SQL> declare
      cursor c1 is select ename,sal from emp;
      v_ename varchar2(10);
      v_sal number(10);
```

```

begin
open c1;
loop
fetch c1 into v_ename, v_sal;
exit when c1%notfound;
dbms_output.put_line(v_ename || ' ' || v_sal);
end loop;
close c1;
end;
/

```

- Whenever we are creating a cursor Oracle Server automatically creates four memory locations along with cursor.
- These memory locations are identified through cursor attributes.
- These memory locations behaves like variables i.e; these variables store single value at a time.

| $c1\%notfound$ | $c1\%found$ | $c1\%.isopen$ | $c1\%.rowcount$ |
|----------------|-------------|---------------|-----------------|
| T              | F           | T             | 1               |

### i). FOUND :-

This attribute returns any of the following values,

TRUE : If a row is successfully returned.

FALSE : If no row is returned.

INVALID\_CURSOR : If the cursor is closed or has not been opened.

NULL : If no FETCH is performed on the opened cursor.

### ii) % ISOPEN :-

This attribute returns any of the following values,

TRUE : If the cursor is opened

FALSE : If the cursor is closed.

### iv) % ROWCOUNT :-

This attribute returns the number of rows fetched from the cursor. The attribute raises an INVALID\_CURSOR error if the respective cursor has not been opened or has been closed.

In addition, if a cursor is opened and no FETCH is performed then % ROWCOUNT attribute is equal to zero. The value of the attribute is incremented by one with every FETCH being performed.

→ This cursor attribute always returns number datatype i.e., this attribute stores number of records fetched from the cursor.

Example:-

→ Write a PL/SQL cursor program to display first five highest salaries from Emp table using % rowcount attribute.

SQL> declare

cursor c1 is select ename, sal from Emp  
order by sal desc;

v\_ename varchar2(10);

v\_sal number(10);

begin

open c1;

loop

fetch c1 into v\_ename, v\_sal;

dbms\_output.put\_line(v\_ename || v\_sal);

exit when c1.%rowcount >= 5;

end loop;

close c1;

end;

/

Another Method :-

SQL> declare

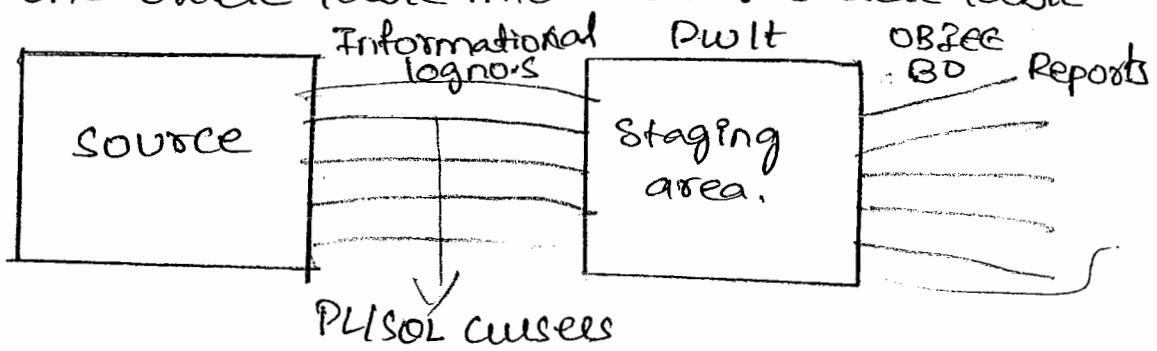
cursor c1 is select \* from Emp order by  
sal desc;

```

i Emp% Rowtype;
begin
  Open c1;
  loop
    fetch c1 into i;
    dbms-output.put-line(i.ename || i.sal);
    exit when c1%RowCount >= 5;
  end loop;
  close c1;
end;
/

```

NOTE :- By using cursors, we can also transfer data from one oracle table into another oracle table.



DBH = Datawarehouses

Ex :- SQL> create table target (name varchar2(10),
 sal number(10));

```

SQL> declare
cursor c1 is select * from emp where sal>2000;
i emp% Rowtype;
begin
  Open c1;
  loop

```

```
fetch c1 into i;
exit when c1%not found;
insert into target(name, sal) values (i.ename,
    i.sal);
end loop;
close c1;
end;
/
```

SQL> select \* from target;

Note:- In procedures, functions, triggers, packages if we are processing with multiple records means we should use cursors.

\* Eliminating explicit cursor life cycle :-

→ using cursor for loop we are eliminating explicit cursor life cycle i.e we are not allowed to use close, fetch, open stmts

\* cursor for loop :-

Syntax:-

```
for indexname in cursorname
loop
    statements;
end loop;
```

→ This loop used in executable section of the PL/SQL block. This method is also called as shortcut method of the cursor. When we are using this loop PL/SQL runtime engine internally uses open, fetch, close statements.

NOTE:- In cursor for loop index variable internally behaves like a record type variable i.e "%rowtype"

Ex:- SOL>declare

```

cursor C1 is select * from Emp;
begin
for i in C1
loop
dbms_output.put_line(i.ename||'/'||i.sal);
end loop;
end;
/

```

NOTE :- we can also eliminate declare section of the cursor using cursor for loop, in this code we are using select statement in the place of cursor name in cursor 'for' loop

Syntax:-

for indexvariable in (select statement)

loop;

stmts;

end loop;

Ex:- SOL>

begin

for i in (select \* from Emp where rownum<=10)

loop

dbms\_output.put\_line(i.ename||'/'||i.sal);

end loop;

end;

/

## \* parameterized cursors:-

- we can also pass parameters to the cursors same like a sub program, "in parameters". i.e here we must define formal parameters in cursor definition.
- ↳ execute this cursor we are using actual parameters in opening the cursors
- formal parameters specifies names of the parameters & type of the parameter

Note:- whenever we are passing parameters to the cursors, procedures, functions we are not allowed to use "datatype size" in the formal parameter definition in Oracle database.

Syntax: ①

```
cursor Cursorname(formal parameter)
                  ↑
cursor Cursorname(actual parameters) is
select * from tablename where columnname =
parameter name;
```

Syntax: ②

```
Open Cursorname(actual parameters);
```

- The datatype of formal & actual parameters should be same.
- Number of parameters used is also should be
  - ↳ (no. of actual parameters & no. of formal parameters should be equal).

Ex:- declare

```
cursor (,P-deptno number) is select * from
Emp where deptno=P-deptno;
```

12

```
i Emp% Rowtypes;
begin
  open c1(10);
  loop
    fetch c1 into i;
    exit when c1% not found;
    dbms_output.put_line(i.enamell || i.deptno);
  end loop;
  close c1;
end;
/
```

Q:- write a PL/SQL program by using "parameterized cursor" to display following static report for job as a parameter from Emp table?

Employees working as a manager

JAMES  
BLACK  
CLARK

Employees working as a analyst

SCOTT  
FORD

SQL>declare

cursor c (P-job varchar2) is select \* from Emp where job = P-job;

```
i Emp% Rowtypes;
begin
  open c, (&job);
  loop.
```

```
fetch c1 into i;
exit when c1%not found;
dbms-output.put-line (i.ename||' || i.job);
end loop;
close cl;
end;
/
```

SQL> declare

```
cursor c1 (P-job varchar2) is. Select *
from Emp where job = P-job;
i emp% low type;
begin
open c1 ('Manager');
dbms-output.put-line ('Employees working
as managers');
loop
fetch c1 into i;
exit when c1%not found;
dbms-output.put-line (i.ename);
end loop;
close c1;
open e1 ('Analyst');
dbms-output.put-line ('Employees working
as analysts');
loop
fetch c1 into i;
exit when (1%not found);
dbms-output.put-line (i.ename);
end loop;
```

```

close c1;
end;
/

```

- If we want to display static report i.e line by line means we should use open multiple times.
- If we are reopen the cursor we should close the cursor before reopen.

Note:- Before reopening the cursor we must close the cursor, otherwise Oracle server returns on error

Ora-6511: cursor already open

Note:- when we are not opening the cursor then Oracle server returns on error

Ora-1001: invalid cursor

NOTE:- we can convert normal cursor, parameterized cursors into cursor for loop to reduce the code

Ex:- SQL> declare

```

cursor c1(p_deptno number) is select *
from Emp where deptno = p_deptno;

```

begin

for i in c1(10)

loop

dbms-output.put-line(i.ename || '|| i.deptno);

end loop;

end;

/

\* WRITE A PLSQL program to retrieve all deptno from dept table by using static cursor and also pass these values into another parameterized cursor to retrieve employee details based on parameter value.

```
SOL > declare
    cursor c1 is select deptno from dept;
    cursor c2(p-deptno number) is select *
        from emp where deptno=p.deptno;
begin
for i in c1
loop
dbms_output.put_line('my deptno is '|| i.deptno);
for j in c2(i.deptno)
loop
dbms_output.put_line(j.ename || '|| j.sal
|| '|| j.deptno);
end loop;
end loop;
end;
/
```

NOTE :- we can also pass default values into parameterized cursor by using default(?) operator

Syntax :-

parametername datatype default[?:=]actualvalue;

Ex:-

```
SOL > declare
```

```
cursor c1(p-deptno number default 30) is
```

```

Select * from Emp where deptno = p_deptno;
i Emp% Rowtype;
begin
open c1(10);
loop
fetch c1 into i;
exit when c1% not found;
dbms_output.put_line (i.ename || ' ' || i.deptno);
end loop;
close c1;
end;
/

```

NOTE:- In procedures, cursors, functions (parameterized) if we are using default (so) := at that time we may pass actual parameters (so) may not in [open c1(10);] → If we are using actual parameters in [open c1(10);] then this [open c1(10)] is executed but not [default(so)]

write a PL/SQL cursor program to modify salaries of the employees from Emp table by using following conditions.

- (i) if job = 'CLERK' then increment sal → 100
- (ii) if job = 'SALESMAN' then decrement sal → 200
- (iii) if job = 'ANALYST' then increment sal → 100

SQL > declare  
cursor c1 is Select \* from Emp;  
i Emp% Rowtype;  
begin  
open c1;  
loop  
fetch c1 into i;

```

exit when c1%not found;
if i.job='CLERK' then
update Emp set sal=i.sal+100 where Empno=i.empno;
else if i.job='SALESMAN' then
update Emp set sal=i.sal-200 where Empno=i.empno;
else if i.job='ANALYST' then
update Emp set sal=i.sal+100 where Empno=i.empno;
end if;
end loop;
Close cl;
end;
/

```

\* update, delete statements used in cursor

(or)

- for update, where clause used in cursor;
- Generally whenever we are using update, delete statements automatically all database systems uses default lock mechanism
- If we want to establish the locks we are using for update clause in cursor select statement.
- whenever we are opening the cursors then only oracle server internally uses exclusive locks on the resource bases on for update lock.
- Syntax:
 

|                      |                                                                 |
|----------------------|-----------------------------------------------------------------|
| Cursor cursorname is | Select * from tablename where condition<br>for update [Nowait]; |
|----------------------|-----------------------------------------------------------------|

\* WHERE Current of clause :-

This clause used in update, delete stmts in cursor

→ Syntax :-

update tablename set columnname=new value  
where current of Cursorname;

→ Syntax :-

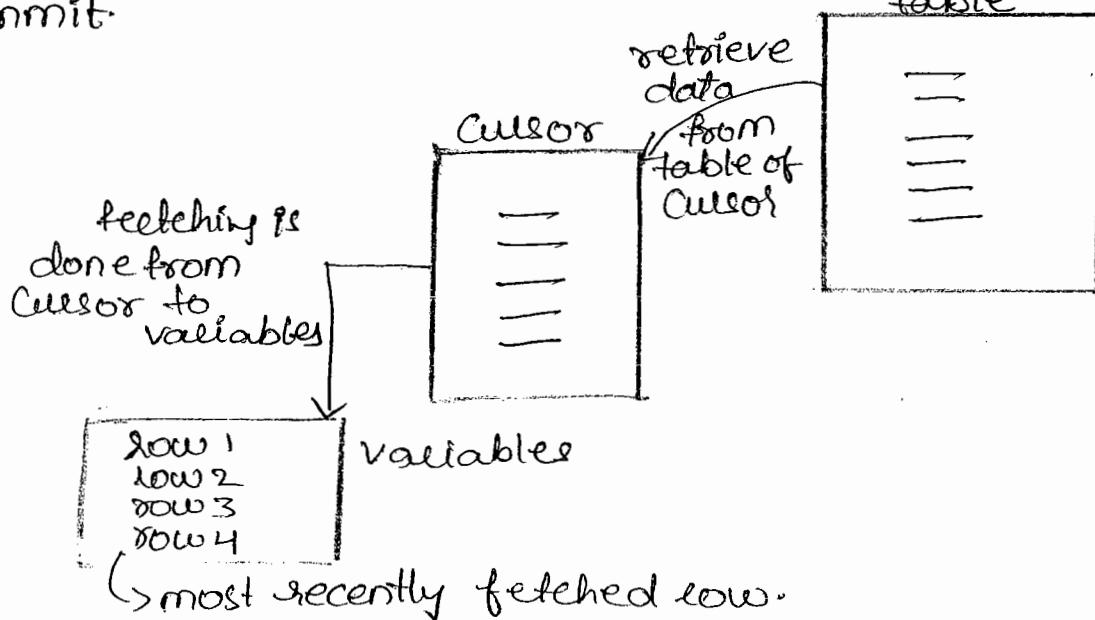
delete from tablename where current of Cursorname;

→ where current of clause uniquely identifies the record in each process. because where current of clause internally uses rowids

→ Note:- whenever we are using where current of clause then we must use for update clause

\* NOTE :- where current of clause used to update  
(a) delete most recently fetched row from the cursor -

→ After processing we must release locks by using Commit.



write a PL/SQL program by using cursor lock mechanism to modify salaries of the CLERKS from emp table.

SQL> declare

```
{ cursor c1 is select * from Emp
{ for update ;
i emp%. rowtype;
begin
open c1;
loop
fetch c1 into i;
exit when c1% not found;
if i.job = 'CLERK' then
update Emp set sal = i.sal + 100 where current
of c1;
end if;
end loop;
Commit;
close c1;
end;
/
```

| C1    |      |             |
|-------|------|-------------|
| Ename | sal  | job         |
| BLAKE | 3000 | MANA<br>GAR |
| Smith | 2800 | CLERK       |
| JONES | 6000 | Clerk       |

oracle database

| emp   |       |             |      |
|-------|-------|-------------|------|
| Ename | Empno | Job         | Sal  |
| BLAKE | 7839  | MANA<br>GAR | 3000 |
| SMITH | 7880  | CLERK       | 2800 |
| JONES | 7830  | CLERK       | 6000 |

write a PL/SQL program by using cursor lock in mechanism, all the employees being as manager raise 5% Salary in Emp table.

SQL > declare

cursor C1(p\_mgr number) is select sal from emp  
where mgr = p\_mgr for update;

V\_mgr number(10);

begin

select empno into V\_mgr from emp where ename  
= 'KING';  
for i in C1(V\_mgr)

loop

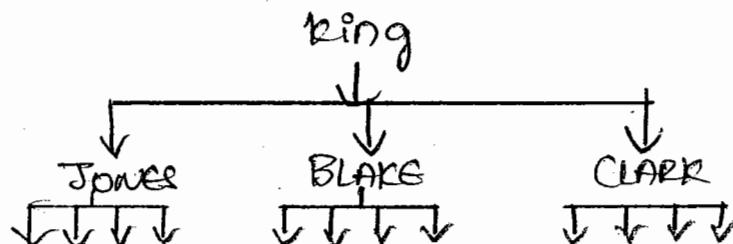
update emp set sal = i.sal \* 1.05 where current  
of C1;

end loop;

Commit;

end;

/



\* implicit cursor attributes :-

→ when a PL/SQL block contains select --- into --- clause (or) pure dml stmts (without cursor or dml stmts is called pure dml stmts) then Oracle server creates a memory area. This memory area is called as SQL area (or) implicit cursor.

→ This memory area internally having 4 attributes

- (i) SQL% not found
- (ii) SQL% found
- (iii) SQL%. is open
- (iv) SQL%. row count

- Here, always sql%isopen returns false and also sql%not found, sql%found attributes returns either true (or) false, when we are using DML stmts in PL/SQL block.
- And also sql%lowcount attributes returns number datatype.

```
* SQL>begin
  delete from emp where ename='ameerpet';
  if sql%found then
    dbms-output.put-line ('your record deleted');
  end if;
  if sql%not found then
    dbms-output.put-line ('your record does not exist');
  end if;
end;
/
O/P:- your record does not exist.
```

Ex: we can use sql%lowcount in which conditions

```
SQL>begin
  update emp set sal=sal+100 where job='CLERK';
  dbms-output.put-line ('affected number of
  clerks are '|| sql%lowcount);
end;
/
O/P:- affected number of clerks are 4.
```

→ In the above program sql%lowcount is used how many number of clerks are effected, because sql%lowcount return number datatype.

## EXCEPTIONS

- Exception is an error occurred during runtime, whenever runtime error is occurred use an appropriate exception name in exception handler.
- There are 3 types of exceptions supported by oracle.
  - (i) predefined Exceptions
  - (ii) userdefined Exceptions
  - (iii) unnamed Exceptions
- (i) predefined Exceptions :-
- Oracle defined 20 predefined Exception names for regularly occurred runtime errors. whenever error occurred use an appropriate predefined exception name in the following exception handler under exception section in PL/SQL block.
- Syntax :-

```
when predefined exception name1 then
statements;
```

```
when predefined exception name2 then
statements;
```

```
-----  
when others then
statements;
```

### \* predefined exception names :-

- (1) no-data-found
- (2) too-many-rows.
- (3) zero-divide
- (4) invalid\_CURSOR

(C) cursor - already - open

(E) invalid - number

(F) value - error.

(D) no-data-found:-

→ when a PL/SQL block contains select ---  
into -- clause and also if requested data is  
not available in a table then oracle server  
returns an error as

ORA - 1403 : no data found.

→ To handle this error we are using "no-data-found"  
exception name.

→ Ex :-

```
SQL> declare
      v_ename  varchar2(10);
      v_sal    number(10);
begin
  select ename, sal into v_ename, v_sal
  from emp where empno = &no;
  dbms_output.put_line(v_ename || ' ' || v_sal);
exception
  when no_data_found then
    dbms_output.put_line("your employee does
                           not exist");
end;
```

SQL> Enter value for no: 7902

FORD 3400

/

Enter value for no: 1111

Your employee does not exist.

② too-many-rows :-

→ when select --- into --- clause try to return more than one value (or) more than one record then oracle server returns an error as

ORA-1422: exact fetch returns more than requested number of rows.

→ To handle this error we are using "two-many-rows" exception name.

| Emp |      |
|-----|------|
|     | Sal  |
|     | 3000 |
|     | 4000 |
|     | 9000 |
|     | 3500 |
|     | 3400 |



select sal into v-sal  
from Emp;

error, because

Select -- into -- clause does not take all records at a time it takes only one record.

Ex:- SQL> declare

```

    v_sal number(10);
begin
  Select sal into v_sal from Emps;
  dbms_output.put_line(v_sal);
exception
  when too_many_rows then
    dbms_output.put_line('not to return more
values');
end;
/

```

③ Zero-divide :-

[ORA-1476: divisor is equal to zero]

Ex:- SQL> declare

```

    a number(10);
    b number(10);
    c number(10);

```

```
begin
a := 70;
b := 0;
c := a/b;
dbms_output.put_line(c);
```

exception

when zero\_divide then

```
dbms_output.put_line('b cannot be zero');
```

end;

/

③ zero-divide-

[ORA-1476 : divisor is equal to zero].

Ex - SQL > declare

```
a number(10);
b number(10);
c number(10);
```

begin

```
a := 70;
```

```
b := 0;
```

```
c := a/b;
```

```
dbms_output.put_line(c);
```

exception

when zero\_divide then

```
dbms_output.put_line('b cannot be zero');
```

end;

/

④ invalid-cursor :- When we are not opening the cursor then oracle server returns an error.

[ORA-14001 : invalid cursor] To handle this error we are using "invalid-cursor" exception name.

Ex - SQL > declare

```
cursor a is select * from Emp where Sal > 2000;
```

```

1%.rowtypes;
begin
loop
fetch a into i;
exit when c.%notfound;
dbms_output.put_line(i.ename||' '||i.sal);
end loop;
close cl;
Exception
when invalid_cursor then
dbms_output.put_line('first we must open the
cursor');
end;
/

```

⑤ cursor-already-open: - when we are try to reopeive cursor without closing the cursor properly then oracle server returns an error as

**ORA-6511: cursor already open**

→ To handle this error we are using cursor-already-open exceptionname.

→ ex: declare  
cursor c1 is select \* from emp where sal>2000;  
i emp%.rowtypes;  
begin  
open c1;  
loop  
fetch c1 into i;  
exit when c.%notfound;  
dbms\_output.put\_line(i.ename||' '||i.sal);  
end loop;  
open c1;  
Exception

when cursor-already-open then  
dbms\_output\_line('we must close the cursor before  
reopen the cursor');

end;

/

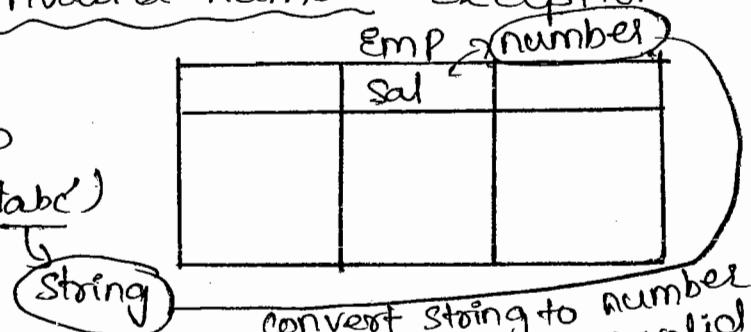
#### ⑥ invalid-number & value-error :-

→ whenever we are trying to convert string type to number type Oracle server returns an error "invalid number", "value error"

→ when a PL/SQL block contains SQL statements and also if we are trying to convert string type to number type then Oracle server returns an error ORA-1722: invalid number To handle this error we are using "invalid-number" exception name.

⑦ :-

select sal into emp  
(empno,sal) values (1,'abc')



⑧ begin  
sql\_stmts; (string) → (number) } error; invalid number  
end

⑨ begin  
PL/SQL stmts; (string) → (number) } error; value error.  
end

Ex:- SQL> begin  
insert into emp(empno,sal) values (1,'abc');  
exception  
when invalid\_number then  
dbms\_output.put\_line('insert proper data');  
end;  
/

Ex:- SQL> declare  
 v-deptno varchar2(10):='fdeptno';  
 v-deptname varchar2(10):='fdname';  
 v-loc varchar2(10):='floc';  
 begin  
 insert into dept values(v-deptno, v-dname, v-loc);  
 exception  
 when invalid\_number then  
 dbms\_output.put\_line('enter properdata');  
 end;  
 /

O/P:-  
 enter value for deptno: 1  
 enter value for dname: a  
 enter value for loc: b

There is no error, even we are giving the datatype for deptno as varchar2(10) & we are giving 'l'. Because it takes actual table datatype of particular column.

Enter value for deptno: x

Enter value for dname: y } error  
 Enter value for loc: z } "invalid number"

\* Value-error - When a PL/SQL block contains procedural stmts (or) PL/SQL stmts and also if we are try to convert string type to number type then oracle server returns an error.

ORA-6502: numeric or value\_error; character to number conversion error.

→ To handle this error we are using 'value-error' exception name

Ex:- SQL> declare  
 z number(10);

```

begin
z := 'fx'+fy';
dbms_output.put_line(z);
exception
when value_error then
dbms_output.put_line('enter proper data');
end;
/

```

Note - when we are trying to store more than the datatype size specified in a variable then oracle server returns an error.

ORA-6502: numeric or value errors character string buffer too small.

→ To handle this error we are using value\_error exception name.

→ Ex: SQL> declare

```

z varchar2(3);
begin
z := 'abcd';
dbms_output.put_line(z);
exception
when value_error then
dbms_output.put_line('invalid string length');
end;
/

```

Ex - SQL> begin
declare
z varchar2(3);= 'abcd';
begin
dbms\_output.put\_line(z);
exception

```

when value-error then
dbms-output.put-line('invalid string length');
end;
exception
when value-error then
dbms-output.put-line('!Invalid String length handled
by using outer block only');
end;
/

```

O/P 1- Invalid string length handled by using outer block only.

\*Exception propagation:- Exceptions also raised in either declare section (or) in executable section (or) in exception section. When an exceptions are raised in executable section, those exceptions are handled either in inner blocks (or) in outer blocks. Whereas when exceptions are raised in declare section (or) in exception section those exceptions must be handled outer blocks only. This is called as "Exception propagation".

\*User defined Exceptions:- We can also create our own exceptions & also raise whenever necessary. These type of exceptions are called userdefined exception.

Step 1: declare

Step 2: raise

Step 3: handle exception.

① declare:- In declare section of the PL/SQL block we are creating our own exception name by using exception predefined type.

## Syntax :-

user-defined name exception;

exit SQL>declare

a exception;

Step 2: raise - By using raise statement we can also raise Exceptions explicitly either in executable Section (or) in exception Section.

Syntax :-

```
when user-defined Exception name1 then
    stmts;
when user-defined Exception name2 then
    stmts;
-
-
-
-
-
-
-
-
-
When others then
    stmts;
```

Ex:- SQL> declare

*z exception;*

begin

if to\_char(sysdate, 'DY') = 'THU'

then

raise -z;

end if;

# Exception

when  $z$  then

```
dbms_output.put_line('my exception raised on  
end;                                         thursday');
```

end;

8

Ex: SQL > declare

z exception;

V-Sal numbers(10);

begin

Select sal into v-sal from Emp where Empno=7902;

13

```
end if;
exception
when z then
dbms_output.put_line('salary already high');
end;
/
```

NOTE :- Generally userdefined exceptions are used, when a PL/SQL blocks contains number of blocks & also if we want to pass control to the number of blocks without checking each individual conditions.

NOTE :- Generally these userdefined exceptions also used to test exception propagation.

### Testing Exception Propagation:-

① Exceptions raised in executable section :- when Exceptions raised in executable sections those exceptions are handled either using inner blocks or using outer blocks.

#### using inner block-

```
sql> declare
      z exception;
begin
raise z;
exception
when z then
dbms_output.put_line('handled using inner
block');
end;
/
```

O/P: handled using inner block.

## using outer block

SOL> declare

```
z exception;
begin
begin
raise z;
end;
Exception
when z then
dbms_output.put_line('handled using outer
block');
end;
```

Exceptions raised in exception section :-

When exceptions are raised in exception section those exceptions must be handled by using outer block only.

Ex:- SOL> declare

```
z1 exception;
z2 exception;
begin
begin
raise z1;
Exception
when z1 then
dbms_output.put_line('z1 handled');
raise
when z2 then
dbms_output.put_line('z2 handled through
outer blocks only');
end;
```

O/p :- z<sub>1</sub> handled

z<sub>2</sub> handled through outer blocks only.

### 2. Exceptions raised in declare section :-

When exception raised in declare section thus exceptions must be handled using outer blocks only

NOTE:- By using raise statement we can also raise predefined exceptions explicitly.

Ex :- Cursors.

Ex :- SQL declare

Cursor c1 is select \* from Emp  
where job = 'welcome';

i emp% rowtype;

begin

Open c1;

fetch c1 into i,

if c1% rowcount = 0 then

raise no\_data\_found;

end if;

close c1;

exception

when no\_data\_found then

dbms\_output.put\_line('u & job not available in  
my table');

end;

/

O/p :- u & job not available in my table.

### \* Unnamed Exceptions :-

If we want to handle other than oracle 20 predefined exception errors then we are using

unnamed exceptions. In this case we are creating our own exception name & associate that exceptionname with appropriate error number by using exception-init() function.

Syntax

Pragma exception-init(<sup>user defined , error</sup>  
<sub>exceptionname number</sub>)

- Here pragma is a Compiler directive i.e. this function associates error number with exception name at compile time.
- This function should be used in PL/SQL block of declare section,

Ex :- SQL> declare  
z exceptions;  
pragma exception-init(z,1400);  
begin  
insert into emp(Empno,ename) values(null,  
exception  
when z then  
dbms\_output.put\_line('not to insert null values');  
end;  
/

O/p :- not to insert null values.

Ex :- SQL> declare  
z exceptions;  
pragma exception-init(z,-2292);  
begin  
delete from dept where deptno<10;  
exception  
when z then

dbms-output.put-line('not to delete master record');  
 Q:- write a PL/SQL program handle - 2291 error number by using emp,dept table through exceptioninit method.

SQL > begin  
 insert into Emp(Empno,DeptNo) values(1,50);  
 end;  
 /  
Error : 2291

so if we want handle the above exception,

SQL > declare

z exception;

pragma exception-init(z,-2291);

begin

delete from dept where DeptNo=10;

exception

when z then

dbms-output.put-line('not to delete master record');

end;

/

Q:- write a PL/SQL program handle - 2292 error number by using emp,dept table through exception-init method.

SQL > begin

insert into Emp(Empno,DeptNo) values(1,50);

end;

/

Error : 2291

so if we want handle the above exception,

SQL > declare

z exception;

```

    pragma exception-init(z,-2291);
begin
insert into emp(empno,deptno) values(1,10);
exception
when z then
dbms-output.put-line('not to insert other than
primary key values');
end;
/

```

### Error trapping functions :-

- ① sqlcode
- ② sqlerrm

→ There are 2 error trapping functions supported by PL/SQL. These functions are used in exception section within PL/SQL block.

→ These are sqlcode, sqlerrm

→ sqlcode always returns numbers whereas sqlerrm always returns error number with error message.

| sql code return values | Meaning                 |
|------------------------|-------------------------|
| 0                      | no errors               |
| negative               | oracle errors           |
| 100                    | no data found           |
| 1                      | User defined exceptions |

Ex :- SQL> declare  
v-sal number(10);  
begin  
select sal into v-sal from emp;

dbms-output.put-line (v-sal);

exception

when others then

dbms-output.put-line (eglcode);

end;

/

Ex1-sol> declare

z exception;

begin

raise z;

Exception

when z then

dbms-output.put-line (egl code);

dbms-output.put-line (eglerrm);

end;

/

O/P:- 1

user-defined exception

→ NOTE:- we can also use these function return values into insert statements. In this case we must declare some variables and assign these function return values into variable & use these variables into insert statement.

\* But directly we can't use these functions into insert statement.

Ex1-sol>create table test(erno number(10),msg

voucher2(100));

SOL > declare

v-sal number(10);

k-erno number(10);

v-msg voucher2(100);

begin

```
select sal into v-sal from emps  
exception  
when others then  
    v-errno := sqlcode;  
    v-msg := sqlerrm;  
    insert into test values (v-errno, v-msg);  
end;
```

SQL> select \* from test;

NOTE :- Generally these SQL code return values used by another application programmers when they are deconstructing PL/SQL block & properly handling appropriate exceptionnames base on sql code

return values

\* raise-application-error :- (predefined function)  
If we want to display userdefined exception msgs in more descriptive form then we are using raise-application-error function i.e. using this function we are displaying userdefined exception messages as same as Oracle error displayed format. This function used in either executable section (o1) in exception section.

Syntax      raise application-error(errornumber, message)

-20000 to -20999

upto 12  
characters

ex:- SQL> declare

v-sal number(10);

z- exceptions

begin

```

{ select sal into v-sal from Emp where Empno= 7902;
  if v-sal > 200 then
    raise z
  else
    update Emp set Sal = Sal + 100 where Empno = 7902;
  end if;
  exception
    when z then
      { raise-application-error(-20123,'Salary already high');
      end;
    }

```

ORA-20123 : Salary already high.

NOTE :- Generally raise-application-error function is used in triggers. Because, this function stops invalid data entry when the condition is true.

Rough  
 dbms-output.put-line('---')  
 Stmt Executed

Subprograms

- Subprograms are named PL/SQL blocks, which is used to solve some particular task.
- There are two types of subprograms supported by Oracle. Those are

- Procedures
- functions

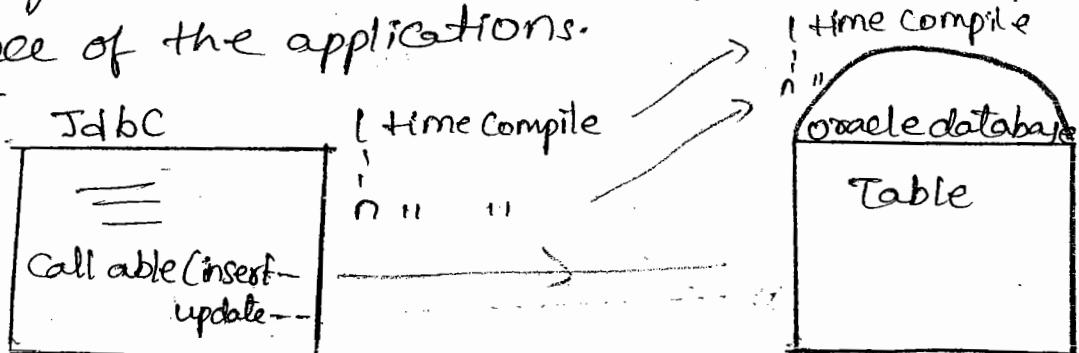
(1) procedures :- procedure is a named PL/SQL block which is used to solve some particular task and also procedure may or may not return values  
 → Some procedures are stored in database

automatically those procedures are called stored procedures)

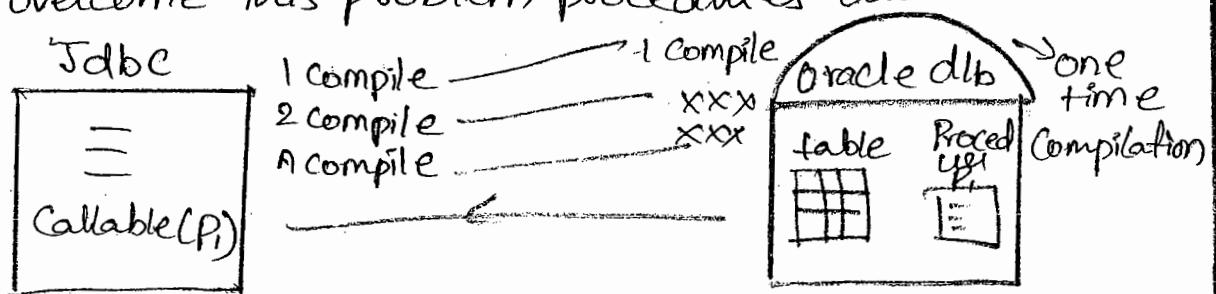
→ when we are using create (or) replace keywords in front of the procedures those procedures automatically permanently stored in database. That's why these procedures are also called as stored procedures.

→ Generally procedures are used to improve performance of the application, because procedures internally having "One time Compilation". By default in all database systems one time compilation program units automatically improves performance of the applications.

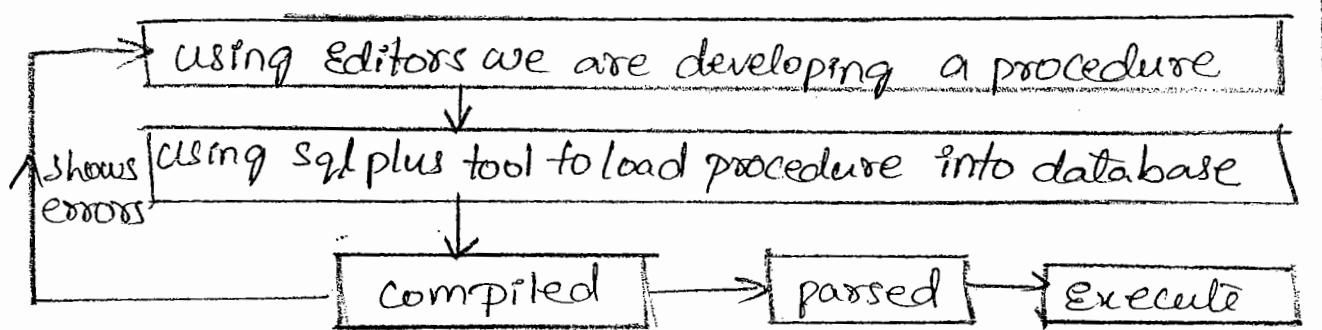
Ex:-



→ Time is wasted because of number of times compilation. To overcome this problem procedures are used.



→ procedure execution process -



→ procedure having 2 parts

(i) procedure specification

(ii) procedure body.

→ In procedure specification we are specifying name of the procedure, type of the parameters

→ Whereas in procedure body we are solving actual task.

→ Syntax i-

Create or replace procedure procedurename (formal parameters)

→ is / as

nation

→ variable declaration, cursors, user defined exceptions;

Begin

≡

[Exception]

≡

end [procedure name];

procedure  
body

Syntax i- parameter name [mode] datatype

\* mode is optional

[mode]

In  
Out

Inout

NOTE i- Declare section of anonymous block & triggers is declare.

→ Declare section of procedures, functions is "is / as"

write a PL/SQL stored procedure for passing Empno as a parameter, display name of the Employee and his Salary from Emp table?

SOL > Create or replace procedure P\_1(p-Empno number)

is

v-ename varchar2(20);

v-Sal number(10);

```
begin
select ename, sal into v_ename, v_sal from emp
  where empno = p_empno;
dbms_output.put_line(v_ename || ' ' || v_sal);
end;
/
```

\* Creating a procedure:-

\* method①:-

```
SQL> exec procedurename (actual parameters);
```

\* method②:- (by using anonymous block)

```
SQL> begin
      procedure name (actual parameters);
    end;
/
```

\* method③:-

```
SQL> call procedurename (actual parameters);
```

\* to view errors:- SQL> show errors;

\* Execution:-

method①:- SQL> exec P1(7902);

Output: FORD 3500

method②:- (By using anonymous block);

```
SQL> begin
```

```
  P1(7889);
end;
/
```

Output: KING 5400

method③:- SQL> call P1(7566);

Output: JONES 3334

→ All stored procedures information is stored under

User-procedures, user-source data dictionary

→ NOTE :- If we want to view text (coding) of the procedure then we are using user-source data-dictionary.

Ex :- SQL > desc user\_source;

SQL > select text from user\_source where name

= 'PI';

↳ (procedure name)

it must be in Capital letters

WRITE a PLSQL stored procedure for passing deptno as a parameter, display employee details for that deptno from emp table?

|   |       |      |    |
|---|-------|------|----|
| [ | SMITH | 2000 | 20 |
|   | ALLEN | 3999 | 20 |
|   | WARD  | 4000 | 20 |

SQL > create or replace procedure PI (P\_deptno number)  
is

Cursor C1 is select \* from emp where deptno = P\_deptno;  
i emp%rowtype.

begin

open C1;

loop

fetch C1 into i;

exit when C1%not found;

dbms\_output.put\_line (i.ename || ' || i.deptno);

end loop;

close C1;

end;

/

Execution :- SQL > exec P1(10);

method① :-

method② :- (using anonymous block)

SQL> begin

    P1(20);

end;

/

Parameters in procedure :-

→ There are 2 types of parameters in procedures

① formal parameters ② Actual parameters.

① Formal parameters :-

→ It is defined in procedure specification

→ These parameters specifies name of the parameter and type of parameter.

Syntax :- [parametername mode] datatype

\* mode :- Based on the purpose of the parameters 3 types of modes supported by oracle parameter

① in mode ② out mode ③ in out mode.

① in mode :- The mode is used to pass the value into procedure body,

→ This mode internally behaves like a constant in procedure body

→ By default mode is in mode, through the in mode, we can also pass default values using either default or := (i.e. colon equal to operator).

(Q) Write a PL/SQL stored procedure, insert a record into dept table using in parameters.

SQL > create or replace procedure p1 (P\_deptno in number,  
P\_dname in varchar2, P\_loc in varchar2)

is  
begin

insert into dept values (P\_deptno, P\_dname, P\_loc);

dbms\_output.put\_line ('Record inserted through  
procedure');

end;  
/

Execution :-

SQL > exec P1(1,'a','b');

→ There are 3 types of execution methods supported by in parameters.

① positional notations

② named notations

③ mixed notations

① Positional Notations :- SQL > exec P1(1,'a','b');

② Named Notations :-

SQL > exec P1 (P\_dname => 'x', P\_loc => 'y', P\_deptno => '2');

③ Mixed Notations :-

→ It is a combination of positional, named notations.

But after positional there can be all named notations.

But after named they can not be positional

i.e

SQL > exec P1(3, P\_loc => 'm', P\_dname => 'z');

record inserted through procedure

out mode :-

→ This mode is used to return values from procedure body.

→ This mode internally behaves like a uninitialized variable in procedure body.

→ Here, explicitly we must specify out keyword

Syntax :- parametername out datatype

Ex :- SQL> Create or replace procedure p1(<sup>constant</sup> a in number,  
variable b out number);  
is

```
begin  
b:=a*a;  
end;
```

/

→ When a procedure contains out, in out parameter those procedures are executed by using following 2 methods

method① :- By using bind variable

method② :- By using anonymous block.

method① :- By using bind variable:-

SQL> variable z number;

SQL> exec p1(3,:z);

SQL> print z;

method② :- By using anonymous block:-

SQL> declare

x number(10);

begin

p1(8,x);

dbms-output.put-line(x);

end;

/

O/P 64

Q) Write a PL/SQL stored procedure for passing a parameter return salary of the Employee by using out parameter.

SQL> Create or replace procedure P1 (P-ename in  
varchar2 , P-sal out number)

is ↳ variable

begin

Select sal into P-sal from Emp where empname = P-ename;  
end;

↳ behave like variable

/

Execution :- Method ① :-

① By using bind variable :-

SQL> Variable a number;

SQL> exec P1 ('SMITH', :a);

SQL> print a;

A  
1400

② Method ② :- By using anonymous blocks :-

SQL> declare

x number(10);

begin

P1 ('KING', x);

dbms-output.put\_line(x);

end;

/

O/P 1 - 5400

→ Q :- Write a PL/SQL stored procedures for passing deptno as a in parameter return number of employees number using out parameter from Emp table

SQL> Create or replace procedure P2 (P-deptno in number,  
P-Count out number)

is  
begin

Select count(\*) into p\_count from Emp where deptno  
end;  
= p-deptno;

Execution: method①: using bind variable:

SOL> Variable z number;

SOL> exec P1(10; :z);

SOL> print z;

-z-  
3

(or)

SOL> Variable z number;

SOL> exec P2(20; :z);

SOL> print z;

-z-  
5  
!

method②: Using anonymous block:-

SOL> declare

  y number(10);

  begin

    P1(20, y);

    dbms-output.put-line(y);

  end;

/

O/P: 5

③ \* in out :-

This mode behave like a constant, initialized variable in procedure body. Here also explicitly we must specify "in out" keyword.

→ Syntax:-

Parametername in out datatype;

② SOL> create or replace procedure P1(a in out number)

is

begin

a:=a\*a;

end;

/

NOTE:-

In out clause acts as a variable. So no need to declare variable in begin-end section.

→ Execution: Method ①: by using bind variable :-

SOL>Variable Z number;

SOL> exec :z :=9;

SOL> exec P1(:Z);

SOL> print z;

Z -

81

Method ② using anonymous block :-

SOL> declare

z number(10):=4no;

begin

P1(z);

dbms-output-line(z);

end;

/

Q1P1:- Enter value for no: 8

64

③ write a PL/SQL stored procedure for passing Employee number return salary of the employee from emp table by using "inout" parameter?

SOL>create or replace procedure P1(P2 in out number)

IS

begin

Select sal into P-X from Emp where Empno = P-X;  
end;

Execution Method ① :- by using bind variable :-

SOL>variable Z number;

SOL>exec :Z:= 7839;

SOL>exec PI({Z});

SOL>print Z;

Z -

5400

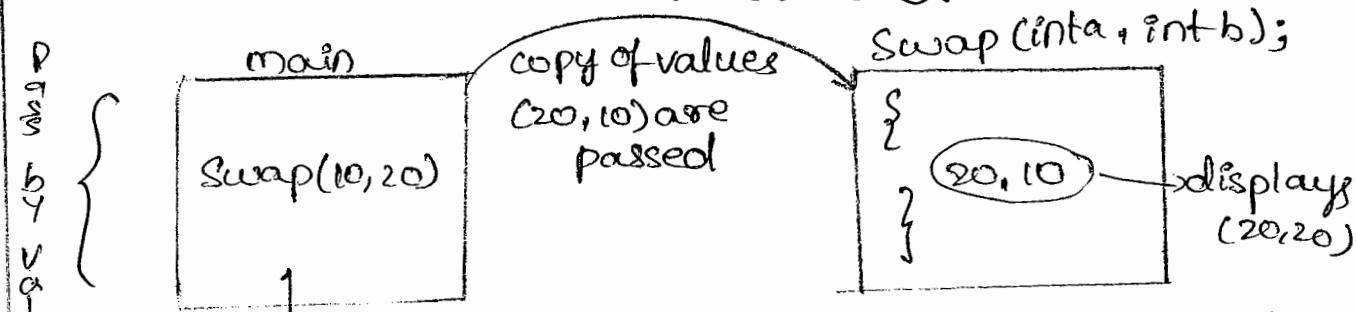
Method ② :- By using anonymous block :-

\* NOCOPY :-

→ In "out" parameter we will get this Nocopy.

→ Before this nocopy we should know pass by value and pass by reference.

→ pass by value pass by reference :-



but it displays (10, 20), that means actual values are not changed in main

→ so to change the actual values we should use pass by reference

→ In parameter behaves like pass by reference for out

10/12

parameter behaves like pass by value.

\* When we are using modular programming (functions, procedures---) we are using two passing parameter mechanisms. These are

- ① Pass by value
- ② Pass by reference.

→ These 2 mechanisms specifies when we are changing formal parameters then actual parameters are effected (or) not.

→ In pass by value method actual value does not change in main program. Because internally copy of the values passed into called program.

→ If we want to change actual values based on the formal parameter modification then we are using pass by reference method.

\* Oracle database also, Suppose these 2 methods, when we are passing parameter to the subprograms. By default all inparameters uses pass by reference method And also by default all outparameters uses pass by value method.

\* When we are returning large amount of data by using outparameters Oracle Server internally creates copy of the values. This process automatically degrades performance. To Overcome this problem Oracle introduce "no copy" hint in the out parameters.

Syntax :- parametername out nocopy datatype.

Eg:- SQL> Create or replace procedure P,(P\_ename in  
varchar2, P\_sal out nocopy number)

is

begin

Select sal into P\_sal from emp where ename = p\_ename;

end;

/

\* Autonomous transactions :- (∴ autonomous = independent)

→ Autonomous transactions are independent transactions used in either in anonymous block or in procedures or in triggers.

→ These transactions are behaves independently from main transactions. When a procedure convert into autonomous procedure then we are using autonomous transaction pragma, commit

→ Syntax :-

pragma autonomous - transaction; (or)

→ This pragma is used in declare section (IS/LAS) of the procedure.

Create or replace procedure procedurename  
(formal parameters)

IS/LAS

pragma autonomous - transaction;

---

begin

---

commit;

[exception]

---

end [procedurename];

NOTE:- Begin

[P<sub>i</sub>;] → not modified

↑ P<sub>2</sub>;

↑ P<sub>3</sub>;

↑ P<sub>4</sub>;

} Here P<sub>1</sub>, P<sub>2</sub>, P<sub>3</sub>, P<sub>4</sub> are child procedure  
} If we are using roll back the  
} procedure are modified.  
} → But P<sub>i</sub> is not modified because for  
} P<sub>i</sub> we are using program autonomous  
} transaction & Commit.

\* Using autonomous transaction:-

SQL > Create table test (name varchar2(10));

SQL > Create or replace procedure P<sub>i</sub>

is

begin

insert into test values ('India');

Commit;

end;

/

Executions:- SQL > begin

insert into test values ('Hyd');

insert into test values ('Mumbai');

P<sub>i</sub>;

roll back;

end;

/

SQL > select \* from test;

NAME

India

\* SQL > delete from test;

\* without Using Autonomous transaction:-

SQL > Create or replace procedure P<sub>i</sub>

is

begin

insert into test values ('India');

Commit;

end;

/

Execution :- SQL > begin

insert into test values ('Hyd');

insert into test values ('Mumbai');

PL;

rollback;

end;

/

SQL > select \* from test;

NAME

Hyd

Mumbai

India

NOTE :- Before Oracle 8.1.6 if a procedure having Commit or rollback then Oracle server not only effects procedure transactions but also effect above of the all procedural transactions.

begin

Hyd

Mumbai

PL;

rollback;

end;

To overcome this problem Oracle 8.1.6 introduced autonomous transactions.

\* Handled or unhandled exceptions in procedure :-

→ whenever we are calling inner procedure into outer procedures then we must handle inner procedure exceptions. Otherwise Oracle server executes outer procedure default exception handler.

main P

outer P

inner P

→ we must use inner procedure exception handler.

Otherwise default exception of outer procedure.

① inner procedure (or) child procedure :-

SOL> Create or replace procedure P<sub>1</sub>(x numbers if in  
numbers)

is

begin

dbms-output.put-line(x/y);

{ exception

{ when zero-divide then

dbms-output.put-line('y cannot be zero');

end;

② outer procedure :-

SOL> Create or replace procedure P<sub>2</sub>

is

begin

P<sub>1</sub>(6,0);

exception

{ when others then

dbms-output.put-line('any error');

end;

/

SOL> exec p2;

y cannot be zero

\* Authid current-user: [Authid = authorized]

→ whenever we are reading data from a table and perform some DML operations then only development using authorized current user clause in procedure for data security point of view. If a procedure contains this clause, if another user giving privilege also then user does not execute this procedure. This clause used in procedure specification only.

## ~~#~~ Syntax

```
SOL>create or replace procedure procedurename  
authid current_user      (formal parameter)  
is  
-----  
begin  
[exception]  
end[procedure];  
/
```

\* Giving privilege of procedure to another user :-  
Syntax :-

grant execute on procedurename to user1, user2, ...;

Eg:- SQL>show user;  
User is "SCOTT"

SQL>create or replace procedure P1(P-Empno number)  
authid current\_user  
is  
V-ename Varchar2(10);  
V-Sal number(10);  
begin  
select ename, sal into V-ename, V-Sal from  
Emp where Empno = P-Empno;  
dbms\_output.put\_line(V-ename||' '||V-Sal);  
end;  
/  
SQL>grant execute on P1 to Soujanya;  
SQL>Conn Soujanya/Soujanya;  
SQL>Set Server output on;  
SQL>exec Scott.P1(75 66);

error: table or view does not exist

\* we can also drop procedure by using  
SQL>drop procedure procedurename;

\* functions \*

→ Function is a named PL/SQL block, which is used to solve some particular task. And also function must return a value.

→ function also having 2 parts

(1) Function specification

(2) Function body.

→ In function specification we are specifying name of the function & type of parameter whereas in function body we are solving actual task.

→ syntax :-

```
SQL>create or replace function functionname(formal
      return datatype           → [don't use ;. It is called
      is/as                      parameters]
      Variable declarations, cursors;
      begin
      →
      return expression;
      end [function name];
```

NOTE:- we can't use ; for the specification in PL/SQL except for package

\* Executing a function :-

(1) method (2) : Cusing Select Statement :-

Syntax :-

```
SQL>select functionname (actual Parameters) from dual;
```

NOTE :- dual table is used for testing

dept(01) emp Any table is used for application

② Method ② :- [using anonymous block] :-

Syntax :-

SOL > begin

Variable name := function name(actual parameters);

end;

/

Eg:- SOL > create or replace function f1 (a varchar2)

return varchar2

is

begin

return a;

end;

↳ (there we can use anything like hi, hello, bye  
Soujanya, etc.)

Execution :-

① method ① :- (using select statement)

SOL > select f1('hi') from dual;

OP :- hi

② method ② :- (using anonymous block)

SOL > declare

z varchar2(10);

begin

z := f1('hi');

dbms\_output\_line(z);

end;

/

OP :- ~~odd~~ hi

③ method ③ :- (By using bind variables)

SOL > Variable X varchar2(10);

```

SQL>begin
:z1:=f1('welcome');
end;
/
SQL>print z;
- - -
WELCOME

```

[ we are using here varchar2(20) we should mention size but if we are using number then no need to use size ]

NOTE:- All function information is also stored in user-procedure, user-source data dictionaries.

→ If we want to view text of the function then we are using user-source datadictionary

→ Eg:- SQL> desc user-source;

SQL> select text from user-source  
where name = 'F1';

Q:-  
write a PL/SQL stored function for passing number as a parameter return a message either even or odd based on that number.

```

SQL>create or replace function f1(a number)
return varchar2
is
begin
if mod(a,2)=0 then
return 'even';
else
return 'odd';
end if;
end;
/

```

\* Execution :-

Method ① :- (by using select stmt)

SOL > select f<sub>i</sub>(3) from dual;

O/P :- odd.

SOL > select f<sub>i</sub>(4) from dual;

O/P :- even

Method ② :- (by using anonymous block)

SOL > declare

  x varchar2

  begin

  x := f<sub>i</sub>(3);

  dbms\_output.put\_line(x);

  end;

  /

O/P :- odd

method ③ :- (by using bind variables)

SOL > variable x varchar2(10);

SOL > begin

  :x := f<sub>i</sub>(7);

  end;

  /

SOL > print x;

  -x-

  odd.

method ④ :-

NOTE :- We can also pass functions into procedure

SOL > exec dbms\_output.put\_line(f<sub>i</sub>(8));

O/P :- even.

method ⑤ :-

```
SOL >begin  
    dbms_output.put_line (f1(9));  
end;  
/
```

O/P :- odd,

NOTE - We can also use user defined functions in insert statement.

```
SOL>insert into test1 values(1, f1(6));
```

SQL>Select \* from test1;

SNO      MSG

NOTE- we can also create our own functions by using predefined aggregate functions And also use this functions in non-aggregatable columns.

Eg:- SOL> create or replace function f,

return number

15

V\_sal number(10);

begin

Select max(sal) into v-Sal from Emp;

return v-Sal;

end;

1

execution:-

SQl>select ename, sal, f1, f1+sal, f1-sal from emp;

Q-1- write a PL/SQL stored function for passing Empno as a parameter from Emptable return gross salary based on following conditions.

Here gross := basic +hra+da-pf;

hra → 10% of sal

da → 20% of sal

pf → 10% of sal

SQL> create replace function f1(P\_Empno number)

return number

is

v\_sal number(10);

gross number(10);

hra number(10);

da number(10);

pf number(10);

begin

Select sal into v\_sal from emp where empno = P\_Empno;

hra:= v\_sal \* 0.1;

da:= v\_sal \* 0.2;

pf := v\_sal \* 0.1;

gross := v\_sal + hra + da - pf;

return gross;

end;

/

Execution:

Method 0:-

SQL> Select f1(7566) from dual;

Output:-

13-

```

2 from Emp where Empno=p-Empno
return round(z);
end;
/

```

\* Note :- In the above program z may return floating point value so to convert that floating point values to integer values then we are using "round" function

\* Execution :-

SOL>Select Empno,ename, fi(Empno,Sysdate) ||' '|| 'Years' "Exp" from Emp where Empno = 7566;  
 here ("Exp" is Experience)

O/P:- EmpNo    ENAME    Exp  
 7566      Jones      32 years

↓  
 [here we  
 can give any  
 Empno]

Note :- In procedures we can also named, mixed notations when a Subprogram Executed Using select Stmt (functions)

Note :- Prior to oracle 11g we are not allowed to used named, mixed notations when a Subprogram Executed using Select Statement (functions) But in oracle 11g we can also use named, mixed notations when a Subprogram Executed Using

AI(7566)

4001

Note:-

→ We are not allowed to use DML stmts in functions (function body)

→ When we are using function and when we are using procedure:

→ In our application returns multiple values then procedures are used

→ In our application returns single value the functions are used

\* In procedures, functions we can use cursors, update, insert, delete; select ... into ... clause. But in procedures we can use DML Stmt's but in functions we can't use

Qs :- Write a PL/SQL stored function that passing Empno, date as parameters return number of years that employee is working from Emp table?

Sol :- SQL> Create or replace function f1(p\_Empno  
number, p\_date date)

return number

is number(10);

begin

Select months\_between(p\_date, hiredate)/12 into

begin

Select dname, loc into p-dname, p-loc from dept

Where dept-no = p-deptno;

return p-dname;

end;

/

Execution:-

method①:- By using bind variables

SOL>Variable a varchar2(10);

SOL>Variable b varchar2(10);

SOL>Variable c varchar2(10);

SOL> :a := f1(10, :b, :c);

end;

/

SOL>point b c;

O/Ps - B --

Accounting

- C --

Newyork

Notes- In procedures if 2 out parameters are there in a pgm, then 2 bind variables are required, while execution by using bind variable

Notes- But in functions if 2 out parameters are there in a pgm like (dname, loc in previous pgm)

## Select Statement (function)

Eg:- SQL> Select Empno,ename,fi(p-Empno=>Empno,  
p-date=>sysdate)||' '||'years' "Exp" from Emp  
Where Empno=7566;

I/O:

XOUT- This mode is used to return more values  
from the functions

Note- By default function return single value

But

→ By using OUT mode function returns multiple  
values, but datatypes should be same for our  
return values

→ Note- When a function contain OUT, IN OUT  
Parameters those functions are not Executed by  
Using Select Statement

→ Those are Executed by Using bind variable and  
anonymous block same as procedure

Q8- Write a PL/SQL Stored function for passing  
deptno as in-parameter, return deptname, loc  
from dept table by using OUT parameter

SQL> Create or replace function fi(p-deptno  
number, p-dname OUT varchar2, p-loc OUT varchar2)  
return varchar2  
is

```
begin
for i in c,
loop
a:=a || '|| i,ename;
end loop;
return a;
end;
/
```

### \*Execution:-

SOL>select deptno, fn(deptno) from Emp  
group by deptno;

Notes:- oracle 10g introduced WM-concat  
predefined aggregate function, which returns  
multiple values group wise. And also this  
function accepts all datatype columns  
(number, character, date etc)

Ex:- Select deptno, wm-concat(ename) from  
Emp group by deptno;

Ex:- SOL>Select deptno,wm-concat(hiredate) from  
Emp group by deptno;

O:- Display Employee names job wise from Emp  
table by using wm-concat) option from Emp  
table

then 3 bind variables (a, b, c) are required while execution, by using bind variable

In SQL :- SQL> select deptno, count(\*) from emp  
group by deptno;

| O/P :- | Deptno | Count(*) |
|--------|--------|----------|
|        | 10     | 3        |
|        | 20     | 5        |
|        | 30     | 6        |

### \* Cursors using functions -

→ we can also use cursors in user defined functions which returns multiple values. These user defined functions are executed by using Select Statement

→ we can also develop user defined aggregate functions same like a predefined aggregate functions and also use these user defined functions in group by clause.

Ex :- SQL> create or replace function f1(p\_deptno  
number)

return varchar2  
is

a varchar2(200);

cursor c1 is select ename from emp

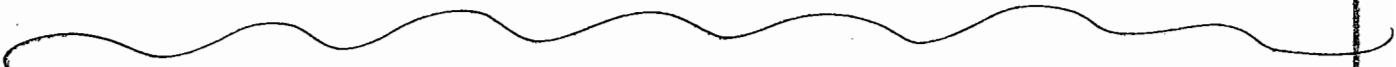
where deptno = p\_deptno;

15v

SQL>select job, wcm\_concat(ename) from Emp  
group by job;

→ We can doop function by using

SQL>doop function functionname;



C C C C C C C C

1 1 1 1 1 1 1 1

## \* DATABASE DESIGN

### \* Information System :-

- Information System allows data storage collection and retrieval.
- It also helps in managing data as well as information.
- It facilitates in transforming data into information. Therefore an information system can be defined as a collection of users, software, hardware, procedures, database applications programs
- Information system in the present era possess an important features called strategic value of information.
- Due to this, there exist a need to align information systems with strategic business goals and invalidate the other features such as isolation and independency.
- Systems development framework shows how applications software transforms data into information which forms a basis for decision making.

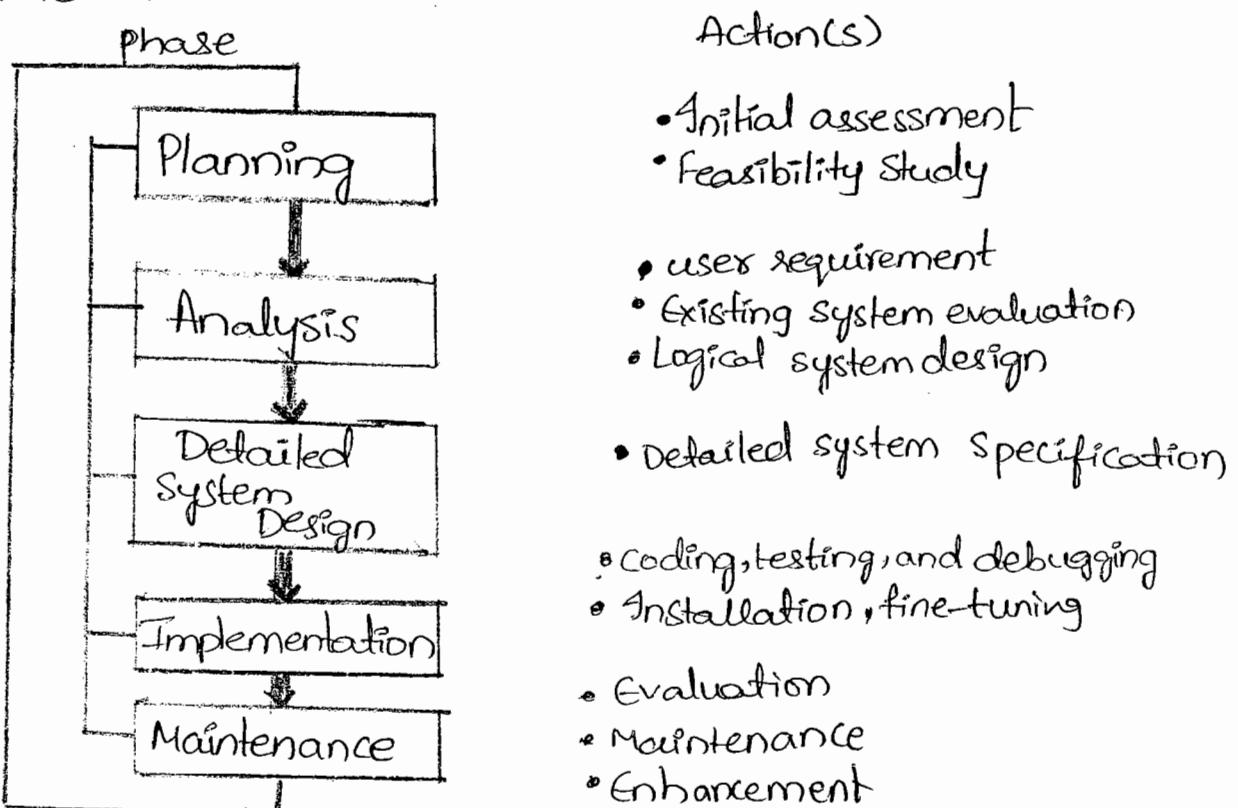
- Generally these applications consist of code and data which reflects the real world strategic business goals and invalidate the other features such as isolation and independency.
- Thus they are used to create formal reports, tabulations and graphic display in order to take a brief look at the information.

Performance of an information systems solely relies on the below factors.

- (i) Application design and implementation
  - (ii) Database design and implementation
  - (iii) Administrative procedures.
- Database design and implementation is the most important factor in all the three but this does not mean that the other two factors are not significant.
  - Therefore the main aim of database design is to develop a model which is normalized, complete, non-redundant, logical and physical database, fully integrated conceptual model.
  - The objective of implementation phase is to develop a database storage structure, load data into it and then provide it for data management.

## \* Systems Development Life cycle (SDLC) :-

- The systems Development life cycle tracks the history of an Information system.
- The SDLC provides the big picture within which the database design and application development can be mapped out and evaluated.
- The SDLC is an iterative rather than a Sequential process.
- The traditional SDLC is divided into five phases:-
  - ① planning
  - ② Analysis
  - ③ Detailed system Design
  - ④ Implementation
  - ⑤ Maintenance.



\*planning :-

Problem definition :-

Examination and evaluation of the problems of the current system.

Feasibility study :-

Development of objectives, analysis alternate design options, technical and economic feasibility of each alternative, projected schedule and proposed costs.

→ In this phase, the entire project strategy along with the organizational objectives is planned.

→ Initially requirements and information flow of the SDLC cycle should be determined which can help in answering few important questions like,

- 1) whether the existing system should be revised.
- 2) whether the existing system should be modified.
- 3) whether the existing system should be replaced.

Besides this the developer feels that the existing system is beyond fixing, then at this point feasibility of the cycle should be checked by the following factors.

→ Technical Hardware and software Requirements :-

Hardware and software requirements should satisfy the system requirements which is being developed but it should be according

to the vendor.

→ System cost :-

Before developing the system, an assessment should be done whether the system is affordable therefore review should be done keenly because spending millions of dollars for a solution to a thousand dollar problem is not feasible and unrealistic.

→ Operational cost :-

Before moving forward the developer should also focus whether the company possess the required operational staff for human, technical resources so that they can keep the system running.

2) System Analysis :-

(a) Detailed study of the current system, its procedures, information flows and method of works organization and control.

(b) Development of logical model of the current system.

→ In this phase, all the problems specified in planning phase are keenly observed and examined. It should cover the organizational as well as individual needs.

### 3) Detailed system design :-

→ In this phase, designing of the System process are Considered Such as

- Development of objectives for the proposed System.
- Development of a logical model, process logic definition, logical data dictionary, and logical design for proposed data System.
- Development of Cost-Benefit analysis, evaluate the economic implications.

### 4) implementation :-

- During the implementation phase, the hardware DBMS Software, and application programs are installed and the database design is implemented.
- During the initial stage of the implementation phase the system enters into a cycle of coding, testing and debugging until it is ready to be delivered.

→ Development of security, audit and test procedures.

### 5) Maintenance :-

Almost as soon as the system is operational end user begin to request changes in it. Those

16

Chances generate system maintenance activities, which can be grouped into three types:

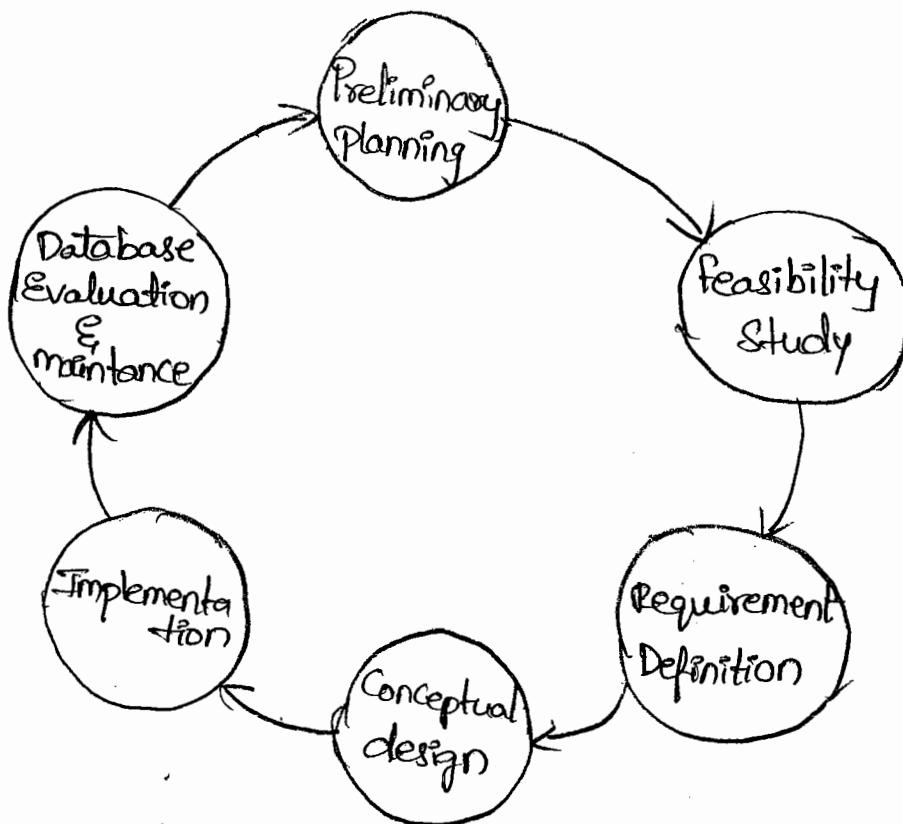
1. Corrective Maintenance :- It corrects any system errors in response to the user's request.
2. Adaptive Maintenance :- It tries to maintain any change caused due to business environment.
3. perfective Maintenance :- It enhances the system's efficiency and performance.

→ Each and every system has its own operational life span because maintaining a system for prolonged time leads to increase in cost therefore a system is maintained only when it is useful.

## \* THE DATABASE LIFE CYCLE :-

- Basically database is carefully designed and constructed Repository of facts.
- The database is a part of larger whole known as an information system.
- The information system also facilitates the transformation of data into information and allows to manage both data and information,
- The performance of information system requires Database design and implementation.
- In broad sense the term database development itself describes the process of database design and implementation.
- Within the larger information system, the database also is subject to life cycle.
- The database life cycle (DBLC) contains six phases:
  - (i) Preliminary planning
  - (ii) Feasibility study
  - (iii) Requirement definition
  - (iv) Conceptual design
  - (v) Implementation
  - (vi) Database evaluation and maintenance.

→ The DDL is concerned with the development of a comprehensive database and the programs needed to process it.



### "A. Database Development life cycle"

#### (i) Preliminary planning:-

→ This is the first stage of the life cycle where planning takes place during the strategic database planning project.

→ During this phase, the firm collects information such as;

- (i) How many application programs are in use
- (ii) What files are associated with each of these applications?
- (iii) What new application and files are under development

## ② Feasibility study :-

→ An important outcome of the investigation is the determination that the system requested is feasibility.

→ There are three major areas to consider while determining the feasibility of a project are;

(i) Technical feasibility

(ii) Economic feasibility

(iii) Operational feasibility

(i) Technical feasibility :-

→ The Analyst findout whether current technical resources, which are available in the organization are capable of handling the user requirements.

→ If not, then Analyst should confirm whether the technology is available and capable of meeting the user's request.

(ii) Economic feasibility :-

→ Economic or financial feasibility is the next part of resource determination.

→ This feasibility study concentrate, whether it is economically feasible in certain cases such as installation etc...

### (iii) operational feasibility :-

→ It is a measure of how well a proposed system solves the problems and takes advantage of the opportunities identified during scope definition and how it's satisfies the requirements identified in the requirements analysis phase of system development.

### (3) Requirement Definition:-

→ The result of this stage are described in four tasks;

(i) The database system scope was defined by analysing management information requirements. The team should consider whether the database should be distributed or centralised.

(ii) User requirements at management and operational levels were documented with a generalized information model.

(iii) General hardware & software requirements were established.

(iv) A plan was made for a time phased development of a system.

## (4) Conceptual Design:

- This design stage creates the Conceptual schema for the database.
- Specification are developed to the point where implementation can begin.
- During this stage, detailed models of user are created and integrated into a conceptual data model recording all corporate data elements of be maintained in the database.
- Development of the model was guided by information contained in policy and procedure manuals.
- They were then integrated as part of conceptual design and became the basis for implementation.

## (5) Implementation:-

- During this phase, a DBMS is selected and acquired then the detailed conceptual model is converted to the implementation model. Populated of the dbms, the data dictionary built, the database populated, application programs developed and user trained.

- The next step was mapping the firm's conceptual model to a relational implementation model. This was done by using the procedures. This was done by using a data definition language (DDL) supplied with the DBMS to develop the data dictionary.
- The next step was to populate the database by loading data into the database. This was done by data manipulation language (DML) supplied with the DBMS.
- The final step was to develop procedures for using the database and to setup training sessions on these procedures and the other facilities of the system.

#### ⑥ Evaluation & maintenance :-

- Evaluation involves interviewing user to determine if any data needs are unmet. Changes are made as needed.
- Then the system is maintained via the introduction of enhancement and the addition of new programmes and data elements.

## \* THE DATABASE DESIGN STRATEGIES:

→ The two database design strategies includes,

- 1) Top-down design
- 2) Bottom-up design.

1) Top-down design:- The steps performed in this design strategy are as follows,

- (a) Identify the data set.
- (b) Define the data element associated with each of the identified data set.

In this process, the focus is on identifying the different entity types and defining the attributes for each of these entities.

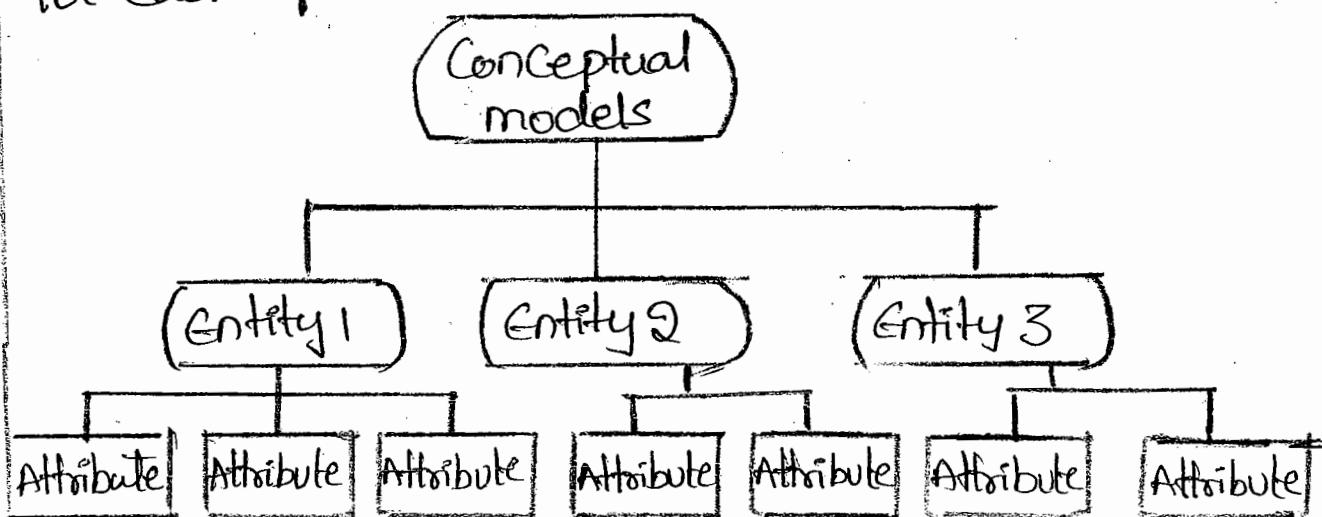


Fig:- Top-down Design strategy

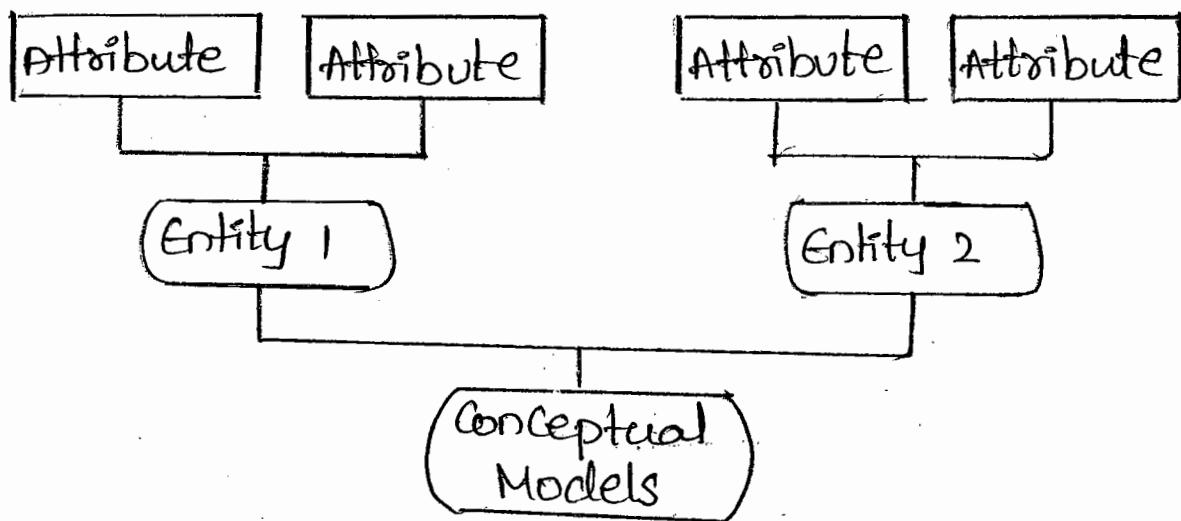
2) Bottom-up Design:

The steps performed in this strategy are as follows,

(a) Identify the data element

(b) Merge the data elements together to form a data set.

Basically, bottom-up design is complementary to top-down design as the former initially defines the attributes and then forms the entity, whereas, the latter initially defines the entity and then its attribute.



\* Centralization vs decentralized conceptual database design.

→ Database design works on two approaches

1. Top down approach
2. Bottom up approach

These two approaches can reflect an impact on the factors such as,

- (i) Company's structure i.e., centralized and decentralized.

- (ii) Company's management style.
- (iii) Size and scope of the system.

Thus, based on the above factors, database can be designed either using,

- (i) Centralized design
- (ii) Decentralized design

(i) Centralized design:-

Centralized design is adopted only when the number of objects and procedures are less. It can be carried out and represented in a simpler manner by a single DBA. A single DBA can efficiently manage the operations and scope of the problem. He/she can develop a conceptual design and verify it with the users. It can define system processes and data constraints in order to make sure that the design will meet the requirements of the end users and designers efficiently.

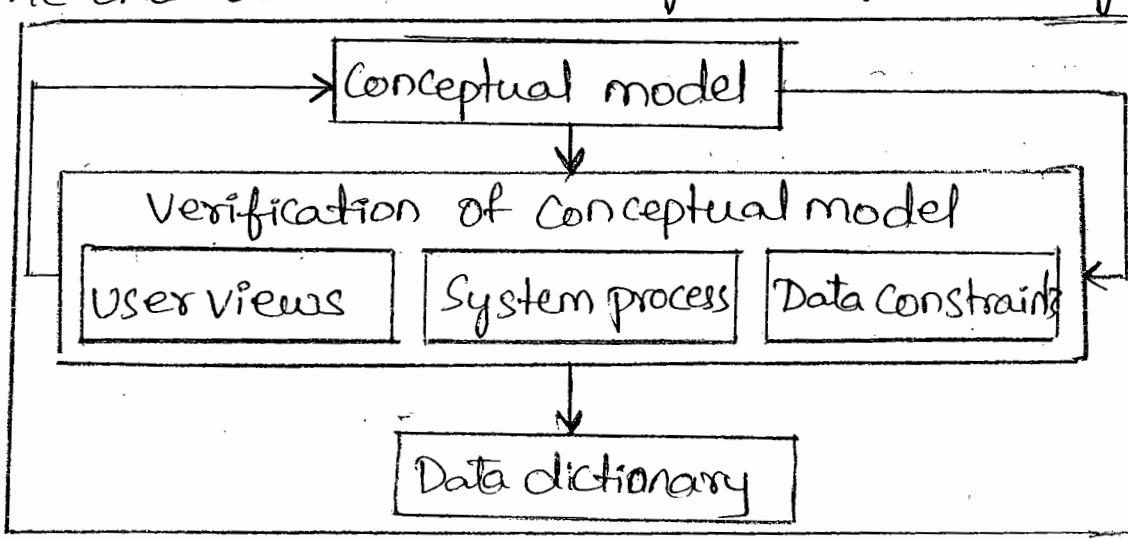


Fig :- Centralized Design.

(ii) Decentralized Design :-

Decentralized design is adopted when the number of data components of a system are more such as entities, relations complex operations. It is generally used when there are many operational sites wherein every element belongs to entire data set.

In decentralized design a group of members are selected in the design team in order to solve complex relations and operations of a complex database project. Decentralized design is divided into several module and each module is assigned to a design team leader. Each design group starts working on its models and creates a conceptual data model for the given subset/module. Thus each conceptual model is verified on individual basis based on the criteria such as user views, processes and constraints. As soon as the verification and validation phase is completed all the modules are combined together to a single conceptual model. Now it is responsibility of the

lead designer to verify whether the larger Conceptual model is capable of performing its operations properly or not.

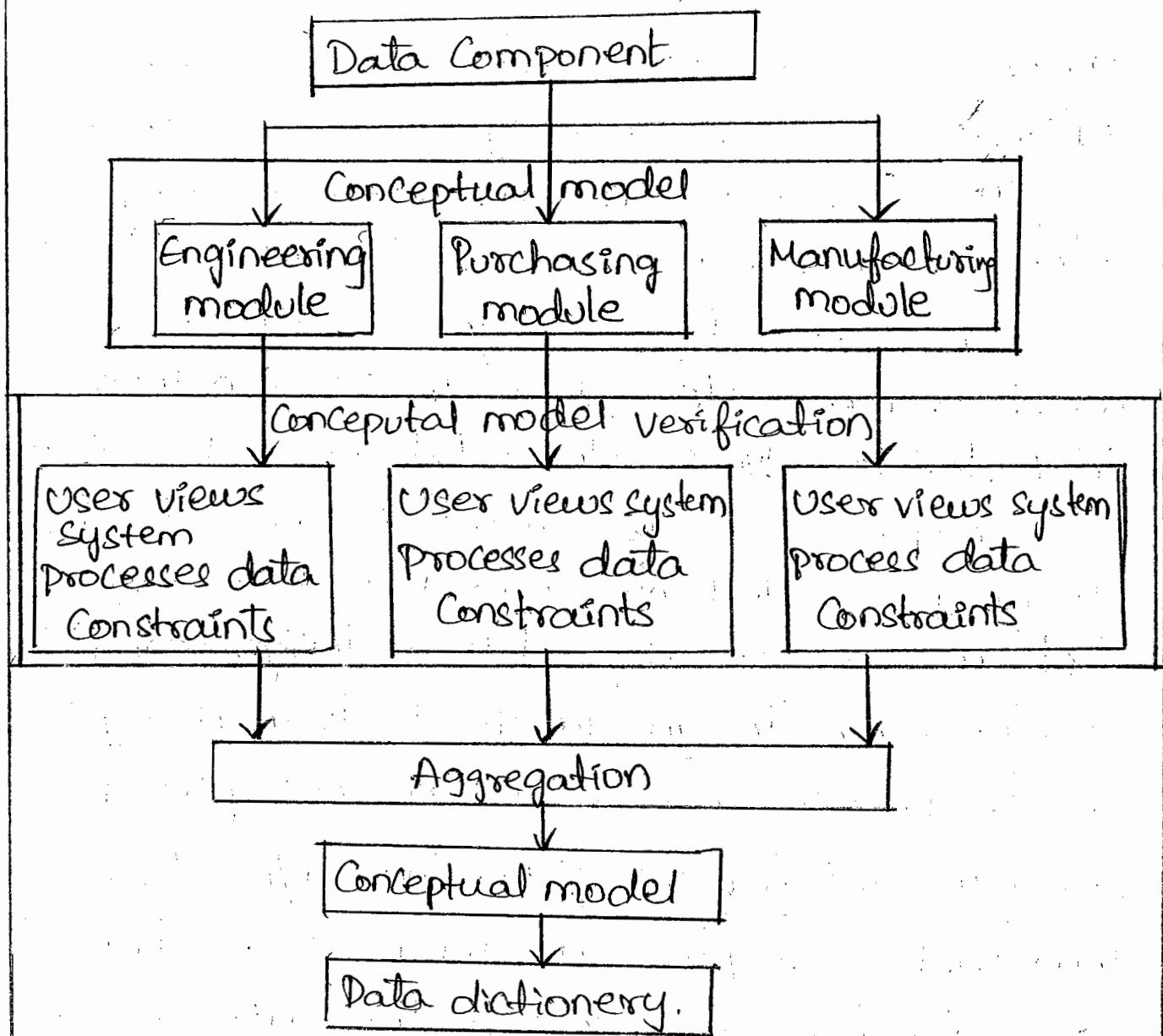


Figure:- Decentralized Design