ORACLE19C

INTRODUCTION OF DBMS:

DATA: IT IS A RAW FACT (I.E. CHARACTERS & NUMBERS)

EX: EMPID IS DATA, ENAME IS DATA, SALARY IS DATA, DOJ IS DATA……ETC.

NOTE: DATA IS NEVER GIVES ACCURATE MEANINGFULL INFORMATION.

INFORMATION: PROCESSING DATA IS CALLED AS INFORMATIONS.

EX:

EMPID	ENAME	SALARY	DOJ	DEPTNAME
======	======	======	====	=====
1021	X	25000.00	05-08-2020	DB
1022	Y	45000.00	24-12-2019	HR

NOTE: INFORMATION IS ALWAYS PROVIDES ACCURATE MEANINGFULL DATA OF PARTICULAR EMPLOYEE, CUSTOMER, STUDENT AND PRODUCT……. ETC.

DATA STORAGES: IT A LOCATION WHERE WE CAN STORE DATA / INFORMATION.WE HAS DIFFERENT TYPES OF DATA STORAGES.

1. BOOKS & PAPERS
2. FLAT FILE (FILE MANAGEMENT SYSTEM)
3. DBMS / DATABASE

1. DISADVANTAGES OF BOOKS & PAPERS:

> IT IS COMPLETE MANUAL PROCEE / SYSTEM.

> REQUIRED MORE MAN POWER.

> COSTLY IN MAINTANANCE

> THERE IS NO SECURITY

> HANDLING A VERY SMALL DATA

> RETRIEVING DATA WILL BE TIME CONSUME.

2. FLAT FILE (FILE MANAGEMENT SYSTEM):

IN FILE MANAGEMENT DATA CAN BE STORED IN FILES.

DISADVANTAGES:

1. **DATA REDUNDANCY & DATA INCONSISTANCY:**
   =====================================
   THESE PROBLEMS COMES INTO PICTURE WHEN WE STORE DATA IN MULTIPLE FILES.WHERE THE CHANGES ARE MADE IN ONE FILE WILL NOT BE REFELECTED TO ANOTHER COPY OF FILE .
   BUT IN CASE OF DATABASE WE CAN MAINTAINE NO.OF COPIES OF SAME DATA AND STILL THE CHANGES MADE IN ONE COPY THEN REFELECTED TO OTHER COPY BECAUSE INTERNALLY MAINTAIN ACID PROPERTIES BY DEFAULT IN DATABASE.

2. **DATAINTEGRITY :**
   ==============
   THIS IS ABOUT MAINTAING PROPER DATA IN EVERY ORAGANIZATION IMPOSE SET INTEGRITY RULES ON DATA AND WE WILL CALL THESE RULES ARE BUSINESS RULES.
   DATABASE PROVIDES AN OPTIONS FOR IMPOSING THE BUSINESS RULES WITH THE HELP OF CONSTRAINTS AND TRIGGERS.

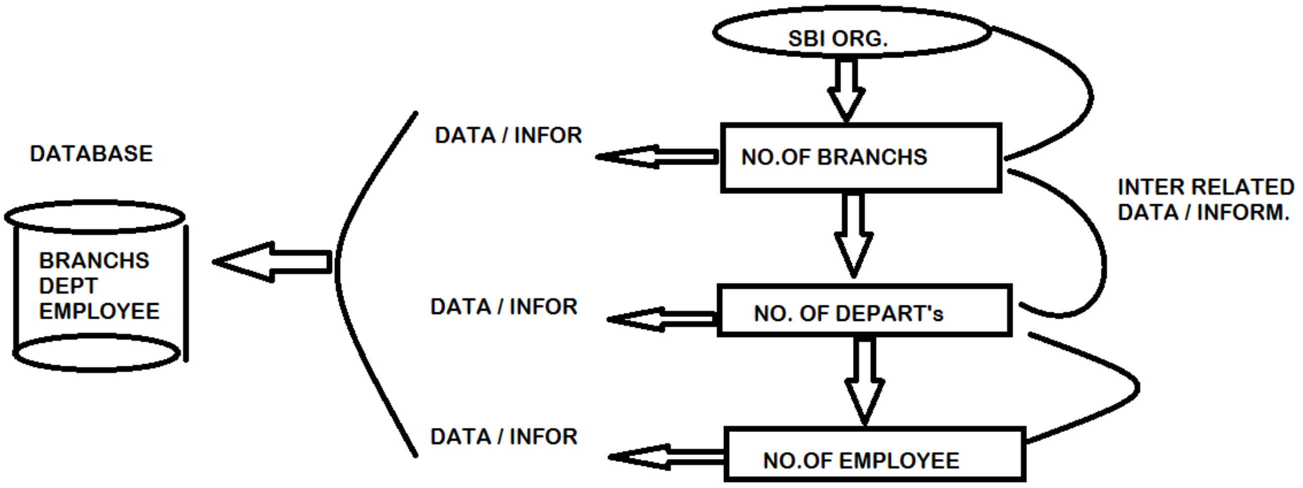
3. **DATARETRIEVE:**
   ==============
   IT IS PROCESS OF DATA RETRIEVING FROM DATA SOURCES.WHICH IS VERY COMPLEX WHILE RETRIEVING DATA FROM FILES WHICH WAS ADDRESSED WITH HIGH LEVEL LANGUAGE.
   WHERE AS IF YOU WANT TO RETRIEVE DATA FROM DATABASE THEN WE ARE USING SQL LANGUAGE.

4. **DATASECURITY:**
   ==============
   DATA IS NEVER SECURE UNDER BOOKS AND FLAT FILE WHERE AS DATABASE ARE PROVIDING AN EXCELLENT CONCEPT IS CALLED AS ROLE BASED SECURITY MECHANISM FOR ACCESSING DATA FROM DATABASE WITH SECURITY MANNER WITH THE HELP OF AUTHENTICATION AND AUTHORIZATION.

5. **DATAINDEXING:**
   ==============
   INDEXES ARE USING FOR ACCESSING DATA MUCH MORE FASTER BUT FLAT FILES DOES NOT PROVIDE ANY INDEX MECHANISM WHERE AS DATABASE WILL PROVIDE INDEXING MECHANISM.

3. DBMS / DATABASE:

DATABASE: **IT IS COLLECTION OF INTER RELATED INFORMATION.BY USING DATABASE WE CAN STORE, MODIFY, SELECT AND DELETE DATA FROM DATABASE WITH SECURITY MANNER.**



<u>TYPES DATABASES:</u>

1) OLTP (ONLINE TRANSACTION PROCESSING)

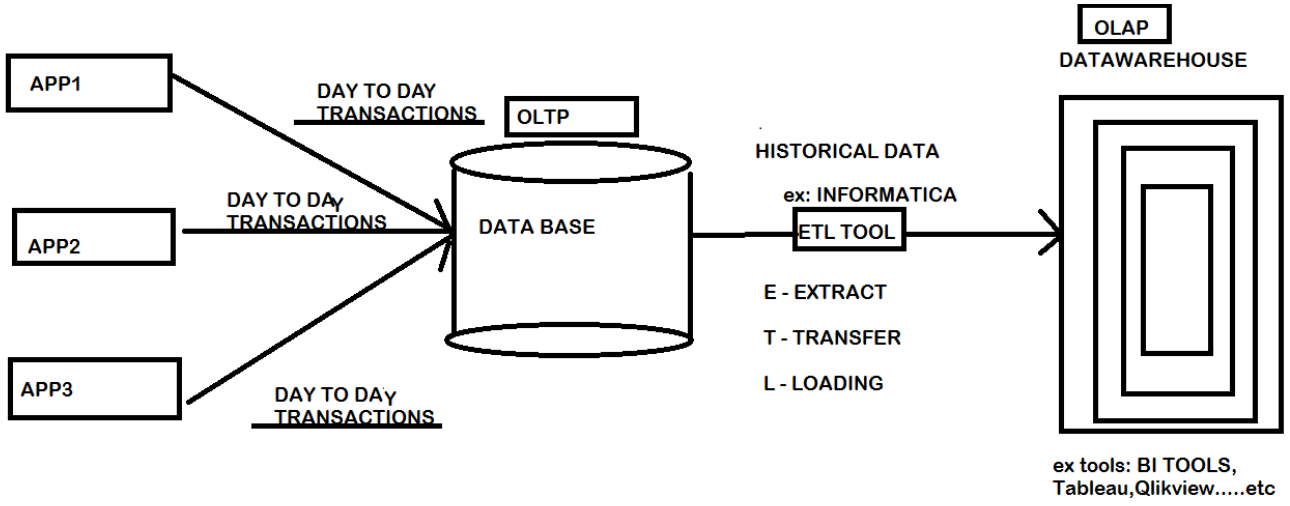
2) OLAP (ONLINE ANALYTICAL PROCESSING)

<u>OLTP:</u>ORGANIZATIONS ARE MAINTAINE OLTP FOR STORING "DAY - TO - DAY TRANSACTIONS INFROMATION ". USING FOR "RUNNING BUSINESS ".

EX: SQLSERVER, ORACLE, MYSQL,.............ETC.

<u>OLAP:</u> USED FOR DATA ANALYSIS (OR) DATA SUMMERIZED (OR) HISTORY OF DATA OF PARTICULAR BUSSINESS.

EX: DATAWAREHOUSE.



<u>DBMS:</u> IT IS A SOFTWARE.WHICH IS USED TO MANAGE DATA IN DATABASE.

<u>WHY DBMS:</u>

EX:

```
                         BUSSINESS ----> (COLLECTION OF ENTITIES)
                                    |
     BRANCH     | EMPLOYEE |PRODUCTS | CUSTOMERS ----> (ENTITIES)
                                    | < STARTING >
      1 BRAN    |  10 EMP       |   10 PRO     |  NO CUST. -→(GEN. A VERY SMALL DATA)
                                    | < AFTER 5 YEARS >
     25 BRAN    | 500 EMP   | 25 PRO      | 50000 CUST. ---->(GEN. SMALL DATA)
                                    | < AFTER 10 YEARS >
     100 BRAN | 10000 EMP   | 100 PRO     | 5 LAK'S CUST. ------>(GEN. BIG/LARGE DATA)
```

<u>ADVANTAGES OF DBMS:</u>

1. TO REDUCE DATA REDUNDANCY.

2. TO AVOID DATAINCONSISTANCY.

3. EASY TO ACCESS DATA.

4. EASY TO MANIPULATE DATA.

5. MORE SECURITY (AUTHENTICATION & AUTHORIZATION)

6. IT SUPPORTS DATAINTEGRITY RULES.

7. SUPPORTING DATA SHARING

8. SUPPORTS TRANSACTIONS AND "ACID" PROPERTIES.

<u>DBMS MODELS / DATABASE MODELS:</u>

HOW DATA CAN BE ORGANIZED / STORE IN DIFFERENT DATABASE MODELS.THER ARE THREE TYPES OF DATABASE MODELS ARE,

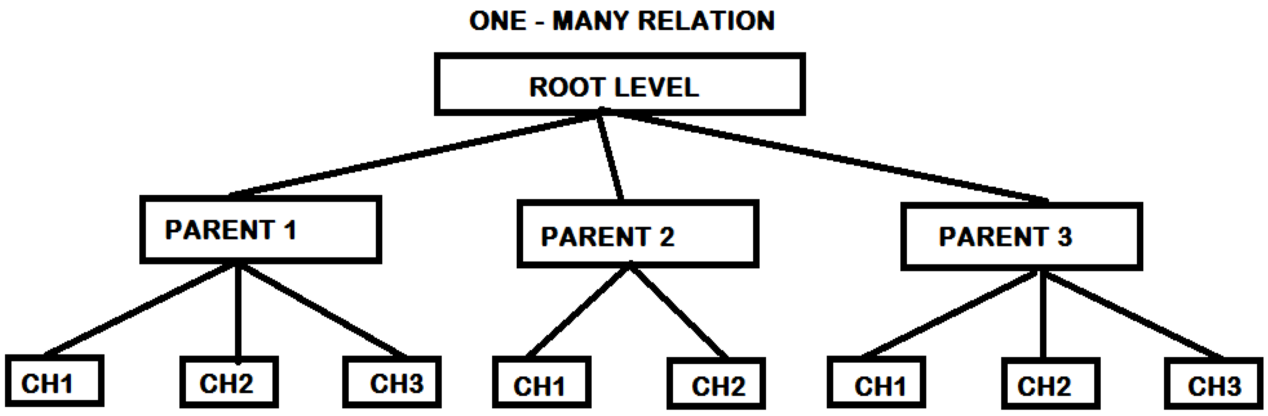
1. **HIERARCHICAL DATABASE MANAGEMENT SYSTEM(HDBMS)**
2. **NETWORK DATABASE MANAGEMENT SYSTEM(NDBMS)**
3. **RELATIONAL DATABASE MANAGEMENT SYSTEM(RDBMS)**
    - **i) OBJECT RELATIONAL DBMS(ORDBMS)**
    - **ii) OBJECT ORIENTED RELATIONAL DBMS(OORDBMS)**

HDBMS:

IT IS A FIRST MODEL OF DATABASE THAT CAME INTO EXISTANCE IN THE 1960'S WHICH WILL ORGANIZE THE DATA IN THE FORM OF A "TREE STRUCTURE " AND WHICH WAS DESIGN BASED ON "ONE – MANY RELATION"

IN ONE – MANY RELATION EVERY CHILD IS HAVING ONLY ONE PARENT.THIS TREE IS CONTAINS THE FOLLOWING THREE LEVEL ROOT,PARENT AND CHILD LEVEL.

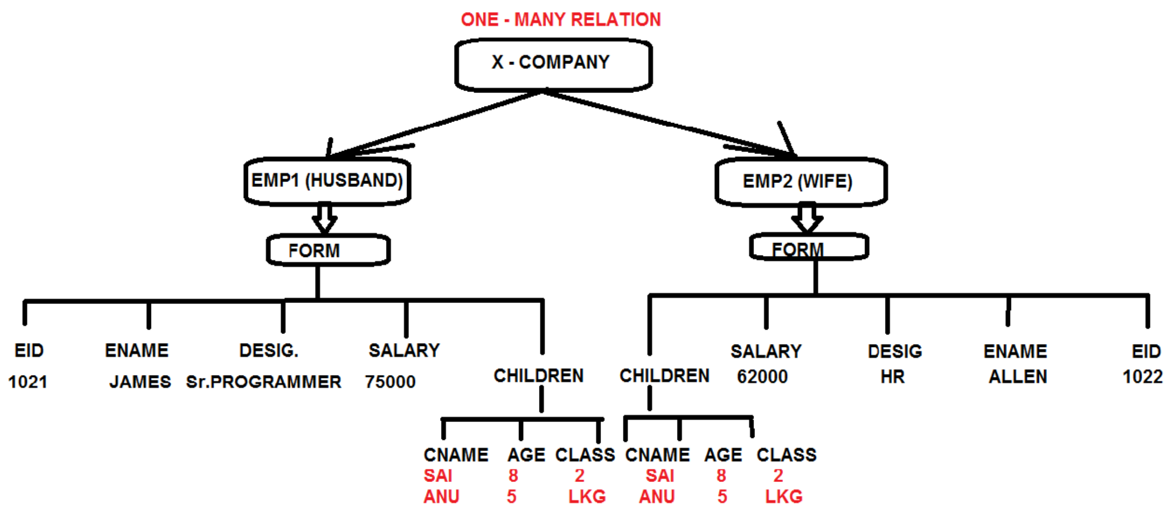
EX: IMS SOFTWARE (INFORMATION MANAGEMENT SYSTEM)



DISADVANTAGES:

1. WHEN WE WANT TO ADD A NEW LEVEL (PARENT / CHILD) TO AN EXISTING STRCTURE THEN USER HAS TO RE CONSTRCT THE ENTIRE STRCTURE SO THAT IT LEADS TIME CONSUME.
2. WHEN WE WANT TO ACCESS DATA FROM THIS MODEL THEN WE NEED TO TRAVEL FROM ROOT LEVEL TO CHILD LEVEL WHICH WILL TIME TAKEN PROCESS.
3. THIS MODEL DESIGNED BASED ON ONE – MANY RELATION I.E EVER CHILD IS HAVING ONLY ONE PARENT SO THAT THERE IS A CHANCES TO OCCURE DATA DUPLICATE.

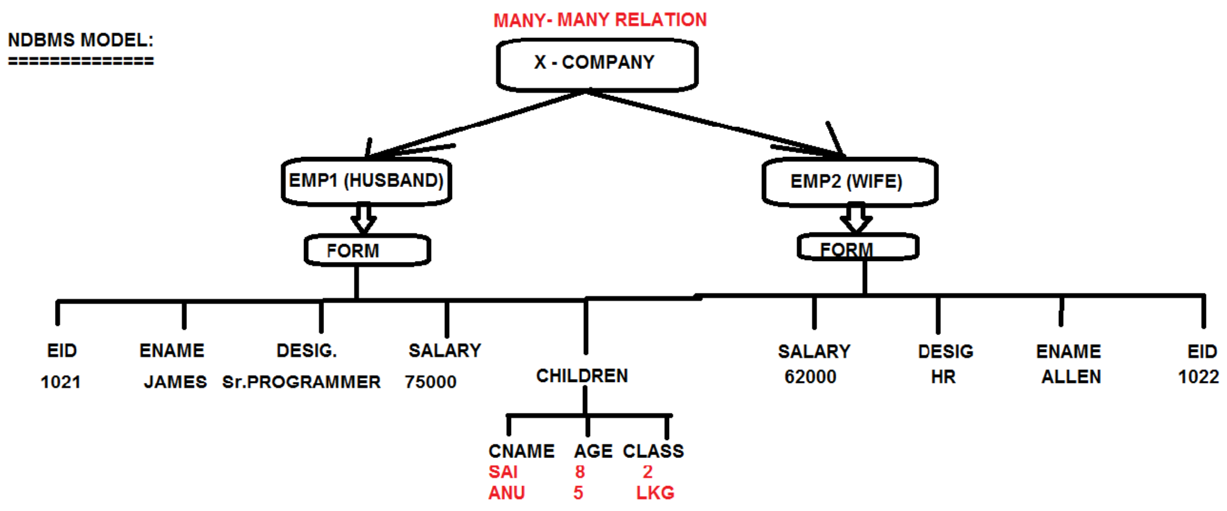
EX:



NDBMS:

THIS MODEL IS A MODIFICATION OF AN EXISTING HIERARCHICAL MODEL BRINGING "MANY – TO – MANY "RELATIONSO THAT A CHILD CAN HAVE MORE THAN ONE PARENT WHICH WILL REDUCE DUPLICATE DATA.IN 1969 THE FIRST NDBMS SOFTWARE LAUNCHED WITH THE NAME AS "IDBMS" (INTEGRATED DATABASE MANAGEMENT SYSTEM).

EX:



ADVANTAGES OF NDBMS:

1. **TO REDUCE DUPLICATE DATA BECAUSE SUPPORTING MANY – MANY RELATION (A CHILD CAN HAVE MULTIPLE PARENTS)**
2. **BY USING POINTERS MECHANISM WE CAN ADD NEW LEVEL (PARENT / CHILD) TO AN EXISTING STRUCTURE WITHOUT RECONSTRUCTION.**
3. **ACCESSING DATA IN THIS MODEL IS VAERY FAST BECAUSE IT USES POINTERS.**

DISADVANTAGES OF NDBMS:

1. **WHEN WE USE NUMBER OF POINTERS IN AN APPLICATION THEN IT WILL INCRESE COMPLEXITY(DIFFICULT) TO IDENTIFYING WHICH POINTER IS BELONGS TO WHICH PARENT OR WHICH CHILD AND ALSO DEGRADE PERFORMANCE.**
2. **NDBMS MODEL WAS NOT MORE SUCCESSFUL MODEL IN REAL TIME BECAUSE IMMEDIATE TAKE OVER BY RDBMS MODEL IN 1970'S WITH EFFECTIVE FEATURES.**

RDBMS:

IN HDBMS AND NDBMS DATA IS ORGANIZED IN THE FORM OF A TREE STRUCTURE WHICH IS LOOKS COMPLEX TO MANAGE AND UNDERSTAND ALSO SO TO OVERCOME THIS PROBLEM IN 1970'S MR.E.F.CODD FROM IBM CAME WITH A NEW CONCEPT ON STORING DATA IN A TABLE STRUCTURE I.E. ROWS AND COLUMNS FORMAT.

E.F.CODD WITH ALL THESE IDEAS FOR THE NEW MODEL CALLED AS "RELATIONAL MODEL" HAS PUBLISHED AN ARTICLE WITH THE TITLE AS "A RELATIONAL MODEL OF DATA FOR LARGE SHARED DATA BANK".

BASING ON THIS ABOVE ARTICLE MANY COMPANIES CAME FORWARD LIKE IBM, RELATIONAL SOFTWARE INC (PRESENT IT IS ORACLE COR.)......ETC. HAS STARTED THE DESIGNED FOR THE NEW DATABASE MODEL I.E. RDBMS.

RDBMS IS MAINLY BASED ON TWO MATHEMATICAL PRINCIPLES ARE "RELATIONAL ALGEBRA" AND "CALCULATIONS".IN THE YEAR 1970'S IBM HAS GIVEN THE PROTOTYPE FOR RDBMS KNOWN AS "SYSTEM R".

IN THE YEAR 1974 IBM HAS LAUNCHED A LANGUAGE FOR COMMUNICATION WITH RDBMS KNOWN AS "SEQUEL" AND LATER CHANGED AS "SQL".

FEATURES OF RDBMS:

- DATA CAN BE ORGANIZED IN TABLE FORMAT.

- MORE SECURITY WITH THE HELP OF "AUTHENTICATION &AUTHORIZATION".

- REDUCE DATAREDUNDANCY & DATAINCONSISTANCY USING NORMALIZATION.

- EASY TO MANIPULATION DATA USING DML COMMANDS.

- EASY TO ACCESS DATA FROM DB WITH THE HELP OF "SQL QUERY (SELECT)".

- FASTLY RETRIEVE DATA USING "INDEXES ".

- DATA SHARING USING "VIEWS ".

- SUPPORTING TRANSACTIONS WITH "ACIDPROPERTIES".

- SUPPORTING DATATYPES, OPERATORS, FUNCTIONS / PROCEDURE, CLAUSES .ETC

- SUPPORTING ALL RELATIONSHIPS THOSE ARE "ONE – ONE","ONE – MANY /MANY – ONE" AND "MANY-MANY".

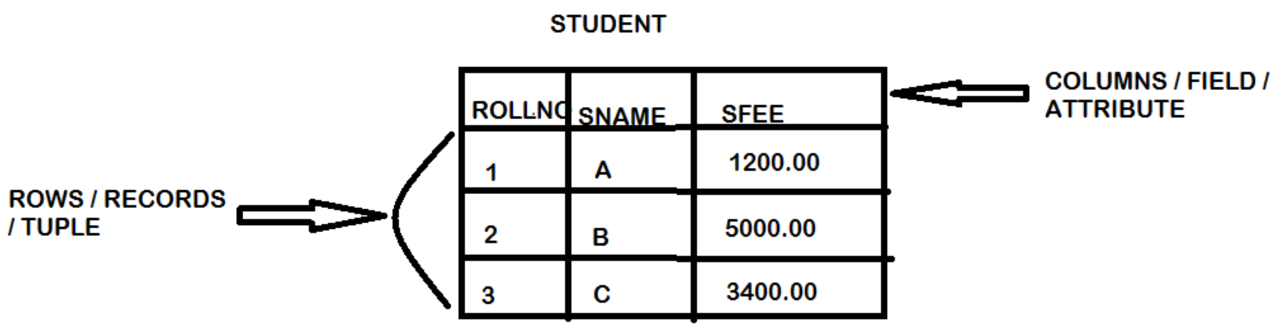
- SUPPORTING DATAINTEGRITY RULES WITH CONSTRAINTS & TRIGGERS

- SUPPORTING SQL & PL/SQL LANGUAGES.

<u>EX.OF AN RDBMS PRODUCTS:</u>

ORACLE, SQL SERVER, MYSQL, DB2, SYBASE, INFORMIX, INGREES, TERADATA,

MAXDB, POSTGRESQL ...ETC



ROLLNO	SNAME	SFEE
1	A	1200.00
2	B	5000.00
3	C	3400.00

DATABASE : COLLECTION OF TABLES

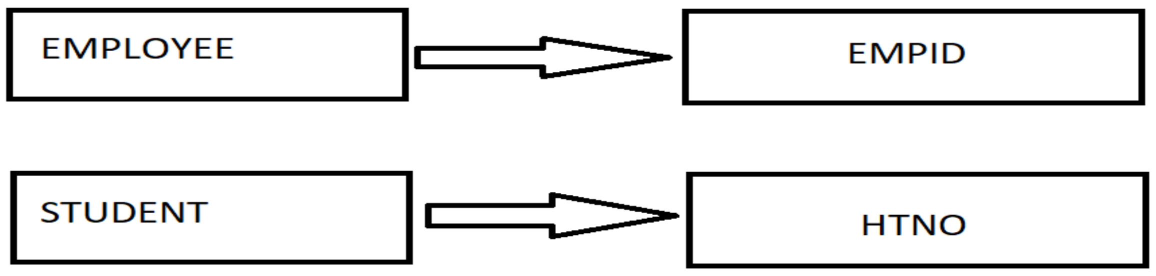
TABLE : COLLECTION OF ROWS AND COLUMNS

ROW : COLLECTION OF COLUMNS

- **HERE RELATION CAN BE DEFINED AS COMMONNESS BETWEEN OBJECTS THESE RELATIONS ARE CLASSIFIED INTO 3 TYPES**
  - **ONE TO ONE RELATION**
  - **ONE TO MANY RELATION / MANY TO ONE RELATION**
  - **MANY TO MANY RELATION**

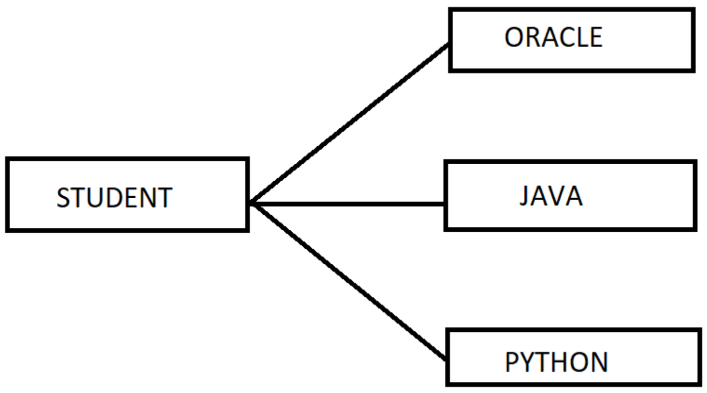
<u>ONE – ONE RELATIONSHIP:</u>

- **IN THIS RELATIONSHIP ONE OBJECT CAN HAVE A RELATIONSHIP WITH ANOTHER OBJECT**



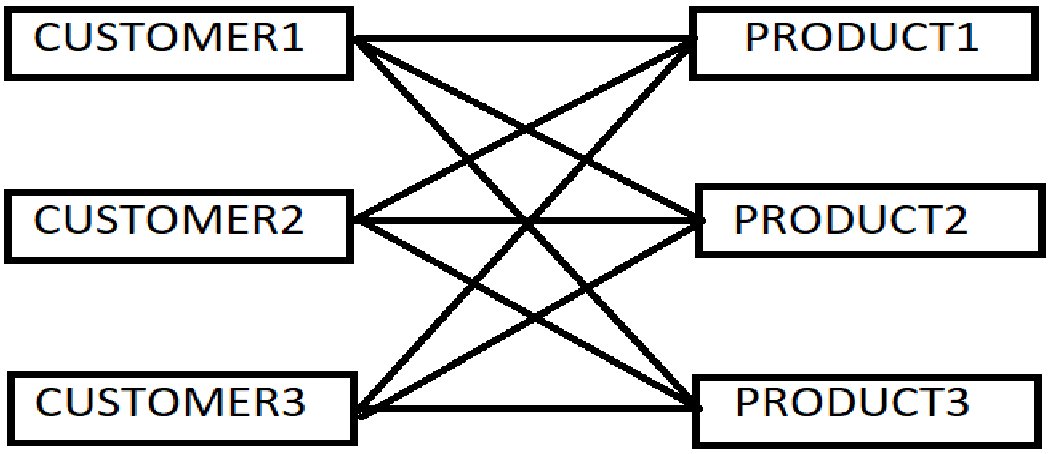
ONE – MANY / MANY – ONERELATIONSHIP:

- **IN THIS RELATIONSHIP ONE OBJECT CAN HAVE A RELATIONSHIP WITH MANY OBJECTS**



MANY – MANY RELATIONSHIP:

- **IN THIS RELATIONSHIP MANY VENDORS (OR) MANY OBJECTS CAN HAVE THE RELATIONSHIP WITH MANY OTHER OBJECTS**



INTRODUCTION TO ORACLE:

ORACLE IS A RELATIONAL DATABASE AN RDBMS PRODUCT FROM ORACLE CORPORATION IN 1979.WHICH IS USED TO STORE DATA (OR) INFORMATION PERMANANTLY I.E., IN HARD DISK ALONG WITH SECURITY.

ORACLE IS A PLATFORM INDEPENDENT AN RDBMS PRODUCT.IT MEANS THAT IT CAN DEPLOYEE (INSTALL) IN ANY OS LIKE WINDOWS, LINUX, UNIX, SOLARIES, MAC…. ETC.

PLATFORM:

- IT A COMBINATION OF OPERATING SYSTEM AND MICRO PROCESSOR.THESE ARE AGAIN CLASSIFIED INTO TWO TYPES.

1) PLATFORM INDEPENDENT:

- IT SUPPORTS ANY OS WITH THE COMBINATION OF ANY MICRO PROCESSOR.

EX: ORACLE, MYSQL, JAVA, .NET…. ETC.

2) PLATFORM DEPENDENT:

- IT SUPPORTS ONLY ONE OS WITH COMBINATION OF ANY MICRO PROCESSOR.

EX: C - LANGUAGE.

VERSIONS OF ORACLE:

YEAR	VERSION	FEATURES
1979	ORACLE 1.0	NOT PUBLIC RELEASED
1980	ORACLE 2.0	FIRST PUBLIC RELEASED, BASIC SQL FUNCTIONALITIES.
1982	ORACLE 3.0	FIRST PORTABLE DB.
1984	ORACLE 4.0	INTRODUCED READ CONSISTENCY.
1986	ORACLE 5.0	INTRODUCED CLIENT-SERVER ARCHITECTURE.
1988	ORACLE 6.0	INTRODUCED PL/SQL
1992	ORACLE 7.0	INTEGRITY CONSTRAINTS INTRODUCED, VARCHAR DATA TYPE CHANGED INTO VARCHAR2,STOREDPROCEDURES,

		FUNCTIONS AND TRIGGERS
1997	ORACLE 8.0	OBJECT ORIENTED FEATURES, TABLEPARTITIONING, INSTEAD TRIGGERS
1998	ORACLE 8I(INTERNET)	ROLLUP, CUBEMETHODS, COLUMNS INCREASED PER A TABLE UP TO 1000
2001	ORACLE 9I	RENAMING COLUMN,ANSI JOINS
2004	ORACLE 10G(GRID TECHNOLOGIES)	INTRODUCED ADMIN SIDE OPERATIONS, FLASHBACKQUERY, INDICATE OF CLAUSES, REGULAR EXPRESSIONS
2007	ORACLE 11G	READ ONLY TABLES, VIRTUALTABLES, INTEGER DATA TYPE, USINGSEQUENCE, ENABLES AND DISABLES TRIGGERS.
2013	ORACLE12C (CLOUD TECHNOLOGY)	TRUNCATE TABLE CASCADE, MULTIPLEINDEXES,INVISIABLECOLUMN, SEQUENCESESSION, NEW AUTO INCREMENT BY USING IDENTITY.
2018	ORACLE18C	POLYMORPHIC TABLE FUNCTIONS, ACTIVE DIRECTORY INTEGRATION
2019	ORACLE19C	ACTIVE DATA GUARD DML REDIRECTION, AUTOMATIC INDEX CREATION,SQL QUERIES ON OBJECT STORES.

<u>WORKING WITH ORACLE:</u>

WHEN WE INSTALL ORACLE SOFTWARE INTERNALLY TWO COMPONENTS ARE INSTALLED.THOSE ARE,

1.ORACLE CLIENT

2. ORACLE SERVER

1. ORACLE CLIENT:

BY USING ORACLE CLIENT TOOL USER CAN PERFORM THE FOLLOWING THREE OPERATIONS ARE

- **USER CAN CONNECT TO ORACLE SERVER**
- **USER CAN SEND REQUEST TO ORACLE SERVER**
- **USER CAN RECEIVE RESPONSE FROM ORACLE SERVER.**

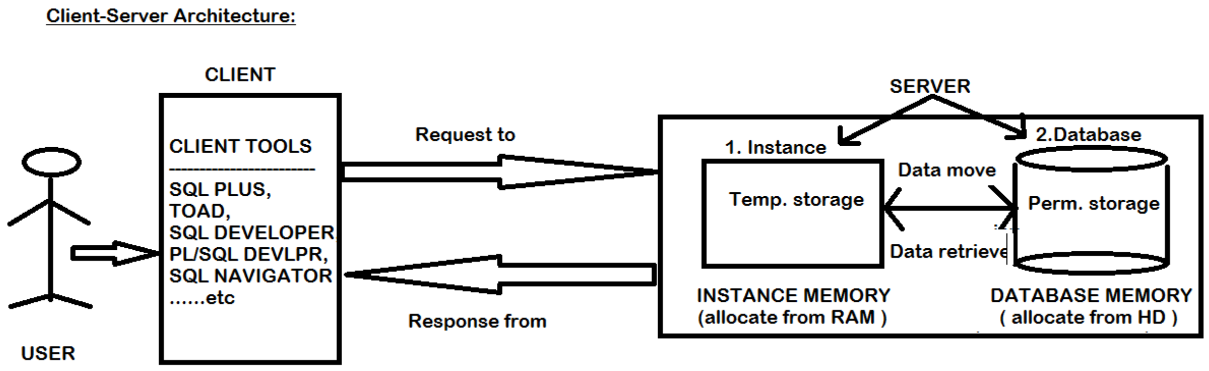
EX: SQLPLUS, TOAD, SQL DEVELOPER, SQL NAVIGATOR…………………ETC.

2. ORACLE SERVER:

ORACLE SERVER MANAGE TWO MORE SUB COMPONENTS INTERNALLY THOSE ARE,

- **INSTANCE**
- **DATABASE**

INSTANCE WILL ACT AS TEMPORARY MEMORY WHICH WILL ALLOCATE FROM RAM AND STORED DATA / INFORMATION TEMPORARY WHERE AS DATABASE IS A PERMANENT MEMORY WHICH WILL ALLOCATE FROM HARDDISK AND STORED DATA PERMANENTLY.



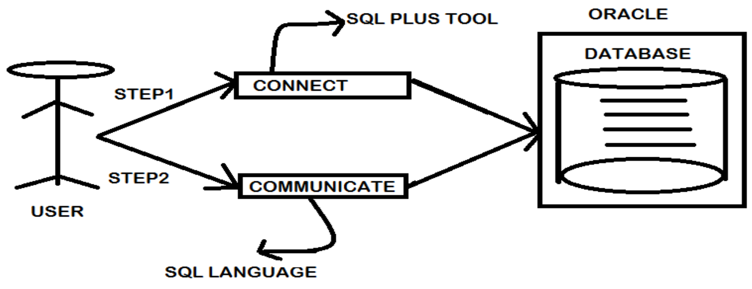
NOTE: WHEN WE WANT TO WORK ON ORACLE DATABASE THEN WE FOLLOW THE FOLLOWING TWO STEPS PROCEDURE

1) **CONNECT TO ORACLE:**

   IF USER WANTS TO CONNECT TO ORACLE THEN WE REQUIRED A DATABASE TOOL IS CALLED AS "SQLPLUS" WHICH WAS INBUILTED IN ORALCE SOFTWARE.

2) **COMMUNICATE WITH DATABASE:**

IF USER WANTS TO COMMUNICATE WITH DATABASE THEN WE NEED A DATABASE COMMUNICATION LANGUAGE IS CALLED AS "SQL".



HOW TO CONNECT TO ORACLE:

BEFORE CONNECT TO ORACLE DATABASE WE NEED TO KNOW THE TYPES OF EDITIONS IN ORACLE SOFTWARE.EVERY ORACLE SOFTWARE IS HAVING TWO TYPES OF EDITIONS THOSE ARE

1) **ORACLE EXPRESS EDITION (PARTIAL SUPPORTING FEATURES)**
2) **ORACLE ENTERPRISE EDITION (FULLY SUPPORTING FEATURES)**

THE ABOVE TWO EDITIONS ARE HAVING DEFAULT USERNAME IS "SYSTEM" AND PASSWORD IS CREATED AT INSTALLATION OF ORACLE SOFTWARE.

STEPS TO CONNECT TO ORACLE:

> GO TO ALL PROGRAMS

> GO TO ORACLE19C_HOME1 FOLDER

> CLICK ON SQL PLUSE ICON

> ENTER USERNAME: SYSTEM

>ENTER PASSWORD: MANAGER (AT INSTALLATION TIME PASSWORD)

TO CREATE A NEW USERNAME & PASSWORD IN ORACLE DB:

SYNTAX: CREATE USER <USERNAME> IDENTIFIED BY <PASSWORD>;

EX: CREATE USER SUDHAKAR IDENTIFIED BY SUDHAKAR;

NOTE: USER IS CREATED BUT THIS USER IS DUMMY USER BECAUSE IS NOT HAVING PERMISSION TO CONNECT AND CREATE NEW TABLE IN DB.SO PERMISSIONS MUST BE GIVEN TO USER(SUDHAKAR) BY USING "GRANT" COMMAND BY DBA(SYSTEM).

NOTE: EVERY USER IN ORACLE SERVER IS CALLED AS "SCHEMA".

GRANTING PERMISSIONS TO USER:

STEP1: **USERNAME: SYSTEM**

PASSWORD: MANAGER

CONNECTED.

STEP2:

GRANT CONNECT, CREATE TABLE TO SUDHAKAR;

HERE,

CONNECT ------- TO CONNECT TO ORACLE DB

CREATE TABLE ------- TO CREATE NEW TABLES IN DB.

NOTE: WHEN WE CONNECT TO ORACLE DB SOME TIMES, WE WILL FACE A PROBLEM IS,

ERROR: ORA-28000: THE ACCOUNT IS LOCKED.

TO OVERCOME THE ABOVE ERROR THEN WE FOLLOW THE FOLLOWING STEPS ARE

SOLUTION:

STEP1: CONNECT TO ORACLE WITH SYSTEM DATABASE ADMIN:

SYNTAX:

ENTER USERNAME: SYSTEM

ENTER PASSWORD: MANAGER

CONNECTED.

STEP2: TO UNLOCK USER:

SYNTAX:

```
SQL> ALTER USER <USER NAME> ACCOUNT UNLOCK / LOCK;
```

EX:

```
SQL> ALTER USER SUDHAKAR ACCOUNT UNLOCK;
```

STEP3: NOW CONNECT TO ORACLE WITH EITHER SYSTEM (OR) SUDHAKAR USER:

```
ENTER USERNAME: SUDHAKAR

ENTER PASSWORD: SUDHAKAR

CONNECTED.
```

HOW TO CHANGE A PASSWORD:

```
SQL> PASSWORD

CHANGING PASSWORD FOR SUDHAKAR

OLD PASSWORD: SUDHAKAR

NEW PASSWORD:123

RETYPE NEW PASSWORD:123

PASSWORD CHANGED

SQL> CONN

ENTER USER-NAME: SUDHAKAR / SUDHAKAR

ERROR: ORA-01017: INVALID USERNAME/PASSWORD; LOGON DENIED

WARNING: YOU ARE NO LONGER CONNECTED TO ORACLE.

SQL> CONN

ENTER USER-NAME: SUDHAKAR / 123

CONNECTED.
```

HOW TO CREATE A NEW PASSWORD IF WE FORGOT A PASSWORD:

SYNTAX:

```
ALTER USER <USER NAME> IDENTIFIED BY

<NEW PASSWORD>;
```

EX:

```
ENTER USER-NAME: SYSTEM / MANAGER

CONNECTED.

SQL> ALTER USER SUDHAKAR IDENTIFIED BY SUDHAKAR;

USER ALTERED.

SQL> CONN

ENTER USER-NAME: SUDHAKAR / 123

ERROR:

ORA-01017: INVALID USERNAME/PASSWORD; LOGON DENIED

WARNING: YOU ARE NO LONGER CONNECTED TO ORACLE.

SQL> CONN

ENTER USER-NAME: SUDHAKAR / SUDHAKAR

CONNECTED.
```

NOTE: **WHEN WE WANT TO CONNECT TO ORACLE DB SERVER SOME TIMES, WE FACED ANOTHER PROBLEM IS CALLED AS "TNS PROTOCAL ADAPTER ERROR".**

```
ENTER USER-NAME: SUDHAKAR / SUDHAKAR

ERROR:

ORA-12560: TNS:PROTOCOL ADAPTER ERROR

ENTER USER-NAME: SYSTEM / MANAGER

ERROR:

ORA-12560: TNS:PROTOCOL ADAPTER ERROR
```

NOTE: TO OVERCOME THE ABOVE PROBLEM THEN WE FOLLOW THE FOLLOWING STEPS ARE,

STEP1: GO TO SERVICES

STEP2: GO TO ORACLESERVICEORCL AND CLICK ON IT

STEP3: SELECT STARTUP TYPE IS AUTOMATIC

STEP4: CLICK ON START BUTTON

STEP5: CLICK ON OK

ENTER USER-NAME: SYSTEM / MANAGER

CONNECTED TO:ORACLE DATABASE 19C ENTERPRISE EDITION RELEASE 19.0.0.0.0 - PRODUCTION

VERSION 19.3.0.0.0

SQL> CONN

ENTER USER-NAME: SUDHAKAR / SUDHAKAR

CONNECTED.

STRUCTURE QUERY LANGUAGE(SQL):SQL IS A DATABASE LANGUAGE WHICH WAS USED TO COMMUNICATE WITH DATABASE.INTRODUCED BY "IBM" AND INITIAL NAME OF THIS LANGUAGE WAS "SEQUEL" AND LATER RENAMED WITH "SQL".

"SQL" IS ALSO CALLED AS "CLI"(COMMON LANGUAGE INTERFACE) BECAUSE THIS IS THE LANGUAGEWHICH IS USED TO COMMUNICATE WITH ANY RDBMS PRODUCTS SUCH AS ORACLE, SQLSERVER, MYSQL, DB2…………ETC.

SQL PRE-DEFINE QUERIES ARE NOT A CASE - SENSITIVE (WRITE QUERIES IN EITHER UPPER & LOWER-CASE CHARACTERS) BUT EVERY SQL QUERY SHOULD ENDS WITH "; ".

SUB - LANGUAGES OF SQL:

1) DDL (DATA DEFINITION LANGUAGE):

>CREATE, ALTER, RENAME, TRUNCATE, DROP

>RECYCLEBIN, FLASHBACK, PURGE (LATEST FEATURES)

2) DML (DATA MANIPULATION LANGUAGE):

> INSERT, UPDATE, DELETE

3) DQL / DRL(DATA QUERY / DATA RETRIVE LANGUAGE):

> SELECT

4) TCL (TRANSACTION CONTROL LANGUAGE):

> COMMIT, ROLLBACK, SAVEPOINT

5) DCL (DATA CONTROLL LANGUAGE):

> GRANT, REVOKE

1) DDL(DATA DEFINITION LANGUAGE):

> CREATE

> ALTER

> RENAME

> TRUNCATE

> DROP

1. CREATE: CREATE A NEW TABLE IN ORACLE DB.

SYNTAX:

CREATE TABLE <TABLE NAME> (<COLUMN NAME1><DATATYPE>[SIZE], <COLUMN NAME2><DATATYPE>[SIZE], ………………………………………………………………………………);

EX:

CREATE TABLE STUDENT(STID INT, SNAMECHAR (10), SFEENUMBER (6,2));

TO VIEW THE STRUCTURE OF A TABLE IN ORACLE DB:

SYNTAX:

SQL> DESC <TABLE NAME>;

EX:

SQL> DESC STUDENT;

2.ALTER:

IT IS USED TO MODIFY THE STRUCTURE OF A TABLE IN DATABASE.THIS COMMAND IS HAVING THE FOLLOWING FOUR SUB COMMANDS ARE

I) ALTER - MODIFY: TO CHANGE DATATYPE AND ALSO SIZE OF DATATYPE OF A PARTICULAR COLUMN.

SYNTAX:

ALTER TABLE <TN> MODIFY <COLUMN NAME><NEW DATATYPE>[NEW SIZE];

EX:

SQL> ALTER TABLE STUDENT MODIFY SNAME VARCHAR2(20);

<u>II) ALTER - ADD:</u> ADDING A NEW COLUMN TO AN EXISTING TABLE.

<u>SYNTAX:</u>

ALTER TABLE <TN> ADD <NEW COLUMN NAME><DATATYPE>[SIZE];

EX:

SQL> ALTER TABLE STUDENT ADD SADDRESS VARCHAR2(30);

<u>III) ALTER - RENAME:</u> TO CHANGE A COLUMN NAME IN A TABLE.

<u>SYNTAX:</u>

ALTER TABLE <TN> RENAME <COLUMN><OLD COLUMN NAME> TO <NEW COLUMN NAME>;

EX:

SQL> ALTER TABLE STUDENT RENAME COLUMN SNAME TO STUDENTNAMES;

<u>IV) ALTER - DROP:</u> TO DROP AN EXISTING COLUMN FROM A TABLE.

<u>SYNTAX:</u>

ALTER TABLE <TN> DROP <COLUMN><COLUMN NAME>;

EX:

SQL> ALTER TABLE STUDENT DROP COLUMN SFEE;

<u>3. RENAME:</u> IT IS USED TO CHANGE A TABLE NAME IN DATABASE.

<u>SYNTAX:</u>

RENAME <OLD TABLE NAME> TO <NEW TABLE NAME>;

EX:

SQL> RENAME STUDENT TO STUDENTDETAILS;

<u>4. TRUNCATE:</u> TO DELETE ALL ROWS FROM A TABLE AT A TIME. BY USING TRUNCATE COMMAND, WE CANNOT DELETE A SPECIFIC ROW FROM A TABLE BECAUSE TRUNCATE DOES NOT SUPPORTS "WHERE CLAUSE" CONDITION. IS DELETING ROWS BUT NOT COLUMNS OF A TABLE.

<u>SYNTAX:</u>

TRUNCATE TABLE <TABLE NAME>;

EX:

```
SQL> TRUNCATE TABLE STUDENT;
```

<u>5.DROP:</u> TO DROP A TABLE (I.E., ROWS AND COLUMNS) FROM DATABASE.

<u>SYNTAX:</u>

```
DROP TABLE <TABLE NAME>;
```

EX:

```
SQL> DROP TABLE STUDENT;
```

<u>NOTE:</u> FROM ORACLE 10G ENTERPRISE EDITION ONCE WE DROP A TABLE FROM DATABASE THEN IT WILL DROP TEMPORARILY. AND USER HAS A CHANCE TO RESTORE DROPPED TABLE AGAIN INTO DATABASE BY USING THE FOLLOWING COMMANDS ARE,

1) **RECYCLEBIN**
2) **FLASHBACK**
3) **PURGE**

<u>1) RECYCLEBIN:</u> IT IS A PRE-DEFINE TABLE WHICH IS USED TO STORED INFORMATION ABOUT DROPPED TABLES.IT WILL WORK AS A WINDOWS RECYCLEBIN IN SYSTEM.

<u>HOW TO VIEW THE STRUCTURE OF RECYCLEBIN:</u>

<u>SYNTAX:</u>

```
SQL> DESC RECYCLEBIN;

NAME                                      NULL?              TYPE

----------------------------------------- --------           ------------

OBJECT_NAME                                 NOT NULL         VARCHAR2(30)

ORIGINAL_NAME                                                VARCHAR2(32)
```

<u>HOW TO VIEW INFORMATION ABOUT DROPPED TABLES IN RECYCLEBIN:</u>

<u>SYNTAX:</u>

```
SQL> SELECT OBJECT_NAME, ORIGINAL_NAME FROM RECYCLEBIN;

OBJECT_NAME                                           ORIGINAL_NAME

------------------------------------------------------       ----------------------

BIN$EENURFTST7AHNHV7RBI71Q==$0                               STUDENT
```

2) FLASHBACK: THIS COMMAND IS USED TO RESTORE A DROPPED TABLE FROM RECYCLEBIN.

SYNTAX:

SQL> FLASHBACK TABLE <TABLE NAME> TO BEFORE DROP;

EX:

SQL> FLASHBACK TABLE STUDENT TO BEFORE DROP;

PURGE:THIS COMMAND IS USED TO DROP A TABLE FROM RECYCLEBIN PERMANENTLY(OR)TO DROP A TABLE FROM DATABASE PERMANENTLY.

SYNTAX1: (DROPPING A SPECIFIC TABLE FROM RECYCLEBIN)

SQL> PURGE TABLE <TABLE NAME>;

EX:

SQL> PURGE TABLE TEST1;

SYNTAX2: (DROPPING ALL TABLES FROM RECYCLEBIN)

SQL> PURGE RECYCLEBIN;

EX:

SQL> PURGE RECYCLEBIN;

SYNTAX3:(DROP A TABLE FROM DATABASE PERMANENTLY)

SQL> DROP TABLE <TABLE NAME> PURGE;

EX:

SQL> DROP TABLE STUDENT PURGE;

2) DML (DATA MANIPULATION LANGUAGE)

> INSERT

> UPDATE

> DELETE

1. INSERT: INSERTING A NEW ROW DATA INTO A TABLE.

SYNTAX1:

INSERT INTO <TN> VALUES(VALUE1, VALUE2, ……………..);

EX:

SQL> CREATE TABLE STUDENT(STID INT, SNAME VARCHAR2(10), SFEENUMBER (10));

SQL> INSERT INTO STUDENT VALUES(1021,'SAI',2500);

1 ROW CREATED.

NOTE: IN THIS METHOD WE SHOULD INSERT VALUES TO ALL COLUMNS IN A TABLE.

SYNTAX2:

INSERT INTO <TN>(REQ. COLUMN NAMES) VALUES (VALUE1, VALUE2, …….);

EX:

SQL> INSERT INTO STUDENT(STID, SNAME) VALUES (1022,'SMITH');

1 ROW CREATED.

NOTE: IN THIS METHOD WE CAN INSERT VALUES FOR REQUIRED COLUMNS ONLY.AND REMAINING COLUMNS WILL TAKE "NULL" BY DEFAULT.

HOW TO INSERT NULLS INTO A TABLE:

METHOD1:

INSERT INTO EMP VALUES (NULL, NULL, NULL);

METHOD2:

INSERT INTO EMP (EID, ENAME, EADDRESS) VALUES (NULL, NULL, NULL);

SUBSTITUTIONAL OPERATOR: THIS OPERATOR IS USED TO INSERT MULTIPLE ROWS DATA INTO A TABLE CONTINUALLY.

I) &: WE CAN INSERT VALUES TO COLUMNS DYNAMICALLY.

SYNTAX1(&):

INSERT INTO <TN> VALUES(&<COLUMN NAME1>,&<COLUMN NAME2>,………….);

EX:

SQL> INSERT INTO STUDENT VALUES(&STID,'&SNAME',&SFEE);

ENTER VALUE FOR STID: 1023

ENTER VALUE FOR SNAME: ALLEN

ENTER VALUE FOR SFEE: 1500

SQL> /   ------------ →(HERE " / " IS USED TO RE-EXECUTE THE LAST EXECUTED SQL QUERY IN SQLPLUS EDITOR)

ENTER VALUE FOR STID: 1024

ENTER VALUE FOR SNAME: WARD

ENTER VALUE FOR SFEE: 4500

<u>SYNTAX2(&):</u>

INSERT INTO <TN>(REQ.COLUMN NAMES) VALUES (&<COLUMN NAME1>, ...);

EX:

SQL> INSERT INTO STUDENT(STID)VALUES(&STID);

ENTER VALUE FOR STID: 1026

SQL> /

ENTER VALUE FOR STID: 1027

SQL> /

ENTER VALUE FOR STID: 1028

<u>UPDATE:</u>

UPDATING ALL ROWS DATA AT A TIME IN A TABLE(OR) UPDATING A SINGLE ROW DATA IN A TABLE BY USING "WHERE CLAUSE"CONDITION.

<u>SYNTAX:</u>

UPDATE <TN> SET <COLUMN NAME1> = <VALUE1>,<COLUMN NAME2>=<VALUE2>,..............[WHERE <CONDITION>];

EX1:

SQL> UPDATE STUDENT SET SNAME='JONES', SFEE=6500 WHERE STID=1027;

SQL> UPDATE EMP SET COMM=500;

SQL> UPDATE EMP SET SAL=NULL;

SQL> UPDATE EMP SET SAL=5000 WHERE SAL IS NULL;

<u>DELETE:</u>TO DELETE ALL ROWS FROM A TABLE AT A TIME(OR)TO DELETE A SPECIFIC ROW FROM A TABLE BY USING "WHERE CLAUSE"CONDITION.

<u>SYNTAX:</u>

DELETE FROM <TN> [WHERE <CONDITION>];

EX:

SQL> DELETE FROM STUDENT WHERE STID=1023;

EX:

```
SQL> DELETE FROM STUDENT;

SQL> DELETE FROM EMP WHERE COMM IS NULL;
```

DIFFERENCE BETWEEN DELETE & TRUNCATE COMMAND:

DELETE	**TRUNCATE**
1. IT IS A DML COMMAND.	**1. IT IS A DDL COMMAND.**
2. IT CAN DELETE A SPECIFIC ROW FROM A TABLE.	**2. IT CANNOT DELETE A SPECIFIC ROW FROM A TABLE.**
3. IT SUPPORTS "WHERE CLAUSE" CONDITION.	**3. IT DOES NOT SUPPORTS "WHERE CLAUSE" CONDITION.**
4.IT TEMPORARY DATA DELETION.	**4. IT IS PERMANENT DATA DELETION.**
5. WE CAN RESTORE DELETED DATA BY USING "ROLLBACK" COMMAND.	**5. WE CANNOT RESTORE DELETED DATA BY USING "ROLLBACK".**
6. EXECUTION SPEED IS SLOW. (DELETING ROWS IN ONE BY ONE MANNER)	**6. EXECUTION SPEED IS FAST (DELETING GROUP OF ROWS AT A TIME)**

3) DQL / DRL(DATA QUERY LANGUAGE / DATA RETRIVE LANGUAGE):

> SELECT

SELECT: TO RETRIEVE ALL ROWS FROM A TABLE AT A TIME (OR) TO RETRIEVE A SPECIFIC ROW FROM A TABLE BY USING "WHERE CLAUSE"CONDITION.

SYNTAX:

```
SELECT * FROM <TABLE NAME> [WHERE <CONDITION>];
```

HERE, " * " IS REPRESENT ALL COLUMNS IN A TABLE.

EX:

```
SQL> SELECT * FROM DEPT;

(OR)

SQL> SELECT DEPTNO, DNAME, LOC FROM DEPT;
```

EX:

```
SQL> SELECT * FROM EMP WHERE JOB='CLERK';

SQL> SELECT * FROM EMP WHERE COMM IS NULL;

SQL> SELECT * FROM EMP WHERE COMM IS NOT NULL;
```

<u>TO VIEW ALL LIST OF TABLES IN ORACLE DATABASE:</u>

<u>SYNTAX:</u>

```
SQL> SELECT * FROM TAB;
```

<u>TO VIEW DATA OF A PARTICULAR TABLE:</u>

<u>SYNTAX:</u>

```
SQL> SELECT * FROM <TABLE NAME>;
```

EX:

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

<u>NOTE:</u> WHEN WE WANT TO DISPLAY THE INFORMATION / DATA OF A PARTICULAR TABLE IN PROPER SYSTEMATICALLY THEN WE NEED TO SET THE FOLLOWING TWO PROPERTIES ARE,

<u>1) PAGESIZE N:</u>

- NUMBER OF ROWS DISPLAYED PER A PAGE.HERE "N" IS REPRESENTED NO. OF ROWS. BY DEFAULT, A SINGLE PAGE IS DISPLAYED 14 ROWS.RANGE IS 0 TO 50000.

<u>SYNTAX:</u>

```
SQL> SET PAGESIZE N;
```

EX:

```
SQL> SET PAGESIZE 100;
```

<u>2) LINES N:</u>

- NUMBER OF BYTES IN A SINGLE LINE.HERE "N" IS REPRESENT NO. OF BYTES.RANGE IS 1 TO 32767.

<u>SYNTAX:</u>

```
SQL> SET LINES N;
```

EX:

```
SQL> SET LINES 100;
```

<u>TO CLEAR SQL PLUS EDITOR SCREEN:</u>

<u>SYNTAX:</u>

SQL> CL SCR;

(OR)

SHIFT+DELETE (FROM KEYBOARD)

<u>TO DISCONNECT / EXIT FROM ORACLE DATABASE:</u>

SQL> EXIT;

<u>ALIAS NAMES:</u> IT IS AN ALTERNATE (OR) TEMPORARY NAME.USER CAN CREATE ALIAS NAMES ON TWO LEVELS IN DB.

<u>I) COLUMN LEVEL:</u>

- IN THIS LEVEL WE ARE CREATING ALIAS NAMES ON COLUMNS.

<u>SYNTAX:</u>

<COLUMN NAME><COLUMN ALIAS NAME>

EX:

DEPTNO   X

<u>II)TABLE LEVEL:</u>

- IN THIS LEVEL WE CREATING ALIAS NAMES ON TABLE.

<u>SYNTAX:</u>

<TABLE NAME><TABLE ALIAS NAME>

EX:

DEPT   D

<u>SYNTAX TO COMBINED COLUMN + TABLE LEVEL ALIAS NAMES BY USING "SELECT" QUERY:</u>

SELECT <COLUMN NAME1><COLUMN NAME1 ALIAS NAME>,<COLUMN NAME2><COLUMN NAME2 ALIAS NAME>,............. FROM <TN><TABLE ALIAS NAME>;

EX:

SQL> SELECT DEPTNO X,DNAME Y,LOC Z FROM DEPT D;

CONCATENATION OPERATOR (||):

- THIS OPERATOR IS USED TO JOIN TWO STRING VALUES (OR) TWO EXPRESSIONS IN A SELECT QUERY.

EX:

```
SQL> SELECT 'WELCOME'||' '||'TO ORACLE' FROM DUAL;
```

OUTPUT:

```
WELCOME TO ORACLE
```

EX:

```
SQL> SELECT 'MR.'||ENAME||' '||'IS WORKING AS A'||' '||JOB FROM EMP;
```

OUTPUT:

```
MR. SMITH IS WORKING AS A CLERK
```

DISTINCT KEYWORD:

- THIS KEYWORD IS USED TO ELIMINATE DUPLICATE VALUES AND DISPLAY

UNIQUE VALUES IN QUERY RESULT.

EX:

```
SQL> SELECT DISTINCT JOB FROM EMP;

SQL> SELECT DISTINCT DEPTNO FROM EMP;
```

HOW TO CREATE A NEW TABLE FROM THE OLD TABLE:

SYNTAX1:

```
CREATE TABLE <NEW TABLE NAME> AS SELECT * FROM <OLD TABLE NAME>;
```

EX:

```
CREATE TABLE NEWEMP AS SELECT * FROM EMP;
```

NOTE: CREATED A NEW TABLE WITH COPY OF ALL ROWS & COLUMNS FROM THE OLD TABLE.

SYNTAX2:

```
CREATE TABLE <NEW TABLE NAME> AS SELECT * FROM <OLD TABLE NAME> WHERE
<FALSE CONDITION>;
```

EX:

CREATE TABLE DUMMYEMP AS SELECT * FROM EMP WHERE 1=2;

NOTE: CREATED A NEW TABLE WITHOUT COPY ROWS FROM OLD TABLE. (COLUMNS COPY)

EX:

CREATE TABLE SPECEMP AS SELECT EID, EADDRESS FROM EMP;

NOTE: CREATED A NEW TABLE WITH SPECIFIC COLUMNS FROM THE OLD TABLE.

EX:

CREATE TABLE SPECROWS AS SELECT * FROM EMP WHERE EADDRESS='HYD';

NOTE: CREATED A NEW TABLE WITH SPECIFIC ROWS FROM THE OLD TABLE.

<u>HOW TO COPY DATA FROM ONE TABLE TO ANOTHER TABLE:</u>

<u>SYNTAX:</u>

INSERT INTO <DESTINATION TABLE NAME> SELECT * FROM <SOURCE TABLE NAME>;

EX:

CREATE TABLE DESTEMP (EMPNO INT, NAMECHAR (10), LOC VARCHAR2(10));

INSERT INTO DESTEMP SELECT * FROM EMP;

EX:

CREATE TABLE DEMP AS SELECT * FROM EMP WHERE 1=0;

INSERT INTO DEMP SELECT * FROM EMP;

OPERATORS IN ORACLE:

OPERATORS ARE USED TO EXPRESS THE CONDITIONS IN SELECT STATEMENTS. OPERATOR MANIPULATES INDIVIDUAL DATA ITEMS AND RETURNS A RESULT. THE DATA ITEMS ARE CALLED OPERANDS OR ARGUMENTS.

THE DIFFERENT TYPES OF OPERATORS AVAILABLE IN ORACLE SQL ARE:-

- **ARITHMETIC OPERATORS**
- **ASSIGNMENT OPERATOR**
- **RELATIONAL OPERATORS**
- **LOGICAL OPERATORS**
- **SPECIAL OPERATORS**
- **SET OPERATORS**

ARITHMETIC OPERATORS: -

- **THE ARITHMETIC OPERATIONS CAN BE USED TO CREATE EXPRESSIONS ON NUMBER AND DATE DATA.**
- **THE ARITHMETIC OPERATIONS CAN BE USED TO PERFORM ANY ARITHMETIC OPERATIONS LIKE ADDITION, SUBTRACTION, MULTIPLICATION AND DIVIDED BY.**
- **THE ARITHMETIC OPERATORS CAN BE USED IN ANY CLAUSE OF A SQL STATEMENT.**
- **SQL * PLUS IGNORES THE BLANK SPACES BEFORE AND AFTER THEARITHMETIC OPERATOR .**

EXAMPLE:-

DISPLAY SALARY OF EMPLOYEES WITH 2000 INCREMENT IN THEIR SALARY.

SQL> SELECT ENAME,SAL,SAL + 2000 "INCREMENTED SALARY" FROM EMP;

EXPLINATION:-IN EMP TABLE EVERY EMPLOYEE SALARY SUM IT 2000.

ARITHMETIC OPERATOR SUBTRACTION (-):-

- **USED TO PERFORM SUBTRACTION BETWEEN TWO NUMBERS AND DATES.**

EXAMPLE:

DISPLAY THE DETAILS OF EMPLOYEES DECREASING THEIR SALARY BY 200.

SQL> SELECT ENAME,SAL,SAL-200 FROM EMP;

EXPLINATION:-IN EMP TABLE EVERY EMPLOYEE SALARY SUBTRACTED WITH 200.

ARITHMETIC OPERATOR MULTIPLICATION(*) :-USED TO PERFORM MULTIPLICATION.

EXAMPLE:-

DISPLAY THE DETAILS OF THE EMPLOYEES INCREMENTING THEIR SALARY TWO TIMES.

SQL> SELECT SAL * 2 FROM EMP;

EXPLINATION:-EVERY EMP TABLE SALARY IS MULTIPLIED BY 2.

ARITHMETIC OPERATOR DIVISION (/):-

USED TO PERFORM DIVISION TEST. DIVISION WILL DISPLAY ONLY THE QUOTIENT VALUE NOT THE REMAINDER VALUE. EXAMPLE 6/2 GIVES 3 BECAUSE 2 DIVIDES 6 BY 3 TIMES.

EXAMPLE:-

DISPLAY HALF OF THE SALARY OF EMPLOYEES.

```
SQL> SELECT SAL, SAL/2 FROM EMP;
```

EXAMPLES:-

```
        SQL> SELECT EMPNO,ENAME,SAL,12*SAL+100 FROM EMP;

          SQL> SELECT EMPNO,ENAME,SAL,(12*SAL)+100 FROM EMP;

          SQL> SELECT EMPNO,ENAME,SAL,12*(SAL+100) FROM EMP;
```

ASSIGNMENTOPERATORS:-

THIS OPERATOR IS USED FOR EQUALITY TEST. USED TO TEST THE EQUALITY OF TWO OPERANDS.

EXAMPLE:-

DISPLAY THE DETAILS OF EMPLOYEES WHOSE SALARY IS EQUAL TO 2000.

```
SQL> SELECT *FROM EMP WHERE SAL=950;


EMPNO   ENAME                JOB       MGR   HIREDATE    SAL    COMM   DEPTNO

------------------------------------------------------------------------------------
---

 7900    JAMES               CLERK     7698   03-DEC-81   950             30
```

RELATIONAL OPERATORLESSTHAN(<):-

THIS OPERATOR IS USED FOR LESS THAN TEST. EXAMPLE A<B CHECKS THAT OPERAND ‘A’ IS LESS THAN ‘B’ OR NOT.

EXAMPLE: DISPLAY THE DETAILS OF THE EMPLOYEES WHOSE SALARY IS LESS THAN 3000.

```
  SQL> SELECT * FROM EMP WHERE SAL< 3000;
```

```
EMPNO ENAME       JOB              MGR HIREDATE      SAL    COMM          DEPTNO

---------------------------------------------------------------------------------------------
-------

 7369 SMITH               CLERK              7902 17-DEC-80    800    20

 7499 ALLEN              SALESMAN        7698 20-FEB-81     1600  300     30

 7521 WARD               SALESMAN        7698 22-FEB-81     1250  500     30

 7566 JONES                MANAGER        7839 02-APR-81     2975    20

 7654 MARTIN            SALESMAN        7698 28-SEP-81     1250  1400     30

 7698 BLAKE              MANAGER        7839 01-MAY-81    2850    30

 7782 CLARK              MANAGER        7839 09-JUN-81     2450    10

 7844 TURNER                 SALESMAN   7698 08-SEP-81     1500    0      30

 7876 ADAMS             CLERK              7788 23-MAY-87    1100    20

 7900 JAMES              CLERK              7698 03-DEC-81    950      30

 7934 MILLER             CLERK              7782 23-JAN-82    1300    10
```

HERE IF YOU OBSERVE WE GOT A RESULT VALUES WHOSE SALARY IS LESS THAN THE OPERAND 3000.

<u>RELATIONAL OPERATOR GREATER THAN(>):-</u>

THIS OPERATOR IS USED FOR GREATER THAN TEST. FOR EXAMPLE A>B CHECKS THE OPERAND ‘A’ IS GREATER THAN ‘B’ OR NOT.

EXAMPLE:

DISPLAY THE DETAILS OF EMPLOYEES WHOSE SALARY IS GREATER THAN 3000

```
SQL> SELECT * FROM EMP WHERE SAL> 3000;


EMPNO ENAME                   JOB        MGR HIREDATE    SAL  COMM DEPTNO

-------------------------------------------------------------------------------------------

 7839 KING                    PRESIDENT      17-NOV-81   5000                 10
```

RELATIONAL OPERATOR LESS THAN OR EQUALS TO(<=):

THIS OPERATOR IS USED FOR LESS THAN OR EQUAL TO TEST. FOR EXAMPLE A<=B, HERE CHECKS WHETHER OPERAND 'A' IS LESS THAN OR EQUALS TO OPERAND 'B'. IF A<B THEN CONDITION IS TRUE AND IF A=B THEN ALSO CONDITION IS TRUE BUT IF A>B THEN CONDITION IS FALSE.

EXAMPLE :-

DISPLAY THE DETAILS OF EMPLOYEES WHOSE SALARY IS LESS THAN OR EQUAL TO 3000.

```
SQL> SELECT * FROM EMP WHERE SAL<= 3000;
```

RELATIONAL OPERATOR GREATER THAN OR EQUALS TO(>=):

THIS OPERATOR IS USED TO CHECK THE GREATER THAN OR EQUAL TEST. FOR EXAMPLE A>=B CHECKS THE OPERAND 'A' IS GREATER THAN OPERAND 'B' OR OPERAND 'A' IS EQUALS TO THE OPERAND 'B'.

EXAMPLE:-

DISPLAY THE DETAILS OF EMPLOYEES WHOSE SALARY IS GREATER THAN OR EQUAL TO 3000.

```
SQL> SELECT * FROM EMP WHERE SAL>= 3000;


EMPNO ENAME            JOB        MGR   HIREDATE    SAL  COMM     DEPTNO

----------------------------------------------------------------------------------------------------
----

 7788 SCOTT            ANALYST    7566  19-APR-87    3000              20

 7839 KING             PRESIDENT        17-NOV-81    5000              10

 7902 FORD             ANALYST    7566  03-DEC-81    3000              20
```

RELATIONAL OPERATOR NOT EQUALS TO(!= OR ^= OR <>):

THIS OPERATOR IS USED FOR INEQUALITY TEST.

EXAMPLES:

DISPLAY THE DETAILS OF EMPLOYEES WHOSE SALARY IS NOT EQUALS TO 2000.

```
SQL> SELECT * FROM EMP WHERE SAL != 3000;
SQL> SELECT * FROM EMP WHERE SAL ^= 2000;
```

```
SQL> SELECT * FROM EMP WHERE SAL<> 2000;
```

<u>LOGICAL OPERATORS:-</u>

<u>AND OPERATOR:-</u>

RETURNS 'TRUE' IF BOTH COMPONENT CONDITIONS ARE TRUE. RETURNS 'FALSE' IF ANY ONE COMPONENT CONDITION OR BOTH COMPONENT CONDITIONS ARE FALSE.

EXAMPLE:-

DISPLAY THE DETAILS OF EMPLOYEES WHOSE SALARY IS GREATER THAN 1000 AND ALSO WHOSE SALARY IS LESS THAN 2000.

```
  SQL> SELECT *FROM EMP WHERE SAL> 1000 AND SAL<2000;

EMPNO ENAME      JOB           MGR HIREDATE     SAL COMM DEPTNO

-------------------------------------------------------------------------------------------

 7499 ALLEN           SALESMAN       7698 20-FEB-81    1600   300   30

 7521 WARD             SALESMAN      7698 22-FEB-81    1250   500          30

 7654 MARTIN           SALESMAN      7698 28-SEP-81    1250  1400          30

 7844 TURNER           SALESMAN      7698 08-SEP-81    1500    0    30

 7876 ADAMS            CLERK         7788 23-MAY-87    1100         20

 7934 MILLER           CLERK         7782 23-JAN-82    1300         10
```

```
SQL> SELECT ENAME,SAL,JOB FROM EMP

     WHERE (SAL>=1500 AND SAL<=5000) AND

     JOB='MANAGER';
```

<u>THE LOGICAL OR OPERATOR:-</u>

- **RETURNS TRUE IF EITHER COMPONENT CONDITIONS BECOME TRUE. RETURNS FALSE IF BOTH THE COMPONENT CONDITIONS BECOMES FALSE.**

EXAMPLE:

DISPLAY THE DETAILS OF EMPLOYEES WHOSE SALARY IS GREATER THAN 1000 OR ALSO WHOSE SALARY IS LESS THAN 2000.

```
SQL> SELECT *FROM EMP WHERE SAL> 1000 OR SAL< 2000;
```

EXPLINATION:-WHOSE SALARIES MORE THAN 1000 OR LESS THAN 2000 THAT ALL EMP TABLE DISPLAY.

```
SQL> SELECT EMPNO,ENAME,JOB,HIREDATE  FROM EMP

         WHERE JOB='MANAGER' OR DEPTNO=20;

SQL> SELECT EMPNO,ENAME,JOB,HIREDATE  FROM EMP

          WHERE (JOB='MANAGER' OR DEPTNO=10);

SQL> SELECT EMPNO,ENAME,JOB,HIREDATE  FROM EMP

                WHERE (JOB='CLERK' OR JOB='SALESMAN' OR JOB='ANALYST');

SQL> SELECT EMPNO,ENAME,JOB,HIREDATE  FROM EMP

               WHERE (SAL<=2500 OR SAL>=5000) OR JOB='MANAGER';

SQL> SELECT ENAME,JOB ,SAL FROM EMP

               WHERE JOB='CLERK' OR JOB='MANAGER' AND SAL>1500;
```

<u>THE LOGICAL NOT OPERATOR:-</u>

THE NOT OPERATOR RETURNS 'TRUE' IF THE CONDITION IS FALSE AND RETURNS 'FALSE' IF THE FOLLOWING CONDITION IS TRUE.

EXAMPLE:

DISPLAY THE DETAILS OF EMPLOYEES WHOSE SALARY IS GREATER THAN OR EQUALS TO 3000.

```
SQL> SELECT * FROM EMP WHERE SAL< 3000;
```

EXPLINATION:-WHOSE SALARY LESS THAN 3000 THAT SALARIES ALL ARE COMMING.

```
SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP

               WHERE NOT ENAME='SMITH';

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP

          WHERE NOT SAL>=5000;

SQL> SELECT EMPNO,ENAME,JOB,SAL,DEPTNO FROM EMP

           WHERE NOT JOB='CLERK' AND DEPTNO=20;
```

SPECIAL OPERATORS:-

IN OPERATOR:-

- **RETURNS TRUE IF VALUE IS AVAILABLE IN GIVEN LIST OF VALUES**
- **SUPPORTS WITH ALL TYPES OF DATA (DATA TYPES)**

IN THE BELOW EXAMPLE ONLY EMPLOYEES WHOSE EMPNO IS (7125,7369,7782) ARE FETCHED.

```
SQL> SELECT *FROM EMP WHERE EMPNO IN (7125, 7369, 7782);


EMPNO ENAME                JOB       MGR HIREDATE       SAL  COMM DEPTNO

--------------------------------------------------------------------------------------------
--

 7369 SMITH               CLERK        7902 17-DEC-80    800         20

 7782 CLARK               MANAGER   7839 09-JUN-81      2450         10
```

INSIDE DML STATEMENTS:-

```
SQL> UPDATE EMP SET SAL=SAL+200 WHERE ENAME IN ('SMITH','ALLEN','WARD');

SQL> DELETE FROM EMP WHERE HIREDATE IN ('22-DEC-82','17-NOV-81');
```

NOT IN OPERATOR:-

NOT IN' OPERATOR IS QUITE OPPOSITE TO 'IN' CLAUSE.

```
SQL> SELECT *FROM EMP WHERE EMPNO NOT IN (7125,  7369,7782);
```

INSIDE DML STATEMENTS:-

```
SQL> UPDATE EMP SET SAL=SAL+200 WHERE ENAME NOT IN

        ('SMITH','ALLEN','WARD');

SQL> DELETE FROM EMP WHERE HIREDATE NOT IN ('22-DEC-82',' 17-NOV-81');
```

BETWEEN OPERATOR:-

- **RETURNS TRUE IF VALUE SPECIFIED IS WITHIN THE SPECIFIED RANGE.**
- **SUPPORTS WITH NUMBERS AND DATE VALUES.**
- **RETURNS ALL VALUES FROM GIVEN RANGE INCLUDING SOURCE & DESTINATION VALUES.**
- **ALWAYS APPLY ON LOW VALUE TO HIGH VALUE.**

EXAMPLE:- IN THIS EXAMPLE ALL EMPLOYEE RECORDS ARE FETCHED WHOSE SALARY IS BETWEEN 2000 AND 3000

```
SQL> SELECT *FROM EMP WHERE SAL BETWEEN 2000 AND 3000;


EMPNO ENAME       JOB        MGR HIREDATE                  SAL  COMM DEPTNO

--------------------------------------------------------------------------------------------------

 7566 JONES              MANAGER        7839 02-APR-81     2975          20

 7698 BLAKE             MANAGER         7839 01-MAY-81     2850          30

 7782 CLARK             MANAGER         7839 09-JUN-81     2450          10

 7788 SCOTT             ANALYST         7566 19-APR-87     3000          20

 7902 FORD               ANALYST        7566 03-DEC-81     3000          20
```

WHENEVER LOWER BOUND VALUE IS LARGER THAN UPPER BOUND THEN IT SHOWS 'NO ROWS SELECTED'

EXAMPLE:-

```
SQL> SELECT *FROM EMP WHERE SAL BETWEEN 3000 AND 2000;
```

OUTPUT:

```
 -- NO ROWS SELECTED
```

```
SQL> SELECT ENAME,SAL,JOB FROM EMP

        WHERE JOB BETWEEN 'MANAGER' AND 'SALESMAN';

SQL> SELECT ENAME,SAL,JOB,HIREDATE FROM EMP

        WHERE HIREDATE    BETWEEN '17-DEC-81' AND '20-JUN-83';
```

<u>NOT BETWEEN OPERATOR:-</u>

- **RETURNS TRUE IF VALUE SPECIFIED IS NOT WITHIN THE SPECIFIED RANGE.**
- **SUPPORTS WITH NUMBERS AND DATE VALUES.**
- **NOT BETWEEN IS AN EXCLUSIVE OPERATOR WHICH ELIMINATES RANGE LIMITS FROM OUTPUT.**

EXAMPLE:-

```
SQL> SELECT *FROM EMP WHERE SAL NOT BETWEEN 2000 AND 3000;

  EMPNO ENAME     JOB            MGR HIREDATE      SAL  COMM DEPTNO

----------------------------------------------------------------------------------------------------
--------

 7369 SMITH              CLERK             7902 17-DEC-80    800    20

 7499 ALLEN             SALESMAN       7698 20-FEB-81     1600  300    30

 7521 WARD              SALESMAN       7698 22-FEB-81     1250          500             30

 7654 MARTIN            SALESMAN       7698 28-SEP-81     1250  1400    30

 7839 KING               PRESIDENT     17-NOV-81  5000   10

 7844 TURNER            SALESMAN       7698 08-SEP-81     1500  0       30

 7876 ADAMS             CLERK            7788 23-MAY-87    1100          20

 7900 JAMES              CLERK           7698 03-DEC-81    950           30

 7934 MILLER             CLERK           7782 23-JAN-82    1300          10
```

NOTE:-

LOWER BOUND – 'VALUE 'MUST BE LOWER WHEN COMPARE TO 'UPPER BOUND 'VALUE

UPPER BOUND- 'VALUE' MUST BE HIGHER WHEN COMPARE TO 'LOWER BOUND 'VALUE

```
   SQL> SELECT ENAME,SAL,JOB FROM EMP

      WHERE JOB  NOT  BETWEEN 'MANAGER' AND 'SALESMAN';

  SQL> SELECT ENAME,SAL,JOB,HIREDATE FROM EMP

         WHERE HIREDATE NOT   BETWEEN '17-DEC-81' AND '20-JUN-83';
```

<u>LIKE OPERATOR: -</u>

USED TO SEARCH FOR SPECIFIC PATTERN IN A GIVEN INPUT.

% (PERCENTAGE) AND _ (UNDERSCORE) ARE TWO WILDCARD CHARACTERS.

% (PERCENTAGE) REPRESENTS "REMAINING GROUP OF CHARACTERS "IN THE GIVEN INPUT

_ (UNDERSCORE) REPRESENTS "ONE CHARACTER" IN GIVEN INPUT.

SYNTAX:-

SELECT *FROM   <TABLENAME>

WHERE   <CHARACTER DATA TYPE COLUMN> LIKE '<VALUE>';

EXAMPLE:- DISPLAY THE EMPLOYEES WHOSE NAME IS STARTING WITH 'S' IN EMP TABLE.

```
SQL> SELECT * FROM EMP WHERE ENAME LIKE   'S%'

EMPNO ENAME       JOB             MGR HIREDATE    SAL  COMM DEPTNO

--------------------------------------------------------------------------------------------

 7369 SMITH              CLERK             7902 17-DEC-80   800              20

 7788 SCOTT              ANALYST           7566 19-APR-87  3000           20
```

DISPLAY THE EMPLOYEES WHOSE NAME ENDS WITH 'S' IN EMP TABLE

```
SQL> SELECT * FROM EMP WHERE ENAME LIKE '%S'

EMPNO ENAME       JOB             MGR HIREDATE      SAL  COMM DEPTNO

--------------------------------------------------------------------------------------------
----

 7566 JONES              MANAGER          7839 02-APR-81    2975  20

 7876 ADAMS             CLERK             7788 23-MAY-87            1100    20

 7900 JAMES              CLERK             7698 03-DEC-81            950      30
```

DISPLAY THE EMPLOYEES WHOSE NAMES ARE HAVING SECOND LETTER AS 'L' IN EMP TABLE

```
SQL> SELECT * FROM EMP WHERE ENAME LIKE '_L%'


EMPNO ENAME                   JOB        MGR HIREDATE    SAL  COMM DEPTNO

--------------------------------------------------------------------------------------------

 7499 ALLEN                   SALESMAN   7698 20-FEB-81  1600   300           30

 7698 BLAKE                   MANAGER    7839 01-MAY-81  2850         30

 7782 CLARK                   MANAGER    7839 09-JUN-81  2450         10
```

```
SQL> SELECT ENAME ,HIREDATE FROM  EMP WHERE HIREDATE  LIKE '%JAN%';
```

```
SQL> SELECT EMPNO,ENAME,JOB FROM EMP WHERE JOB LIKE '_____';
```

EX:

WHOSE EMPLOYEE NUMBER STARTS WITH 7 AND ENDS WITH 8 DIGIT?

```
SELECT * FROM EMP WHERE EMPNO LIKE '7%8';
```

EX:

WHO ARE JOININED IN THE YEAR 1981?

```
SELECT * FROM EMP WHERE HIREDATE LIKE '%81';
```

EX:

WHO ARE JOINED IN THE MONTH OF DEC?

```
SELECT * FROM EMP WHERE HIREDATE LIKE '%DEC%';
```

EX:

WHO ARE JOINED IN MONTH OF "FEB" , "MAY" ?

```
SELECT * FROM EMP WHERE HIREDATE LIKE '%FEB%' OR HIREDATE LIKE '%MAY%';
```

```
SQL> SELECT EMPNO,ENAME,JOB ,HIREDATE FROM EMP WHERE HIREDATE LIKE '%-
FEB-81';
```

<u>LIKE OPERATOR WITH SPECIAL CHAR'S:</u>

EX:WHOSE EMPLOYEE NAME IS HAVING "#" CHAR?

```
SELECT * FROM EMP77 WHERE ENAME LIKE '%#%';
```

EX:WHOSE EMPLOYEE NAME IS HAVING "@" CHAR?

```
SELECT * FROM EMP77 WHERE ENAME LIKE '%@%';
```

EX:WHOSE EMPLOYEE NAME IS HAVING "_" CHAR?

```
SELECT * FROM EMP77 WHERE ENAME LIKE '%\_%'ESCAPE '\';
```

EX:WHOSE EMPLOYEE NAME IS HAVING "%" CHAR?

```
SELECT * FROM EMP77 WHERE ENAME LIKE '%\%%'ESCAPE '\';
```

NOTE:

GENERALLY ORACLE DB SERVER WILL TREAT THESE SYMBOL(%, _) AS "WILDCARD OPERATORS" BUT NOT "SPECIAL CHAR'S". TO OVERCOME THIS PROBLEM WE MUST USE A PRE-DEFINED STATEMENT IS "ESCAPE '\' " .

NOT LIKE OPERATOR: -

SYNTAX:-

SELECLT *FROM <TABLE NAME>

WHERE <CHARACTER DATA TYPE COLUMN> NOT LIKE '<VALUE>';

DISPLAY THE EMPLOYEES WHOSE NAME IS NOT ENDS WITH 'S' IN EMP TABLE?

```
SQL> SELECT *FROM EMP WHERE ENAME NOT LIKE '%S';


EMPNO ENAME      JOB           MGR HIREDATE     SAL  COMM DEPTNO
-----------------------------------------------------------------

 7369 SMITH      CLERK        7902 17-DEC-80    800          20
 7499 ALLEN      SALESMAN     7698 20-FEB-81    1600   300   30
 7521 WARD       SALESMAN     7698 22-FEB-81    1250   500   30
 7654 MARTIN     SALESMAN     7698 28-SEP-81    1250  1400   30
 7698 BLAKE      MANAGER      7839 01-MAY-81    2850         30
 7782 CLARK      MANAGER      7839 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         30
 7902 FORD       ANALYST      7566 03-DEC-81    3000         20
 7934 MILLER     CLERK        7782 23-JAN-82    1300         10
```

DISPLAY THE EMPLOYEES WHOSE NAMES ARE NOT HAVING SECOND LETTER AS 'L' IN EMP TABLE?

```
SQL> SELECT *FROM EMP WHERE ENAME NOT LIKE '_L%';
```

DISPLAY THE EMPLOYEES WHOSE NAMES ARE NOT START WITH 'S' IN EMP TABLE.?

```
SQL> SELECT *FROM EMP WHERE ENAME NOT LIKE 'S%';

SQL> SELECT ENAME ,HIREDATE FROM  EMP WHERE HIREDATE NOT LIKE '%JAN%';

SQL> SELECT EMPNO,ENAME,JOB FROM EMP WHERE ENAME NOT LIKE '_O%';
```

DISPLAY THE EMPLOYEES WHOSE NAMES ARE SECOND LETTER START WITH 'R' FROM ENDING.?

```
SQL> SELECT *FROM EMP WHERE ENAME LIKE '%R_';

EMPNO ENAME      JOB          MGR HIREDATE    SAL  COMM DEPTNO

------------------------------------------------------------------------------------------

 7521 WARD            SALESMAN      7698 22-FEB-81    1250   500     30

 7782 CLARK           MANAGER       7839 09-JUN-81     2450            10

 7902 FORD            ANALYST       7566 03-DEC-81     3000           20
```

DISPLAY THE NAMES IN EMP TABLE WHOSE NAMES HAVING 'LL'.?

```
SQL> SELECT *FROM  EMP WHERE ENAME LIKE '%LL%';

EMPNO ENAME        JOB                MGR HIREDATE    SAL  COMM DEPTNO

------------------------------------------------------------------------------------------

 7499 ALLEN           SALESMAN        7698 20-FEB-81  1600   300    30

 7934 MILLER          CLERK           7782 23-JAN-82  1300       10
```

IS NULL OPERATOR:COMPARING NULLS IN A TABLE.NULL IS UNKNOW (OR) UNDEFINED VALUE IN DATABASE,NULL != 0 & NULL != SPACE.SO USE " IS " KEYWORD.

SYNTAX:

```
        WHERE <COLUMN NAME> IS NULL;
```

EX:

WAQ TO DISPLAY EMPLOYEE WHOSE COMMISSION IS NULL ?

```
SELECT * FROM EMP WHERE COMM IS NULL;
```

EX:

WAQ TO DISPLAY EMPLOYEE WHOSE COMMISSION IS NOT NULL ?

```
SELECT * FROM EMP WHERE COMM IS NOT NULL;
```

EX:

WAQ TO DISPLAY ENAME,JOB,SAL,COMM AND ALSO SAL+COMM FROM EMP TABLE WHOSE ENAME IS "SMITH" ?

```
SQL> SELECT ENAME,JOB,SAL,COMM,COMM+SAL FROM EMP WHERE ENAME='SMITH';

ENAME      JOB            SAL      COMM    COMM+SAL

---------- ---------    ---------- ----------   ---------------

SMITH      CLERK          800
```

<u>NOTE:</u>IF ANY ARITHMETIC OPERATOR IS PERFORMING SOME OPERATION WITH NULL THEN IT AGIAN RETURNS NULL ONLY.

```
EX:          IF   X=1000;

             I) X+NULL -------> 1000+0 -----------> 1000

             II) X-NULL -------> 1000-NULL ------------> NULL

             III) X*NULL ------> 1000*NULL ------------> NULL

             IV) X/NULL --------> 1000/NULL ------------> NULL
```

-TO OVERCOME THE ABOVE PROBLEM WE SHOULD USE A PREDEFINED FUNCTION IS CALLED AS "NVL()"

<u>NVL(EXP1,EXP2):</u>NVL STANDS FOR NULL VALUE. IT IS PRE-DEFINED FUNCTION. IS USED TO REPLACE A USER DEFINED VALUE IN PLACE OF NULL IN THE EXPRESSION. THIS FUNCTION IS HAVING TWO ARGUMENTS ARE EXPRESSION1 AND EXPRESSION2.

```
    >IF EXP1 IS NULL ------------> RETURNS EXP2 VALUE(USER DEFINED VALUE)

    >IF EXP1 IS NOT NULL -------> RETURNS EXP1 ONLY.
```

EX:

```
SQL> SELECT NVL(NULL,0) FROM DUAL;

NVL(NULL,0)

-----------

         0
```

```
SQL> SELECT NVL(NULL,100) FROM DUAL;

NVL(NULL,100)

-------------

        100

SQL> SELECT NVL(0,100) FROM DUAL;

NVL(0,100)

----------

       0

SQL> SELECT ENAME, JOB, SAL, COMM, NVL (COMM,0)+SAL FROM EMP WHERE
ENAME='SMITH';

ENAME     JOB          SAL      COMM          NVL(COMM,0)+SAL

---------- --------- ---------- ---------- ---------------    --------------------------

SMITH     CLERK         880                             880
```

<u>NVL2(EXP1, EXP2, EXP3):</u> PRE-DEFINED FUNCTION WHICH IS AN EXTENSION OF NVL() HAVING 3 ARGUMENTS ARE EXP1, EXP2, EXP3.

- **IF EXP1 IS NULL -----------> EXP3 VALUE(USER DEFINED VALUE)**
- **IF EXP1 IS NOT NULL ------> EXP2 VALUE(USER DEFINED VALUE)**

EX:WAQ TO UPDATE ALL EMPLOYEE COMMISSIONS IN A TABLE BASED ON THE FOLLOWING CONDITIONS?

I) IF EMPLOYEE COMM IS NULL THEN UPDATE THOSE EMPLOYEES COMM AS 600.

II) IF EMPLOYEE COMM IS NOT NULL THEN UPDATE THOSE EMPLOYEES COMM AS COMM+500.

SOL:-UPDATE EMP SET COMM=NVL2(COMM, COMM+500,600);

<u>SET OPERATORS:</u>

SQL SET OPERATORS ALLOWS COMBINE RESULTS FROM TWO OR MORE SELECT STATEMENTS. AT FIRST SIGHT THIS LOOKS SIMILAR TO SQL JOINS ALTHOUGH THERE IS BIG DIFFERENCE. SQL JOINS TENDS TO COMBINE COLUMNS I.E. WITH EACH ADDITIONALLY JOINED TABLE IT IS POSSIBLE TO SELECT MORE AND MORE COLUMNS. SQL SET OPERATORS ON THE OTHER HAND COMBINE ROWS FROM DIFFERENT QUERIES WITH STRONG PRECONDITIONS - ALL INVOLVED SELECTS MUST. JOINS WE ARE COLLECTING THE DATA FROM TWO TABLES WHEN THERE IS A COMMON DATA. BUT IN SET OPERATORS THE DATA IS NOT JOINED, IN THIS THE DATA IS MERGED

• RETRIEVE THE SAME NUMBER OF COLUMNS AND

• THE DATA TYPES OF CORRESPONDING COLUMNS IN EACH INVOLVED SELECT MUST BE COMPATIBLE (EITHER THE SAME OR WITH POSSIBILITY IMPLICITLY CONVERT TO THE DATA TYPES OF THE FIRST SELECT STATEMENT).

OPERATOR	RETURNS
UNION	ALL DISTINCT ROWS SELECTED BY EITHER QUERY
UNION ALL	ALL ROWS SELECTED BY EITHER QUERY, INCLUDING ALL DUPLICATES
INTERSECT	ALL DISTINCT ROWS SELECTED BY BOTH QUERIES
MINUS	ALL DISTINCT ROWS SELECTED BY THE FIRST QUERY BUT NOT THE SECOND

YOU CAN COMBINE MULTIPLE QUERIES USING THE SET OPERATORS UNION, UNION ALL, INTERSECT, AND MINUS. ALL SET OPERATORS HAVE EQUAL PRECEDENCE. IF A SQL STATEMENT CONTAINS MULTIPLE SET OPERATORS, THEN ORACLE DATABASE EVALUATES THEM FROM THE LEFT TO RIGHT UNLESS PARENTHESES EXPLICITLY SPECIFY ANOTHER ORDER.

THE CORRESPONDING EXPRESSIONS IN THE SELECT LISTS OF THE COMPONENT QUERIES OF A COMPOUND QUERY MUST MATCH IN NUMBER AND MUST BE IN THE SAME DATATYPE GROUP

IF COMPONENT QUERIES SELECT CHARACTER DATA, THEN THE DATATYPE OF THE RETURN VALUES ARE DETERMINED AS FOLLOWS:

• IF BOTH QUERIES SELECT VALUES OF DATATYPE CHAR OF EQUAL LENGTH, THEN THE RETURNED VALUES HAVE DATATYPE CHAR OF THAT LENGTH. IF THE QUERIES SELECT VALUES OF CHAR WITH DIFFERENT LENGTHS, THEN THE RETURNED VALUE IS VARCHAR2 WITH THE LENGTH OF THE LARGER CHARVALUE.

• IF EITHER OR BOTH OF THE QUERIES SELECT VALUES OF DATATYPE VARCHAR2, THEN THE RETURNED VALUES HAVE DATATYPE VARCHAR2.

IN QUERIES USING SET OPERATORS, ORACLE DOES NOT PERFORM IMPLICIT CONVERSION ACROSS DATATYPE GROUPS. THEREFORE, IF THE CORRESPONDING EXPRESSIONS OF COMPONENT QUERIES RESOLVE TO BOTH CHARACTER DATA AND NUMERIC DATA, ORACLE RETURNS AN ERROR.

SET OPERATOR GUIDELINES:-

- **THE EXPRESSIONS IN THE SELECT LISTS MUST MATCH IN NUMBER AND DATA TYPE.**
- **PARENTHESES CAN BE USED TO ALTER THE SEQUENCE OF EXECUTION.**
- **THE ORDER BY CLAUSE:**

  – CAN APPEAR ONLY AT THE VERY END OF THE STATEMENT

  – WILL ACCEPT THE COLUMN NAME, ALIASES FROM THE FIRST SELECT STATEMENT, OR THE POSITIONAL NOTATION
- **COLUMN NAMES FROM THE FIRST QUERY APPEAR IN THE RESULT.**

ADVANTAGE OF SET OPERATOR:-

☐ **USE A SET OPERATOR TO COMBINE MULTIPLE QUERIES INTO A SINGLE QUERY**

☐ **THESE OPERATORS ARE USED TO COMBINE THE INFORMATION OF SIMILAR DATA TYPE FROM ONE OR MORE THAN ONE TABLE.**

RESTRICTIONS ON THE SET OPERATORS:-

THE SET OPERATORS ARE SUBJECT TO THE FOLLOWING RESTRICTIONS:

- **THE ORDER BY CLAUSE DOESN'T RECOGNIZE THE COLUMN NAMES OF THE SECOND SELECT**
- **THE SET OPERATORS ARE NOT VALID ON COLUMNS OF TYPE BLOB, CLOB, BFILE, VARRAY, ORNESTED TABLE.**
- **THE UNION, INTERSECT, AND MINUS OPERATORS ARE NOT VALID ON LONG COLUMNS.**
- **SET OPERATIONS ARE NOT ALLOWED ON SELECT STATEMENTS CONTAINING TABLE COLLECTION EXPRESSIONS.**
- **SELECT STATEMENTS INVOLVED IN SET OPERATIONS CAN'T USE THE FOR UPDATE CLAUSE.**

SQL STATEMENTS CONTAINING THESE SET OPERATORS ARE REFERRED TO AS COMPOUND QUERIES, AND EACH SELECT STATEMENT IN A COMPOUND QUERY IS REFERRED TO AS A COMPONENT QUERY. TWO SELECTS CAN BE COMBINED INTO A COMPOUND QUERY BY A SET OPERATION ONLY IF THEY SATISFY THE FOLLOWING TWO CONDITIONS:

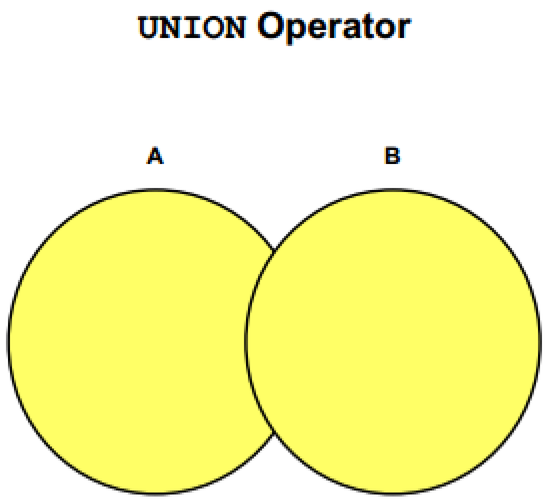
1. THE RESULT SETS OF BOTH THE QUERIES MUST HAVE THE SAME NUMBER OF COLUMNS.

2. THE DATATYPE OF EACH COLUMN IN THE SECOND RESULT SET MUST MATCH THE DATATYPE OF ITS CORRESPONDING COLUMN IN THE FIRST RESULT SET.

THE GENERIC SYNTAX OF A QUERY INVOLVING A SET OPERATION IS:

```
<COMPONENT QUERY>

    {UNION | UNION ALL | MINUS | INTERSECT}

    <COMPONENT QUERY>
```

- UNION:- UNION OPERATOR COMBINES THE RESULTS OF TWO SELECT STATEMENTS INTO ONE RESULT SET, AND THEN ELIMINATES ANY DUPLICATES ROWS FROM THE FINAL RESULT SET.



The UNION operator returns results from both queries after eliminating duplications.

EXAMPLE:-

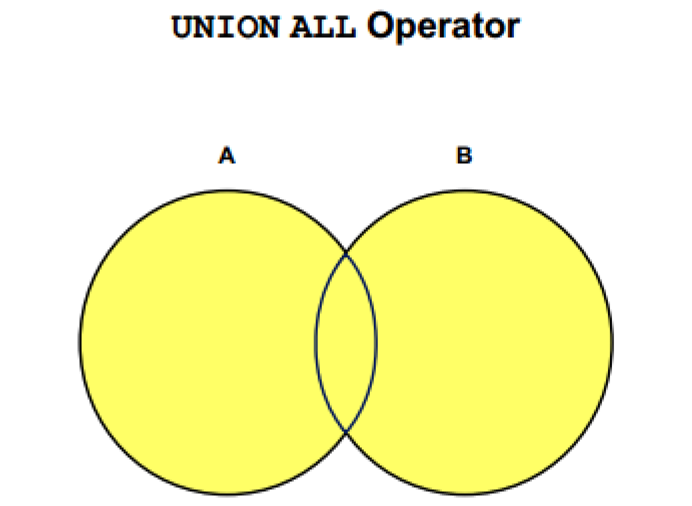
- SQL> SELECT EMPNO,ENAME FROM EMP WHERE DEPTNO=20
- UNION
- SELECT EMPNO,ENAME FROM EMP WHERE DEPTNO=30 ORDER BY 1;

EXPLINATION:- THE ABOVE STATEMENT COMBINES THE RESULTS OF TWO QUERIES WITH THE UNION OPERATOR, WHICH ELIMINATES DUPLICATE SELECTED ROWS. THIS STATEMENT SHOWS THAT YOU MUST MATCH DATATYPE (USING THE TO_CHAR FUNCTION) WHEN COLUMNS DO NOT EXIST IN ONE OR THE OTHER TABLE:

```
SQL> SELECT EMPNO,ENAME,JOB FROM EMP

WHERE DEPTNO=(SELECT DEPTNO FROM DEPT

WHERE  DNAME='SALES')

UNION

SELECT EMPNO,ENAME,JOB FROM EMP

WHERE DEPTNO=(SELECT DEPTNO FROM DEPT WHERE DNAME='ACCOUNTING')
ORDER BY 1;
```

UNION ALL:-

UNION ALL OPERATOR COMBINES THE RESULTS OF TWO SELECT STATEMENTS INTO ONE RESULT SET INCLUDING DUPLICATES.



The UNION ALL operator returns results from both queries, including all duplications.

EXAMPLE:-

```
SQL> SELECT EMPNO,ENAME FROM EMP WHERE DEPTNO=10

    UNION ALL

    SELECT EMPNO,ENAME FROM EMP WHERE DEPTNO=30

    ORDER BY 1;
```

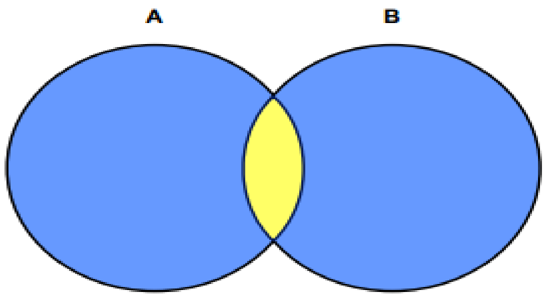
EXPLINATION:- THE UNION OPERATOR RETURNS ONLY DISTINCT ROWS THAT APPEAR IN EITHER RESULT, WHILE THE UNION ALL OPERATOR RETURNS ALL ROWS. THE UNION ALL OPERATOR DOES NOT ELIMINATE DUPLICATE SELECTED ROWS:

```
SQL> SELECT JOB FROM EMP WHERE DEPTNO=20

               UNION ALL

               SELECT JOB FROM EMP WHERE DEPTNO=30;
```

INTERSECT:-

INTERSECT OPERATOR RETURNS ONLY THOSE ROWS THAT ARE COMMON IN BOTH TABLES.

INTERSECT Operator



The INTERSECT operator returns rows that are common to both queries.

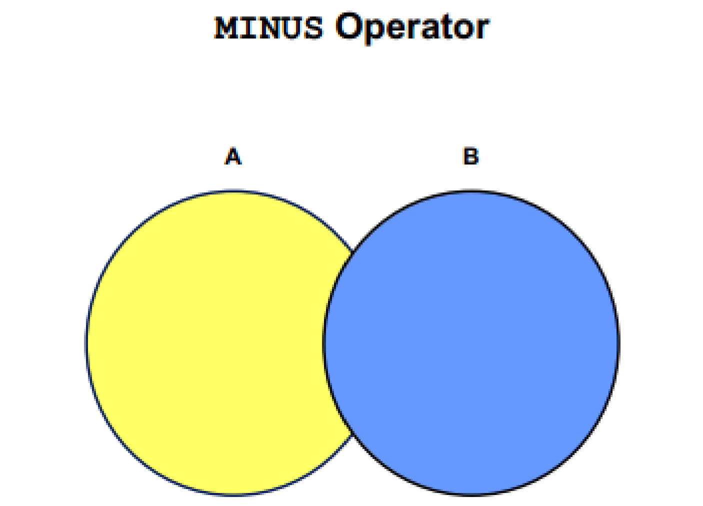
```
   SQL> SELECT EMPNO,ENAME FROM EMP WHERE DEPTNO=10

        INTERSECT

            SELECT EMPNO,ENAME FROM EMP WHERE DEPTNO=30

        ORDER BY 1;
```

EXPLINATION:- THE ABOVE STATEMENT COMBINES THE RESULTS WITH THE INTERSECT OPERATOR, WHICH RETURNS ONLY THOSE ROWS RETURNED BY BOTH QUERIES.

```
   SQL> SELECT JOB FROM EMP WHERE DEPTNO=20

               INTERSECT

               SELECT JOB FROM EMP WHERE DEPTNO=30;
```

MINUS:-

MINUS OPERATOR TAKES THE RESULT SET OF FIRST SELECT STATEMENT AND REMOVES THOSE ROWS THAT ARE RETURNED BY A SECOND SELECT STATEMENT.



The MINUS operator returns rows in the first query that are not present in the second query.

EXAMPLE:-

```
                SQL> SELECT EMPNO,ENAME FROM EMP
                           WHERE     DEPTNO=10
                            MINUS
                          SELECT EMPNO,ENAME FROM EMP
                             WHERE DEPTNO=30 ORDER BY 1;
```

EXPLINATION:- THE ABOVE STATEMENT COMBINES RESULTS WITH THE MINUS OPERATOR, WHICH RETURNS ONLY UNIQUE ROWS RETURNED BY THE FIRST QUERY BUT NOT BY THE SECOND:

```
 SQL>SELECT JOB FROM EMP WHERE DEPTNO=20
          MINUS
          SELECT JOB FROM EMP WHERE DEPTNO=30;
```

USING SET OPERATIONS TO COMPARE TWO TABLES:-

DEVELOPERS, AND EVEN DBAS, OCCASIONALLY NEED TO COMPARE THE CONTENTS OF TWO TABLES TO DETERMINE WHETHER THE TABLES CONTAIN THE SAME DATA. THE NEED TO DO THIS IS ESPECIALLY COMMON IN TEST ENVIRONMENTS, AS DEVELOPERS MAY WANT TO COMPARE A SET OF DATA GENERATED BY A PROGRAM UNDER TEST WITH A SET OF "KNOWN GOOD" DATA. COMPARISON OF TABLES IS ALSO USEFUL FOR AUTOMATED TESTING PURPOSES, WHEN WE HAVE TO COMPARE ACTUAL RESULTS WITH A GIVEN SET OF EXPECTED RESULTS. SQL'S SET OPERATIONS PROVIDE AN INTERESTING SOLUTION TO THIS PROBLEM OF COMPARING TWO TABLES.

THE FOLLOWING QUERY USES BOTH MINUS AND UNION ALL TO COMPARE TWO TABLES FOR EQUALITY. THE QUERY DEPENDS ON EACH TABLE HAVING EITHER A PRIMARY KEY OR AT LEAST ONE UNIQUE INDEX.

EXAMPLE:-

```
 SQL>DESC CUSTOMER_KNOWN_GOOD

 NAME                       NULL?    TYPE

 --------------------------- -------- ----------------

 CUST_NBR                      NOT NULL NUMBER(5)

 NAME                       NOT NULL VARCHAR2(30)

SELECT * FROM CUSTOMER_KNOWN_GOOD;


  CUST_NBR NAME

----------- -----------------------------

        1 SONY

        1 SONY

        2 SAMSUNG

        3 PANASONIC

        3 PANASONIC

        3 PANASONIC
```

```
SQL>DESC CUSTOMER_TEST

NAME                    NULL?    TYPE

 --------------------------- -------- ----------------

 CUST_NBR                    NOT NULL NUMBER(5)

 NAME                     NOT NULL VARCHAR2(30)


SQL>SELECT * FROM CUSTOMER_TEST;

  CUST_NBR NAME

----------- -----------------------------

        1 SONY

        1 SONY

        2 SAMSUNG

        2 SAMSUNG

        3 PANASONIC
```

AS WE CAN SEE THE CUSTOMER_KNOWN_GOOD AND CUSTOMER_TEST TABLES HAVE THE SAME STRUCTURE, BUT DIFFERENT DATA. ALSO NOTICE THAT NONE OF THESE TABLES HAS A PRIMARY OR UNIQUE KEY; THERE ARE DUPLICATE RECORDS IN BOTH. THE FOLLOWING SQL WILL COMPARE THESE TWO TABLES EFFECTIVELY.

EXAMPLE:-

```
     SQL>(SELECT * FROM CUSTOMER_KNOWN_GOOD

          MINUS

         SELECT * FROM CUSTOMER_TEST)

            UNION ALL

         (SELECT * FROM CUSTOMER_TEST

             MINUS

           SELECT * FROM CUSTOMER_KNOWN_GOOD);
```

EXPLINATION:-LET'S TALK A BIT ABOUT HOW THIS QUERY WORKS. WE CAN LOOK AT IT AS THE UNION OF TWO COMPOUND QUERIES. THE PARENTHESES ENSURE THAT BOTH MINUS OPERATIONS TAKE PLACE FIRST BEFORE THE UNION ALL OPERATION IS PERFORMED. THE RESULT OF THE FIRST MINUS QUERY WILL BE THOSE ROWS IN

CUSTOMER_KNOWN_GOOD THAT ARE NOT ALSO IN CUSTOMER_TEST. THE RESULT OF THE SECOND MINUS QUERY WILL BE THOSE ROWS IN CUSTOMER_TEST THAT ARE NOT ALSO IN CUSTOMER_KNOWN_GOOD. THE UNION ALL OPERATOR SIMPLY COMBINES THESE TWO RESULT SETS FOR CONVENIENCE. IF NO ROWS ARE RETURNED BY THIS QUERY, THEN WE KNOW THAT BOTH TABLES HAVE IDENTICAL ROWS. ANY ROWS RETURNED BY THIS QUERY REPRESENT DIFFERENCES BETWEEN THE CUSTOMER_TEST AND CUSTOMER_KNOWN_GOOD TABLES.

```
   SQL>(SELECT C1.*, COUNT(*)

FROM CUSTOMER_KNOWN_GOOD C1

GROUP BY C1.CUST_NBR, C1.NAME

MINUS

SELECT C2.*, COUNT(*)

FROM CUSTOMER_TEST C2

GROUP BY C2.CUST_NBR, C2.NAME)

UNION ALL

(SELECT C3.*, COUNT(*)

FROM CUSTOMER_TEST C3

GROUP BY C3.CUST_NBR, C3.NAME

MINUS

SELECT C4.*, COUNT(*)

FROM CUSTOMER_KNOWN_GOOD C4

GROUP BY C4.CUST_NBR, C4.NAME);
```

EXPLINATION:- THESE RESULTS INDICATE THAT ONE TABLE (CUSTOMER_KNOWN_GOOD) HAS ONE RECORD FOR "SAMSUNG", WHEREAS THE SECOND TABLE (CUSTOMER_TEST) HAS TWO RECORDS FOR THE SAME CUSTOMER. ALSO, ONE TABLE (CUSTOMER_KNOWN_GOOD) HAS THREE RECORDS FOR "PANASONIC", WHEREAS THE SECOND TABLE (CUSTOMER_TEST) HAS ONE RECORD FOR THE SAME CUSTOMER. BOTH THE TABLES HAVE THE SAME NUMBER OF ROWS (TWO) FOR "SONY", AND THEREFORE "SONY" DOESN'T APPEAR IN THE OUTPUT.

TIP: DUPLICATE ROWS ARE NOT POSSIBLE IN TABLES THAT HAVE A PRIMARY KEY OR AT LEAST ONE UNIQUE INDEX. USE THE SHORT FORM OF THE TABLE COMPARISON QUERY FOR SUCH TABLES.

TYPE CONVERSION FUNCTIONS:-

IF CORRESPONDING EXPRESSIONS DOES NOT BELONGS TO SAME DATA TYPE ALSO WE CAN RESTRICTIVE DATA FROM MULTIPLE QUERIES ,IN THIS CASE WE USE APPROPRIATE TYPE CONVERSION FUNCTION.

EXAMPLE:-

```
        SQL> SELECT ENAME "NAME",TO_NUMBER(NULL) "DEPTNO" FROM EMP

                        UNION

              SELECT TO_CHAR(NULL),DEPTNO FROM DEPT;
```

```
SQL> SELECT EMPNO,ENAME,SAL,TO_NUMBER(NULL) FROM EMP

              WHERE DEPTNO=20

           UNION

      SELECT EMPNO,ENAME,TO_NUMBER(NULL),HIREDATE FROM EMP

             WHERE DEPTNO=30;
```

USING NULLS IN COMPOUND QUERIES

WE DISCUSSED UNION COMPATIBILITY CONDITIONS AT THE BEGINNING OF THIS CHAPTER. THE UNION COMPATIBILITY ISSUE GETS INTERESTING WHEN NULLS ARE INVOLVED. AS WE KNOW, NULL DOESN'T HAVE A DATATYPE, AND NULL CAN BE USED IN PLACE OF A VALUE OF ANY DATATYPE. IF WE PURPOSELY SELECT NULL AS A COLUMN VALUE IN A COMPONENT QUERY, ORACLE NO LONGER HAS TWO DATATYPES TO COMPARE IN ORDER TO SEE WHETHER THE TWO COMPONENT QUERIES ARE COMPATIBLE. FOR CHARACTER COLUMNS, THIS IS NO PROBLEM. FOR EXAMPLE:

```
        SQL>SELECT 1 NUM, 'DEFINITE' STRING FROM DUAL

                     UNION

              SELECT 2 NUM, NULL STRING FROM DUAL;
```

NOTICE THAT ORACLE CONSIDERS THE CHARACTER STRING 'DEFINITE' FROM THE FIRST COMPONENT QUERY TO BE COMPATIBLE WITH THE NULL VALUE SUPPLIED FOR THE CORRESPONDING COLUMN IN THE SECOND COMPONENT QERY. HOWEVER, IF A NUMBER OR A DATE COLUMN OF A COMPONENT QUERY IS SET TO NULL, WE MUST EXPLICITLY TELL ORACLE WHAT "FLAVOR" OF NULL TO USE. OTHERWISE, WE'LL ENCOUNTER ERRORS. FOR

EXAMPLE:

```
SQL>SELECT 1 NUM, 'DEFINITE' STRING FROM DUAL

          UNION

       SELECT NULL NUM, 'UNKNOWN' STRING FROM DUAL;
```

ERROR AT LINE 1:

ORA-01790: EXPRESSION MUST HAVE SAME DATATYPE AS CORRESPONDING EXPRESSION

NOTE THAT THE USE OF NULL IN THE SECOND COMPONENT QUERY CAUSES A DATATYPE MISMATCH BETWEEN THE FIRST COLUMN OF THE FIRST COMPONENT QUERY, AND THE FIRST COLUMN OF THE SECOND COMPONENT QUERY. USING NULL FOR A DATE COLUMN CAUSES THE SAME PROBLEM, AS IN THE FOLLOWING

EXAMPLE:-

```
SQL>SELECT 1 NUM, SYSDATE DATES FROM DUAL

          UNION

       SELECT 2 NUM, NULL DATES FROM DUAL;

SELECT 1 NUM, SYSDATE DATES FROM DUAL

        ERROR AT LINE 1:
```

ORA-01790: EXPRESSION MUST HAVE SAME DATATYPE AS CORRESPONDING EXPRESSION

IN THESE CASES, WE NEED TO CAST THE NULL TO A SUITABLE DATATYPE TO FIX THE PROBLEM, AS IN THE FOLLOWING EXAMPLES:-

```
SQL>SELECT 1 NUM, 'DEFINITE' STRING FROM DUAL

          UNION

       SELECT TO_NUMBER(NULL) NUM, 'UNKNOWN' STRING FROM DUAL;
```

```
SQL>SELECT 1 NUM, SYSDATE DATES FROM DUAL

           UNION

    SELECT 2 NUM, TO_DATE(NULL) DATES FROM DUAL;
```

THIS PROBLEM OF UNION COMPATIBILITY WHEN USING NULLS IS ENCOUNTERED IN ORACLE8I. HOWEVER, THERE IS NO SUCH PROBLEM IN ORACLE9I, AS WE CAN SEE IN THE FOLLOWING EXAMPLES GENERATED FROM AN ORACLE9I DATABASE.

```
SQL>SELECT 1 NUM, 'DEFINITE' STRING FROM DUAL

           UNION

        SELECT NULL NUM, 'UNKNOWN' STRING FROM DUAL;

SQL>SELECT 1 NUM, SYSDATE DATES FROM DUAL

  UNION

 SELECT 2 NUM, NULL DATES FROM DUAL;
```

EX:-

-
```
SQL>SELECT JOB FROM EMP WHERE DEPTNO=20

      UNION

      SELECT JOB FROM EMP WHERE DEPTNO=30;
```

EX:-

```
SQL> SELECT * FROM EMP_HYD;


       EID ENAME             SAL
---------- ---------- ----------
      1021 SAI             85000
      1022 JONES             48000
      1023 ALLEN             35000
```

```
SQL> SELECT * FROM EMP_CHENNAI;

       EID ENAME             SAL

---------- ---------- ----------

      1021 SAI             85000

      1024 WARS            36000

      1025 MILLER          28000
```

EX:WAQ TO DISPLAY ALL EMPLOYEE DETAILS WHO ARE WORKING IN NARESHIT ORGANIZATION?

SELECT * FROM EMP_HYD UNION SELECT * FROM EMP_CHENNAI;

(OR)

SELECT * FROM EMP_HYD UNION ALL SELECT * FROM EMP_CHENNAI;

EX:WAQ TO DISPLAY EMPLOYEE DETAILS WHO ARE WORKING IN BOTH BRANCHS ?

SELECT * FROM EMP_HYD INTERSECT SELECT * FROM EMP_CHENNAI;

EX:WAQ TO DISPLAY EMPLOYEE DETAILS WHO ARE WORKING IN HYD BUT NOT IN CHENNAI BRANCHS ?

SELECT * FROM EMP_HYD MINUS SELECT * FROM EMP_CHENNAI;

EX:WAQ TO DISPLAY EMPLOYEE DETAILS WHO ARE WORKING IN CHENNAI BUT NOT IN HYD BRANCHS ?

SELECT * FROM EMP_CHENNAI MINUS SELECT * FROM EMP_HYD;

<u>FUNCTIONS IN ORACLE</u>:

> TO PERFORM TASK & MUST RETURN VALUE.
> ORACLE SUPPORTS TWO TYPES FUNCTIONS. THOSE ARE

1) PRE-DEFINE / BUILT IN FUNCTIONS (USE IN SQL & PL/SQL)
2) USER DEFINE FUNCTIONS (USEIN PL/SQL)

1) PRE-DEFINE FUNCTIONS:
--
> THESE ARE AGAIN CLASSIFIED INTO TWO CATEGORIES.

A) SINGLE ROW FUNCTIONS (SCALAR FUNCTIONS)
B) MULTIPLE ROW FUNCTIONS (GROUPING FUNCTIONS)

SINGLE ROW FUNCTIONS:

> THESE FUNCTIONS ARE RETURNS A SINGLE ROW (OR) A SINGLE VALUE.

> NUMERIC FUNCTIONS
> STRING FUNCTIONS
> DATE FUNCTIONS
> CONVERSION FUNCTIONS

HOW TO CALL AFUNCTION:

SYNTAX:

SELECT <FNAME>(VALUES) FROM DUAL;

WHAT IS DUAL:

> PRE-DEFINE TABLE IN ORACLE.
> HAVING SINGLE COLUMN & SINGLE ROW
> IS CALLED AS DUMMY TABLE IN ORACLE.
> TESTING FUNCTIONS(PRE-DEFINE & USER DEFINE) FUNCTIONALITIES.

TO VIEW STRC.OF DUAL TABLE:

```
SQL> DESC DUAL;
```

TO VIEW DATA OF DUAL TABLE:

```
SQL> SELECT * FROM DUAL;
```

NUMERIC FUNCTIONS:
----
1) ABS():
> CONVERTS (-VE) VALUE INTO (+VE) VALUE.

SYNTAX:

```
        ABS(NUMBER)
```

EX:
```
SQL> SELECT ABS(-12) FROM DUAL; --------> 12
SQL> SELECT ENAME,SAL,COMM,ABS(COMM-SAL) FROM EMP;
```

2) CEIL():
> RETURNS A VALUE WHICH IS GREATER THAN OR EQUAL TO GIVEN VALUE.

SYNTAX:

```
        CEIL(NUMBER)
```

EX:
```
SQL> SELECT CEIL(9.0) FROM DUAL;------9
SQL> SELECT CEIL(9.3) FROM DUAL;-------10
```

3) FLOOR():

SYNTAX:
```
        FLOOR(NUMBER)
```

EX:
```
SQL> SELECT FLOOR(9.0) FROM DUAL;------9
SQL> SELECT FLOOR(9.8) FROM DUAL;------9
```

4) MOD():
RETURNS REMAINDER VALUE.

SYNTAX:
```
        MOD(M,N)
```

EX:
```
SQL> SELECT MOD(10,2) FROM DUAL;-------0
```

5) POWER():
THE POWER OF GIVEN EXPRESSION

SYNTAX:
```
        POWER(M,N)
```

EX:
```
SQL> SELECT POWER(2,3) FROM DUAL;----------8
```

ROUND():
> NEAREST VALUE GIVEN EXPRESSION.

SYNTAX:
```
        ROUND(NUMBER,[DECIMAL PLACES])
```

EX:
```
SQL> SELECT ROUND(5.50) FROM DUAL;------6
SQL> SELECT ROUND(32.456,2) FROM DUAL;------32.46
```

TRUNC:

> RETURNS A VALUE WHICH WILL SPECIFIED NUMBER OF DECIMAL PLACES.

SYNTAX:
```
        TRUNC(NUMBER,DECIMAL PLACES)
```

EX:
```
SQL> SELECT TRUNC(5.50) FROM DUAL;---------5
SQL> SELECT TRUNC(32.456,2) FROM DUAL;----32.45
```

STRING FUNCTIONS:

LENGTH():

> LENGTH OF GIVEN STRING.

SYNTAX:
```
        LENGTH(STRING)
```

EX:
```
SQL> SELECT LENGTH('HELLO') FROM DUAL;-----------------------5
SQL> SELECT LENGTH('GOOD MORNING') FROM DUAL;--------12

SQL> SELECT ENAME,LENGTH(ENAME) FROM EMP;
SQL> SELECT * FROM EMP WHERE LENGTH(ENAME)=4;
```

LOWER():

TO CONVERT UPPER CASE CHAR'S INTO LOWER CASE CHAR'S.

SYNTAX:
```
        LOWER(STRING)
```

EX:
```
SQL> SELECT LOWER('HELLO') FROM DUAL;
SQL> UPDATE EMP SET ENAME=LOWER(ENAME) WHERE JOB='CLERK';
```

UPPER():

SYNTAX:
```
        UPPER(STRING)
```

EX:
```
SQL> SELECT LOWER('HELLO') FROM DUAL;
```

INITCAP():

TO CONVERT FIRST CHAR. IS CAPITAL.

SYNTAX:
```
        INITCAP(STRING)
```

EX:
```
SQL> SELECT INITCAP('HELLO') FROM DUAL;
SQL> SELECT INITCAP('GOOD MORNING') FROM DUAL;
```

LTIRM():

TO REMOVE UNWANTED SPACES (OR) UNWANTED CHARACTERS FROM LEFT
SIDE
OF GIVEN STRING.

SYNTAX:
```
        LTRIM(STRING1[,STRING2])
```

EX:
```
SQL> SELECT LTRIM('    SAI') FROM DUAL;
SQL> SELECT LTRIM('XXXXXXSAI','X') FROM DUAL;
SQL> SELECT LTRIM('123SAI','123') FROM DUAL;
```

RTRIM():

TO REMOVE UNWANTED SPACES (OR) UNWANTED CHARACTERS FROM RIGHT SIDE
OF GIVEN STRING.

SYNTAX:

```
        RTRIM(STRING1[,STRING2])
```

EX:

```
SQL> SELECT RTRIM('SAIXXXXXXX','X') FROM DUAL;
```

TRIM():

TO REMOVE UNWANTED SPACES (OR) UNWANTED CHARACTERS FROM BOTH SIDES
OF GIVEN STRING.

SYNTAX:

```
        TRIM('TRIMMING CHAR' FROM 'STRING')
```

EX:

```
SQL> SELECT TRIM('X' FROM 'XXXXXXSAIXXXXX') FROM DUAL;
```

LPAD():

TO FILL A STRING WITH SPECIFIC CHAR. ON LEFT SIDE OF GIVEN
STRING.

SYNTAX:

```
        LPAD(STRING1,LENGTH,STRING2)
```

EX:

```
SQL> SELECT LPAD('HELLO',10,'@') FROM DUAL;
@@@@@HELLO
```

RPAD():

TO FILL A STRING WITH SPECIFIC CHAR. ON RIGHT SIDE OF GIVEN STRING.

SYNTAX:

```
        RPAD(STRING1,LENGTH,STRING2)
```

EX:
```
SQL> SELECT RPAD('HELLO',10,'@') FROM DUAL;
HELLO@@@@@
```

CONCAT():

ADDING TWO STRING EXPRESSIONS.

SYNTAX:

```
        CONCAT(STRING1,STRING2)
```

EX:
```
SQL> SELECT CONCAT('GOOD','BYE') FROM DUAL;
```

REPLACE():

TO REPLACE ONE STRING WITH ANOTHER STRING.

SYNTAX:

```
        REPLACE(STRING1,STRING2,STRING3)
```

EX:
```
SQL> SELECT REPLACE('HELLO','ELL','XYZ') FROM DUAL;
HXYZO

SQL> SELECT REPLACE('HELLO','L','ABC') FROM DUAL;
HEABCABCO
```

TRANSLATE():

TO TRANSLATE A SINGLE CHAR WITH ANOTHER SINGLE CHAR.

SYNTAX:

```
        TRANSLATE(STRING1,STRING2,STRING3)
```

EX:
```
SQL> SELECT TRANSLATE('HELLO','ELO','XYZ') FROM DUAL;
HXYYZ
        SOL:   E = X , L=Y , O=Z
        HELLO =>HXYYZ
```

EX:
```
SQL> SELECT ENAME,SAL,TRANSLATE(SAL,'0123456789','$B@GH*V#T%')
SALARY FROM EMP;

ENAME           SAL              SALARY
----------      ---------------            -------------------------
SMITH           800              T$$

        SOL: 0=$,1=B,2=@,3=G,4=H,5=*,6=V,7=#,8=T,9=%.
```

SUBSTR():

IT RETURNS REQ.SUBSTRINGFROM GIVEN STRING EXPRESSION.

SYNTAX:

```
        SUBSTR(STRING1,<STARTING POSITION OF CHAR.>,<LENGTH OF
CHAR'S>)
```

EX:
```
SQL> SELECT SUBSTR('HELLO',2,3) FROM DUAL;
ELL

SQL> SELECT SUBSTR('WELCOME',4,2) FROM DUAL;
CO

SQL> SELECT SUBSTR('WELCOME',-6,3) FROM DUAL;
ELC
```

INSTR():

RETURNS OCCURENCE POSITION OF A CHAR. IN THE GIVEN STRING.

SYNTAX:

INSTR(STRING1,STRING2,<STARTING POSITION OF CHAR.>,<OCCURENCE POSITION OF CHAR.>)

EX:

```
SQL> SELECT INSTR('HELLO WELCOME','O') FROM DUAL;---------> 5
SQL> SELECT INSTR('HELLO WELCOME','Z') FROM DUAL;-----> 0
SQL> SELECT INSTR('HELLO WELCOME','O',1,2) FROM DUAL;-----11
SQL> SELECT INSTR('HELLO WELCOME','E',5,2) FROM DUAL;-------13
SQL> SELECT INSTR('HELLO WELCOME','E',1,4) FROM DUAL;--------8
```

NOTE:

POSITION OF CHAR'S ALWAYS FIXED EITHER COUNT FROM LEFT TO RIGHT (OR) RIGHT TO LEFT.

```
        SOL: HELLO   WELCOME
             12345 6 78910111213
```

EX:

```
SQL> SELECT INSTR('HELLO WELCOME','E',-1,3) FROM DUAL;--------2
SQL> SELECT INSTR('HELLO WELCOME','L',-4,3) FROM DUAL;-------3
SQL> SELECT INSTR('HELLO WELCOME','L',-6,3) FROM DUAL;----------0
```

DATE FUNCTIONS:

----

1) SYSDATE:

> CURRENT DATE INFORMATION OF THE SYSTEM.

EX:

```
SQL> SELECT SYSDATEFROM DUAL;
SQL> SELECT SYSDATE+10 FROM DUAL;
SQL> SELECT SYSDATE-10 FROM DUAL;
```

ADD_MONTHS():

> ADDING NO.OF MONTHS TO THE DATE.

SYNTAX:

```
        ADD_MONTHS(DATE,<NO.OF MONTHS>)
```

EX:

```
SQL> SELECT ADD_MONTHS(SYSDATE,3) FROM DUAL;
SQL> SELECT ADD_MONTHS(SYSDATE,-3) FROM DUAL;
```

LAST_DAY():

> RETURNS THE LAST DAY OF THE MONTH.

SYNTAX:

```
        LAST_DAY(DATE)
```

EX:

```
SQL> SELECT LAST_DAY(SYSDATE) FROM DUAL;
```

NEXT_DAY():

> RETURNS THE NEXT SPECIFIED DAY FROM THE GIVEN DATE.

SYNTAX:

```
        NEXT_DAY(DATE,'<DAY NAME>')
```

EX:

```
SQL> SELECT NEXT_DAY(SYSDATE,'SUNDAY') FROM DUAL;
```

MONTHS_BETWEEN():

> RETURNS NO.OF MONTHS BETWEEN TWO DATE EXPRESSIONS.

SYNTAX:

```
        MONTHS_BETWEEN(DATE1,DATE2)
```

EX:

```
SQL> SELECT MONTHS_BETWEEN('05-JAN-81','05-JAN-80') FROM DUAL;-----
12
SQL> SELECT MONTHS_BETWEEN('05-JAN-80','05-JAN-81') FROM DUAL;----- -
12
```

NOTE: HERE, DATE1 IS ALWAYS GREATER THAN DATE2 OTHERWISE ORACLE RETURNS NAGATIVE VALUE.

CONVERSION FUNCTIONS:
----

1. **TO_CHAR()**
2. **TO_DATE()**

TO_CHAR():

> DATE TYPE TO CHAR TYPE TO DISPLAY DATE IN DIFFERENT FROMAT.

SYNTAX:

```
TO_CHAR(DATE,[<FORMAT>])
```

YEAR FORMATS:

YYYY	-	2020
YY	-	20
YEAR	-	TWENTY TWENTY
CC	-	CENTUARY 21
AD / BC	-	AD YAER / BC YEAR

EX:

```
SQL> SELECT TO_CHAR(SYSDATE,'YYYY  YY  YEAR CC AD') FROM DUAL;


TO_CHAR(SYSDATE,'YYYYYYYEARCCAD')
-----------------------------------------------------
2020  20  TWENTY TWENTY 21 AD
```

Q: TO DISPLAY EMPLOYEE WHO ARE JOINED IN YEAR 1982 BY USING TO_CHAR() FUNCTION ?

SOL:

```
SQL> SELECT * FROM EMP WHERE TO_CHAR(HIREDATE,'YYYY')=1982;
```

Q: TO DISPLAY EMPLOYEE WHO ARE JOINED IN YEAR 1980,1982,1987 BY USING TO_CHAR() FUNCTION ?

SOL:

```
SQL> SELECT * FROM EMP WHERE TO_CHAR(HIREDATE,'YYYY')
IN(1980,1982,1987);
```

MONTH FORMAT:

MM   - MONTH NUMBER
MON - FIRST THREE CHAR FROM MONTH SPELLING
MONTH   - FULL NAME OF MONTH

EX:

```
SQL> SELECT TO_CHAR(SYSDATE,'MM MON MONTH') FROM DUAL;

TO_CHAR(SYSDATE,
----------------
08 AUG AUGUST
```

```
SQL> SELECT TO_CHAR(SYSDATE,'MM MON MONTH') FROM DUAL;

TO_CHAR(SYSDATE,
----------------
08 AUG AUGUST
```

Q: TO DISPLAY EMPLOYEE WHO ARE JOINED IN FEB,MAY,DEC MONTHS BY USING TO_CHAR() ?

SOL:

```
SQL> SELECT * FROM EMP WHERE TO_CHAR(HIREDATE,'MM') IN(02,05,12);
```

Q: TO DISPLAY EMPLOYEE WHO ARE JOINED IN FEB 1981 BY USING TO_CHAR() ?

SOL:

```
SQL> SELECT * FROM EMP WHERE TO_CHAR(HIREDATE,'MMYYYY')='021981';
```

DAY FORMATS:

DDD	- DAY OF THE YEAR.
DD	- DAY OF THE MONTH.
D	- DAY OF THE WEEK

SUN	- 1
MON	- 2
TUE	- 3
WEN	- 4
THU	- 5
FRI	- 6
SAT	- 7

DAY	- FULL NAME OF THE DAY
DY	- FIRST THREE CHAR'S OF DAY SPELLING

```
EX:SQL> SELECT TO_CHAR(SYSDATE,'DDD DD D DAY DY') FROM DUAL;

TO_CHAR(SYSDATE,'DDDDD
---------------------------------------------
220 07 6 FRIDAY    FRI
```

Q: TO DISPLAY EMPLOYEE WHO ARE JOINED ON "FRIDAY" BY USING TO_CHAR() ?

SOL:

```
SQL> SELECT * FROM EMP WHERE TO_CHAR(HIREDATE,'DAY')='FRIDAY';
```

Q: TO DISPLAY EMPLOYEE ON WHICH DAY EMPLOYEES ARE JOINED ?

SOL:

```
SQL> SELECT ENAME||' '||'JOINED ON'||' '||TO_CHAR(HIREDATE,'DAY') FROM
EMP;
```

NOTE:

IN ORACLE WHENEVER WE USING TO_CHAR() AND ALSO WITHIN TO_CHAR() WHEN WE USE DAY / MONTH FORMAT THEN ORACLE SERVER INTERNALLY ALLOCATE SOME EXTRA MEMORY FOR DAY/MONTH FORMAT OF DATA.

TO OVERCOME THE ABOVE PROBLEM THAT IS TO REMOVE EXTRA MEMORY WHICH WAS ALLOCATE BY ORACLE SERVER THEN WE USE A PRE-DEFINE SPECIFIER IS
CALLED "FM" (FILL MODE).

EX:

```
SELECT * FROM EMP WHERE TO_CHAR(HIREDATE,'FMDAY')='FRIDAY';
```

QUATER FORMAT:

Q - ONE DIGIT QUATEROF THE YEAR

1 - JAN - MAR
2 - APR - JUN
3 - JUL - SEP
4 - OCT - DEC

EX:

```
SQL> SELECT TO_CHAR(SYSDATE,'Q') FROM DUAL;
T
---
3
```

Q : WHO ARE JOINED IN 2ND QUATER OF 1981 ?

SOL:

```
SQL> SELECT * FROM EMP WHERE TO_CHAR(HIREDATE,'YYYY')='1981' AND
TO_CHAR(HIREDATE,'Q')=2;
```

WEEK FORMAT:

WW - WEEK OF THE YEAR
W - WEEK OF MONTH

EX:

```
SQL> SELECT TO_CHAR(SYSDATE,'WW W') FROM DUAL;

TO_C
---------
32  2
```

TIME FORMAT:

HH - HOUR PART
HH24 - 24 HRSFROMAT
MI - MINUTE PART
SS - SECONDS PART
AM / PM - AM TME (OR) PM TIME

EX:

```
SQL> SELECT TO_CHAR(SYSDATE,'HH:MI:SS AM') FROM DUAL;

TO_CHAR(SYS
-----------------------
12:04:21 PM
```

TO_DATE():

TO CONVERT CHAR TYPE TO ORACLE DATE FORMAT TYPE.

SYNTAX:

```
        TO_DATE(STRING[,FROMAT])
```

EX:

```
SQL> SELECT TO_DATE('08/AUGUST/2020') FROM DUAL;

TO_DATE('
---------
08-AUG-20

SQL> SELECT TO_DATE('08-AUG-2020')+10 FROM DUAL;

TO_DATE('
---------
18-AUG-20
```

MULTIPLE ROW FUNCTIONS:

THESE FUNCTIONS ARE RETURNS EITHER GROUP OF VALUES (OR) A SINGLE VALUE.

SUM():

> IT RETURNS SUM OF A SPECIFIC COLUMN VALUES.

EX:
```
SQL> SELECT SUM(SAL) FROM EMP;
SQL> SELECT SUM(SAL) FROM EMP WHERE JOB='CLERK';
```

AVG():

> IT RETURNS AVERAGE OF A SPECIFIC COLUMN VALUES.

EX:
```
SQL> SELECT AVG(SAL) FROM EMP;
SQL> SELECT AVG(SAL) FROM EMP WHERE DEPTNO=10;
```

MIN():

> IT RETURNS MIN.VALUEFROM GROUP OF VALUES.

EX:
```
SQL> SELECT MIN(HIREDATE) FROM EMP;
SQL> SELECT MIN(HIREDATE) FROM EMP WHERE JOB='MANAGER';
SQL> SELECT MIN(SAL) FROM EMP;
```

MAX():

> IT RETURNS MAX.VALUEFROM GROUP OF VALUES.

EX:
```
SQL> SELECT MAX(SAL) FROM EMP;
```

COUNT():

> IT RETURNS NO.OF ROWS IN A TBALE / NO.OF VALUES IN A COLUMN
> THREE TYPES,
I) COUNT(*)
II) COUNT(<COLUMN NAME>)
III) COUNT(DISTINCT <COLUMN NAME>)
EX:

```
TEST
--------
SNO  NAME
---  -----
101  A
102  B
103
104  C
105  A
106  C
```

COUNT(*):

> COUNTING ALL ROWS (DUPLICATES & NULLS) IN A TABLE.

EX:
```
SQL> SELECT COUNT(*) FROM TEST;

COUNT(*)
------------------
       6
```

COUNT(<COLUMN NAME>):
--
> COUNTING ALL VALUES INCLUDING DUPLICATE VALUES BUT NOT NULL VALUES
FROM A COLUMN.

EX:
```
SQL> SELECT COUNT(NAME) FROM TEST;

COUNT(NAME)
-----------------------
        5
```

COUNT(DISTINCT <COLUMN NAME>):

> COUNTING UNIQUE VALUES FROM A COLUMN.HERE "DISTINCT" KEYWORD IS ELIMINATING DUPLICATE VALUES.

EX:
```
SQL> SELECT COUNT(DISTINCT NAME) FROM TEST;-------→ 3
```

CLAUSES:

- CLAUSE IS STATEMENT WHICH IS USED TO ADD TO SQL PRE-DEFINE QUERY FOR PROVIDING ADDITIONAL FACILITIES ARE LIKE FILTERING ROWS, SORTING VALUES, GROUPING SIMILAR VALUES, FINDING

SUB TOTAL AND GRAND TOTAL BASED ON THE GIVEN VALUES AUTOMATICALLY.

- ORACLE SUPPORTS THE FOLLOWING CLAUSES. THOSE ARE,

> WHERE	- FILTERING ROWS (BEFORE GROUPING DATA)
> ORDER BY	- SORTING VALUES
> GROUP BY	- GROUPING SIMILAR DATA
> HAVING	- FILTERING ROWS (AFTER GROUPING DATA)
> ROLLUP	- FINDING SUB TOTAL & GRAND TOTAL (SINGLE COLUMN)
> CUBE	- FINDING SUB TOTAL & GRAND TOTAL (MULTIPLE COLUMNS)

SYNTAX:

<SQL PER-DEFINE QUERY> + <CLAUSES>;

WHERE CLAUSE:

> FILTERING ROWS IN ONE BY ONE MANNER BEFORE GROUPING DATA IN TABLE.

SYNTAX:

WHERE <FILTERING CONDITION>

EX:

```
SELECT * FROM EMP WHERE EMPNO=7788;

UPDATE EMP SET SAL=8500 WHERE JOB='CLERK';

DELETE FROM EMP WHERE DEPTNO=10;
```

NOTE: "WHERE" CLAUSE CAN BE USED IN "SELECT " ,"UPDATE" AND "DELETE" COMMANDS ONLY.

ORDER BY CLAUSE:

> SORTING VALUES BASED ON COLUMNS. IT CAN BE USED IN "SELECT" COMMAND ONLY.

> BY DEFAULT ORDER BY CLAUSE ARRANGE VALUES IN ASCENDING ORDER BUT IF WE WANT TO ARRANGE VALUES IN DESCENDING ORDER THEN WE USE "DESC" KEYWORD.

SYNTAX:

SELECT * / <LIST OF COLUMN NAMES> FROM <TN> ORDER BY <COLUMN NAME1> <ASC / DESC>,<COLUMN NAME2> <ASC/DESC>,…………;

EX1:

WAQ TO DISPLAY EMPLOYEE SALARIES IN ASCENDING ORDER?

SOL:

```
SQL> SELECT * FROM EMP ORDER BY SAL;
```

(OR)

```
SQL> SELECT SAL FROM EMP ORDER BY SAL;
```

EX2:

WAQ TO ARRANGE EMPLOYEE NAMES IN DESCENDING ORDER?

SOL:

```
SQL> SELECT ENAME FROM EMP ORDER BY ENAME DESC;
```

EX3:

WAQ TO DISPLAY EMPLOYEE WHO ARE WORKING IN THE

DEPTNO IS 20 AND ARRANGE THOSE EMPLOYEE SALARIES IN

DESCENDING ORDER?

SOL:

```
SQL> SELECT * FROM EMP WHERE DEPTNO=20 ORDER BY SAL DESC;
```

EX4:

WAQ TO ARRANGE EMPLOYEE DEPTNO'S IN ASCENDING ORDER

AND THOSE EMPLOYEE SALARIES IN DESCENDING ORDER FROM

EACH DEPTNO WISE?

SOL:

```
SQL> SELECT * FROM EMP ORDER BY DEPTNO, SAL DESC;
```

NOTE:

ORDER BY CLAUSE NOT ONLY ON COLUMN NAMES EVEN THOUGH

WE CAN APPLY ON POSITION OF COLUMN IN SELECT QUERY.

EX:

```
SQL> SELECT * FROM EMP ORDER BY 6;

SQL> SELECT ENAME, JOB, SAL FROM EMP ORDER BY 3;

SQL> SELECT ENAME, SAL FROM EMP ORDER BY 2;

SQL> SELECT SAL FROM EMP ORDER BY 1;
```

NOTE:

USING ORDER BY CLAUSE ON "NULL" VALUES COLUMN THEN ORACLE RETURNS "NULL" VALUES LAST IN ASCENDING ORDER AND "NULL" VALUES ARE DISPLAYED FIRST IN DESCENDING BY DEFAULT.

IF WE WANT TO CHANGE THIS DEFAULT ORDER OF "NULL" THEN WE USE NULL CLAUSES ARE “NULLS FIRST" AND " NULLS LAST ".

EX:

```
SQL> SELECT * FROM EMP ORDER BY COMM NULLS FIRST;

SQL> SELECT * FROM EMP ORDER BY COMM DESC NULLS LAST;
```

<u>GROUP BY:</u>

> GROUPING SIMILAR DATA BASED ON COLUMNS.

> WHEN WE USE "GROUP BY "WE MUST USE "AGGREGATIVE FUNCTIONS" ARE

SUM(),AVG(),MIN(),MAX(),COUNT().

> WHENEVER WE IMPLEMENT "GROUP BY" CLAUSE IN SELECT STATEMENT THEN

FIRST GROUPING SIMILAR DATA BASED COLUMNS AND LATER AN AGGREGATIVE FUNCTION/(S) WILL EXECUTE ON EACH GROUP OF DATA TO PRODUCE ACCURATE RESULT.

SYNTAX:

SELECT <COLUMN NAME1>,<COLUMN NAME2>,......,<AGGREGATIVE FUNCTION NAME1>,....... FROM <TN> GROUP BY <COLUMN NAME1>,<COLUMN NAME2>,..........................;

```
                                    GROUP BY

AGGREGATIVE FUNCTIONS.                                |

SUM (), AVG (),                  JOB (NO. OF IN EACH JOB)

MIN (), MAX (), COUNT ()                      |

CLERK |     ANALYST       |  PRESIDENT    |   MANAGER      |   SALESMAN

CLERK       ANALYST              (1)          MANAGER          SALESMAN

CLERK        (2)                              MANAGER          SALESMAN

CLERK                                         (3)              SALESMAN

  (4)                                                           (4)
```

EX1:

WAQ TO FIND OUT NO. OF EMPLOYEE WORKING IN EACH JOB?

SOL:

SQL> SELECT JOB,COUNT(*) NUM_OF_EMPLOYEE FROM EMP GROUP BY JOB;

EX2:

WAQ TO CALCULATE DEPARTMENT NUMBER WISE TOTAL SALARY ?

SOL:

SQL> SELECT DEPTNO,SUM(SAL) TOTAL_SALARY FROM EMP

GROUP BY DEPTNO ORDER BY DEPTNO;

EX3:

WAQ TO DISPLAY NO.OF EMPLOYEE WORKING IN EACH JOB ALONG WITH

DEPTNO WISE ?

SOL:

```
SQL> SELECT JOB,DEPTNO,COUNT(*) NUM_OF_EMPLOYEE FROM EMP

GROUP BY JOB,DEPTNO;
```

EX4:

WAQ TO CALCULATE DEPTNO WISE TOTALSALARY WHERE DEPTNO'S ARE 10,20 ?

SOL:

```
SQL> SELECT DEPTNO,SUM(SAL) TOTAL_SALARY FROM EMP

WHERE DEPTNO IN(10,20) GROUP BY DEPTNO;
```

EX5:

WAQ TO CALCULATE DEPTNO WISE AVG,MIN,MAX SALARIES ?

SOL:

```
SQL> SELECT DEPTNO,AVG(SAL) AVGSAL,MIN(SAL) MINSAL,MAX(SAL) MAXSAL FROM
EMP GROUP BY DEPTNO ORDER BY DEPTNO;
```

<u>HAVING:</u>

> FILTERING ROWS AFTER GROUPING DATA IN TABLE. IT CAN BE USED ALONG WITH "GROUP BY" CLAUSE.

SYNTAX:

```
SELECT <COLUMN NAME1>,<COLUMN NAME2>,......,<AGGREGATIVE FUNCTION
NAME1>,....... FROM <TN> GROUP BY <COLUMN NAME1>,<COLUMN
NAME2>,...............................HAVING <FILTERING CONDITION>;
```

EX1:

WAQ TO FIND OUT NO.OF EMPLOYEE OF EACH JOB IN WHICH JOB NO.OF EMPLOYEE

ARE MORE THAN 3 ?

SOL:

```
SQL> SELECT JOB,COUNT(*) FROM EMP GROUP BY JOB

HAVING COUNT(*)>3;
```

EX2:

WAQ TO DISPLAY SUM OF SALARY OF DEPTNO'S FROM EMP TABLE.IF SUM OF SALARY OF DEPTNO IS LESS THAN 9000 ?

SOL:

```
SQL> SELECT DEPTNO,SUM(SAL) FROM EMP GROUP BY DEPTNO
HAVING SUM (SAL)<9000;
```

<u>DIFF. B/W "WHERE" AND "HAVING" CLAUSE:</u>

WHERE	HAVING
1. WHERE CLAUSE CONDITION IS EXECUTED ON EACH ROW OF A TABLE.	1. HAVING CLAUSE CONDITION IS EXECUTED ON GROUP OF ROWS OF A TABLE.
2. IT CAN BE APPLY BEFORE GROUP BY CLAUSE.	2. IT CAN BE APPLY AFTER GROUP BY CLAUSE.
3. IT CANNOT SUPPORT AGGREGATIVE FUNCTIONS.	3. IT CAN SUPPORTS AGGREGATIVE FUNCTIONS.
4. WITHOUT GROUP BY WE CAN USE WHERE CLAUSE.	4. WITHOUT GROUP BY WE CANNOT USE HAVING CLAUSE.

<u>USING ALL CLAUSES IN A SINGLE SELECT STATEMENT:</u>

SYNTAX:

```
SELECT <COL1>,<COL2>,................,<AGGREGATIVE FUNCTION
NAME1>,....................
FROM <TABLE NAME> [WHERE <FILTERING CONDITION>
                        GROUP BY <COL1>,<COL2>,.....................
                        HAVING <FILTERING CONDITION>
                        ORDER BY <COL1> [ASC/DESC],<COL2> [ASC/DESC],.............
                      ];
```

EX:

```
 SELECT DEPTNO,COUNT(*) FROM EMP
 WHERE SAL>1000
 GROUP BY DEPTNO
 HAVING COUNT(*)>3
 ORDER BY DEPTNO;
```

```
DEPTNO   COUNT(*)
------ ----------
   20        4
   30        5
```

ORDER OF EXECUTION:

> FROM
> WHERE
> GROUP BY
> HAVING
> SELECT
> ORDER BY

ROLLUP & CUBE:

> SPECIAL CLAUSES.
> TO FINDING SUB TOTAL & GRAND TOTAL BASED ON COLUMNS.
> WORKING ALONG WITH "GROUP BY" CLAUSE.
> ROLLUP WILL FIND SUB & GRAND TOTAL BASED ON A SINGLE COLUMN.
> CUBE WILL FIND SUB & GRAND TOTAL BASED ON MULTIPLE COLUMNS.

SYNTAX:

```
        GROUP BY ROLLUP (<COL1>,<COL2>,<COL3>,........,<COL N>)
```

EX. ON ROLLUP WITH A SINGLE COLUMN:

```
SQL> SELECT DEPTNO, COUNT (*) FROM EMP GROUP BY ROLLUP(DEPTNO);

   DEPTNO          COUNT(*)
----------         ----------
     10            3
     20            5
     30            6
                   14
```

EX. ON ROLLUP WITH MULTIPLE COLUMNS:

```
SQL> SELECT DEPTNO, JOB, COUNT (*) FROM EMP GROUP BY ROLLUP(DEPTNO,JOB);
```

NOTE: IN THE ABOVE EX. ROLLUP IS FINDING SUB & GRAND TOTAL BASED ON A SINGLE COLUMN (DEPTNO).IF WE WANT TO FIND SUB & GRAND TOTAL THEN USE "CUBE" CLAUSE.

SYNTAX:

```
        GROUP BY CUBE (<COL1>,<COL2>,....................,<COL N>)
```

EX. ON CUBE WITH A SINGLE COLUMN:

```
SQL> SELECT DEPTNO, COUNT (*) FROM EMP GROUP BY CUBE(DEPTNO) ORDER BY
DEPTNO;
```

EX. ON CUBE WITH MULTIPLE COLUMNS:

```
SQL> SELECT DEPTNO, JOB, COUNT (*) FROM EMP GROUP BY CUBE(DEPTNO , JOB)

       ORDER BY DEPTNO;
```

JOINS:

- IN RDBMS DATA CAN BE STORED IN MULTIPLE TABLES.IF WE WANT TO RETRIEVE OUR REQUIRED DATA / INFORMATION FROM THOSE MULTIPLE TABLES THEN WE USE A TECHNIQUE IS CALLED AS "JOINS".

- JOINS ARE USED TO RETRIEVE DATA / INFORMATION FROM MULTIPLE TABLES AT A TIME.

- ORACLE IS SUPPORTING THE FOLLOWING JOINS ARE,

1. INNER JOINS:

- **EQUI JOIN**
- **NON-EQUI JOIN**

2. OUTER JOINS:

- **LEFT OUTER JOIN**
- **RIGHT OUTER JOIN**
- **FULL OUTER JOIN**

3. CROSS JOIN

4. NATURAL JOIN

5. SELF JOIN

SYNTAX:

=======

SELECT * / <LIST OF COLUMNS> FROM <TN1> <JOIN KEY> <TN2> ON <JOINING CONDITION>;

1. INNER JOINS:

===========

- EQUI JOIN:

=========

- WHEN WE ARE RETRIEVING DATA FROM MULTIPLE TABLES BASED ON

AN " = " OPERATOR IS CALLED AS "EQUI JOIN".

- WHEN WE USE EQUI JOIN WE SHOULD HAVE AT LEAST ONE COMMON COLUMN IN BETWEEN TABLES.

- A COMMON COLUMN DATATYPE MUST BE MATCH IN BOTH TABLES.

- THERE IS NO NEED TO HAVE RELATIONSHIP BETWEEN TABLES IT IS A OPTIONAL.

- RETRIEVING MATCHING ROWS FROM TABLES.

SYNTAX FOR JOINING CONDITION:

=======================

ON <TN1 / TABLE ALIAS NAME1>.<COMMON COLUMN> = <TN2 / TABLE ALIAS NAME2>.<COMMON COLUMN>;

DEMO_TABLES:

============

```
SQL> SELECT * FROM COURSE;


      CID       CNAME                CFEE

----------      ----------           ----------

        1       C          500

        2       ORACLE               2500

        3       JAVA                 3500


SQL> SELECT * FROM STUDENTS;


     STID          SNAME            CID

----------         ----------  ----------

     1021      SMITH                2

     1022      ALLEN                3

     1023      WARD                 2

     1024      JONES
```

EX:

WAQ TO RETRIEVE STUDENTS AND THEIR CORRESPONDING COURSE DETAILS FROM STUDENTS,COURSE TABLES?

```
SQL> SELECT * FROM STUDENTS INNER JOIN COURSE ON
STUDENTS.CID=COURSE.CID;
```

(OR)

```
SQL> SELECT * FROM STUDENTS S INNER JOIN COURSE  C ON S.CID=C.CID;
```

RULE OF JOINS:

==============

- A ROW IN A TABLE IS COMPARING WITH ALL ROWS OF ANOTHER TABLE.

EX:

WAQ TO RETRIEVE STUDENTS,COURSE DETAILS FROM STUDENTS,COURSE TABLES WHO ARE JOINED

IN ORACLE COURSE?

```
SQL> SELECT STID,SNAME,CNAME,CFEE FROM STUDENTS S

  2  INNER JOIN COURSE C ON S.CID=C.CID WHERE CNAME='ORACLE';
```

(OR)

```
SQL> SELECT STID,SNAME,CNAME,CFEE FROM STUDENTS S

  2  INNER JOIN COURSE C ON S.CID=C.CID AND CNAME='ORACLE';
```

EX:

WAQ TO RETRIEVE EMPLOYEES DETAILS WHO ARE WORKING IN THE LOCATION IS "CHICAGO"

FROM EMP,DEPT TABLES?

```
SQL> SELECT ENAME,DNAME,LOC,D.DEPTNO FROM EMP E

  2  INNER JOIN DEPT D ON E.DEPTNO=D.DEPTNO

  3  AND LOC='CHICAGO';
```

EX:

WAQ TO DISPLAY SUM OF SALARIES OF DEPARTMENTS FROM EMP,DEPT TABLES?

```
SQL> SELECT DNAME,SUM(SAL) FROM EMP E INNER JOIN DEPT D

  2  ON E.DEPTNO=D.DEPTNO GROUP BY DNAME;
```

EX:

WAQ TO DISPLAY DEPTNOS,SUM OF SALARIES OF DEPARTMENTS FROM EMP,DEPT TABLES?

```
SQL> SELECT D.DEPTNO,DNAME,SUM(SAL)FROM EMP E INNER JOIN

  2  DEPT D ON E.DEPTNO=D.DEPTNO GROUP BY D.DEPTNO,DNAME;
```

EX:

WAQ TO DISPLAY SUM OF SALARIES OF DEPARTMENTS FROM EMP,DEPT TABLES IF THE SUM

OF SALARY OF A DEPARTMENT IS MORE THAN 10000?

```
SQL> SELECT DNAME,SUM(SAL) FROM EMP E INNER JOIN DEPT D

  2  ON E.DEPTNO=D.DEPTNO GROUP BY DNAME

  3  HAVING SUM(SAL)>10000;
```

NON-EQUI JOIN:

==========

- RETRIEVING DATA FROM MULTIPLE TABLES BASED ON ANY OPERATOR EXCEPT AN" = " OPERATOR.

- IT SUPPORTS THE FOLLOWING OPERATORS ARE < , > , <= , >= , != , AND, OR,BETWEEN,....ETC.

DEMO_TABLES:

==========

```
SQL> SELECT * FROM TEST1;

       SNO              NAME
----------              ----------
        10              SMITH
        20              ALLEN

SQL> SELECT * FROM TEST2;

       SNO              SAL
----------              ----------
        10       23000
        30       33000
```

EX:

```
SQL> SELECT * FROM TEST1 T1 INNER JOIN TEST2 T2 ON T1.SNO < T2.SNO;
SQL> SELECT * FROM TEST1 T1 INNER JOIN TEST2 T2 ON T1.SNO > T2.SNO;
SQL> SELECT * FROM TEST1 T1 INNER JOIN TEST2 T2 ON T1.SNO <= T2.SNO;
SQL> SELECT * FROM TEST1 T1 INNER JOIN TEST2 T2 ON T1.SNO >= T2.SNO;
SQL> SELECT * FROM TEST1 T1 INNER JOIN TEST2 T2 ON T1.SNO != T2.SNO;
```

EX:

WAQ TO DISPLAY EMPLOYEES WHOSE SALARY IS BETWEEN LOW SALARY AND HIGH SALARYFROM EMP,SALGRADE TABLES?

```
SQL> SELECT ENAME,SAL,LOSAL,HISAL FROM EMP INNER JOIN

        SALGRADE ON SAL BETWEEN LOSAL AND HISAL;

                   (OR)

SQL> SELECT ENAME,SAL,LOSAL,HISAL FROM EMP INNER JOIN SALGRADE

        ON (SAL>=LOSAL) AND (SAL<=HISAL);
```

OUTER JOINS:

=============

- IN THE ABOVE INNER JOINS WE ARE LOOSING MATCHING ROWS FOR NON-EQUI JOIN

AND UNMATCHING ROWS FOR EQUI JOIN.

- TO OVERCOME THE ABOVE PROBLEMS WE USE "OUTER JOINS" TECHNIQUE FOR

RETREIVING MATCHING AND ALSO UNMATCHING ROWS FROM MULTIPLE TABLES.

- OUTER JOINS ARE THREE TYPES,

I) LEFT OUTER JOIN:

==================

- RETRIEVING MATCHING ROWS FROM BOTH TABLES BUT UNMATCHING ROWS FROM

LEFT SIDE TABLE ONLY.

EX:

```
SQL> SELECT * FROM STUDENTS S LEFT OUTER JOIN COURSE C ON S.CID=C.CID;
```

II) RIGHT OUTER JOIN:

==================

- RETRIEVING MATCHING ROWS FROM BOTH TABLES BUT UNMATCHING ROWS FROM

RIGHT SIDE TABLE ONLY.

EX:

SQL> SELECT * FROM STUDENTS S RIGHT OUTER JOIN COURSE C ON S.CID=C.CID;

III) FULL OUTER JOIN:

==================

- IT IS A COMBINATION OF LEFT OUTER AND RIGHT OUTER JION.

- BY USING FULL OUTER JOIN WE CAN RETRIEVE MATCHING AND UNMATCING ROWS

FROM BOTH TABLES AT A TIME.

EX:

SQL> SELECT * FROM STUDENTS S FULL OUTER JOIN COURSE C ON S.CID=C.CID;

CROSS JOIN:

===========

- JOINING TABLES WITHOUT ANY CONDITION.

- IT IS A DEFAULT JOIN IN DATABASE.

- IN CROSS JOIN MECHANISM,A TABLE IS HAVING (M) NO.OF ROWS AND ANOTHER

TABLE IS HAVING (N) NO.OF ROWS THEN THE RESULT OF CROSS JOIN IS (MXN) ROWS.

EX:

```
SQL> SELECT * FROM STUDENTS CROSS JOIN COURSE;
```

DEMO_TABLES:

==============

```
SQL> SELECT * FROM ITEMS1;

       SNO INAME           PRICE
---------- ----------      ----------
         1 PIZZA           180
         2 BURGER          75

SQL> SELECT * FROM ITEMS2;

       SNO INAME           PRICE
---------- ----------      ----------
       101 PEPSI           25
       102 COCACOLA        20
```

EX:

```
SQL> SELECT I1.INAME,I1.PRICE,I2.INAME,I2.PRICE,
  2  I1.PRICE+I2.PRICE AS TOTAL_BILL_AMOUNT FROM
  3  ITEMS1 I1 CROSS JOIN ITEMS2 I2;
```

NATURAL JOIN:

==============

- IT IS A SIMILAR TO EQUI JOIN BUT PREPARING A JOIN CONDITION BY ORACLE SERVER

BY DEFAULT BASED ON AN " = " OPERATOR ONLY.

- IT IS ALSO RETRIEVING MATCHING ROWS ONLY JUST LIKA A EQUI-JOIN.

- THIS JOIN WILL REMOVE DUPLICATE COLUMNS FROM A RESULT SET.

EX:

```
SQL> SELECT * FROM STUDENTS S NATURAL JOIN COURSE C;
```

SELF JOIN:

==========

- THIS TECHNIQUE WILL WORK ON A SINGLE TABLE IN DATABASE.

- SELF JOIN IS NOTHING BUT JONING A TABLE BY ITSELF (OR)

COMPARING A TABLE DATA BY ITSELF.

- WHEN WE USE SELF JOIN WE SHOULD CREATE ALIAS NAMES ON A TABLE NAME

OTHERWISE WE CANNOT IMPLEMENT SELF JOIN MECHANISM.

- ONCE WE CREATED ALIAS NAME ON A TABLE NAME INTERNALLY ORACLE SERVER

IS PREPARING VIRTUAL TABLE ON EACH ALIAS NAME.

- WE CAN CREATE ANY NO.OF ALIAS NAMES ON A SINGLE TABLE BUT EACH ALIAS

NAME MUST BE DIFFERENT.

- THERE ARE TWO CASES WHERE WE CAN USE SELF JOIN TECHNIQUE.

CASE-1: COMPARING A SINGLE COLUMN VALUES BY ITSELF WITH IN THE TABLE.

CASE-2: COMPARING TWO DIFFERENT COLUMNS VALUES TO EACH OTHER WITH IN THE TABLE.

CASE-1: COMPARING A SINGLE COLUMN VALUES BY ITSELF WITH IN THE TABLE.

==

EX:

WAQ TO DISPLAY EMPLOYEES WHO ARE WORKING IN THE SAME LOCATION WHERE THE

EMPLOYEE "SMITH" IS ALSO WORKING?

```
SQL> SELECT T1.ENAME,T1.LOC FROM TEST T1 JOIN TEST T2
  2  ON T1.LOC=T2.LOC AND T2.ENAME='SMITH';
```

EX:

WAQ TO FETCH EMPLOYEES WHOSE SALARY IS SAME AS THE EMPLOYEE SCOTT SALARY?

```
SQL> SELECT E1.ENAME,E1.SAL FROM EMP E1 JOIN EMP E2
  2  ON E1.SAL=E2.SAL AND E2.ENAME='SCOTT';
```

CASE-2: COMPARING TWO DIFFERENT COLUMNS VALUES TO EACH OTHER WITH IN THE TABLE:

==

EX:

WAQ TO DISPLAY MANAGERS AND THEIR EMPLOYEES FROM EMP TABLE?

```
SQL> SELECT M.ENAME AS MANAGERS,E.ENAME AS EMPLOYEES
   FROM EMP E JOIN EMP M ON M.EMPNO=E.MGR
```

EX:

WAQ TO DISPLAY EMPLOYEES WHO ARE WORKING UNDER "BLAKE" MANAGER?

```
SQL> SELECT M.ENAME AS MANAGERS,E.ENAME AS EMPLOYEES
  2     FROM EMP E JOIN EMP M ON M.EMPNO=E.MGR
  3  AND M.ENAME='BLAKE';
```

EX:

WAQ TO DISPLAY MANAGER OF "BLAKE"?

```
SQL> SELECT M.ENAME AS MANAGERS
  2     FROM EMP E JOIN EMP M ON M.EMPNO=E.MGR
  3  AND E.ENAME='BLAKE';
```

EX:

WAQ TO DISPLAY EMPLOYEES WHO ARE JOINED BEFORE THEIR MANAGER?

```
SQL> SELECT E.ENAME AS EMPLOYEES,E.HIREDATE AS E_DOJ,
  2  M.ENAME AS MANAGER,M.HIREDATE AS M_DOJ FROM
  3  EMP E JOIN EMP M ON M.EMPNO=E.MGR
  4  AND E.HIREDATE < M.HIREDATE;
```

EX:

WAQ TO DISPLAY EMPLOYEES WHOSE SALARY IS MORE THAN THEIR MANAGER?

```
SQL> SELECT E.ENAME AS EMPLOYEE,E.SAL AS E_SALARY,
  2  M.ENAME AS MANAGER,M.SAL AS M_SALARY FROM
  3  EMP E JOIN EMP M ON M.EMPNO=E.MGR AND
  4  E.SAL>M.SAL;
```

HOW TO JOIN MORE THAN TWO TABLES:

==============================

SYNTAX:

======

```
SELECT * FROM <TN1> <JOIN KEY> <TN2> ON <JOIN CONDITION1>
<JOIN KEY> <TN3> ON <JOIN CONDITION2>
<JOIN KEY> <TN4> ON <JOIN CONDITION3>
.................................................................
.................................................................
<JOIN KEY> <TN N> ON <JOIN CONDITION N-1>;
```

EX:

```
SQL> SELECT * FROM STUDENTS S INNER JOIN COURSE C
  2  ON S.CID=C.CID INNER JOIN REGISTER R ON C.CID=R.CID;
```

DATAINTEGRITY:

TO MAINTAIN ACCURATE & CONSISTENCY DATA IN DB TABLES.THIS DATAINTEGRITY AGAIN CLASSIFIED INTO TWO WAYS THOSE ARE

1. DECLARATIVE / PRE -DEFINE DATAINTEGRITY (USING CONSTRAINTS)

2. PROCEDURAL / USER - DEFINE DATAINTEGRITY (USING TRIGGERS)

DECLARATIVE / PRE -DEFINE DATAINTEGRITY:

- THIS DATAINTEGRITY CAN BE IMPLEMENTED WITH HELP OF "CONSTRAINTS”. THESE ARE AGAIN THREE TYPES,

- ENTITY INTEGRITY

- REFERENCIAL INTEGRITY

- DOMAIN INTEGRITY

ENTITY INTEGRITY:

- IT ENSURE THAT EACH ROW UNIQULY IDENTIFY IN A TABLE.TO IMPLEMENT THIS MECHANISM WE USE PRIMARY KEY OR UNIQUE CONSTRAINT.

REFERENCIAL INTEGRITY:

- IT ENSURE THAT TO CREATE RELATIONSHIP BETWEEN TABLES.TO IMPLEMENT THIS MECHANISM THEN WE USE FOREIGN KEY (REFERENCIAL KEY).

DOMAIN INTEGRITY:

- DOMAIN IS NOTHING BUT COLUMN.IT ENSURE THAT TO CHECK VALUES WITH USER DEFINE CONDITION BEFORE ACCEPTING VALUES INTO A COLUMN.TO PERFORM THIS MECHANISM WE USE CHECK, DEFAULT, NOT NULL CONSTRAINTS.

CONSTRAINTS:

- CONTSRAINTS ARE USED TO RESTRICTED UNWANTED(INVALID) DATA INTO TABLE.ALL DATABASES ARE SUPPORTING THE FOLLOWING CONSTRAINT TYPES ARE

- UNIQUE

- NOT NULL

- CHECK

- PRIMARY KEY

- FOREIGN KEY (REFERENCES KEY)

- DEFAULT

- ALL DATABASES ARE SUPPORTING THE FOLLOWING TWO TYPES OF METHODS TO DEFINE CONSTRAINTS.THOSE ARE

I) COLUMN LEVEL:

- IN THIS METHOD WE ARE DEFINING CONSTRAINTS ON INDIVIDUAL COLUMNS.

SYNTAX:

CREATE TABLE <TN> (<COLUMN NAME1><DATATYPE>[SIZE] <CONSTRAINT TYPE>, ……);

II) TABLE LEVEL:

- IN THIS METHOD WE ARE DEFINING CONSTRAINTS AFTER ALL COLUMNS ARE DECLARED. (END OF THE TABLE DEFINITION)

SYNTAX:

CREATE TABLE <TN>(<COLUMN NAME1><DATATYPE>[SIZE], <COLUMN NAME2><DATATYPE>[SIZE], ………………………………., <CONSTRAINT TYPE> (<COLUMN NAME1>, <COLUMN NAME2>, ……………..));

UNIQUE:

- TO RESTRICTED DUPLICATE VALUES BUT ACCEPTING NULLS INTO A COLUMN.

I) COLUMN LEVEL:

EX:

SQL> CREATE TABLE TEST1(SNO INT UNIQUE, NAME VARCHAR2(10) UNIQUE);

TESTING:

SQL> INSERT INTO TEST1 VALUES(1,'A');---ALLOWED

SQL> INSERT INTO TEST1 VALUES(1,'A');---NOT ALLOWED

SQL> INSERT INTO TEST1 VALUES(NULL,NULL);----ALLOWED

II) TABLE LEVEL:

EX:

SQL> CREATE TABLE TEST2(SNO INT, NAME VARCHAR2(10), UNIQUE (SNO, NAME));

TESTING:

SQL> INSERT INTO TEST2 VALUES(1,'A');---ALLOWED

SQL> INSERT INTO TEST2 VALUES(1,'A');---NOT ALLOWED

SQL> INSERT INTO TEST2 VALUES(2,'A');---ALLOWED

NOTE: WHEN WE APPLY UNIQUE CONSTRAINT ON GROUP OF COLUMNS THEN WE CALLED AS "COMPOSITE UNIQUE" CONSTRAINT.IN THIS MECHANISM INDIVIDUAL COLUMNS ARE ACCEPTING DUPLICATE VALUES BUT DUPLICATE COMBINATION OF COLUMNS DATA IS NOT ALLOWED.

NOT NULL:

- TO RESTRICTED NULLS BUT ACCEPTING DUPLICATE VALUES INTO A COLUMN.

- NOT NULL CONSTRAINT NOT SUPPORTS "TABLE LEVEL".

COLUMN LEVEL:

EX:

SQL> CREATE TABLE TEST3(STID INT NOT NULL, SNAME VARCHAR2(10) NOT NULL);

TESTING:

SQL> INSERT INTO TEST3 VALUES(101,'A');----ALLOW

SQL> INSERT INTO TEST3 VALUES(101,'A');---ALLOW

SQL> INSERT INTO TEST3 VALUES(NULL,NULL);----NOT ALLOW

CHECK:

- TO CHECK VALUES WITH USER DEFINED CONDITION BEFORE ACCEPTING VALUES INTO A COLUMN.

I) COLUMN LEVEL:

EX:

SQL> CREATE TABLE TEST4(EID INT, SALNUMBER(10) CHECK(SAL>=10000));

TESTING:

SQL> INSERT INTO TEST4 VALUES(1021,9500);---NOT ALLOW

SQL> INSERT INTO TEST4 VALUES(1021,10000);---ALLOW

II) TABLE LEVEL:

EX:

SQL> CREATE TABLE TEST5(ENAME VARCHAR2(10),SAL NUMBER(10),

CHECK(ENAME=LOWER(ENAME) AND SAL>8000));

TESTING:

SQL> INSERT INTO TEST5 VALUES('SAI',7500);---NOT ALLOW

SQL> INSERT INTO TEST5 VALUES('SAI',9500);---ALLOW

PRIMARY KEY:

============

- TO RESTRICTED DUPLICATES & NULLS INTO A COLUMN.

- A TABLE SHOULD HAVE ONLY "ONE PRIMARY KEY".

I) COLUMN LEVEL:

EX:

SQL> CREATE TABLE TEST6(PCODE INT PRIMARY KEY, PNAME VARCHAR2(10)

PRIMARY KEY);

ERROR AT LINE 1:

ORA-02260: TABLE CAN HAVE ONLY ONE PRIMARY KEY.

SOL:

SQL> CREATE TABLE TEST6(PCODE INT PRIMARY KEY, PNAME VARCHAR2(10));

TESTING:

SQL> INSERT INTO TEST6 VALUES(10021,'C');-----ALLOW

SQL> INSERT INTO TEST6 VALUES(10021,'C++');----NOT ALLOW

SQL> INSERT INTO TEST6 VALUES(NULL,'C++');----NOT ALLOW

II) TABLE LEVEL:

EX:

SQL> CREATE TABLE TEST7(BCODE INT, BNAME VARCHAR2(10),

LOC VARCHAR2(10), PRIMARY KEY(BCODE, BNAME));

TESTING:

```
SQL> INSERT INTO TEST7 VALUES(1021,'SBI','SRNAGAR');---ALLOW

SQL> INSERT INTO TEST7 VALUES(1021,'SBI','MADHAPUR');---NOT ALLOW

SQL> INSERT INTO TEST7 VALUES(1022,'SBI','MADHAPUR');---ALLOW

SQL> INSERT INTO TEST7 VALUES(1021,'ICICI','SRNAGAR');----ALLOW
```

NOTE: WHEN WE APPLY PRIMARY KEY CONSTRAINT ON GROUP OF COLUMNS THEN WE CALLED AS "COMPOSITE PRIMARY KEY" CONSTRAINT.IN THIS MECHANISM INDIVIDUAL COLUMNS ARE ACCEPTING DUPLICATE VALUES BUT DUPLICATE COMBINATION OF COLUMNS DATA IS NOT ALLOWED.

FOREIGN KEY (REFERENCES KEY):

- FOREIGN KEY IS USED TO ESTABLISH RELATIONSHIP BETWEEN TABLES.

BASIC THINGS:

1. WE HAVE A COMMON COLUMN NAME(OPTIONAL) BUT RECOMMENDED.

2. COMMON COLUMN DATATYPE MUST MATCH.

3. ONE TABLE FOREIGN KEY MUST BELONGS TO ANOTHER TABLE PRIMARY KEY.

AND HERE PRIMARY KEY & FOREIGN KEY COLUMN MUST BE COMMON COLUMN.

4. PRIMARY KEY TABLE IS CALLED AS "PARENT TABLE" AND FOREIGN KEY TABLE IS CALLED AS "CHID TABLE"(I.E PARENT & CHILD RELATIONSHIP).

5. FOREIGN KEY COLUMN VALUES SHOULD BE MATCH WITH PRIMARY KEY COLUMN

VALUES ONLY.

6. GENERALLY PRIMARY KEY IS NOT ALLOWED DUPLICATE AND NULL VALUES WHERE AS FOREIGN KEY IS ALLOWED DUPLICATE & NULL VALUES.

I) COLUMN LEVEL:

SYNTAX:

```
<COMMON COLUMN NAME OF CHILD><DT>[SIZE] REFERENCES

<PARENT TABLE NAME> (<COMMON COLUMN NAME OF PARENT>)
```

EX:

STEP1:

```
SQL> CREATE TABLE DEPT1(DEPTNO INT PRIMARY KEY, DNAME VARCHAR2(10));
```

STEP2:

```
SQL> INSERT INTO DEPT1 VALUES(10,'ORACLE');

SQL> INSERT INTO DEPT1 VALUES(20,'JAVA');
```

STEP3:

```
SQL> CREATE TABLE EMP1(EID INT PRIMARY KEY, ENAME VARCHAR2(10),

         DEPTNO INT REFERENCES DEPT1(DEPTNO));
```

STEP4:

```
SQL>INSERT INTO EMP1 VALUES(1021,'SAI',10);

SQL>INSERT INTO EMP1 VALUES(1022,'JONES',10);

SQL>INSERT INTO EMP1 VALUES(1023,'MILLER',20);
```

- ONCE WE ESTABLISH RELATIONSHIP BETWEEN TABLES THERE ARE TWO RULES ARE COME INTO PICTURE.THOSE ARE

1) INSERTION RULE:

- WE CANNOT INSERT VALUES INTO FOREIGN KEY(REFERENCES KEY) COLUMN THOSE VALUES ARE NOT EXISTING UNDER PRIMARY KEY COLUMN OF PARENT TABLE.

EX:

```
SQL> INSERT INTO EMP1 VALUES(1026,'SCOTT',30);

ERROR AT LINE 1:

ORA-02291: INTEGRITY CONSTRAINT (SCOTT.SYS_C005468) VIOLATED - PARENT KEY NOT
FOUND.
```

2) DELETION RULE:

- WHEN WE TRY TO DELETE A RECORD FROM PARENT TABLE AND THOSE ASSOCIATED RECORDS ARE AVAILABLE IN CHILD TABLE THEN ORACLE RETURNS

AN ERROR IS,

EX:

```
SQL> DELETE FROM DEPT1 WHERE DEPTNO=20;

ERROR AT LINE 1:

ORA-02292: INTEGRITY CONSTRAINT (SCOTT.SYS_C005468) VIOLATED - CHILD RECORD
FOUND.
```

NOTE:

IF WE WANT TO DELETE A RECORD FROM PARENT TABLE WHEN THEY HAVE CORRESPONDING CHILD RECORDS IN CHILD TABLE THEN WE PROVIDE SOME SET OF RULES TO PERFORM DELETE OPERATIONS ON PARENT TABLE.THOSE RULES ARE CALLED AS "CASCADE RULES".

I) ON DELETE CASCADE

II) ON DELETE SET NULL

I) ON DELETE CASCADE:

- WHENEVER WE ARE DELETING A RECORD FROM PARENT TABLE THEN THAT ASSOCIATED CHILD RECORDS ARE DELETED FROM CHILD TABLE AUTOMATICALLY.

EX:

STEP1:

```
SQL> CREATE TABLE DEPT2(DEPTNO INT PRIMARY KEY,DNAME VARCHAR2(10));
```

STEP2:

```
SQL> INSERT INTO DEPT2 VALUES(10,'ORACLE');

SQL> INSERT INTO DEPT2 VALUES(20,'JAVA');
```

STEP3:

```
SQL> CREATE TABLE EMP2(EID INT PRIMARY KEY, ENAME VARCHAR2(10),

        DEPTNO INT REFERENCES DEPT2(DEPTNO) ON DELETE CASCADE);
```

STEP4:

```
SQL>INSERT INTO EMP2 VALUES(1021,'SAI',10);

SQL>INSERT INTO EMP2 VALUES(1022,'JONES',10);

SQL>INSERT INTO EMP2 VALUES(1023,'MILLER',20);
```

TESTING:

```
SQL> DELETE FROM DEPT2 WHERE DEPTNO=20; ----ALLOWED
```

<u>II) ON DELETE SET NULL:</u>

WHENEVER WE ARE DELETING A RECORD FROM PARENT TABLE THEN THAT ASSOCIATED CHILD RECORDS ARE SET TO NULL IN CHILD TABLE AUTOMATICALLY.

EX:

STEP1:

```
SQL> CREATE TABLE DEPT3(DEPTNO INT PRIMARY KEY,DNAME VARCHAR2(10));
```

STEP2:

```
SQL> INSERT INTO DEPT3 VALUES(10,'ORACLE');

SQL> INSERT INTO DEPT3 VALUES(20,'JAVA');
```

STEP3:

```
SQL> CREATE TABLE EMP3(EID INT PRIMARY KEY, ENAME VARCHAR2(10),

     DEPTNO INT REFERENCES DEPT3(DEPTNO) ON DELETE SET NULL);
```

STEP4:

```
SQL>INSERT INTO EMP3 VALUES(1021,'SAI',10);

SQL>INSERT INTO EMP3 VALUES(1022,'JONES',10);

SQL>INSERT INTO EMP3 VALUES(1023,'MILLER',20);
```

TESTING:

```
SQL> DELETE FROM DEPT3 WHERE DEPTNO=10; ----ALLOWED
```

SYNTAX FOR TABLE LEVEL:

CREATE TABLE<TN>(<COL1><DT>[SIZE], <COL2><DT>[SIZE], ..., FOREIGN KEY(<COL1>, <COL2>, ...) REFERENCES <PARENT TABLE NAME>(<COL1>, <COL2>, ...);

PRE-DEFINE CONSTRAINT NAME:

- WHENEVER WE ARE APPLYING CONSTRAINT ON A PARTICULAR COLUMN THEN DB SERVER(SYSTEM) INTERNALLY GENERATE AN UNIQUE ID NUMBER (OR) AN UNIQUE CONSTRAINT KEY NAME AUTOMATICALLY FOR IDENTIFYING A CONSTRANT.

EX:

SQL> CREATE TABLE TEST8(SNO INT PRIMARY KEY, NAME VARCHAR2(10));

TESTING:

SQL> INSERT INTO TEST8 VALUES(1,'A');---ALLOWED

SQL> INSERT INTO TEST8 VALUES(1,'B');---NOT ALLOWED

ERROR:

ORA-00001: UNIQUE CONSTRAINT (SCOTT.SYS_C005475) VIOLATED

USER DEFINE CONSTRAINT NAME:

- IN PLACE OF PRE-DEFINE CONSTRAINT NAME WE CAN ALSO CREATE A USER DEFINED CONSTRAINT KEY NAME (OR) CONSTRAINT ID FOR IDENTIFYING A CONSTRAINT.

SYNTAX:

<COLUMN NAME><DT>[SIZE] CONSTRAINT <USER DEFINED CONSTRAINT NAME><CONSTRAINT TYPE>

EX:

SQL> CREATE TABLE TEST10(SNO INT CONSTRAINT PK_SNO PRIMARY KEY, NAME VARCHAR2(10) CONSTRAINT UQ_NAME UNIQUE);

TESTING:

SQL> INSERT INTO TEST10 VALUES(1,'A');

```
SQL> INSERT INTO TEST10 VALUES(1,'B');

ERROR AT LINE 1:

ORA-00001: UNIQUE CONSTRAINT (SCOTT.PK_SNO) VIOLATED


SQL> INSERT INTO TEST10 VALUES(2,'A');

ERROR AT LINE 1:

ORA-00001: UNIQUE CONSTRAINT (SCOTT.UQ_NAME) VIOLATE
```

DATA DICTIONARIES (OR) READ ONLY TABLES:

- WHENEVER WE ARE INSTALLING ORACLE S/W INTERNALLY ORACLE SERVER IS CREATING SOME PRE-DEFINE TABLES ARE CALLED AS "DATA DICTIONARIES". THESE DATA DICTIONARIES ARE USED TO STORE THE INFORMATION ABOUT DB OBJECTS SUCH AS TABLES,INDEXES,VIEWS,SYNONYMS,.............ETC.

- THESE DATA DICTIONARIES ARE SUPPORTING "SELECT" AND "DESC" COMMANDS ONLY.SO THAT DATA DICTIONARIES ARE ALSO CALLED AS "READ ONLY TABLES" IN ORACLE DB.

- IF WE WANT TO VIEW ALL DATA DICTIONARIES IN ORACLE DB THEN WE FOLLOW THE FOLLOWING SYNTAX IS,

SYNTAX:

```
SQL> SELECT * FROM DICT;
```

NOTE1:

IF WE WANT TO VIEW ALL CONSTRAINTS INFROMATION OF A PARTICULAR TABLE THEN WE USE "USER_CONSTRAINTS" DATA DICTIONARY.

EX:

```
SQL> DESC USER_CONSTRAINTS;

SQL> SELECT CONSTRAINT_NAME, CONSTRAINT_TYPE FROM USER_CONSTRAINTSWHERE
TABLE_NAME='TEST10';


CONSTRAINT_NAME              CONSTRAINT_TYPE

----------------------------------        ------------------------------

PK_SNO                                P

UQ_NAME                               U
```

NOTE2:

- IF WE WANT TO VIEW CONSTRAINT NAME ALONG WITH COLUMN NAME OF A PARTICULAR TABLE THEN WE USE " USER_CONS_COLUMNS " DATADICTIONARY.

EX:

```
SQL> DESC USER_CONS_COLUMNS;

SQL> SELECT CONSTRAINT_NAME, COLUMN_NAME FROM USER_CONS_COLUMNSWHERE
TABLE_NAME='TEST10';


CONSTRAINT_NAME            COLUMN_NAME

---------------------------------         ---------------------------

UQ_NAME                                   NAME

PK_SNO                                    SNO
```

NOTE3:

TO VIEW A LOGICAL CONDITION OF CHECK CONSTRAINT THEN WE NEED TO CALL "SEARCH_CONDITION" COLUMN FROM "USER_CONSTRAINTS" DATA DICTIONARY.

EX:

```
SQL> CREATE TABLE TEST11(SNO INT, SAL NUMBER(10) CHECK(SAL>5000));
```

EX:

```
SQL> DESC USER_CONSTRAINTS;

SQL> SELECT SEARCH_CONDITION FROM USER_CONSTRAINTS

        WHERE TABLE_NAME='TEST11';


SEARCH  CONDITION

SAL>5000
```

NOTE4:

TO VIEW ALL COLUMNS INFORMATION OF A PARTICULAR TABLE THEN WE USE "USER_TAB_COLUMNS" DATADICTIONARY.

EX:

```
SQL> DESC USER_TAB_COLUMNS;

SQL> SELECT COLUMN_NAME FROM USER_TAB_COLUMNS

        WHERE TABLE_NAME='EMP';
```

<u>HOW TO FIND NO. OF ROWS IN A TABLE:</u>

```
SQL> SELECT COUNT (*) FROM EMP;

COUNT (*)

    14
```

<u>HOW TO FIND NO. OF COLUMNS IN A TABLE:</u>

```
SQL> SELECT COUNT (*) FROM USER_TAB_COLUMNS WHERE TABLE_NAME='EMP';


COUNT (*)

      8
```

<u>HOW TO ADD CONSTRAINTS TO AN EXISTING TABLE:</u>

SYNTAX:

=======

ALTER TABLE <TN> ADD CONSTRAINT <CONSTRAINT KEY NAME><CONSTRAINT TYPE>(<COLUMN NAME>);

EX:

```
SQL> CREATE TABLE TEST12(EID INT, ENAME VARCHAR2(10),SAL NUMBER(10));
```

<u>I) ADDING PRIMARY KEY:</u>

```
SQL> ALTER TABLE TEST12 ADD CONSTRAINT PK_EID PRIMARY KEY(EID);
```

<u>II) ADDING UNIQUE, CHECK CONSTRAINT:</u>

```
SQL> ALTER TABLE TEST12 ADD CONSTRAINT UQ_ENAME UNIQUE(ENAME);

SQL> ALTER TABLE TEST12 ADD CONSTRAINT CHK_SAL CHECK(SAL=10000);
```

III) ADDING "NOT NULL" CONSTRAINT:

SYNTAX:

ALTER TABLE <TN> MODIFY <COLUMN NAME> CONSTRAINT <CONSTRAINT KEY NAME>

NOT NULL;

EX:

```
SQL> ALTER TABLE TEST12 MODIFY ENAME CONSTRAINT NN_ENAME NOT NULL;
```

IV) ADDING FOREIGN KEY CONSTRAINT:

SYNTAX:

ALTER TABLE <TN> ADD CONSTRAINT <CONSTRAINT KEY NAME>FOREIGN KEY(<COMMON COLUMN OF CHILD TABLE>) REFERENCES<PARENT TABLE>(<COMMON COLUMN OF PARENT TABLE>) ON DELETE CASCADE /ON DELETE SET NULL;

EX:

```
SQL> CREATE TABLE TEST13(DNAME VARCHAR2(10), EID INT);

TABLE CREATED.
```

EX:

```
SQL> ALTER TABLE TEST13 ADD CONSTRAINT FK_EID FOREIGN KEY(EID)

        REFERENCES TEST12(EID) ON DELETE CASCADE;
```

HOW TO DROP CONSTRAINT FROM AN EXISTING TABLE:

SYNTAX:

ALTER TABLE <TN> DROP CONSTRAINT <CONSTRAINT KEY NAME>;

I) DROPPING PRIMARY KEY:

METHOD1:

```
SQL> ALTER TABLE TEST13 DROP CONSTRAINT FK_EID; -------FIRST

SQL> ALTER TABLE TEST12 DROP CONSTRAINT PK_EID; --------LATER
```

METHOD2:

- WHEN WE DROP PRIMARY KEY ALONG WITH FOREIGN KEY CONSTRAINT FROM PARENT AND CHILD TBALES THEN WE USE "CASCADE" STATEMENT.

EX:

```
SQL> ALTER TABLE TEST12 DROP CONSTRAINT PK_EID CASCADE;
```

II) DROPPING UNIQUE, CHECK, NOT NULL CONSTRANT:

```
SQL> ALTER TABLE TEST12 DROP CONSTRAINT UQ_ENAME;

SQL> ALTER TABLE TEST12 DROP CONSTRAINT CHK_SAL;

SQL> ALTER TABLE TEST12 DROP CONSTRAINT NN_ENAME;
```

HOW TO RENAME CONSTRAINT NAME:

SYNTAX:

ALTER TABLE <TN> RENAME CONSTRAINT < OLD CONSTRAINT NAME> TO <NEW CONSTRAINT NAME>;

EX:

```
SQL> CREATE TABLE TEST14(SNO INT PRIMARY KEY);

SQL> SELECT CONSTRAINT_NAME FROM USER_CONS_COLUMNS

         WHERE TABLE_NAME='TEST14';


CONSTRAINT_NAME

------------------------------

SYS_C005489

SQL> ALTER TABLE TEST14 RENAME CONSTRAINT SYS_C005489 TO SNO_PK;

SQL> SELECT CONSTRAINT_NAME FROM USER_CONS_COLUMNS

        WHERE TABLE_NAME='TEST14';
```

```
CONSTRAINT_NAME
------------------------------
SNO_PK
```

HOW DISABLE / ENABLE CONSTRAINT:

- BY DEFAULT, CONSTRAINTS ARE ENABLE MODE.IF WE WANT TO DISABLE CONSTRAINT TEMP. THEN WE USE "DISABLE" KEYWORD.IT MEANS THAT CONSTRAINT IS EXISTING IN DB BUT NOT WORK TILL IT MAKE AS "ENABLE".

- WHENEVER WE WANT TO COPY HUGE AMOUNT OF DATA FROM ONE TABLE TO ANOTHER TABLE THERE WE USE "DISABLE" KEYWORD.

SYNTAX:

```
ALTER TABLE <TN> DISABLE / ENABLE CONSTRAINT <CONSTRAINT KEY NAME>;
```

EX:

```
SQL> CREATE TABLE TEST15(ENAME VARCHAR2(10), SALNUMBER (10)
     CHECK(SAL=5000));

SQL> INSERT INTO TEST15 VALUES('SAI',5000); ----ALLOWED

SQL> INSERT INTO TEST15 VALUES('JONES',3000); -----NOT ALLOWED
ERROR AT LINE 1:
ORA-02290: CHECK CONSTRAINT (SCOTT.SYS_C005492) VIOLATED

SQL> ALTER TABLE TEST15 DISABLE CONSTRAINT SYS_C005492;
SQL> INSERT INTO TEST15 VALUES('JONES',3000); ----ALLOWED
```

EX:

```
SQL> ALTER TABLE TEST15 ENABLE CONSTRAINT SYS_C005492;
ERROR AT LINE 1:
ORA-02293: CANNOT VALIDATE (SCOTT.SYS_C005492) - CHECK CONSTRAINT VIOLATED
```

- TO OVERCOME THE ABOVE PROBLEM THEN WE USE "NOVALIDATE" KEYWORD AT THE TIME OF ENABLE CONSTRAINT.ONCE WE USE "NOVALIDATE" KEYWORDTHEN CONSTRAINT IS ENABLE WITH "NOVALIDATE" AND ORACLE SERVER WILL NOT CHECK EXISTING DATA IN TABLE BUT CHECKING NEW DATA WHILE INSERTING TIME.

EX:

```
SQL> ALTER TABLE TEST15 ENABLE NOVALIDATE CONSTRAINT SYS_C005492;

TABLE ALTERED.
```

TESTING:

```
SQL> INSERT INTO TEST15 VALUES('SCOTT',6000); ---NOT ALLOWED

ERROR AT LINE 1:

ORA-02290: CHECK CONSTRAINT (SCOTT.SYS_C005492) VIOLATED


SQL> INSERT INTO TEST15 VALUES('SCOTT',5000); ------ALLOWED
```

DEFAULT CONSTRAINT:

- IT A SPECIAL TYPE OF CONSTRAINT WHICH IS USED TO ASSIGN A USER DEFINE DEFAULT VALUE TO A COLUMN.

SYNTAX:

```
<COLUMN NAME><DATATYPE>[SIZE] DEFAULT <VALUE / EXPRESSION>
```

EX:

```
SQL> CREATE TABLE TEST17(SNO INT, SALNUMBER (10) DEFAULT 5000);

TABLE CREATED.
```

TESTING:

```
SQL> INSERT INTO TEST17 VALUES (1,8500);

SQL> INSERT INTO TEST17(SNO)VALUES (2);
```

OUTPUT:

```
      SNO        SAL
---------------            ----------
        1                  8500
        2                  5000
```

HOW TO ADD DEFAULT VALUE TO AN EXISTING TABLE:

SYNTAX:

```
ALTER TABLE <TN> MODIFY <COLUMN NAME> DEFAULT <VALUE / EXPRESSION>;
```

EX:

```
SQL> CREATE TABLE TEST18(EID INT, SALNUMBER (10));

SQL> ALTER TABLE TEST18 MODIFY SAL DEFAULT 8000;
```

TESTING:

```
SQL> INSERT INTO TEST18(EID)VALUES (1021);
```

NOTE:

- IF WE WANT TO VIEW DEFAULT VALUE OF A COLUMN THEN WE USE "USER_TAB_COLUMNS" DATADICTIONARY.

EX:

```
SQL> DESC USER_TAB_COLUMNS;

SQL> SELECT COLUMN_NAME, DATA_DEFAULT FROM USER_TAB_COLUMNS

        WHERE TABLE_NAME='TEST18';


COLUMN_NAME                  DATA_DEFAULT
--------------------------   -------------------------------
SAL                          8000
```

HOW TO REMOVE DEFAULT VALUE OF A COLUMN:

EX:

```
ALTER TABLE TEST18 MODIFY SAL DEFAULT NULL;
```

```
COLUMN_NAME              DATA_DEFAULT
-------------------------  -------------------------------
SAL                      NULL
```

TRANSACTION CONTROL LANGUAGE (TCL)

TRANSACTION:

- A TRANSACTION IS A UNIT OF WORK THAT IS PERFORMED AGAINST DATABASE.

EX:

- IF WE ARE INSERTING / UPDATING / DELETING DATA TO / FROM A TABLE THEN WE ARE PERFORMING A TRANSACTION ON A TABLE.

- TO MANAGE TRANSACTIONS ON DATABASE TABLES THEN WE PROVIDE THE FOLLOWING COMMAND ARE

1) COMMIT

2) ROLLBACK

3) SAVEPOINT

COMMIT:

- THIS COMMAND IS USED TO MAKE A TRANSACTION IS PERMANENT.THESE ARE TWO TYPES.

I) IMPLICIT COMMIT:

- THESE TRANSACTIONS ARE COMMITTED BY SYSTEM (ORACLE DB) BY DEFAULT.

EX: DDL COMMANDS

II) EXPLICIT COMMIT:

- THESE TRANSACTIONS ARE COMMITTED BY USER AS PER REQUIREMENT.

EX: DML COMMANDS

EX:

```
SQL> CREATE TABLE BRANCH (BCODE INT, BNAME VARCHAR2(10), BLOC
VARCHAR2(10));
```

STEP1:

```
SQL> INSERT INTO BRANCH VALUES (1021,'SBI','HYD');
SQL> COMMIT;
```

STEP2:

```
SQL> UPDATE BRANCH SET BLOC='MUMBAI' WHERE BCODE=1021;
SQL> COMMIT;
```

STEP3:

```
SQL> DELETE FROM BRANCH WHERE BCODE=1021;
SQL> COMMIT;
```

NOTE: THE ABOVE DML OPERATIONS ARE NOT POSSIBLE TO "ROLLBACK" BECAUSE THOSE OPERATIONS ARE COMMITTED BY USER EXPLICITLY.

ROLLBACK:

- THIS COMMAND IS USED TO CANCEL TRANSACTION.BUT ONCE A TRANSACTION IS COMMITTED THEN WE CANNOT "ROLLBACK(CANCEL)".

EX:

```
SQL> DELETE FROM BRANCH WHERE BCODE=1021;
SQL> ROLLBACK;
```

NOTE: THE ABOVE "DELETE" OPERATION IS NOT COMMITTED SO THAT USER HAS A CHANCE TO ROLLBACK THAT OPERATION.

RULE OF TRANSACTION:

- THE RULE OF TRANSACTION TELLS THAT EITHER ALL THE STATEMENTS IN THE TRANSACTION SHOULD BE EXECUTED (ALL ARE COMMITTED) SUCCESSFULLY OR NONE OF THOSE STATEMENTS TO BE EXECUTED. (I.E., ALL ARE CANCELLED)

SAVEPOINT:

- WHENEVER A USER CREATE SAVEPOINT WITH IN THE TRANSACTION THEN INTERNALLY SYSTEM IS ALLOCATING A SPECIAL MEMORY FOR A SAVEPOINT AND STORE A TRANSACTION INFROMATION WHICH WE WANT TO ROLLBACK(CANCEL).

HOW TO CREATE A SAVEPOINT:

SYNTAX:

```
SQL> SAVEPOINT <POINTER NAME>;
```

HOW TO ROLLBACK A SAVEPOINT:

SYNTAX:

```
SQL> ROLLBACK TO <POINTER NAME>;
```

EX1:

```
SQL> DELETE FROM BRANCH WHERE BCODE=1021;

SQL> DELETE FROM BRANCH WHERE BCODE=1025;

SQL> SAVEPOINT S1;

SAVEPOINT CREATED.

SQL> DELETE FROM BRANCH WHERE BCODE=1023;
```

CASE1:

======

SQL> ROLLBACK TO S1; ------1023 RECORD ONLY

CASE2:

======

SQL> ROLLBACK; ----- 1021,1025 ROLLBACK

(OR)

SQL>COMMIT; ------ 1021,1025 COMMITTED

EX2:

```
SQL> DELETE FROM BRANCH WHERE BCODE=1021;
SQL> SAVEPOINT S1;
SQL> DELETE FROM BRANCH WHERE BCODE IN (1023,1025);
```

CASE1:

SQL> ROLLBACK TO S1; ----- 1023,1025 RECORDS ARE ROLLBACK

CASE2:

SQL>ROLLBACK; ------1021 ROLLBACK

(OR)

SQL>COMMIT; ------1021 COMMITTED

SUBQUERY / NESTED QUERY:

- A QUERY INSIDE ANOTHER QUERY IS CALLED AS "SUBQUERY / NESTED QUERY".

SYNTAX:

=======

SELECT * FROM EMP WHERE <CONDITION>(SELECT * FROM……….(SELECT * FROM…………));

- A SUBQUERY STATEMENT IS HAVING TWO MORE QUERIES THOSE ARE,

I) MAIN QUERY / PARENT QUERY / OUTER QUERY

II) SUBQUERY / CHILD QUERY / INNER QUERY

- AS PER THE EXECUTION PROCESS OF SUBQUERY STATEMENT IT AGAIN CLASSIFIED

INTO TWO TYPES,

1) NON-CORELATED SUBQUERY

2) CO-RELATED SUBQUERY

1) NON-CORELATED SUBQUERY:

=======================

- IN THIS NCSQ FIRST INNER QURY IS EXECUTED AND LATER OUTER QUERY WILL

EXCUTE.

I) SINGLE ROW SUBQUERY

II) MULTIPLE ROW SUBQUERY

III) MULTIPLE COLUMN SUBQUERY

I) SINGLE ROW SUBQUERY:

===================

- WHEN A SUBQUERY RETURN A SINGLE VALUE IS CALLED AS "SRSQ".

- IN SRSQ WE CAN USE THE FOLLOWING OPERATORS ARE " = , < , > , <= , >= ,!= ".

EX:

WAQ TO DISPLAY EMPLOYEE DETAILS WHO ARE GETTING THE FIRST HIGHEST SALARY?

===

SUBQUERY STATEMENT = OUTER QUERY + INNER QUERY

===

STEP1: INNER QUERY:

=================

```
SELECT MAX(SAL) FROM EMP;---------5000
```

STEP2: OUTER QUERY:

=================

```
SELECT * FROM EMP WHERE SAL = (INNER QUERY);
```

STEP3: SUBQUERY STATEMENT:

========================

```
SELECT * FROM EMP WHERE SAL=(SELECT MAX(SAL) FROM EMP);
```

EX:

WAQ TO DISPLAY THE SENIOR MOST EMPLOYEE DETAILS FROM EMP TABLE?

```
SQL> SELECT * FROM EMP WHERE HIREDATE=(SELECT MIN(HIREDATE) FROM
EMP);
```

EX:

WAQ TO DISPLAY EMPLOYEES DETAILS WHOSE SALARY IS MORE THAN THE MAXIMUM SALARY

OF "SALESMAN"?

```
SQL> SELECT * FROM EMP WHERE SAL>
  2  (SELECT MAX(SAL) FROM EMP WHERE JOB='SALESMAN');
```

EX:

WAQ TO FIND OUT THE SECOND HIGHEST SALARY?

```
SQL> SELECT MAX(SAL) FROM EMP WHERE SAL<(SELECT MAX(SAL) FROM
EMP);
```

EX:

WAQ TO DISPLAY EMPLOYEES DETAILS WHO ARE EARNING THE SECOND HIGHEST SALARY?

```
SQL> SELECT * FROM EMP WHERE SAL=
         (SELECT MAX(SAL) FROM EMP WHERE SAL <
         (SELECT MAX(SAL) FROM EMP));
```

EX:

WAQ TO DISPLAY EMPLOYEES DETAILS WHO ARE EARNING THE 3RD HIGHEST SALARY?

```
SQL> SELECT * FROM EMP WHERE SAL=

        (SELECT MAX(SAL) FROM EMP WHERE SAL <

        (SELECT MAX(SAL) FROM EMP WHERE SAL <

        (SELECT MAX(SAL) FROM EMP )));
```

II) MULTIPLE ROW SUBQUERY:

==========================

- **WHEN A SUBQUERY RETURNS MORE THAN ONE VALUE.**
- **IT SUPPORTS THE FOLLOWING OPERATORS ARE "IN,ANY,ALL".**

EX:

WAQ TO DISPLAY EMPLOYEES WHOSE JOB IS SAME AS THE JOB OF "SMITH","MARTIN"?

```
SQL> SELECT * FROM EMP WHERE JOB IN(SELECT JOB FROM EMP

     WHERE ENAME='SMITH' OR ENAME='MARTIN');

                (OR)

SQL> SELECT * FROM EMP WHERE JOB IN(SELECT JOB FROM EMP

     WHERE ENAME IN('SMITH','MARTIN'));
```

EX:

WAQ TO DISPLAY EMPLOYEES WHO ARE GETTING MINIMUM,MAXIMUM SALARIES FROM EMPTABLE?

```
SQL> SELECT * FROM EMP WHERE SAL IN

  2  (

  3  SELECT MIN(SAL) FROM EMP

  4  UNION

  5  SELECT MAX(SAL) FROM EMP

  6  );
```

EX:

WAQ TO DISPLAY THE SENIOR MOST EMPLOYEES DETAILS FROM EACH DEPTNO WISE?

```
SQL> SELECT * FROM EMP WHERE HIREDATE IN(SELECT MIN(HIREDATE) FROM

        EMP GROUP BY DEPTNO);
```

EX:

WAQ TO DISPLAY EMPLOYEES DETAILS WHO ARE GETTING MAXIMUM SALARY FROM EACH JOB

WISE?

```
SQL> SELECT * FROM EMP WHERE SAL IN(SELECT MAX(SAL) FROM

        EMP GROUP BY JOB);
```

ANY OPERATOR:

============

- IT RETURNS A VALUE IF ANY ONE VALUE IS SATISFIED WITH USER DEFINED VALUE FROM

THE GIVEN GROUP OF VALUES.

EX:

```
        X(40) > ANY(10,20,30)


        IF X=09 ===> FALSE

        IF X=25 ===> TRUE

        IF X=40 ===> TRUE
```

ALL OPERATOR:

============

- IT RETURNS A VALUE IF ALL VALUES ARE SATISFIED WITH USER DEFINED VALUE FROM

THE GIVEN GROUP OF VALUES.

EX:

X(40) > ALL(10,20,30)

IF X=09 ===> FALSE

IF X=25 ===> FALSE

IF X=40 ===> TRUE

EX:

WAQ TO DISPLAY EMPLOYEES WHOSE SALARY IS MORE THAN ANY SALESMAN SALARY?

```
SQL> SELECT * FROM EMP WHERE SAL>ANY(SELECT SAL FROM EMP WHERE
JOB='SALESMAN');
```

EX:

WAQ TO DISPLAY EMPLOYEES WHOSE SALARY IS MORE THAN ALL SALESMAN SALARIES?

```
SQL> SELECT * FROM EMP WHERE SAL>ALL(SELECT SAL FROM EMP WHERE
JOB='SALESMAN');
```

ANY OPERATOR	ALL OPERATOR
===========	===========
X > ANY	X > ALL
X >= ANY	X >= ALL
X < ANY	X < ALL
X <= ANY	X <= ALL
X = ANY	X = ALL
X != ANY	X != ALL

NOTE:

=====

```
SQL> UPDATE EMP SET SAL=1300 WHERE ENAME='FORD';
```

EX:

WAQ TO DISPLAY EMPLOYEES DETAILS WHO ARE GETTING MAXIMUM SALARY FROM EACH JOB

WISE?

```
SQL> SELECT * FROM EMP WHERE SAL IN(SELECT MAX(SAL) FROM

        EMP GROUP BY JOB);
```

OUTPUT:

========

```
JOB            MAX(SAL)
====           =========
CLERK          1300
SALESMAN       1600
MANAGER        2975
ANALYST        3000
PRESIDENT      5000
ANALYST        1300
```

- BY USING MULTIPLE ROW SUBQUERY WHEN WE ARE COMPARING GROUP OF VALUES

THERE IS A CHANCE TO RETURN THE WRONG RESULT.

- TO OVERCOME THE ABOVE PROBLEM WE SHOULD USE "MULTIPLE COLUMN SUBQUERY"

MECHANISM.

III) MULTIPLE COLUMN SUBQUERY:

=============================

- IN THIS MECHANISM MULTIPLE COLUMNS VALUES OF INNER QUERY IS COMPARING WITH

MULTIPLE COLUMNS VALUES OF OUTER QUERY IS CALLED AS "MCSQ"

SYNTAX:

======

```
SELECT * FROM <TN> WHERE (<COLUMN NAME1>,<COLUMN NAME2>,………….)
 IN ( SELECT <COLUMN NAME1>,<COLUMN NAME2>,………FROM <TN>);
```

EX:

WAQ TO DISPLAY EMPLOYEES DETAILS WHO ARE GETTING MAXIMUM SALARY FROM EACH JOB

WISE?

```
SQL> SELECT * FROM EMP WHERE(JOB,SAL) IN(SELECT JOB,MAX(SAL) FROM

        EMP GROUP BY JOB);
```

OUTPUT:

```
========

JOB                 MAX(SAL)

====                =========

CLERK               1300

SALESMAN            1600

MANAGER             2975

ANALYST             3000

PRESIDENT           5000
```

EX:

WAQ TO DISPLAY EMPLOYEES WHOSE JOB,MGR ARE SAME AS THE JOB,MGR OF THE EMPLOYEE "SCOTT"?

```
SQL> SELECT * FROM EMP WHERE(JOB,MGR) IN(SELECT JOB,MGR

  2  FROM EMP WHERE ENAME='SCOTT');
```

2) CO-RELATED SUBQUERY:

=======================

- IN THIS CRSQ FIRST OUTER QUERY IS EXECUTED AND LATER INNER QUERY WILL

EXECUTE.

SYNTAX FOR FINDING "NTH" HIGH / LOW SALARY:

==================================

SELECT * FROM <TN> <TABLE ALIAS NAME1> WHERE N-1=(SELECT COUNT(DISTINCT <COLUMN NAME>)

FROM <TN> <TABLE ALIAS NAME2> WHERE <TABLE ALIAS NAME2>.<COLUMN NAME>

< / > <TABLE ALIAS NAME1>.<COLUMN NAME>);

HERE,

< - LOW SALARY

> - HIGH SALARY

EX:

WAQ TO FINDOUT THE FIRST HIGHEST SALARY EMPLOYEE DETAILS?

```
SQL> SELECT * FROM TEST T1 WHERE 0=(SELECT COUNT(DISTINCT SAL)
       FROM TEST T2 WHERE T2.SAL > T1.SAL);
```

EX:

WAQ TO FINDOUT THE 4TH HIGHEST SALARY EMPLOYEE DETAILS?

```
SQL> SELECT * FROM TEST T1 WHERE 3=(SELECT COUNT(DISTINCT SAL)
       FROM TEST T2 WHERE T2.SAL > T1.SAL);
```

EX:

WAQ TO FINDOUT THE FIRST LOWEST SALARY EMPLOYEE DETAILS?

```
SQL> SELECT * FROM TEST T1 WHERE 0=(SELECT COUNT(DISTINCT SAL)

        FROM TEST T2 WHERE T2.SAL < T1.SAL);
```

SYNTAX TO DISPLAY "TOP N " HIGH / LOW SALARIES:

==

```
SELECT * FROM <TN> <TABLE ALIAS NAME1> WHERE N>(SELECT
COUNT(DISTINCT <COLUMN NAME>)

FROM <TN> <TABLE ALIAS NAME2> WHERE <TABLE ALIAS
NAME2>.<COLUMN NAME>

< /  > <TABLE ALIAS NAME1>.<COLUMN NAME>);
```

HERE,

< - LOW SALARY

> - HIGH SALARY

EX:

WAQ TO DISPLAY TOP 3 HIGHEST SALARIES EMPLOYEE DETAILS?

```
SQL> SELECT * FROM TEST T1 WHERE 3>(SELECT COUNT(DISTINCT SAL)

        FROM TEST T2 WHERE T2.SAL > T1.SAL);
```

EX:

WAQ TO DISPLAY TOP 3 LOWEST SALARIES EMPLOYEE DETAILS?

```
SQL> SELECT * FROM TEST T1 WHERE 3>(SELECT COUNT(DISTINCT SAL)

        FROM TEST T2 WHERE T2.SAL < T1.SAL);
```

NOTE:

=====

> TO FIND OUT "NTH" HIGH / LOW SALARY ------------> N-1

> TO DISPLAY "TOP N" HIGH / LOW SALARIES -------> N>

EXISTS OPERATOR:

==============

- IT IS A SPECIAL OPERATOR WHICH WAS USED IN CO-RELATED SUBQUERY ONLY.

- THIS OPERATOR IS USED TO CHECK OUR REQUIRED ROW IS EXISTING IN A TABLE OR NOT.

- IF A ROW IS EXISTING IN A TABLE THEN IT RETURN TRUE.

- IF A ROW IS NOT EXISTING IN A TABLE THEN IT RETURN FALSE.

SYNTAX:

======

```
        WHERE EXISTS(SUBQUERY)
```

EX:

WAQ TO DISPLAY DEPARTMENT DETAILS IN WHICH DEPARTMENT THE EMPLOYEES ARE WORKING?

```
SQL> SELECT * FROM DEPT D WHERE EXISTS(SELECT DEPTNO FROM
  2  EMP E WHERE E.DEPTNO=D.DEPTNO);
```

EX:

WAQ TO DISPLAY DEPARTMENT DETAILS IN WHICH DEPARTMENT THE EMPLOYEES ARE NOT WORKING?

```
SQL> SELECT * FROM DEPT D WHERE NOT EXISTS(SELECT DEPTNO FROM
        EMP E WHERE E.DEPTNO=D.DEPTNO);
```

VIEWS:

VIEW IS DB OBJECT IS CALLED SUBSET OF A TABLE.VIEW IS ALSO CALLED AS VIRTUAL TABLE BECAUSE IT DOESN'TSTORE DATA AND IT DOESN'T OCCUPY ANY MEMORY.

VIWE IS CREATING BY USING "SELECT QUERY" FOR GETTING THE REQ.INFORMATIONFROM TABLE (BASE TABLE).

TYPES OF VIEWS:

A USER CAN CREATE THE FOLLOWING TWO TYPES OF VIEWS ON BASE TABLESTHOSE ARE,

1. SIMPLE VIEWS
2. COMPLEX VIEWS

1. SIMPLE VIEWS:

WHEN WE CREATE A VIEW TO ACCESS REQUIRED DATA FROM A SINGLE BASE TABLEIS CALLED AS SIMPLE VIEWS.

THROUGH A SIMPLE VIEW WE CAN PERFORM ALL DML (INSERT, UPDATE, DELETE)OPERATIONS ON BASE TABLE.

SYNTAX:

```
CREATE VIEW <VIEW NAME> AS SELECT * FROM <TN> [ WHERE
<CONDITION>];
```

EX1:

```
SQL> CREATE VIEW SV1 AS SELECT * FROM DEPT;

SQL> SELECT * FROM SV1;
```

DML OPERATIONS THROUGH A SIMPLE VIEW:

```
QL> INSERT INTO SV1 VALUES (50,'DBA','HYD');

SQL> UPDATE SV1 SET LOC='INDIA' WHERE DEPTNO=50;

SQL> DELETE FROM SV1 WHERE DEPTNO=50;
```

NOTE: WHENEVER WE PERFORM DML OPERATIONS ON VIEW INTERNALLY THE VIEW WILL PERFORM THOSE OPERATIONS ON BASE TABLE.HERE VIEW WILL ACT ASAN INTERFACE BETWEEN USER AND BASE TABLE.

USER <----------><VIEW><-----------> BASE TABLE

EX2:

SQL> CREATE VIEW SV2 AS SELECT EMPNO, ENAME, JOB, SAL FROM EMP;

TESTING:

SQL> INSERT INTO SV2 VALUES (1122,'SAI','HR',8000); ---ALLOW

SQL> INSERT INTO SV2 VALUES (1122,'WARNER','SR.HR',9500); ----NOT ALLOW (EMPNO COLUMN IS PRIMARY KEY COLUMN IN EMP TABLE)

<u>WITH CHECK OPTION:</u>

IT IS A CONSTRAINT WHICH IS USED TO RESTRICT ROWS ON BASE TABLE THROUGH

A VIEW WHILE PERFORMING DML OPERATIONS.

EX:

SQL> CREATE VIEW SV3 AS SELECT * FROM TEST1 WHERE SAL=18000 WITH CHECK OPTION;

TESTING:

SQL> INSERT INTO SV3 VALUES (1025,'SCOTT',12000); ---NOT ALLOW

SQL> INSERT INTO SV3 VALUES (1025,'SCOTT',58000); ---NOT ALLOW

SQL> INSERT INTO SV3 VALUES (1025,'SCOTT',18000); ---ALLOWED

WITH READ ONLY:

IF WE CREATED A VIEW "WITH READ ONLY" CLAUSE THEN WE RESTRICT DML OPERATIONS.WE ALLOW "SELECT" AND "DESC" COMMANDS.

EX:

```
SQL> CREATE VIEW SV4 AS SELECT * FROM DEPT WITH READ ONLY;
```

NOTE: NOW WE CANNOT PERFORMDMLOPERATIONSTHROUGH A VIEW ON BASE TABLE.

2.COMPLEX VIEWS:

- A VIEW IS CALLED AS COMPLEX VIEW,

I) WHEN WE CREATE ON MULTIPLE BASE TABLES.

II) WHEN WE CREATE A VIEW WITH AGGREGATIVE FUNCTIONS, GROUPBY, HAVINGCLAUSES, SETOPERATORS, SUB-QUERY, DISTINCT KEY WORD.COMPLEX VIEW ARE NOT ALWAYS SUPPORTS DML OPERATIONS.

EX1:

```
SQL> CREATE VIEW CV1 AS SELECT * FROM STUDENT S INNER JOIN COURSE C

     ON S.CID=C.CID;


ERROR AT LINE 1:

ORA-00957: DUPLICATE COLUMN NAME
```

NOTE: WHEN WE CREATE A VIEW ON BASE TABLES THEN WE SHOULD NOT ALLOWDUPLICATE COLUMN NAMES.TO AVOID THIS PROBLEM THEN USE "USING"CLAUSE.

```
SQL> CREATE VIEW CV1 AS SELECT * FROM STUDENT S INNER JOIN COURSE C
USING(CID);
```

NOW WE CREATED A COMPLEX VIEW ON MULTIPLE TABLES.BUT NOT ALLOW DMLOPERATIONS.

EX2:

```
SQL> CREATE VIEW CV2 AS

        SELECT * FROM EMP_HYD

        UNION

        SELECT * FROM EMP_CHENNAI;
```

> THE ABOVE COMPLEX VIEW CV2 IS NOT ALLOW DML OPERATIONS.

EX3:

```
SQL> CREATE VIEW CV3 AS SELECT DEPTNO, SUM(SAL) FROM EMP GROUP BY DEPTNO;

ERROR AT LINE 1:

ORA-00998: MUST NAME THIS EXPRESSION WITH A COLUMN ALIAS
```

NOTE: WHEN WE CREATE A VIEW WITH FUNCTION THEN WE MUST CREATE ALIAS NAME FOR THOSE FUNCTIONS OTHERWISE ORACLE RETURNS AN ERROR.

EX:

```
SQL> CREATE VIEW CV3 AS SELECT DEPTNO, SUM(SAL) AS SUMSAL FROM EMP

        GROUP BY DEPTNO;
```

> THE ABOVE COMPLEX VIEW CV3 NOT ALLOWED DML OPERATIONS.

EX4:

```
SQL> CREATE VIEW CV4 AS SELECT EMPNO, ENAME, SAL, D. DEPTNO, DNAME, LOC FROMEMP E INNER JOIN DEPT D ON E. DEPTNO=D.DEPTNO;
```

<u>TESTING:</u>

```
SQL> UPDATE CV4 SET SAL=500 WHERE EMPNO=7788; ---ALLOWED

SQL> DELETE FROM CV4 WHERE EMPNO=7782; ----ALOOWED

SQL> INSERT INTO CV4 VALUES (1122,'SAI',6000,10,'SAP','HYD'); ---NOT ALLOW
```

NOTE:GENERALLY COMPLEX VIEW ARE NOT ALLOWED TO PERFORM DML OPERATIONSBUT WE PERFORM UPDATE, DELETE OPERATIONS ON KEY PRESERVED TABLE (I.E PRIMARY KEY)SO THAT COMPLEX VIEWS ARE SUPPORTING DML OPERATION PARTIALLY.

NOTE:TO VIEW ALL VIEWS DETAILS IN ORACLE DB THEN WE USE THE FOLLOWING DATADICTIONAY IS "USER_VIEWS".

EX:

```
SQL> DESC USER_VIEWS;

SQL> SELECT VIEW_NAME FROM USER_VIEWS;
```

SYNTAX TO DROP A VIEW:

```
SQL> DROP VIEW <VIEW NAME>;
```

EX:

```
SQL> DROP VIEW SV1;

SQL> DROP VIEW CV1;

SQL> DROP VIEW FV1;
```

<u>ADVANTAGES OF VIEWS:</u>

1. IT IS PROVIDING SECURITY.IT MEANS THAT TO EACH USER CAN BE GIVEN PERMISSION TO ACCESS SPECIFIC COLUMNS & SPECIFIC ROWS FROM A TABLE.

2. IF DATA IS ACCESSED AND ENTERED THROUGH A VIEW,THE DB SERVER WILL CHECKDATA TO ENSURE THAT IT MEETS SPECIFIED INTERGRITY CONSTRAINTS RULES ORNOT.

3. QUERY SIMPLIFY IT MEANS THAT TO REDUCE COMPLEX QUERY.

SEQUENCE:

- SEQUENCE IS A DB OBJECT.WHICH IS USED TO GENERATE SEQUENCE NUMBERS ON A PARTICULAR COLUMN AUTOMATICALLY.

SYNTAX:

```
CREATE SEQUENCE <SEQUENCE NAME>

[ START WITH N]

[ MINVALUE N]

[ INCREMENT BY N]

[ MAXVALUE N]

[ NO CYCLE / CYCLE]

[ NO CACHE / CACHE N];
```

PARAMETERS OF SEQUENCE OBJECT:

START WITH N:

- IT REPRESENT THE STARTING SEQUENCE NUMBER.HERE "N" IS REPRESENT WITN NUMBER.

MINVALUE N:

- IT SPECIFY THE MINIMUM VALUE OF THE SEQUENCE.HERE "N" IS REPRESENT WITN NUMBER.

INCREMENT BY N:

- IT SPECIFY THE INCREMENTAL VALUE IN BETWEEN SEQUENCE NUMBERS.HERE "N" IS REPRESENT WITN NUMBER.

MAXVALUE N:

- IT SPECIFY THE MAXIMUM VALUE OF THE SEQUENCE.HERE "N" IS REPRESENT WITN NUMBER.

NO CYCLE:

- IT IS DEFAULT PARAMETER.IF WE CREATED SEQUENCE WITH " NO CYCLE " THEN SEQUENCE STARTS FROM START WITH VALUE AND GENERATE VALUES UPTO MAX VALUE.AFTER REACHING MAX VALUE THEN SEQUENCE IS STOP.

CYCLE:

- IF WE CREATED A SEQUENCE WITH "CYCLE" THEN SEQUENCE STARTS FROM START WITH VALUE AND GENERATE VALUES UPTO MAXVALUE.AFTER REACHING MAX VALUETHEN SEQUENCE WILL STARTS WITH MINVALUE.

NO CACHE:

- IT IS DEFAULT PARAMETER.WHEN WE CREATED A SEQUENCE WITH "NO CACHE"PARAMETER THEN THE SET OF SEQUENCE VALUES ARE STORING INTO DATABASE MEMORY.EVERY TIME WE WANT ACCESS SEQUENCE NUMBERS THEN ORACLE SERVER WILL GO TO DATABASE MEMORY AND RETURN TO USER.SO THAT IT WILL DEGRADE THE PERFORMANCE OF AN APPLICATION.

CACHE N:

- WHEN WE CREATED A SEQUENCE WITH "CACHE " PARAMETER THEN SYSTEM IS ALLOCATING TEMP. MEMORY(CACHE) AND IN THIS MEMORY WE WILL STORE THE SET SEQUENCE NUMBERS.WHENEVER USER WANT TO ACCESS SEQUENCE NUMBERS THEN ORACLE SERVER WILL GO TO CACHE MEMORY AND RETURN TO USER.

- ACCESSING DATA FROM CACHE IS MUCH FASTER THAN ACCESSING DATA FROM DATABASE.IT WILL INCRESE THE PERFORMANCE OF AN APPLICATION.HERE "N" IS REPRESENT THE SIZE OF CACHE FILE.MINIMUM SIZE OF CACHE IS 2KB AND MAXIMUM SIZE OF CACHE IS 20KB.

<u>NOTE:</u>

- TO WORK WITH SEQUENCE OBJECT WE SHOULD USE THE FOLLOWING TWO PSEUDO COLUMNS ARE "NEXTVAL" AND "CURRVAL".

<u>NEXTVAL:</u>

- IT IS USED TO GENERATE SEQUENCE NUMBERS ON A PARTICULAR COLUMN.

<u>SYNTAX:</u>

```
SELECT <SEQUENCE NAME>. <NEXTVAL> FROM DUAL;
```

<u>CURRVAL:</u>

- IT IS USED TO SHOW THE CURRENT VALUE OF THE SEQUENCE.

<u>SYNTAX:</u>

```
SELECT <SEQUENCE NAME>. <CURRVAL> FROM DUAL;
```

EX1:

STEP1:

```
SQL> CREATE SEQUENCE SQ1
        START WITH 1
        MINVALUE 1
        INCREMENT BY 1
        MAXVALUE 3;
SEQUENCE CREATED.
```

STEP2:

```
SQL> CREATE TABLE TEST1(SNO INT, NAME VARCHAR2(10));

TABLE CREATED.
```

TESTING:

=========

```
SQL> INSERT INTO TEST1 VALUES (SQ1.NEXTVAL,'&NAME');

ENTER VALUE FOR NAME: A

/

ENTER VALUE FOR NAME: B

/

ENTER VALUE FOR NAME: C

/

ENTER VALUE FOR NAME: D

ERROR AT LINE 1:

ORA-08004: SEQUENCE SQ1.NEXTVAL EXCEEDS MAXVALUE AND CANNOT BE
INSTANTIATED.
```

ALTERING A SEQUENCE:

SYNTAX:

```
ALTER SEQUENCE <SEQUENCE NAME><PARAMETER NAME> N;
```

EX:

```
SQL> ALTER SEQUENCE SQ1 MAXVALUE 5;

SEQUENCE ALTERED.
```

TESTING:

```
SQL> INSERT INTO TEST1 VALUES (SQ1.NEXTVAL,'&NAME');
ENTER VALUE FOR NAME: D
/
ENTER VALUE FOR NAME: E
```

<u>OUTPUT:</u>

```
SQL> SELECT * FROM TEST1;

SNO         NAME
-------     ----------
  1         A
  2         B
  3         C
  4         D
  5         E
```

NOTE: WE CAN ALTER ALL PARAMETERS EXCEPT "START WITH " PARAMETER.

EX2:

```
SQL> CREATE SEQUENCE SQ2
        START WITH 1
        MINVALUE 1
        INCREMENT BY 1
        MAXVALUE 3
        CYCLE
        CACHE 2;
```

```
SQL> CREATE TABLE TEST2(SNO INT, NAME VARCHAR2(10));

TABLE CREATED.
```

TESTING:

```
SQL> INSERT INTO TEST2 VALUES (SQ2.NEXTVAL,'&NAME');

ENTER VALUE FOR NAME: A

/

ENTER VALUE FOR NAME: B

/

ENTER VALUE FOR NAME: C

/

....................................

....................................
```

OUTPUT:

```
SQL> SELECT * FROM TEST2;

SNO        NAME

-------    ----------

  1        A

  2        B

  3        C

  1        D

  2        E

  3        F
```

```
SQL> CREATE SEQUENCE SQ3
        START WITH 3
        MINVALUE 1
        INCREMENT BY 1
        MAXVALUE 5
        CYCLE
        CACHE 2;
SEQUENCE CREATED.

SQL> CREATE TABLE TEST3(SNO INT, NAME VARCHAR2(10));
TABLE CREATED.
```

<u>TESTING:</u>

```
SQL> INSERT INTO TEST3 VALUES (SQ3.NEXTVAL,'&NAME');
ENTER VALUE FOR NAME: A
/
.....................................
.....................................
```

<u>OUTPUT:</u>

```
SQL> SELECT * FROM TEST3;

      SNO             NAME
--------------        ----------
        3             A
        4             B
```

5	C
1	M
2	N
3	O
4	P
5	Q

NOTE: IF WE WANT TO VIEW ALL SEQUENCES IN ORACLE DATABASE THEN WE USE "USER_SEQUENCES" DATA DICTIONARY.

EX:

```
SQL> DESC USER_SEQUENCES;

SQL> SELECT SEQUENCE_NAME FROM USER_SEQUENCES;
```

SYNTAX TO DROP A SEQUENCE:

============================

```
SQL> DROP SEQUENCE <SEQUENCE NAME>;
```

EX:

```
SQL> DROP SEQUENCE SQ1;
```

INDEXES:

- INDEX IS AN DATABASE OBJECT WHICH IS USED TO RETRIEVE DATA FROM A TABLE FASTLY.

- A DATABASE INDEX WILL WORK AS A BOOK INDEX PAGE IN TEXT BOOK.IN TEXT BOOK BY USING INDEX PAGE WE CAN RETRIEVE A PARTICULAR TOPIC FROM A TEXT BOOK VERY FASTLY SAME AS BY USING DATABASE INDEX OBJECT WE CAN RETRIEVE A PARTICULAR ROW FROM A TABLE VAERY FASTLY.

- BY USING INDEXES, WE CAN SAVE TIME AND IMPROVE THE PERFORMANCE OF DATABASE.THESE INDEXES ARE CREATED BY DBA.

- INDEX OBJECT CAN BE CREATED ON A PARTICULAR COLUMN (OR) COLUMNSOF A TABLE AND THESE COLUMNS ARE CALLED AS "INDEX KEY COLUMNS".

- ALL DATABASES ARE SUPPORTING THE FOLLOWING TWO TYPES OF SEARCHING MECHANISMS THOSE ARE,

1. TABLE SCAN(DEFAULT)
2. INDEX SCAN

1.TABLE SCAN:

- IT IS A DEFAULT SCANNING MECHANISM FOR RETRIEVING DATA FROM TABLE.IN THIS MECHANISM ORACLE SERVER IS SCANNING ENTIRE TABLE (TOP - BOTTOM)

EX:

```
SQL> SELECT * FROM EMP WHERE SAL=3000;


SOL:

 SAL

--------

 800

1600

1250
```

2975

1250

2850

2450

3000 **(IN THIS TABLE SCAN WE ARE COMPARING WHERE CONDITION 14 TIMES)**

5000

1500

1100

950

3000

1300

2) INDEX SCAN:

- IN INDEX SCAN MECHANISM ORACLE SERVER SCANNING ONLY INDEXED COLUMN FROM A TABLE. IN THIS MECHANISM WE AGAIN FOLLOW THE FOLLWOING TWO METHODS,

I) AUTOMATICALLY / IMPLICITLY:

- WHENEVER WE ARE CREATING A TABLE ALONG WITH "PRIMARY KEY " (OR) "UNIQUE" KEY CONSTRAINT THEN INTERNALLY SYSTEM IS CREATING AN INDEX OBJECT ON THAT PARTICULAR COLUMN AUTOMATICALLY.

EX:

```
SQL> CREATE TABLE TEST1(EID INT PRIMARY KEY, ENAME VARCHAR2(10));

SQL> CREATE TABLE TEST2(SNO INT UNIQUE, NAME VARCHAR2(10));
```

NOTE:

- IF WE WANT TO VIEW INDEX NAME ALONG WITH COLUMN NAME OF A PARTICULAR TABLE THEN WE USE "USER_IND_COLUMNS" DATA DICTIONARY.

EX:

```
SQL> DESC USER_IND_COLUMNS;

SQL> SELECT COLUMN_NAME, INDEX_NAME FROM
USER_IND_COLUMNSWHERE TABLE_NAME='TEST1';
```

```
COLUMN_NAME                 INDEX_NAME
-------------------------   ----------------------
EID                         SYS_C005501
```

SQL> SELECT COLUMN_NAME, INDEX_NAME FROM USER_IND_COLUMNSWHERE TABLE_NAME='TEST2';

```
COLUMN_NAME                 INDEX_NAME
-------------------------   ----------------------
SNO                         SYS_C005502
```

<u>II) MANUALLY / EXPLICITLY:</u>

- WHEN USER WANT TO CREATE AN INDEX OBJECT ON A PARTICULAR COLUMN/(S) THEN WE FOLLOW THE FOLLOWING SYNTAXS,

<u>TYPES OF INDEXES:</u>

1. B - TREE INDEX (DEFAULT INDEX)

2. BITMAP INDEX

1. **<u>B - TREE INDEX:</u>**

- WHEN WE CREATED AN INDEX ON A SINGLE COLUMN THEN WE CALLED AS SIMPLE INDEX.

SYNTAX:

CREATE INDEX <INDEX NAME> ON <TN> (<COLUMN NAME>);

EX:

```
SQL> CREATE INDEX SIND ON EMP(SAL);

INDEX CREATED.
```

EX:

```
SQL> SELECT * FROM EMP WHERE SAL=3000;
```

SOL:

```
               B-TREE (BINARY TREE)

            (<) |LP| 3000 |RP| (>=)

                    |

        LP| 2975 | RP                                LP | 5000 | RP

                    |                                    |

2850|*, 2450|*, 1600|*, 1500|*           | 3000 |*, *|

1300|*, 1250|*, *, 1100|*, 950|*, 800|*
```

NOTE: IN INDEX SCAN WE ARE COMPARING 3 TIMES.WHICH IS MUCH FASTER THAN TABLE SCAN (14 TIMES COMPARING). HERE " * " IS REPRESENT ROWID.

2. BITMAP INDEX:

- BITMAP INDEX IS CREATED ON DISTINCT VALUES OF A PARTICULAR COLUMN.GENERALLY BITMAP INDEXES ARE CREATED ON LOW CARDINALITY OF COLUMNS.

- WHEN WE CREATE BITMAP INDEX INTERNALLY ORACLE SERVER IS PREPARING BITMAP INDEXED TABLE WITH BIT NUMBERS ARE 1 AND 0. HERE 1 IS REPRESENT CONDITION IS TRUE WHERE AS 0 IS REPRESENT CONDITION IS FALSE.

CARDINALITY:

- IT REFERES TO THE UINQUENESS OF DATA VALUES CONTAINE IN PARTICULAR COLUMN OF TABLE.

<u>HOW TO FIND CARDINALITY OF A COLUMN:</u>

CARDINALITY OF COLUMN = NO. OF DISTINCT VALUES OF A COLUMN

NO. OF ROWS IN A TABLE

EX:

CARDINALITY OF EMPNO = 14

14

CARDINALITY OF EMPNO IS "1" ----(CREATING BTREE INDEX)

EX:

CARDINALITY OF JOB = 5

14

CARDINALITY OF JOB = 0.35 ------ (CREATING BIT MAP INDEX)

SYNTAX:

CREATE BITMAP INDEX <INDEX NAME> ON <TN>(<COLUMN NAME>);

EX:

CREATE BITMAP INDEX BITIND ON EMP(JOB);

EX:

```
SELECT * FROM EMP WHERE JOB='MANAGER';
```

BITMAP INDEXED TABLE

=====================

JOB	1	2	3	4	5	6	7	8	9	10	11	12	13	14
CLERK	1	0	0	0	0	0	0	0	0	0	1	1	0	1
SALESMAN	0	1	1	0	1	0	0	0	0	1	0	0	0	0
MANAGER	0	0	0	1	0	1	1	0	0	0	0	0	0	0
ANALYST	0	0	0	0	0	0	0	1	0	0	0	0	1	0
PRESIDENT	0	0	0	0	0	0	0	0	1	0	0	0	0	0

NOTE: HERE "1" IS REPRESENTED WITH ROWID OF A PARTICULAR ROW IN A TABLE.

<u>NOTE:</u>

- IF WE WANT TO VIEW INDEX NAME ALONG WITH INDEX TYPE THEN WE USE"USER_INDEXES" DATADICTIONARY.

EX:

```
SQL> DESC USER_INDEXES;

SQL> SELECT INDEX_NAME, INDEX_TYPE FROM USER_INDEXES
     WHERE TABLE_NAME='EMP';

INDEX_NAME                INDEX_TYPE
-----------------         ---------------------
SIND                 NORMAL(B-TREE)
BITIND               BITMAP
FIND                 FUNCTION-BASED NORMAL(B-TREE)
UIND                 NORMAL(B-TREE)
CIND                 NORMAL(B-TREE)
```

HOW TO DROP AN INDEX:

```
SQL> DROP INDEX <INDEX NAME>;
```

EX:

```
SQL> DROP INDEX SIND;

SQL> DROP INDEX BITIND;
```

NORMALIZATION:

- NORMALIZATION IS A TECHNIQUE OF ORGANIZING THE DATA INTO MULTIPLE TABLES. NORMALIZATION PROCESS AUTOMATICALLY ELIMINATES DATA REDUNDANCY (REPETITION) AND ALSO AVOIDING INSERTION, UPDATE AND DELETION PROBLEMS.

PROBLEMS WITHOUT NORMALIZATION: IF A TABLE IS NOT PROPERLY NORMALIZED AND HAVE DATA REDUNDANCY THEN IT WILL NOT ONLY OCCUPYEXTRA MEMORY SPACE BUT WILL ALSO MAKE IT DIFFICULT TO HANDLE INSERT, DELETE AND UPDATE OPERATIONS IN STUDENT TABLE.

STUDENT DETAILS

ROLL NO	**NAME**	**BRANCH**	**HOD**	**OFFICE NUMBER**
101	**SAI**	**CSE**	**MR. X**	**040-53337**
102	**ALLEN**	**CSE**	**MR. X**	**040-53337**
103	**JAMES**	**CSE**	**MR. X**	**040-53337**
104	**MILLER**	**CSE**	**MR. X**	**040-53337**

IN THE TABLE ABOVE, WE HAVE DATA OF 4 COMPUTER SCI. STUDENTS. AS WE CAN SEE, DATA FOR THE FIELDS BRANCH, HODAND OFFICE_ NUMBER IS REPEATED FOR THE STUDENTS WHO ARE IN THE SAME BRANCH IN THE COLLEGE, THIS IS DATA REDUNDANCY.

INSERTION PROBLEM:

IF WE HAVE TO INSERT DATA OF 100 STUDENTS OF SAME BRANCH, THEN THE BRANCH INFORMATION WILL BE REPEATED FOR ALL THOSE 100 STUDENTS. THESE SCENARIOS ARE NOTHING BUT INSERTION PROBLEM. REASON FOR DATA REDUNDANCY IS TWO DIFFERENT RELATED DATA STORED IN THE SAME TABLE.

STUDENT DATA + BRANCH DATA

UPDATION PROBLEM:

IF WE WANT TO CHANGE HOD NAME THEN SYSTEM ADMIN HAS TO UPDATE ALL STUDENTS RECORDS WITH NEW HOD NAME.AND IF BY MISTAKE WE MISS ANY RECORD, IT WILL LEAD TO DATA INCONSISTENCY. THIS IS UPDATION PROBLEM.

EX: MR. X LEAVES AND MR. Y JOIN AS A NEW HOD FOR CSE. THEN THE TABLE WILL BE LIKE BELOW,

STUDENT DETAILS

ROLL NO	NAME	BRANCH	HOD	OFFICE NUMBER
101	SAI	CSE	MR. Y	040-53337
102	ALLEN	CSE	MR. Y	040-53337
103	JAMES	CSE	MR. Y	040-53337
104	MILLER	CSE	MR. Y	040-53337
105	WARNER	CSE	MR. Y	040-53337

DELETION PROBLEM:

IN OUR STUDENT DETAILS TABLE, TWO DIFFERENT INFORMATION'S ARE KEPT TOGETHER, STUDENT INFORMATION AND BRANCH INFORMATION. HENCE, AT THE END OF THE ACADEMIC YEAR, IF STUDENT RECORDS ARE DELETED, WE WILL ALSO LOSE THE BRANCH INFORMATION. THIS IS CALLED AS DELETION PROBLEM.

HOW NORMALIZATION WILL SOLVE ALL THESE PROBLEMS:

STUDENT DETAILS

ROLL NO	NAME	BRANCH	HOD	OFFICE NUMBER
101	SAI	CSE	MR. Y	040-53337
102	ALLEN	CSE	MR. Y	040-53337
103	JAMES	CSE	MR. Y	040-53337
104	MILLER	CSE	MR. Y	040-53337

105	**WARNER**	**CSE**	**MR. Y**	**040-53337**

NOTE: NOW WE NEED TO DECOMPOSING A STUDENT TABLE INTO TWO TABLES LIKE BELOW,

STUDENT DETAILS

ROLL NO	**NAME**	**BRANCH (FK)**
101	**SAI**	**CSE**
102	**ALLEN**	**CSE**
103	**JAMES**	**CSE**
104	**MILLER**	**CSE**
105	**WARNER**	**CSE**

BRANCH DETAILS

BRANCH (PK)	**HOD**	**OFFICE NUMBER**
CSE	**MR. Y**	**040-53337**

NOTE: BY THE ABOVE EXAMPLE WE AVOID INSERTION, DELETION AND UPDATION PROBLEMS.

<u>TYPES OF NORMAL FORMS:</u> NORMALIZATION CAN BE ACHIEVED IN MULTIPLE WAYS:

1. FIRST NORMAL FORM
2. SECOND NORMAL FORM
3. THIRD NORMAL FORM
4. BCNF
5. FOURTH NORMAL FORM
6. FIFTH NORMAL FORM

<u>FIRST NORMAL FORM (1NF):</u>

FOR A TABLE TO BE IN THE FIRST NORMAL FORM, IT SHOULD FOLLOW THE FOLLOWING 4 RULES:

1. **EACH COLUMN SHOULD CONTAIN ATOMIC VALUE(ATOMIC = SINGLE VALUE).**

EX:

COLUMN1	COLUMN2
A	X, Y
B	W, X
C	Y
D	Z

2. **A COLUMN SHOULD CONTAIN VALUES THAT ARE SAME DATATYPE.**

EX:

NAME	DOB
SAI	01-JAN-92
JONES	24-APR-84
18-DEC-85	MILLER

3. **ALL THE COLUMNS IN A TABLE SHOULD HAVE UNIQUE NAMES.**

EX:

NAME	NAME	DOB
SAI	SAI	16-OCT-93

4. THE ORDER IN WHICH DATA IS STORED, DOES NOT MATTER.

EX:

ROLLNO	FIRST_NAME	LAST_NAME
1	SAI	KUMAR
2	JONES	ROY
4	MILLER	JOY
3	JAMES	WARTON

EX: STUDENT TABLE

ROLL NO	NAME	SUBJECT
101	SAI	JAVA, ORACLE
102	JONES	PYTHON
103	ALLEN	C, C++

THE ABOVE TABLE ALREADY SATISFIES 3 RULES OUT OF THE 4 RULES, AS ALL OUR COLUMN NAMES ARE UNIQUE, WE HAVE STORED DATA IN THE ORDER WE WANTED TO AND WE HAVE NOT INTER-MIXED DIFFERENT TYPE OF DATA IN COLUMNS.

BUT OUT OF THE 3 DIFFERENT STUDENTS IN OUR TABLE, 2 HAVE OPTED FOR MORE THAN 1 SUBJECT. AND WE HAVE STORED THE SUBJECT NAMES IN A SINGLE COLUMN. BUT AS PER THE 1ST NORMAL FORM EACH COLUMN MUST CONTAIN ATOMIC VALUE.

TO AVOID THIS PROBLEM, WE HAVE TO BREAK THE VALUES INTO ATOMIC VALUES. HERE IS OUR UPDATED TABLE AND IT NOW SATISFIES THE FIRST NORMAL FORM.

< COMPOSITE PRIMARY KEY>

ROLL NO	NAME	SUBJECT
101	SAI	ORACLE
101	SAI	JAVA
102	JONES	PYTHON
103	ALLEN	C
103	ALLEN	C++

NOTE: BY DOING SO, ALTHOUGH A FEW VALUES ARE GETTING REPEATED BUT VALUES FOR THE SUBJECT COLUMN ARE NOW ATOMIC FOR EACH RECORD/ROW.

SECOND NORMAL FORM (2NF):

FOR A TABLE TO BE IN THE SECOND NORMAL FORM, IT MUST SATISFY TWO CONDITIONS:

1. **THE TABLE SHOULD BE IN THE FIRST NORMAL FORM.**
2. **THERE SHOULD BE NO PARTIAL DEPENDENCY.**

WHAT IS DEPENDENCY: IN A TABLE IF NON-KEY COLUMNS (NON-PRIMARY KEY) ARE DEPENDS ON KEY COLUMN(PRIMARY KEY) THEN IT IS CALLED AS FULLY DEPENDENCY / FUNCTIONAL DEPENDENCY.

```
            (PK)

      EX:   A     B     C     D
```

HERE, "A "IS A KEY COLUMN → "B"," C"," D" ARE NON-KEY COLUMNS.

EX:

(PK) STUDENT TABLE

STUDENT_ID	NAME	BRANCH	ADDRESS
101	SAI	CSE	HYD
102	SAI	IT	MUM
103	JAMES	CSE	CHENNAI
104	MILLER	CSE	HYD

NOTE: A PRIMARY KEY COLUMN(STID) CAN BE USED TO FETCH DATA ANY COLUMN IN THE TABLE.

<u>WHAT IS PARTIAL DEPENDENCY:</u> IN A TABLE IF NON-KEY COLUMN DEPENDS ON PART OF THE KEY COLUMN, THEN IT IS CALLED AS PARTIAL DEPENDENCY

<PRIMARY KEY (A,B) / COMPOSITE PRIMARY KEY>

EX: A B C D

HERE, "A AND B "IS A KEY COLUMNS → " C"," D" ARE NON-KEY COLUMNS. THEN "D" DEPENDS ON "B" BUT NOT "A" COLUMN.

EX: LET'S CREATE ANOTHER TABLE FOR SUBJECT, WHICH WILL HAVE SUBJECT_ID AND SUBJECT_NAME FIELDS AND SUBJECT_ID WILL BE THE PRIMARY KEY.

<PRIMARY KEY>   SUBJECT TABLE

SUBJECT_ID	**SUBJECT_NAME**
1	**ORACLE**
2	**JAVA**
3	**PYTHON**

NOW WE HAVE A STUDENT TABLE WITH STUDENT INFORMATION AND ANOTHER TABLE SUBJECT FOR STORING SUBJECT INFORMATION.

LET'S CREATE ANOTHER TABLE SCORE, TO STORE THE MARKS OBTAINED BY STUDENTS IN THE RESPECTIVE SUBJECTS.

WE WILL ALSO BE SAVING NAME OF THE TEACHER WHO TEACHES THAT SUBJECT ALONG WITH MARKS.

(COMPOSITE PRIMARY KEY) SCORE TABLE

STUDENT_ID	**SUBJECT_ID**	**MARKS**	**TEACHER**
101	**1**	**70**	**ORACLE TEACHER**
101	**2**	**75**	**JAVA TEACHER**
102	**1**	**80**	**OACLE TEACHER**
103	**3**	**68**	**PYTHON TEACHER**

- **IN THE SCORE TABLE WE ARE SAVING THE STUDENT_ID TO KNOW WHICH STUDENT'S MARKS ARE THESE AND SUBJECT_ID TO KNOW FOR WHICH SUBJECT THE MARKS ARE FOR.**

TOGETHER STUDENT_ID + SUBJECT_ID FORMS COMPOSITE PRIMARY KEY FOR THIS TABLE, WHICH CAN BE THE PRIMARY KEY.

NOTE:

1. **IN ABOVE SCORE TABLE,” TEACHER COLUMN” IS ONLY DEPENDS ON SUBJECT_ID BUT NOT ON STUDENT_ID IS CALLED AS “PARTIAL DEPENDENCY”.**
2. **IF THERE IS NO COMPOSITE PRIMARY KEY ON A TABLE THEN THERE IS NO PARTIAL DEPENDENCY.**

<u>HOW TO REMOVE PARTIAL DEPENDENCY:</u>THERE ARE MANY DIFFERENT SOLUTIONS TO REMOVE PARTIAL DEPENDENCY.SO OUR OBJECTIVE IS TO REMOVE “TEACHER” COLUMN FROM SCORE TABLE AND ADD TO SUBJECT TABLE. HENCE, THE SUBJECT TABLE WILL BECOME

SUBJECT TABLE

SUBJECT_ID	SUBJECT_NAME	TEACHER
1	ORACLE	ORACLE TEACHER
2	JAVA	JAVA TEACHER
3	PYTHON	PYTHON TEACHER

AND OUR SCORE TABLE IS NOW IN THE SECOND NORMAL FORM, WITH NO PARTIAL DEPENDENCY.

<COMPOSITE PRIMARY KEY>

STUDENT_ID	SUBJECT_ID	MARKS
101	1	70
101	2	75
102	1	80
103	3	68

THIRD NORMAL FORM (3NF):

FOR A TABLE TO BE IN THE THIRD NORMAL FORM THERE IS TWO CONDITIONS.

1. **IT SHOULD BE IN THE SECOND NORMAL FORM.**
2. **AND IT SHOULD NOT HAVE TRANSITIVE DEPENDENCY.**

TRANSITIVE DEPENDENCY: IN TABLE IF NON-KEY COLUMN DEPENDS ON NON-KEY COLUMN, THEN IT IS CALLED AS TRANSITIVE DEPENDENCY.

(COMPOSITE PRIMARY KEY)

EX: A B C D

HERE, "A AND B "ARE KEY COLUMNS → " C"," D" ARE NON-KEY COLUMNS. THEN "D" DEPENDS ON "C" BUT NOT "A &B" COLUMNS.

NOTE: IN THE SCORE TABLE, WE NEED TO STORE SOME MORE INFORMATION, WHICH IS THE EXAM NAME AND TOTAL MARKS, SO LET'S ADD 2 MORE COLUMNS TO THE SCORE TABLE.

<COMPOSITE PRIMARY KEY>   SCORE TABLE

STUDENT_ID	SUBJECT_ID	MARKS	EXAM_NAME	TOTAL_MARKS

- **WITH EXAM NAME AND TOTAL MARKS ADDED TO OUR SCORE TABLE, IT SAVES MORE DATA NOW. PRIMARY KEY FOR OUR SCORE TABLE IS A COMPOSITE KEY, WHICH MEANS IT'S MADE UP OF TWO ATTRIBUTES OR COLUMNS → STUDENT +SUBJECT**
- **OUR NEW COLUMN EXAM NAME DEPENDS ON BOTH STUDENT AND SUBJECT. FOR EXAMPLE, A MECHANICAL ENGINEERING STUDENT WILL HAVE WORKSHOP EXAM BUT A COMPUTER SCIENCE STUDENT WON'T. AND FOR SOME SUBJECTS YOU HAVE PRACTICAL EXAMS AND FOR SOME YOU DON'T. SO, WE CAN SAY THAT EXAM NAME IS DEPENDENT ON BOTH STUDENT ID AND SUBJECT ID.**
- **WELL, THE COLUMN TOTAL MARKS DEPEND ON EXAM NAME AS WITH EXAM TYPE THE TOTAL SCORE CHANGES. FOR EXAMPLE, PRACTICAL IS LESS MARKS WHILE THEORY EXAMS ARE HAVING MORE MARKS.**
- **BUT EXAM NAME IS JUST ANOTHER COLUMN IN THE SCORE TABLE. IT IS NOT A PRIMARY KEY AND TOTAL MARKS DEPENDS ON IT.**
- **THIS IS TRANSITIVE DEPENDENCY. WHEN A NON-PRIME ATTRIBUTE DEPENDS ON OTHER NON-PRIME ATTRIBUTES RATHER THAN DEPENDING UPON THE PRIME ATTRIBUTES OR PRIMARY KEY.**

<u>HOW TO REMOVE TRANSITIVE DEPENDENCY:</u>AGAIN, THE SOLUTION IS VERY SIMPLE. TAKE OUT THE COLUMN'S EXAM NAME AND TOTAL MARKS FROM SCORE TABLE AND PUT THEM IN AN EXAM TABLE AND USE THE EXAM_ID WHEREVER REQUIRED.

SCORE TABLE: IN 3RD NORMAL FORM

STUDENT_ID	SUBJECT_ID	MARKS	EXAM_ID(FK)

EXAM TABLE

EXAM_ID(PK)	EXAM_NAME	TOTAL_MARKS
1	WORKSHOP	200
2	MAINS	70
3	PRACTICAL'S	30

SUPER KEY & CANDIDATE KEY:

SUPER KEY: A COLUMN (OR) COMBNATION OF COLUMNS WHICH ARE UNIQUELY IDENTIFYING A ROW IN A TABLE IS CALLED AS SUPER KEY.

CANDIDATE KEY: A MINIMAL SUPER KEY WHICH IS UNIQUELY IDENTIFYING A ROW IN A TABLE IS CALLED AS CANDIDATE KEY.

(OR)

A SUPER KEY WHICH IS SUBSET OF ANOTHER SUPER KEY, BUT THE COMBINATION OF SUPER KEYS IS NOT A CANDIDATE KEY.

IN DB DESIGN ONLY DB DESIGNER USES SUPER KEY AND CANDIDATE KEY.THAT MEAN FIRST DESIGNERS SELECT SUPER KEYS AND THEN ONLY THEY ARE SELETING CANDIDATE KEYS FROM THOSE SUPER KEYS.

EX: STUDENT TABLE

STUDENT_ID	NAME	BRANCH	MAILID	REG_NUMBER
101	SAI	CSE	SAI@GAMIL.COM	CS-10021
102	JONES	CSE	JOY@GMAIL.COM	CS-10022
103	ALLEN	IT	ALL@YMAIL.COM	IT-20021
104	SAI	EEE	MI@HOTMAIL.COM	EE-30021

EX. OF SUPER KEYS:

STID | STID + MAILID |

MAILID | MAILID + REG_NUMBER |STID + MAILID + REG_NUMBER

REG_NUMBER| REG_NUMBER + STID |

EX. ON CANDIDATE KEYS:

STID

MAILID

REG_NUMBER

BOYCE-CODD NORMAL FORM (BCNF):

FOR A TABLE TO SATISFY THE BOYCE-CODD NORMAL FORM, IT SHOULD SATISFY THE FOLLOWING TWO CONDITIONS:

1. **IT SHOULD BE IN THE THIRD NORMAL FORM.**
2. **AND, FOR ANY DEPENDENCY A → B, A SHOULD BE A SUPER KEY.**

EX:

(COMPOSITE PRIMARY KEY) COLLEGE ENROLLMENT TABLE

STUDENT_ID	SUBJECT(B)	PROFESSOR(A)
101	JAVA	P. JAVA
101	C++	P. CPP
102	JAVA	P. JAVA2
103	ORACLE	P. ORACLE
104	JAVA	P. JAVA

IN THE TABLE ABOVE, STUDENT ID, SUBJECT FORM PRIMARY KEY, WHICH MEANS SUBJECT COLUMN, IS A PRIME ATTRIBUTE.BUT THERE IS ONE MORE DEPENDENCY, PROFESSOR → SUBJECT. AND WHILE SUBJECT IS A PRIME ATTRIBUTE, PROFESSOR IS A NON-PRIME ATTRIBUTE, WHICH IS NOT ALLOWED BY BCNF.

HOW TO SATISFY BCNF?

TO MAKE THIS RELATION (TABLE) SATISFY BCNF, WE WILL DECOMPOSE THIS TABLE INTO TWO TABLES, STUDENT TABLE AND PROFESSOR TABLE.

BELOW WE HAVE THE STRUCTURE FOR BOTH THE TABLES.

STUDENT TABLE

STUDENT_ID	PROFESSOR_ID
101	1
101	2

PROFESSOR TABLE

(COMPOSITE PRIMARY KEY)

PROFESSOR_ID	PROFESSOR	SUBJECT
1	P. JAVA	JAVA
2	P. CPP	C++

AND NOW, THIS RELATION SATISFIES BOYCE-CODD NORMAL FORM.

FOURTH NORMAL FORM (4NF):

FOR A TABLE TO SATISFY THE FOURTH NORMAL FORM, IT SHOULD SATISFY THE FOLLOWING TWO CONDITIONS:

1. **IT SHOULD BE IN THE BOYCE-CODD NORMAL FORM.**
2. **A TABLE DOES NOT CONTAIN MORE THAN ONE INDEPENDENT MULTI-VALUED ATTRIBUTE / MULTIVALUED DEPENDENCY.**

MULTI VALUED DEPENDENCY: IN A TABLE ONE COLUMN SAME VALUE MATCH WITH MULTIPLE VALUES OF ANOTHER COLUMN IS CALLED AS MULTI VALUED DEPENDENCY.

NOTE: GENERALLY, WHEN A TABLE HAVING MORE THAN ONE INDEPENDENT MULTI VALUED ATTRIBUTES THEN THE TABLE HAVING MORE DUPLICATE DATA FOR REDUCING THIS DUPLICATE DATA THEN DB DESIGNERS USE 4NF PROCESS OTHERWISE NO NEED(IT IS OPTIONAL).

EX: COLLEGE ENROLLMENT TABLE (5NF)

STUDENT_ID	COURSE	HOBBY
1	ORACLE	CRICKET
1	JAVA	READING
1	C#	HOCKEY

IN THE TABLE ABOVE, THERE IS NO RELATIONSHIP BETWEEN THE COLUMNS COURSE AND HOBBY. THEY ARE INDEPENDENT OF EACH OTHER.SO THERE IS MULTI-VALUE DEPENDENCY, WHICH LEADS TO UN-NECESSARY REPETITION OF DATA.

IDENTIFY INDEPENDENT MULTI VALUED ATTRIBUTES AND THOSE ATTRIBUTES MOVE INTO SEPARATE TABLES THESE TABLES ARE CALLED AS 4NF TABLES. THESE TABLES DO NOT CONTAIN MORE THAN ONE INDEPENDENT MULTI VALUED ATTRIBUTE (COLUMN).

HOBBIES TABLE(4NF)

STUDENT_ID	**HOBBY**
1	**CRICKET**
1	**READING**
1	**HOCKEY**

COURSE OPTED TABLE(4NF)

STUDENT_ID	**COURSE**
1	**ORACLE**
1	**JAVA**
1	**C#**

FIFTH NORMAL FORM (5NF):

IF A TABLE HAVING MULTI VALUED ATTRIBUTES AND ALSO THAT TABLE CANNOT DECOMPOSED INTO MULTIPLE TABLES IS CALLED AS FIFTH NORMAL FORM.

GENERALLY, IN 4NF RESOURCE TABLE SOME ATTRIBUTES ARE NOT LOGICALLY RELATED WHERE AS IN 5NF RESOURCE TABLE ALL ATTRIBUTES ARE RELATED TO ONE TO ANOTHER.

FIFTH NORMAL FORM IS ALSO CALLED AS PROJECT JOINED NORMAL FORM BECAUSE IF POSSIBLE DECOMPOSING TABLE INTO NUMBER OF TABLES AND ALSO WHENEVER WE ARE JOINING THOSE TABLES THEN THE RESULT RECORDS MUST BE AVAILABLE IN RESOURCE TABLE.

PL/SQL

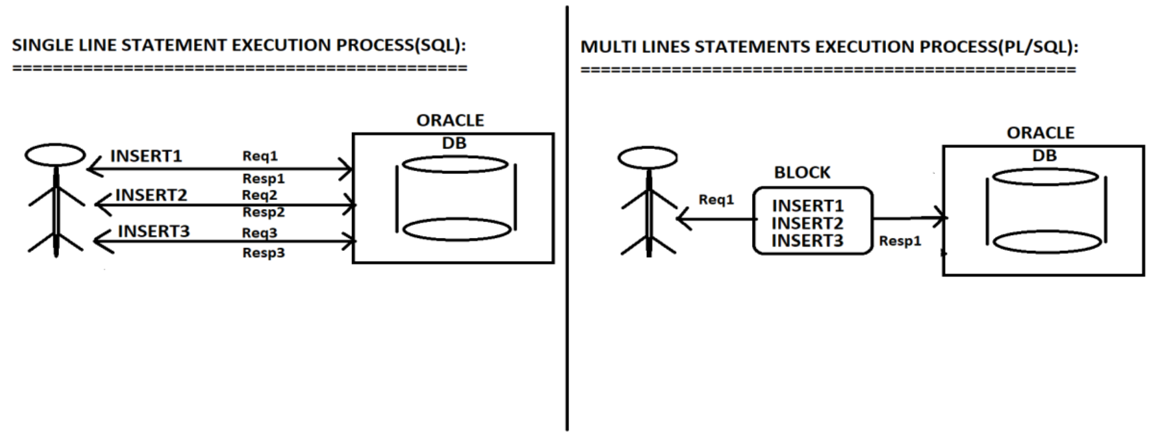
INTRODUCTION TO PL/SQL:

PL/SQL STANDS FOR PROCEDURAL LANGUAGE WHICH IS AN EXTENSION OFSQL.PL/SQL WAS INTRODUCED IN ORACLE 6.0 VERSION.

SQL IS A NON-PROCEDURAL LANGUAGE WHEREAS PL/SQL IS A PROCEDURAL LANGUAGE.

SQL SUPPORTS A SINGLE LINE STATEMENT (QUERY) EXECUTION PROCESS WHEREAS PL/SQL SUPPORTS MULTI LINES STATEMENTS(PROGRAM) EXECUTION PROCESS.

IN SQL EVERY QUERY STATEMENT IS COMPILING AND EXECUTING INDIVIDUALLY.SO THAT NO. OFCOMPILATIONS ARE INCRESED AND REDUCE PERFORMANCE OF DATABASE.



IN PL/SQL ALL SQL QUERIES ARE GROUPED INTO A SINGLE BLOCK AND WHICH WILL COMPILE AND EXECUTE ONLY ONE TIME.SO THAT IT WILL REDUCE NO. OF COMPILATIONS AND IMPROVE PERFORMANCE OF DATABASE.

FEATURES OF PL/SQL:

1. TO IMPROVES PERFORMANCE.

2. SUPPORTING CONDITIONAL & LOOPING STATEMENTS.

3. SUPPORTING REUSABILITY.

4. PROVIDING SECURITY BECAUSE ALL PROGRAMS ARE SAVED

IN DATABASE AND AUTHORIZED USER CAN ONLY ACCESS THE PROGRAMS.

5. SUPPORTING PORTABILITY I.E PL/SQL PROGRAMS CAN BE MOVED FROM ONE

PLATFORM TO ANOTHER PLATFORM WITHOUT ANY CHANGES.

6. SUPPORTING EXCEPTION HANDLING MECHANISM.

7. SUPPORTING MODULAR PROGRAMMING I.E IN A PL/SQL A BIG PROGRAM CAN BE DIVIDED INTO SMALL MODULES WHICH ARE CALLED AS STORED PROCEDURE AND

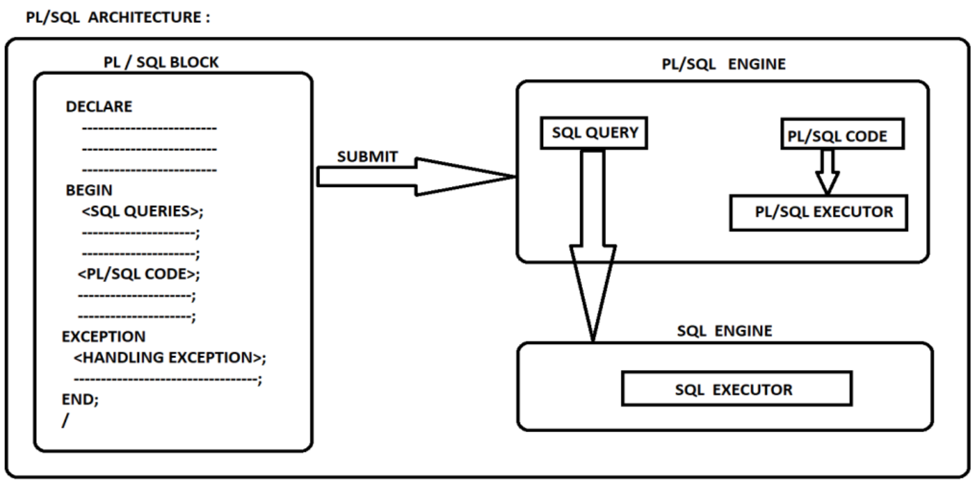
STORED FUNCTIONS.

PL/SQL ARCHITECTURE:

PL/SQL IS BLOCK STRUCTURE PROGRAMMING LANGUAGE.WHICH IS HAVING THE FOLLOWING TWO ENGINES THOSE ARE

1. SQL ENGINE

2. PL/SQL ENGINE



WHENEVER WE ARE SUBMITING A PL/SQL BLOCK INTO ORACLE SERVER THEN ALL SQL STATEMENTS(QUERIES) ARE SEPERATED AND EXECUTING BY SQLQUERY EXECUTOR WITH IN SQL ENGINE.WHERE AS ALL PL/SQL STATEMENTS(CODE) ARE SEPERATED AND EXECUTING BY PL/SQL CODE EXECUTOR WITH IN PL/SQL ENGINE.

WHAT IS BLOCK:

A BLOCK IS A SET OF STATEMENTS WHICH ARE COMPILE & EXECUTED BYORACLE AS A SINGLE UNIT. PL/SQL SUPPORTING THE FOLLOWING TWO TYPES OF BLOCKS THOSE ARE,

1. ANONYMOUS BLOCK
2. SUB BLOCK

DIFFERENCES BETWEEN ANONYMOUS & SUB BLOCK:

ANONYMOUS BLOCK	SUB BLOCK
1. UNNAMED BLOCK	1. NAMED BLOCK
2. THIS BLOCK CODE IS NOT SAVED IN DB.	2. THIS BLOCK CODE IS SAVED IN DB AUTOMATICALLY.
3. IT CANNOT REUSABLE.	3. IT CAN BE REUSABLE.
4. EVERY TIME COMPILATION OF CODE.	4. PRE - COMPILED CODE (FIRST TIME COMPILATION ONLY)
5. ARE USING IN "DB TESTING".	5. ARE USING IN APPLICATION DEVELOPMENT LIKE "JAVA", ".NET" & "DB APPLICATIONS ".

ANONYMOUS BLOCKS:

THESE ARE UNNAMED BLOCKS IN PL/SQL.WHICH CONTAINS THREE MORE BLOCKS THOSE ARE,

I) DECLARATION BLOCK

II) EXECUTION BLOCK

III) EXCEPTION BLOCK

I) DECLARATION BLOCK:

> THIS BLOCK STARTS WITH " DECLARE " STATEMENT.

> DECLARING VARIABLES, CURSORS, USER DEFINE EXCEPTIONS.

> IT IS OPTIONAL BLOCK.

II) EXECUTION BLOCK:

> THIS BLOCK STARTS WITH " BEGIN " STATEMENT & ENDS WITH "END" STATEMENT.

> IMPLEMENTING SQL STATEMENTS(SQL) & LOGICAL CODE OF A PROGRAM (PL/SQL).

> IT IS MANDATORY BLOCK.

III) EXCEPTION BLOCK:

> THIS BLOCK STARTS WITH "EXCEPTION" STATEMENT.

> HANDLING EXCEPTIONS.

> IT IS AN OPTIONAL BLOCK.

STRUCTURE OF PL/SQL BLOCK:

```
DECLARE
<VARIABLES, CURSOR, UD EXCEPTIONS>;
BEGIN
< WRITING SQL STATEMENTS>;
< PL/SQL LOGICAL CODE>;
EXCEPTION
< HANDLING EXCEPTIONS>;
END;
/
```

VARIABLES IN PL/SQL:

STEP1: DECLARING VARIABLES:

SYNTAX:

```
        DECLARE
        <VARIABLE NAME><DT>[SIZE];
```

EX:

```
        DECLARE
        A NUMBER (10) (OR) A INT;
        B VARCHAR2(10);
```

STEP2: ASSIGNING / STORING A VALUE INTO VARIABLE:

SYNTAX:

```
        <VARIABLE NAME>: =<VALUE>;
```

EX:

```
        A: = 1021;
        B: = 'SAI';
```

HERE,

```
        : =     - ASSIGNMENT OPERATOR IN PL/SQL
        =       - COMPARISION OPERATOS IN PL/SQL
```

STEP3: PRINTING VARIABLES VALUES:

SYNTAX:

```
DBMS_OUTPUT.PUT_LINE (<VARIABLE NAME > (OR) '<UD MESSAGE>');
```

EX:

```
        DBMS_OUTPUT.PUT_LINE(A);
        DBMS_OUTPUT.PUT_LINE(B);
        DBMS_OUTPUT.PUT_LINE ('WELCOME TO PL/SQL');
```

EX1:

TO PRINT "WELCOME TO PL/SQL" STATEMENT.

SOL:

```
SQL> BEGIN
        DBMS_OUTPUT.PUT_LINE ('WELCOME TO PL/SQL');
        END;
        /

PL/SQL PROCEDURE SUCCESSFULLY COMPLETED.
```

NOTE:

THE ABOVE PROGRAM WILL NOT DISPLAY THE OUTPUT OF A PL/SQL PROGRAM.IF ORACLE SERVER WANT TO DISPLAY OUTPUT OF A PL/SQL PROGRAM THEN WE USE THE FOLLOWING SYNTAX,

SYNTAX:

```
        SET SERVEROUTPUT OFF / ON;
```

HERE,

OFF:IT IS DEFAULT.OUTPUT IS NOT DISPLAY

ON: OUTPUT IS DISPLAY

```
SQL> SET SERVEROUTPUT ON;

SQL> /

WELCOME TO PL/SQL
```

EX2:

TO PRINT VARIABLES VALUES?

SOL:

```
SQL> DECLARE

         X NUMBER (10);

         Y NUMBER (10);

        BEGIN

         X: =100;

         Y: =200;

 DBMS_OUTPUT.PUT_LINE ('VARIABLES VALUES ARE:'||X||','||Y);

  END;

  /

VARIABLES VALUES ARE:100,200
```

EX3:

TO PRINT SUM OF TWO NUMBERS AT RUNTIME?

SOL:

```
DECLARE

         X NUMBER (2);

         Y NUMBER (2);
```

```
        Z NUMBER (10);
BEGIN
        X: =&X;
        Y: =&Y;
        Z: =X+Y;
        DBMS_OUTPUT.PUT_LINE(Z);
END;
/
```

OUTPUT:

```
ENTER VALUE FOR X: 10
OLD   6: X: =&X;
NEW   6: X: =10;
ENTER VALUE FOR Y: 20
OLD   7: Y: =&Y;
NEW   7: Y: =20;
30
```

VERIFY:

ON = DISPLAY OLD, NEW BIND VARIABELE STATEMENTS

OFF = DOESNOT DISPLAY OLD, NEW BIND VARIABLES STATEEMTNS

SYNTAX:

```
        SET VERIFY ON / OFF
```

EX:

```
SQL> SET VERIFY OFF;
SQL> /

ENTER VALUE FOR X: 10

ENTER VALUE FOR Y: 20

30
```

<u>SELECT…… INTO STATEMENT:</u>

STORING A TABLE COLUMNS VALUES INTO VARIABELS.RETURNS A SINGLE ROW (OR) A SINGLE VALUE.CAN USE IN EXECUTION BLOCK.

SYNTAX:

SELECT <COLUMN NAME1>, <COLUMN NAME2>, ………..INTO <VARIABLE NAME1>, <VARIABLE NAME2>…………… FROM<TN> [WHERE <CONDITION>];

EX1:

WA PL/SQL PRG. TO DISPLAY ENAME, SALARY DETAILS FROM EMP TABLE AS PER THE GIVEN EMPNO BY USING SELECT ……INTO STATEMENT?

SOL:

```
DECLARE

V_ENAME VARCHAR2(10);

V_SAL NUMBER (10);

BEGIN

SELECT ENAME, SAL INTO V_ENAME, V_SAL FROM EMP WHERE
EMPNO=&EMPNO;

DBMS_OUTPUT.PUT_LINE(V_ENAME||','||V_SAL);

END;

/
```

OUTPUT:

```
ENTER VALUE FOR EMPNO: 7788

SCOTT,3000
```

EX:

WA PL/SQL PRG. TO FETCH MAX.SALARY OF EMP TABLE BY USING

"SELECT INTO" STATEMENT?

SOL:

```
DECLARE

V_MAXSAL NUMBER (10);

BEGIN

SELECT MAX(SAL) INTO V_MAXSAL FROM EMP;

DBMS_OUTPUT.PUT_LINE(V_MAXSAL);

END;

/
```

OUTPUT:

```
5000
```

<u>VARIABLES ATTRIBUTES (OR) ANCHOR NOTATIONS:</u>

VARIABLES ATTRIBUTES ARE USED IN PLACE OF DATATYPES AT VARIABLE DECLARATION.

WHENEVER WE ARE USING VARIABLES ATTRIBUTES INTERNALLY ORACLE SERVERIS ALLOCATE SOME MEMORY FOR THESE VARIABLES ATTRIBUTES FOR STORINGTHE CORRESPONDING VARIABLE COLUMN DATATYPE WHICH WAS ASSIGNED AT THE TIME OF TABLE CREATION.

VARIABLES ATTRIBUTES ARE ALSO CALLED AS "ANCHOR NOTATIONS".

THE ADVANTAGE OF VARIABLES ATTRIBUTES ARE WHENEVER WE WANT TO CHANGE

A PARTICULAR COLUMN DATATYPE IN A TABLE THEN THE CORRESPONDING COLUMNVARIABLE DATATYPE ALSO CHANGED IN VARIABLE ATTRIBUTE MEMORY AUTOMATICALLY.

PL/SQL SUPPORTS THE FOLLOWING TWO TYPE VARIABLES ATTRIBUTES ARE,

1. COLUMN LEVEL ATTRIBUTES

2. ROW LEVEL ATTRIBUTES

1. COLUMN LEVEL ATTRIBUTES:

IN THIS LEVEL WE ARE DEFINING VARIABLES ATTRIBUTES FORINDIVIDUAL COLUMNS.IT IS REPRESENTING WITH "%TYPE" STATEMENT.

SYNTAX:

```
        <VARIABLE NAME><TN>. <COLUMN NAME>%TYPE;
```

EX:

```
        V_ENAME   EMP.ENAME%TYPE;
        V_SAL         EMP.SAL%TYPE;
```

PROGRAM1:

```
DECLARE
V_ENAME EMP.ENAME%TYPE;
V_SAL EMP.SAL%TYPE;
BEGIN
SELECT ENAME, SAL INTO V_ENAME, V_SAL FROM EMP WHERE
EMPNO=&EMPNO;
DBMS_OUTPUT.PUT_LINE(V_ENAME||','||V_SAL);
END;
/
```

OUTPUT:

```
ENTER VALUE FOR EMPNO: 7788
SCOTT,3000
```

<u>2. ROW LEVEL ATTRIBUTES:</u>

IN THIS LEVEL WE ARE DECLARING A SINGLE VARIABLE WILL REPRESENT ALLDIFFERENT DATATYPES OF COLUMNS IN A TABLE.IT REPRESENT WITH "%ROWTYPE ".

SYNTAX:

```
        <VARIABLE NAME><TABLE NAME>%ROWTYPE;
```

```
EX:    I      EMP%ROWTYPE;
```

<u>PROGRAM2:</u>

```
DECLARE

I EMP%ROWTYPE;

BEGIN

SELECT ENAME, SAL INTO I. ENAME, I.SAL FROM EMP WHERE
EMPNO=&EMPNO;

DBMS_OUTPUT.PUT_LINE (I. ENAME||','||I.SAL);

END;

/
```

(OR)

```
DECLARE

I EMP%ROWTYPE;

BEGIN

SELECT * INTO I FROM EMP WHERE EMPNO=&EMPNO;

DBMS_OUTPUT.PUT_LINE (I. ENAME||','||I.SAL||','||I.DEPTNO);

END;

/
```

<u>CONTROL STRUCTURES:</u>

-USED TO CONTROL FLOW OF THE PROGRAM.

-THERE ARE THREE TYPES OF CONTROL STRUCTURES.

I. CONDITIONAL CONTROL STRUCTURES

II. BRANCHING CONTROL STRUCTURES

III. ITERATION CONTROL STRUCTURES

<u>I. CONDITIONAL CONTROL STRUCTURES:</u>

<u>I. SIMPLE IF:</u> IT CONTAINS ONLY TRUE BLOCK.

SYNTAX:

```
IF <CONDITION> THEN
<EXEC-STATEMENTS>; -- TRUE BLOCK
END IF;
```

<u>II. IF. ELSE</u>: IT CONTAINS BOTH TRUE BLOCK & FALSE BLOCK.

SYNTAX:

```
IF <CONDITION> THEN
<EXEC-STATEMENTS>; -- TRUE BLOCK
 ELSE
<EXEC-STATEMENTS>; -- FALSE BLOCK
 END IF;
```

III. NESTED IF:

-> IF WITHIN THE IF IS CALLED AS NESTED IF.

SYNTAX:

```
IF <CONDITION> THEN
     IF <CONDITION> THEN
<EXEC-STATEMENT>;
     ELSE
<EXEC-STATEMENTS>;
     END IF;
 ELSE
     IF <CONDITION> THEN
<EXEC-STATEMENT>;
     ELSE
<EXEC-STATEMENTS>;
     END IF;
 END IF;
```

IV. IF..ELSE LADER:

SYNTAX:

```
IF <CONDITION> THEN
<EXEC-STATEMENTS>;
 ELSIF <CONDITION> THEN
<EXEC-STATEMENTS>;
.....................
 ELSE
<EXEC-STATEMENTS>;
 END IF;
```

<u>II. BRANCHING CONTROL STURCTURES:</u>

<u>CASE</u>:

SYNTAX:

```
CASE <VARIABLE/EXPRESSION>
 WHEN <COND> THEN
<EXEC-STATEMENTS>;
 WHEN <COND> THEN
<EXEC-STATEMENTS>;
 WHEN <COND> THEN
<EXEC-STATEMENTS>;
 ELSE
<EXEC-STATEMENT>;
 END CASE;
```

<u>ITERATION CONTROL STATEMENTS:</u>

<u>I. SIMPLE LOOP:</u>

-> IT IS AN INFINITE LOOP.IF WE WANT BREAK A SIMPLE LOOP THEN WE SHOULD USE "EXIT" STATEMENT.

SYNTAX:

```
LOOP
<EXEC-STATEMENTS>;
END LOOP;
```

<u>II. WHILE LOOP:</u>

SYNTAX:

```
WHILE <CONDITION>
LOOP
<EXEC-STATEMENTS>;
<INCRE/DECRE>;
END LOOP;
```

<u>III. FOR LOOP:</u>

-> BY DEFAULT, IT IS INCREMENTED BY 1.

SYNTAX:

```
FOR <INDEX_VARIABLE> IN <START_VALUE>..<END_VALUE>
LOOP
<EXEC-STATEMENTS>;
END LOOP;
```

<u>CURSORS:</u>

CURSOR IS A TEMP.MEMORY / A PRIVATE SQL AREA(PSA) / A WORK SPACE.CURSOR ARE TWO TYPES.THOSE ARE:

I) EXPLICIT CURSOR (USER DEFINE CURSOR)

II) IMPLICIT CURSOR (SYSTEM DEFINE CURSOR)

<u>I) EXPLICIT CURSOR:</u>

THESE CURSOR ARE CREATING BY USER FOR HOLDING MULTIPLE ROWS BUT WE CAN ACCESS ONLY ONE ROW AT TIME. (ONE BY ONE / ROW BY ROW MANNER).

IF WE WANT TO CREATE AN EXPLICIT CURSOR, WE NEED FOLLOW THE FOLLOWINGFOUR STEPS.THOSE ARE

1) DECLARING A CURSOR

2) OPEN A CURSOR

3) FETCH ROWS FROM A CURSOR

4) CLOSE A CURSOR

STEPS TO CREATE EXPLICIT CURSOR:

1)DECLARING A CURSOR:IN THIS PROCESS WE DEFINE A CURSOR.

SYNTAX:

DECLARE CURSOR <CURSORNAME> IS < SELECT STATEMENT>;

2)OPENING A CURSOR:WHEN WE OPEN A CURSOR, IT WILL INTERNALLY EXECUTE THE SELECT STATEMENT THAT IS ASSOCIATED WITH THE CURSOR DECLARTION AND LOAD THE DATA INTO CURSOR.

SYNTAX:

OPEN < CURSORNAME>;

3)FETCHING DATA FROM THE CURSOR:IN THIS PROCESS WE ACCESS ROW BY ROW FROM CURSOR.

SYNTAX:

FETCH <CURSORNAME> INTO <VARIABLES>;

4)CLOSING A CURSOR: IN THIS PROCESS, IT RELEASES THE CURRENT RESULT SET OF THE CURSOR LEAVING THE DATASTRUCTURE AVAILABLE FOR REOPENING.

SYNTAX:

CLOSE <CURSORNAME>;

ATTRIBUTES OF EXPLICIT CURSORS:

IT SHOWS STATUS OF THE CURSOR AND IT RETURNS BOOLEAN VALUE.

SYNTAX:

<CURSOR_NAME>%<ATTRIBUTE>;

A. %ISOPEN:

IT RETURNS TRUE, WHEN THE CURSOR OPENS SUCCESSFULLY.

B. %FOUND:

IT RETURNS TRUE, WHEN THE CURSOR CONTAINS DATA.

C. %NOTFOUND:

IT RETURNS TRUE, WHEN THE CURSOR DOESN'T FIND ANY DATA.

D.%ROWCOUNT:

IT RETURS NO. OF FETCH STATEMENTS EXECUTED.RETURN TYPE IS NUMBER.

EX1:

WA CURSOR PROGRAM TO FETCH A SINGLE ROW FROM EMP TABLE?

SOL:

```
DECLARE CURSOR C1 IS 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);

CLOSE C1;

END;

/
```

OUTPUT:

```
SMITH,800
```

EX2: TO FETCH MULTIPLE ROWS FROM EMP TABLE BY USING LOOPING STATEMENTS?

<u>I) BY USING SIMPLE LOOP:</u>

- IT IS AN INFINITE LOOP.SO THAT WE NEED BREAK A LOOP THEN WE ARE USING "EXIT" STATEMENT.

SOL:

```
 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;

 /
```

OUTPUT:

```
SMITH,800

ALLEN,1600

WARD,1250

.................

..................
```

<u>II) BY USING WHILE LOOP:</u>

```
DECLARE CURSOR C1 IS SELECT ENAME, SAL FROM EMP;

 V_ENAME VARCHAR2(10);

 V_SAL NUMBER (10);

 BEGIN

 OPEN C1;

 FETCH C1 INTO V_ENAME, V_SAL; ---LOOP START FROM 1ST ROW

 WHILE(C1%FOUND)
 LOOP

 DBMS_OUTPUT.PUT_LINE(V_ENAME||','||V_SAL);

 FETCH C1 INTO V_ENAME, V_SAL; ---LOOP CONTINUE UPTO LAST ROW

 END LOOP;

 CLOSE C1;

 END;

 /
```

III) BY USING FOR LOOP:

```
DECLARE CURSOR C1 IS SELECT ENAME, SAL FROM EMP;

BEGIN

FOR I IN C1

LOOP

DBMS_OUTPUT.PUT_LINE (I. ENAME||','||I.SAL);

END LOOP;

END;

/
```

NOTE:

WHENEVER WE ARE USING "FOR LOOP" STATEMENT IN CURSOR FOR FETCHING ROWS FROM A CURSOR MEMORY THEN THERE NO NEED TO OPEN CURSOR, FETCH ROWFROM CURSOR AND CLOSE CURSOR BY EXPLICITLY BECAUSE INTERNALLY ORACLE SERVER WILL OPEN, FETCH AND CLOSE CURSOR BY IMPLICITLY.

HERE, FOR LOOP EXECUTE NO. OF TIMES DEPENDS ON NO. OF ROWS IN CURSOR(C1). EVERY TIME FOR LOOP IS EXECUTE AND FETCH A ROW FROM C1 AND

ASSIGNED / STORED IN LOOP VARIABLE (I) AND LATER I LOOP VARIABLE VALUES AREPRINTED.

EX3:

WA CURSOR PROGRAM TO FETCH TOP FIVE HIGHEST SALARIES EMPLOYEEROWS FROM EMP TABLE?

SOL:

```
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;

EXIT WHEN C1%ROWCOUNT>5;

DBMS_OUTPUT.PUT_LINE(V_ENAME||','||V_SAL);

END LOOP;

CLOSE C1;

END;

/
```

OUTPUT:

```
KING,5000

FORD,3000

SCOTT,3000

JONES,2975

BLAKE,2850
```

EX4:

WA CURSOR PROGRM TO FETCH EVEN POSITION ROWS FROM EMP TABLE?

SOL:

```
 DECLARE CURSOR C1 IS SELECT EMPNO, ENAME FROM EMP;
 V_EMPNO NUMBER (10);
 V_ENAME VARCHAR2(10);
 BEGIN
 OPEN C1;
 LOOP
 FETCH C1 INTO V_EMPNO, V_ENAME;
 EXIT WHEN C1%NOTFOUND;
 IF MOD(C1%ROWCOUNT,2) =0 THEN
 DBMS_OUTPUT.PUT_LINE(V_EMPNO||','||V_ENAME);
 END IF;
 END LOOP;
 CLOSE C1;
 END;
 /
```

OUTPUT:

```
7499, ALLEN
7566, JONES
7698, BLAKE
7788, SCOTT
7844, TURNER
```

EX5: WA CURSOR PROGRAM TO FETCH 9TH POSITION ROW FROM EMP TABLE?

SOL:

```
DECLARE CURSOR C1 IS SELECT ENAME FROM EMP;
V_ENAME VARCHAR2(10);
BEGIN
OPEN C1;
LOOP
FETCH C1 INTO V_ENAME;
EXIT WHEN C1%NOTFOUND;
IF C1%ROWCOUNT=9 THEN
DBMS_OUTPUT.PUT_LINE(V_ENAME);
END IF;
END LOOP;
CLOSE C1;
END;
/
```

OUTPUT:

```
KING
```

IMPLICIT CURSOR:

- THESE CURSOR ARE DECLARING BY ORACLE SERVER BY DEFAULT.ORACLE DECLARE THESE CURSOR AFTER EXECUTION OF DML COMMAND (INSERT / UPDATE / DELETE).

- IMPLICIT CURSOR TELLING US THE STATUS OF LAST DML COMMAND WHETHER SUCCESSFULL OR NOT.

<u>ATTRIBUTES OF IMPLICIT CURSORS:</u>

1. %ISOPEN:

- IT RETURNS TRUE THEN CURSOR SUCCESSFUL OPEN OTHERWISE RETURNS FALSE.

2. %NOTFOUND:

- IT RETURNS TRUE THEN LAST DML COMMAND IS FAIL OTHERWISE RETURNS FALSE.

3. %FOUND:

- IT RETRUNS TRUE THEN LAST DML COMMAND IS SUCCESSFULLY EXECUTED OTHERWISE RETURNS FALSE.

4.%ROWCOUNT:

- IT RETURNS NO. OF ROWS AFFECTED BY LAST DML COMMAND.

EX:

```
 DECLARE

 V_EMPNO NUMBER (10);

 BEGIN

 V_EMPNO: =&V_EMPNO;

 DELETE FROM EMP WHERE EMPNO=V_EMPNO;

 IF SQL%FOUND THEN

 DBMS_OUTPUT.PUT_LINE ('RECORD IS DELETED');

 ELSE

 DBMS_OUTPUT.PUT_LINE ('RECORD IS NOT EXISTS');

 END IF;

 END;

 /
```

<u>OUTPUT:</u>

```
ENTER VALUE FOR V_EMPNO: 7788

RECORD IS DELETED
```

SUB BLOCKS:

A SUB BLOCK IS A NAMED BLOCK OF CODE THAT IS DIRECTLY SAVED ON THE DB SERVER AND IT CAN BE EXECUTED WHEN AND WHERE IT IS REQUIRED. WE HAVE THREE TYPES OF SUB BLOCKS IN ORACLE.

1.STORED PROCEDURES

2.STORED FUNCTIONS

4.TRIGGERS

STORED PROCEDURES:

A STORED PROCEDURE IS A DATABASE OBJECT WHICH CONTAINS PRECOMPILED QUERIES. STORED PROCEDURES ARE A BLOCK OF CODE DESIGNED TO PERFORM A TASK WHENEVER WE CALLED AND MAY BE OR MAY NOT BE RETURN A VALUE.

WHY WE NEED STORED PROCEDURE:

WHENEVER WE WANT TO EXECUTE A SQL QUERY FROM AN APPLICATION THE SQL QUERY WILL BE FIRST PARSED (I.E. COMPLIED) FOR EXECUTION WHERE THE PROCESS OF PARSING IS TIME CONSUMING BECAUSE PARSING OCCURS EACH AND EVERY TIME, WE EXECUTE THE QUERY OR STATEMENT.

TO OVERCOME THE ABOVE PROBLEM, WE WRITE SQL STATEMENTS OR QUERY UNDER STORED PROCEDURE AND EXECUTE, BECAUSE A STORED PROCEDURE IS A PRE-COMPLIED BLOCK OF CODE WITHOUT PARSING THE STATEMENTS GETS EXECUTED WHENEVER THE PROCEDURES ARE CALLED WHICH CAN INCREASE THE PERFORMANCE OF AN APPLICATION.

ADVANTAGES OF STORED PROCEDURE:

- **AS THERE IS NO UNNECESSARY COMPILATION OF QUERIES, THIS WILL REDUCE BURDEN ON DATABASE.**
- **APPLICATION PERFORMANCE WILL BE IMPROVED**
- **USER WILL GET QUICK RESPONSE**
- **CODE REUSABILITY & SECURITY.**

PROCEDURE SYNTAX:

```
      CREATE OR REPLACE PROCEDURE <PROCEDURE_NAME>
                 [ PARAMETER NAME [MODE TYPE] DATATYPE,….]
                         IS
<VARIABLE DECLARATION>;
      BEGIN
<EXEC STATEMENTS>;

      [ EXCEPTION BLOCK
<EXEC-STATEMENTS>;]
      END;
```

TO EXECUTE THE PROCEDURE:

SYNTAX1:

```
EXECUTE / EXEC <PROCEDURE_NAME>;
```

SYNTAX2:(ANONYMOUS BLOCK)

```
   BEGIN
<PROCEDURE_NAME>;
   END;
```

EXAMPLES ON PROCEDURE WITHOUT PARAMATERS:

EX1:

```
CREATE OR REPLACE PROCEDURE MY_PROC
    IS
```

```
 BEGIN
  DBMS_OUTPUT.PUT_LINE ('WELCOME TO PROCEDURES....');
 END MY_PROC;
```

TO EXECUTE THE PROCEDURE:

SYNTAX1:

```
EX: EXEC MY_PROC;
```

SYNTAX2:

```
EX: BEGIN
     MY_PROC;
     END;
```

EX2: WRITE A PROCEDURE TO DISPLAY SUM OF TWO NUMBERS.

```
CREATE OR REPLACE PROCEDURE ADD_PROC
                IS
A NUMBER: =10;
B NUMBER: =20;
BEGIN
 DBMS_OUTPUT.PUT_LINE ('SUM OF TWO NUMBERS = '||(A+B));
END ADD_PROC;
```

EXAMPLES ON PROCEDURES WITH PARAMETERS:

EX3:

```
  CREATE OR REPLACE PROCEDURE ADD_PROC (A NUMBER, B NUMBER)
                         IS
  BEGIN
   DBMS_OUTPUT.PUT_LINE ('SUM OF TWO NUMBERS = '||(A+B));
  END ADD_PROC;
```

<u>TO EXECUTE ABOVE PROCEDURE:</u>

```
EXEC ADD_PROC (10,60);

EXEC ADD_PROC (&A, &B);
```

EX4: WRITE A PROCEDURE TO ACCEPT EMPLOYEE NUMBER AND DISPLAY CORRESPONDING EMPLOYEE   NET SALARY.

```
CREATE OR REPLACE PROCEDURE EMP_PROC (TEMPNO EMP.EMPNO%TYPE)

                    IS

TSAL EMP.SAL%TYPE;

TCOMM EMP.COMM%TYPE;

NETSAL NUMBER;

COMM_NULL EXCEPTION;

BEGIN

 SELECT SAL, COMM INTO TSAL, TCOMM FROM EMP WHERE EMPNO=TEMPNO;

 IF TCOMM IS NULL THEN

    RAISE COMM_NULL;

 END IF;

 NETSAL: =TSAL+TCOMM;

 DBMS_OUTPUT.PUT_LINE ('GIVEN EMPLOYEE NET SALARY = '||NETSAL);

EXCEPTION

 WHEN COMM_NULL THEN

   RAISE_APPLICATION_ERROR (-20001,'GIVEN EMPLOYEE IS NOT GETTING
COMMISSION.');

 WHEN NO_DATA_FOUND THEN

   RAISE_APPLICATION_ERROR (-20002, 'SUCH EMPLOYEE NUMBER IS NOT
EXIST.');

END EMP_PROC;
```

PROCEDURES RETURN VALUES THROUGH PARAMETER MODES:

- THERE ARE THREE TYPES OF PARAMETERS MODES.

IN -> IT ACCEPTS INPUT INTO STORED PROCEDURE(DEFAULT)

OUT -> IT RETURNS OUTPUT THROUGH STORED PROCEDURE

IN OUT -> BOTH ACCEPTING AND ALSO RETURN.

EX. ON "IN" PARAMETERS:

EX5:

```
  CREATE OR REPLACE PROCEDURE ADD_PROC (A IN NUMBER, B IN NUMBER)
                          IS
  BEGIN
   DBMS_OUTPUT.PUT_LINE ('SUM OF TWO NUMBERS = '||(A+B));
  END ADD_PROC;
EXEC ADD_PROC (90,30);
```

EX6: CREATE A SP TO INPUT EMPNO AND DISPLAY THAT EMPLOYEE NAME, SAL FROM EMP TABLE?

```
SQL> CREATE OR REPLACE PROCEDURE SP1(P_EMPNO IN NUMBER)
  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> EXECUTE SP1(7788);
SCOTT,3000
```

<u>EX ON "OUT" PARAMETERS:</u>

<u>EX7:</u>

```
SQL> CREATE OR REPLACE PROCEDURE SP2(X IN NUMBER, Y OUT NUMBER)
 IS
 BEGIN
 Y: =X*X*X;
 END;
  /
PROCEDURE CREATED.

SQL> EXECUTE SP2(5);
ERROR AT LINE 1:
ORA-06550: LINE 1, COLUMN 7:
PLS-00306: WRONG NUMBER OR TYPES OF ARGUMENTS IN CALL TO 'SP2'
```

<u>NOTE: TO OVERCOME THE ABOVE PROBLEM THEN WE FOLLOW THE FOLLOWING 3 STEPS,</u>

<u>STEP1: DECLARE REFERENCED /BIND VARIABLE FOR "OUT" PARAMETERS IN SP:</u>

<u>SYNTAX:</u>

```
        VAR[IABLE] <REF.VARIABLE NAME><DT>[SIZE];
```

<u>STEP2: TO ADD A REFERENCED /BIND VARIABLE TO A SP:</u>

<u>SYNTAX:</u>

```
EXECUTE <PNAME> (VALUE1, VALUE2, ……:<REF.VARIABLE NAME>….);
```

<u>STEP3: PRINT REFERENCED VARIABLES:</u>

<u>SYNTAX:</u>

```
        PRINT <REF.VARIABLE NAME>;
```

EXECUTION PLAN OF "OUT" PARAMETERS IN SP:

```
SQL> VAR RY NUMBER;

SQL> EXECUTE SP2(5,:RY);

PL/SQL PROCEDURE SUCCESSFULLY COMPLETED.

SQL> PRINT RY;


       RY
----------
      125
```

EX8:

CREATE A SP TO INPUT EMPNO AS A "IN" PARAMETER AND RETURNS THAT EMPLOYEE PROVIDENT FUND, PROFESSIONAL TAX AT 10%,20% ON BASIC SALARY BY USING "OUT" PARAMETERS?

```
SQL> CREATE OR REPLACE PROCEDURE SP3(P_EMPNO IN NUMBER, PF OUT NUMBER, PT OUT NUMBER)

 IS

 V_SAL NUMBER (10);

 BEGIN

 SELECT SAL INTO V_SAL FROM EMP WHERE EMPNO=P_EMPNO;

 PF: = V_SAL*0.1;

 PT: = V_SAL*0.2;

 END;

 /

PROCEDURE CREATED.

SQL> VAR RPF NUMBER;

SQL> VAR RPT NUMBER;

SQL> EXECUTE SP3(7788,:RPF,:RPT);

SQL> PRINT RPF RPT;
```

EX9:

```
CREATE OR REPLACE PROCEDURE ADD_PROC (A IN NUMBER, B IN NUMBER, C
OUT NUMBER)
 IS
 BEGIN
  C: =A+B;
 END ADD_PROC;
```

OUTPUT:

```
VAR R NUMBER;
EXECUTE ADD_PROC (10,20,:R);
PRINT R;
```

<u>EX. ON “IN OUT” PARAMETERS:</u>

EX10:

```
SQL> CREATE OR REPLACE PROCEDURE SP4(X IN OUT NUMBER)
  AS
  BEGIN
  X: = X*X;
  END;
  /
PROCEDURE CREATED.
SQL> EXECUTE SP4(5);
ERROR AT LINE 1:
ORA-06550: LINE 1, COLUMN 11:
PLS-00363: EXPRESSION '5' CANNOT BE USED AS AN ASSIGNMENT TARGET
```

<u>NOTE: TO OVERCOME THE ABOVE PROBLEM THEN WE FOLLOW THE FOLLOWING 4 STEPS,</u>

<u>STEP1: DECLARE REFERENCED VARIABLE FOR "OUT" PARAMETERS IN SP:</u>

SYNTAX:

```
VAR[IABLE] <REF.VARIABLE NAME><DT>[SIZE];
```

<u>STEP2: ASSIGN A VALUE TO REFERENCED VARIABLE:</u>

SYNTAX:

```
EXECUTE <REF.VARIABLE NAME> := <VALUE>;
```

<u>STEP3: TO ADD A REFERENCED VARIABLE TO A SP:</u>

SYNTAX:

```
EXECUTE <PNAME> (:<REF.VARIABLE NAME>……);
```

<u>STEP4: PRINT REFERENCED VARIABLES:</u>

SYNTAX:

```
PRINT <REF.VARIABLE NAME>;
```

OUTPUT:

```
SQL> VAR RX NUMBER;

SQL> EXECUTE :RX := 10;

SQL> EXECUTE SP4(:RX);

SQL> PRINT RX;
```

NOTE: ALL PROCEDURES NAMES ARE STORED IN USER_OBJECTS.   SELECT OBJECT_NAME FROM USER_OBJECTS;

EX:

SELECT OBJECT_NAME FROM USER_OBJECTS WHERE OBJECT_TYPE='PROCEDURE';

NOTE: PROCEDURE BODIES ARE STORED IN USER_SOURCE.

EX:

```
SELECT TEXT FROM USER_SOURCE WHERE NAME='EMP_PROC';
```

<u>DROPPING PROCEDURES:</u>

SYNTAX:

```
SQL> DROP PROCEDURE <PROCEDURE_NAME>;
```

EX: DROP PROCEDURE MY_PROC;

STORED FUNCTIONS:

A FUNCTION IS BLOCK OF CODE TO PERFORM SOME TASK AND MUST RETURN A VALUE.THESE FUNCTIONS ARE CREATED BY USER EXPLICITELY.SO THAT WE CAN ALSO CALLED AS "USER DEFINED FUNCTION"

SYNTAX:

```
CREATE OR REPLACE FUNCTION <FUNCTION_NAME> [(ARUGMENT DATATYPE,

                                    ARGUMENT DATATYPE,)]

                    RETURN <DATATYPE>

                        IS

 BEGIN

<EXEC-STATEMENTS>;

  RETURN (VALUE);

 END <FUNCTION_NAME>;

 /
```

<u>HOW TO CALL A STORED FUNCTION:</u>

```
SELECT <FNAME>(VALUES) FROM DUAL;
```

EX: CREATE A SF TO ACCEPT EMPLOYEE NUMBER AND RETURN THAT EMPLOYEE NAME FROM EMP TABLE?

```
CREATE OR REPLACE FUNCTION SF1(P_EMPNO NUMBER)

RETURN VARCHAR2

AS

V_ENAME VARCHAR2(10);

BEGIN

SELECT ENAME INTO V_ENAME FROM EMP WHERE EMPNO=P_EMPNO;

RETURN V_ENAME;

END;

/

FUNCTION CREATED.

SQL> SELECT SF1(7566) FROM DUAL;
```

EX: CREATE A SF TO INPUT DEPARTMENT NAME AND RETURN SUM OF SALARY OF DEPARTMENT?

```
FUNCTION SF1(P_DNAME VARCHAR2)

RETURN NUMBER

AS

V_TOTSAL NUMBER (10);

BEGIN

SELECT SUM(SAL) INTO V_TOTSAL FROM EMP E,DEPT D

WHERE E. DEPTNO=D.DEPTNO AND DNAME=P_DNAME;

RETURN V_TOTSAL;

END;

/

SAL> SELECT SF1('SALES') FROM DUAL;
```

EX: CREATE A SF TO RETURN NO. OF EMPLOYEE IN BETWEEN GIVEN DATES?

```
FUNCTION SF2(SD DATE, ED DATE)

RETURN NUMBER

AS

V_COUNT NUMBER (10);

BEGIN

SELECT COUNT (*) INTO V_COUNT FROM EMP

WHERE HIREDATE BETWEEN SD AND ED;

RETURN V_COUNT;

END;

/

SQL> SELECT SF2('01-JAN-81','31-DEC-81') FROM DUAL;
```

EX: CREATE A SF TO INPUT EMPLOYEE NUMBER AND RETURN THAT EMPLOYEE GROSS SALARY AS PER GIVEN CONDITIONS ARE

I) HRA -------- 10%

II) DA -------- 20%

III) PF --------10%.

```
FUNCTION SF3(P_EMPNO NUMBER)

RETURN NUMBER

AS

V_BSAL NUMBER (10);

V_HRA NUMBER (10);

V_DA NUMBER (10);

V_PF NUMBER (10);

V_GROSS NUMBER (10);

BEGIN

SELECT SAL INTO V_BSAL FROM EMP WHERE EMPNO=P_EMPNO;
```

```
V_HRA: =V_BSAL*0.1;
V_DA: =V_BSAL*0.2;
V_PF: =V_BSAL*0.1;
V_GROSS: =V_BSAL+V_HRA+V_DA+V_PF;
RETURN V_GROSS;
END;
/
SQL> SELECT SF3(7788) FROM DUAL;
```

EX: WRITE A FUNCTION TO FIND SIMPLE INTEREST.

```
CREATE OR REPLACE FUNCTION SI (P NUMBER, T NUMBER, R NUMBER)
                  RETURN NUMBER
                      IS
SIMPLE_INT NUMBER;
BEGIN
 SIMPLE_INT: =(P*T*R)/100;
 RETURN (SIMPLE_INT);
END SI;
/
```

>GENERALLY, FUNCTIONS ARE EXECUTED BY USING 'SELECT' STATEMENT.

```
SQL>SELECT SI (1000,2,10) FROM DUAL;
```

EX: CREATE A SF TO FIND EXPERIENCE OF GIVEN EMPLOYEE?

```
 CREATE OR REPLACE FUNCTION EMP_EXP (TEMPNO EMP.EMPNO%TYPE)
                  RETURN VARCHAR2
```

```
                    IS
 TDATE EMP.HIREDATE%TYPE;
 TEXP NUMBER;
 BEGIN
  SELECT HIREDATE INTO TDATE FROM EMP
                      WHERE EMPNO=TEMPNO;
TEXP: =ROUND((SYSDATE-TDATE)/365);
RETURN (TEMPNO||' EMPLOYEE EXPERIENCE IS '||TEXP||' YEARS.');
 EXCEPTION
  WHEN NO_DATA_FOUND THEN
RETURN ('GIVEN EMPLOYEE RECORD NOT FOUND.');
 END EMP_EXP;


SQL> SELECT EMP_EXP (7788) FROM DUAL;
SQL> SELECT EMP_EXP(EMPNO) FROM EMP;
```

<u>FUNCTION FOR TO CALCULATE EMPLOYEE EXPERIENCE:</u>

```
CREATE OR REPLACE FUNCTION EMP_EXPE (TEMPNO EMP.EMPNO%TYPE)
                  RETURN NUMBER
                      IS
 TEXP NUMBER;
 BEGIN
  SELECT ROUND((SYSDATE-HIREDATE)/365) INTO TEXP FROM EMP
                                       WHERE EMPNO=TEMPNO;
  RETURN(TEXP);
 END EMP_EXPE;
```

NOTE:

ALL FUNCTIONS ARE STORED IN USER_OBJECTS.

ALL FUNCTIONS BODIES ARE STORED IN 'USER_SOURCE' SYSTEM TABLE.

> TO SEE THE FUNCTION BODY.

EX:

```
SQL> SELECT TEXT FROM USER_SOURCE WHERE NAME='EMP_EXPE';
```

<u>DROPPING FUNCTIONS:</u>

SYNTAX:

```
SQL> DROP FUNCTION <FUNCTION_NAME>;
```

EX:

```
SQL> DROP FUNCTION EMP_EXPE;
```

<u>TRIGGERS:</u>

A SET OF PL/SQL STATEMENTS STORED PERMANENTLY IN DATABASE AND "AUTOMATICALLY" ACTIVATED WHEN EVER AN EVENT RAISING STATEMENT (DML / DDL) IS PERFORMED.

THEY ARE USED TO IMPOSE USER DEFINED RESTRICTIONS(OR)BUSINESS RULES ON TABLE / SCHEMA.THEY ARE ALSO ACTIVATED WHEN TABLES ARE MANIPULATED BY OTHER USERS OR BY OTHER APPLICATION S/W TOOLS.THEY PROVIDE HIGH SECURITY ON TABLES.THEY ARE STORED IN "USER_TRIGGERS" SYSTEM TABLE.

1. **DML TRIGGERS**
2. **DDL TRIGGER / DB TRIGGERS**

PURPOSE OF TRIGGERS:

1. TO INVOKING USER DEFINED MESSAGE (OR) ALERTS AT EVENT RAISE.

2. TO CONTROL DML / DDL OPERATIONS.

3. TO IMPLEMENTING BUSINESS LOGICAL CONDITIONS.

4. TO VALIDATING DATA.

5. FOR AUDITING.

1. DML TRIGGERS:

- THESE TRIGGERS ARE EXECUTED BY SYSTEM AUTOMATICALLY WHEN USER PERFORMDML (INSERT / UPDATE / DELETE) OPERATIONS ON A SPECIFIC TABLE.

SYNTAX:

```
    CREATE OR REPLACE TRIGGER <TRIGGER_NAME>

    BEFORE/AFTER INSERT OR UPDATE OR DELETE

    [ OF <COLUMNS>] ON <TABLE NAME>

    [ FOR EACH ROW]

      WHEN <CONDITION> (TRUE -> EXECUTES THE TRIGGER,

                        FALSE - NOT EXECUTE)

      DECLARE

<VARIABLE DECLARATION>;]

    BEGIN

<EXEC STATEMENTS>;

      [ EXCEPTION

<EXEC STATEMENTS>;]

    END;
```

<u>TRIGGER EVENT:</u>

- INDICATES WHEN TO ACTIVATE THE TRIGGER

<u>BEFORE TRIGGER:</u>

1. FIRST TRIGGER BODY IS EXECUTED

2. LATER DML COMMAND EXECUTED

<u>AFTER TRIGGER:</u>

1. FIRST DML COMMAND EXECUTED

2. LATER TRIGGER BODY EXECUTED

<u>TRIGGER LEVELS:</u>

- TRIGGER CAN CREATE AT TWO LEVELS.

<u>A. ROW LEVEL TRIGGER:</u>

- IN THIS LEVEL, TRIGGER BODY(LOGIC) IS EXECUTING FOR EACH ROW WISE FOR A DML OPERATION.

EX:

```
CREATE OR REPLACE TRIGGER TR1

AFTER UPDATE ON TEST

FOR EACH ROW

BEGIN

DBMS_OUTPUT.PUT_LINE('HELLO');

END;

/

TRIGGER CREATED.
```

TESTING:

```
SQL> UPDATE TEST SET SAL=10000 WHERE SAL=15000;
HELLO
HELLO
2 ROWS UPDATED.
```

<u>B. STATEMENT TRIGGER:</u>

- IN THIS LEVEL, TRIGGER BODY IS EXECUTING ONLY ONE TIME FOR A DML OPERATION.

EX:

```
CREATE OR REPLACE TRIGGER TR1
AFTER UPDATE ON TEST
BEGIN
DBMS_OUTPUT.PUT_LINE('HELLO');
END;
/
TRIGGER CREATED.
```

TESTING:

```
SQL> UPDATE TEST SET SAL=12000 WHERE SAL=10000;
HELLO

2 ROWS UPDATED.
```

<u>BIND VARIABLES:</u>

- BIND VARIABLES ARE JUST LIKE VARIABLES WHICH ARE USED TO STORE VALUES WHILE INSERTING, UPDATING, DELETING DATA FROM A TABLE.THESE ARE TWO TYPES,

<u>1.: NEW:</u>

- THIS BIND VARIABLE WILL STORE NEW VALUES WHEN WE INSERT.

<u>SYNTAX:</u>

```
: NEW. <COLUMN NAME>= <VALUE>;
```

EX:

```
: NEW.SAL = 15000;
```

<u>2.: OLD:</u>

- THIS BIND VARIABLE WILL STORE OLD VALUES WHEN WE DELETE.

<u>SYNTAX:</u>

```
:OLD. <COLUMN NAME>= <VALUE>;
```

EX:

```
: OLD.SAL = 12000;
```

NOTE: THESE BIND VARIABLES ARE USED IN "ROW LEVEL TRIGGERS ONLY".

<u>1. TO INVOKING USER DEFINED MESSAGE / ALERT AT EVENT FIRE.</u>

EX:

```
CREATE OR REPLACE TRIGGER TRINSERT

AFTER INSERT ON TEST

BEGIN
```

```
DBMS_OUTPUT.PUT_LINE ('SOME ONE INSERTED DATA INTO YOUR TABLE');
END;
/
TRIGGER CREATED.
```

TESTING:

```
SQL> INSERT INTO TEST VALUES (105,'SCOTT',36000);
SOME ONE INSERTED DATA INTO YOUR TABLE
1 ROW CREATED.
```

EX:

```
CREATE OR REPLACE TRIGGER TRUPDATE
AFTER UPDATE ON TEST
BEGIN
DBMS_OUTPUT.PUT_LINE ('SOME ONE UPDATED DATA INTO YOUR TABLE');
END;
/
TRIGGER CREATED.
```

TESTING:

```
UPDATE TEST SET SAL=22000 WHERE EID=1021;
SOME ONE UPDATING DATA IN YOUR TABLE
1 ROW UPDATED.
```

EX:

```
CREATE OR REPLACE TRIGGER TRDELETE
AFTER DELETE ON TEST
BEGIN
```

DBMS_OUTPUT.PUT_LINE ('SOME ONE DELETED DATA FROM YOUR TABLE');

END;

/

TRIGGER CREATED.

TESTING:

DELETE FROM TEST WHERE EDI=1022;

SOME ONE DELETING DATA FROM YOUR TABLE.

1 ROW DELETED.

EX:

CREATE OR REPLACE TRIGGER TRDML

AFTER INSERT OR UPDATE OR DELETE ON TEST

BEGIN

DBMS_OUTPUT.PUT_LINE ('SOME ONE PERFORMING DML OPERATIONS ON YOUR TABLE');

END;

/

TRIGGER CREATED.

2. TO CONTROL / RESTRICTED DML OPERATIONS ON A TABLE:

EX:

CREATE OR REPLACE TRIGGER TRIN

AFTER INSERT ON TEST

BEGIN

RAISE_APPLICATION_ERROR (-20487,'SOME ONE INSERTING DATA INTO YOUR TABLE');

END;

/

TESING:

```
INSERT INTO TEST VALUES (106,'MILLER',52000)
ERROR AT LINE 1:
ORA-20487: SOME ONE INSERTED DATA INTO UR TABLE
```

EX:

```
CREATE OR REPLACE TRIGGER TRUP
AFTER UPDATE ON TEST
BEGIN
RAISE_APPLICATION_ERROR (-20481,'SOME ONE UPDATING DATA IN YOUR
TABLE');
END;
/
```

TESTING:

```
UPDATE TEST SET SAL=22000 WHERE EID=1021;
ERROR AT LINE 1:
ORA-20487: SOME ONE UPDATING DATA IN YOUR TABLE
```

EX:

```
CREATE OR REPLACE TRIGGER TRDEL
AFTER DELETE ON TEST
BEGIN
RAISE_APPLICATION_ERROR (-20481,'SOME ONE DELETING DATA FROM
YOUR TABLE');
END;
/
```

TESTING:

```
DELETE FROM TEST WHERE EDI=1022;
```

ERROR AT LINE 1:

ORA-20487: SOME ONE DELETING DATA FROM YOUR TABLE.

EX:

```
CREATE OR REPLACE TRIGGER TRDEL
AFTER INSERT OR UPDATE OR DELETE ON TEST
BEGIN
RAISE_APPLICATION_ERROR (-20481,'SOME ONE PERFORMING DML
OPERATIONS ON YOUR TABLE');
END;
/
```

TESTING:

```
INSERT INTO TEST VALUES (106,'MILLER',52000);
UPDATE TEST SET SAL=22000 WHERE EID=1021;
DELETE FROM TEST WHERE EDI=1022;
ERROR AT LINE 1:
ORA-20781: SOME ONE PERFORMING DML OPERATIONS ON YOUR TABLE.
```

<u>3. TO IMPLEMENTING BUSINESS LOGICAL CONDITIONS:</u>

EX:

CREATE A TRIGGER TO RESTRICTED DML OPERATIONS ON EVERY SATURDAY?

```
CREATE OR REPLACE TRIGGER TRDAY
AFTER INSERT OR UPDATE OR DELETE ON TEST
BEGIN
IF TO_CHAR(SYSDATE,'DY') = 'SAT' THEN
```

RAISE_APPLICATION_ERROR (-20456,'YOU CANNNOT PERFORM DML OPERATIONS ON EVERY SATURDAY');

END IF;

END;

/

EX:

CREATE A TRIGGER TO RESTRICTED DML OPERATIONS ON TEST TABLE IN BETWEEN 9AM TO 5PM?

CREATE OR REPLACE TRIGGER TRTIME

AFTER INSERT OR UPDATE OR DELETE ON TEST

BEGIN

IF TO_CHAR(SYSDATE,'HH24') BETWEEN 9 AND 16 THEN

RAISE_APPLICATION_ERROR (-20456,'YOU CANNNOT PERFORM DML OPERATIONS BETWEEN 9AM TO 5PM');

END IF;

END;

/

EX:

TRIGGER NAME: HOLI_TRIG

TABLE NAME: EMP

TRIGGER EVENT: BEFORE INSERT OR UPDATE OR DELETE

SOL:

CREATE OR REPLACE TRIGGER HOLI_TRIG

BEFORE INSERT OR UPDATE OR DELETE

ON EMP

DECLARE

CNT NUMBER;

BEGIN

```
 IF TO_CHAR(SYSDATE,'HH24') NOT BETWEEN 10 AND 16 THEN
    RAISE_APPLICATION_ERROR (-20001,'OFFTIMINGS, TRANS. ARE NOT
ALLOWED.');
 END IF;


 IF TO_CHAR(SYSDATE,'DY') IN ('SAT','SUN') THEN
    RAISE_APPLICATION_ERROR (-20002,'WEEKENDS, TRANS. ARE NOT
ALLOWED.');
 END IF;


 SELECT COUNT(HDATE) INTO CNT FROM HOLIDAY
         WHERE TO_CHAR(SYSDATE,'DD/MM/YY’)
=TO_CHAR(HDATE,'DD/MM/YY');
 IF CNT>0 THEN
    RAISE_APPLICATION_ERROR (-20003,'TODAY PUBLIC HOLIDAY, TRANS.
ARE NOT ALLOWED.');
 END IF;
END;
```

<u>4. TO VALIDATING DATA:</u>

EX:

CREATE A TRIGGER TO VALIDATE INSERT OPERATION IF NEW SALARY IS LESS THAN 5000?

```
SQL> CREATE OR REPLACE TRIGGER TRSAL1
  BEFORE INSERT ON TEST
  FOR EACH ROW
  BEGIN
IF: NEW.SAL< 5000 THEN
  RAISE_APPLICATION_ERROR (-20348,'NEW SAL SHOULD NOT BE LESS THAN
TO 5000');
  END IF;
  END;

  /
```

TESTING:

```
INSERT INTO TEST VALUES (1021,'SMITH',4500); -----NOT ALLOWED

INSERT INTO TEST VALUES (1021,'SMITH',5500); -----ALLOWED
```

EX:

```
CREATE OR REPLACE TRIGGER DEPT_TRIG

BEFORE INSERT ON DEPT

FOR EACH ROW

BEGIN

: NEW.DNAME: =UPPER (: NEW.DNAME);

: NEW.LOC: =UPPER (: NEW.LOC);

END;
```

TESTING:

```
SQL> INSERT INTO DEPT VALUES (50,'ECONOMICS','HYD');
```

EX:

CREATE A TRIGGER TO VALIDATE UPDATE OPERATION ON TEST TABLE IF NEW SALARY IS LESS THAN TO OLD SALARY?

```
SQL> CREATE OR REPLACE TRIGGER TRSAL2

  BEFORE UPDATE ON TEST

  FOR EACH ROW

  BEGIN

IF: NEW.SAL<: OLD.SAL THEN
```

```
  RAISE_APPLICATION_ERROR (-20748,'NEW SAL SHOULD NOT BE LESS THAN
TO OLD SALARY');

  END IF;

  END;

  /
```

TESTING:

UPDATE TEST SET SAL=6000 WHERE SAL=8000; -----NOT ALLOWED

UPDATE TEST SET SAL=9000 WHERE SAL=8000; ------ ALLOWED

EX:

CREATE A TRIGGER TO VALIDATE DELETE OPERATION ON TEST TABLE IF WE TRY TO DELETE THE EMPLOYEE"SMITH" DETAILS?

```
SQL> CREATE OR REPLACE TRIGGER TRDATA

  BEFORE DELETE ON TEST

  FOR EACH ROW

  BEGIN

IF: OLD.ENAME = 'SMITH' THEN

  RAISE_APPLICATION_ERROR (-20648,'WE CANNOT DELETE SMITH
EMPLOYEE DETAILS');

  END IF;

  END;

  /
```

TESTING:

DELETE FROM TEST WHERE ENAME='SMITH'; -----NOT ALLOWED

DELETE FROM TEST WHERE ENAME='ALLEN'; ----ALLOWED

<u>5. FOR AUDITING</u>:

- WHEN WE MANIPULATE DATA IN A TABLE THOSE TRANSACTIONAL VALUES ARE STORED IN ANOTHER TABLE IS CALLED AS AUDITING TABLE.

EX:

SQL> CREATE TABLE EMP1(EID INT, ENAME VARCHAR2(10), SALNUMBER (10));

TABLE CREATED.

SQL> CREATE TABLE AUDITEMP1(EID INT, AUDIT_INFOR VARCHAR2(100));

TABLE CREATED.

EX:

CREATE OR REPLACE TRIGGER TRAUDIT_INSERT

AFTER INSERT ON EMP1

FOR EACH ROW

BEGIN

INSERT INTO AUDITEMP1 VALUES (: NEW.EID,'SOME ONE INSERTED A NEW ROW INTO EMP1 TABLE ON'||' '||

TO_CHAR (SYSDATE,'DD-MON-YYYY HH:MI: SS AM'));

END;

/

TESTING:

INSERT INTO EMP1 VALUES (1021,'SMITH',4500);

EX:

CREATE OR REPLACE TRIGGER TRAUDIT_UPDATE

AFTER UPDATE ON EMP1

FOR EACH ROW

BEGIN

INSERT INTO AUDITEMP1 VALUES (: OLD.EID,'SOME ONE UPDATED DATA IN EMP1 TABLE ON'||' '||

TO_CHAR (SYSDATE,'DD-MON-YYYY HH:MI: SS AM'));

END;

/

TESTING:

UPDATE EMP1 SET SAL=6000 WHERE EID=1021;

EX:

CREATE OR REPLACE TRIGGER TRAUDIT_DELETE

AFTER DELETE ON EMP1

FOR EACH ROW

BEGIN

INSERT INTO AUDITEMP1 VALUES (: OLD.EID,'SOME ONE DELETED DATA FROM EMP1 TABLE ON'||' '||

TO_CHAR (SYSDATE,'DD-MON-YYYY HH:MI: SS AM'));

END;

/

TESTING:

DELETE FROM EMP1 WHERE ENAME='SMITH';

EX:

CREATE OR REPLACE TRIGGER TRAUDIT_DML

AFTER INSERT OR UPDATE OR DELETE ON EMP1

FOR EACH ROW

BEGIN

INSERT INTO AUDITEMP1 VALUES (: OLD.EID,'SOME ONE PERFORMING DML OPERATIONS ON EMP1 TABLE ON'||' '||

TO_CHAR (SYSDATE,'DD-MON-YYYY HH:MI: SS AM'));

END;

/

TESTING:

```
INSERT INTO EMP1 VALUES (1021,'SMITH',4500);

UPDATE EMP1 SET SAL=6000 WHERE EID=1021;

DELETE FROM EMP1 WHERE ENAME='SMITH';
```

DDL TRIGGERS / DB TRIGGERS:

- THESE TRIGGERS ARE EXECUTED BY SYSTEM AUTOMATICALLY WHEN USER PERFORMDDL (CREATE / ALTER / DROP / RENAME) OPERATIONS ON A SPECIFIC SCHEMA / DATABASE.

- THESE TRIGGER ARE HANDLING BY DBA ONLY.

SYNTAX:

```
    CREATE OR REPLACE TRIGGER <TRIGGER_NAME>

    BEFORE/AFTER CREATE OR ALTER OR DROP OR RENAME

    ON USERNAME.SCHEMA

    [ FOR EACH ROW]

    [ DECLARE

<VARIABLE DECLARATION>;]

    BEGIN

<EXEC STATEMENTS>;

    END;
```

EX:

```
SQL> CREATE OR REPLACE TRIGGER TRDDL

  AFTER CREATE ON MYDB9AM.SCHEMA

  BEGIN

  RAISE_APPLICATION_ERROR (-20456,'WE CANNOT CREATE A NEW TABLE IN
MYDB9AM SCHEMA');

  END;
```

```
 /

TRIGGER CREATED.

SQL> CREATE TABLE T1(SNO INT);

ERROR AT LINE 1:

ORA-20456: WE CANNOT CREATE A NEW TABLE IN MYDB9AM SCHEMA
```

EX:

```
SQL> CREATE OR REPLACE TRIGGER TRDDL

  AFTER ALTER ON MYDB9AM.SCHEMA

  BEGIN

  RAISE_APPLICATION_ERROR (-20456,'WE CANNOT ALTER A TABLE IN
MYDB9AM SCHEMA');

END;

  /

SQL> ALTER TABLE EMP1 ADD EADD VARCHAR2(10);

ORA-20456: WE CANNOT ALTER TABLE IN MYDB9AM SCHEMA

SQL> CREATE OR REPLACE TRIGGER TRDDL

  AFTER DROP ON MYDB9AM.SCHEMA

  BEGIN

  RAISE_APPLICATION_ERROR (-20456,'WE CANNOT DROP A TABLE FROM
MYDB9AM SCHEMA');

  END;

  /

TRIGGER CREATED.


SQL> DROP TABLE EMP1 PURGE;

ORA-20456: WE CANNOT DROP A TABLE FROM MYDB9AM SCHEMA
```

```
SQL> CREATE OR REPLACE TRIGGER TRDDL
  AFTER RENAME ON MYDB9AM.SCHEMA
  BEGIN
  RAISE_APPLICATION_ERROR (-20456,'WE CANNOT RENAME A TABLE IN
MYDB9AM SCHEMA');
  END;
  /
TRIGGER CREATED.
SQL> RENAME EMP1 TO EMPDETAILS;
ORA-20456: WE CANNOT RENAME A TABLE IN MYDB9AM SCHEMA
ORA-06512: AT LINE 2
```

EX:

```
SQL> CREATE OR REPLACE TRIGGER TRDDL
  AFTER CREATE OR ALTER OR RENAME OR DROP ON MYDB9AM.SCHEMA
  BEGIN
  RAISE_APPLICATION_ERROR(-20456,'WE CANNOT PERFORM DDL
OPERATIONS ON MYDB9AM SCHEMA');
  END;
  /
TRIGGER CREATED.
```

EX:

CREATE A TRIGGER TO RESTRICTED DDL OPERATIONS ON MYDB9AM SCHEMA IN BETWEEN 9AM TO 5PM?

```
CREATE OR REPLACE TRIGGER TRDDLTIME
AFTER CREATE OR ALTER OR RENAME OR DROP ON MYDB9AM.SCHEMA
BEGIN
IF TO_CHAR(SYSDATE,'HH24') BETWEEN 9 AND 16 THEN
```

RAISE_APPLICATION_ERROR (-20456,'YOU CANNNOT PERFORM DDL OPERATIONS ON MYDB9AM BETWEEN 9AM TO 5PM');

END IF;

END;

/

DROPPING TRIGGERS:

SQL> DROP TRIGGER <TRIGGERNAME>;

EX:

DROP TRIGGER TRDDLTIME;

EXAMPLES QUERIES:

SQL> CREATE TABLE STU_DENT(ROLLNO NUMBER(10),SNAME VARCHAR2(20),COURSE VARCHAR2(20),

FEE NUMBER(5));

SQL> INSERT INTO STU_DENT VALUES(1,'NARESH','ORACLE',2000);

SQL> SELECT *FROM STU_DENT;

SQL>INSERT INTO STU_DENT(ROLLNO,COURSE)VALUES(2,'JAVA');

SQL>INSERT INTO STU_DENT(ROLLNO,COURSE)VALUES(2,'JAVA');

SQL>SELECT *FROM STU_DENT WHERE SNAME='NARESH';

SQL>SELECT *FROM STU_DENT WHERE COURSE='JAVA';

SQL>CREATE TABLE SAMP_TAB(EID NUMBER(10),ENAME VARCHAR2(20),JOB VARCHAR2(20),SAL NUMBER(7,2));

DISPLAY SALARY OF EMPLOYEES WITH 2000 INCREMENT IN THEIR SALARY.

SQL> SELECT ENAME,SAL,SAL + 2000 "INCREMENTED SALARY" FROM EMP

DISPLAY THE DETAILS OF EMPLOYEES DECREASING THEIR SALARY BY 200.

```
SQL> SELECT ENAME,SAL,SAL-200 FROM EMP;
```

EXPLINATION:-IN EMP TABLE EVERY EMPLOYEE SALARY SUBTRACTED WITH 200.

ARITHMETIC OPERATOR MULTIPLICATION(*) :-USED TO PERFORM MULTIPLICATION.

EXAMPLE:-

DISPLAY THE DETAILS OF THE EMPLOYEES INCREMENTING THEIR SALARY TWO TIMES.

```
 SQL> SELECT SAL * 2 FROM EMP;
```

EXPLINATION:-EVERY EMP TABLE SALARY IS MULTIPLIED BY 2.

DISPLAY HALF OF THE SALARY OF EMPLOYEES.

```
SQL> SELECT SAL, SAL/2 FROM EMP;

SQL> SELECT EMPNO,ENAME,SAL,12*SAL+100 FROM EMP;

 SQL> SELECT EMPNO,ENAME,SAL,(12*SAL)+100 FROM EMP;

 SQL> SELECT EMPNO,ENAME,SAL,12*(SAL+100) FROM EMP;
```

DISPLAY THE DETAILS OF EMPLOYEES WHOSE SALARY IS EQUAL TO 2000.

```
SQL> SELECT *FROM EMP WHERE SAL=950;
```

EXAMPLE: DISPLAY THE DETAILS OF THE EMPLOYEES WHOSE SALARY IS LESS THAN 3000.

```
  SQL> SELECT * FROM EMP WHERE SAL < 3000;
```

DISPLAY THE DETAILS OF EMPLOYEES WHOSE SALARY IS LESS THAN OR EQUAL TO 3000.

```
SQL> SELECT * FROM EMP WHERE SAL <= 3000;
```

DISPLAY THE DETAILS OF EMPLOYEES WHOSE SALARY IS NOT EQUALS TO 2000.

```
SQL> SELECT * FROM EMP WHERE SAL != 3000;


SQL> SELECT * FROM EMP WHERE SAL ^= 2000;

SQL> SELECT * FROM EMP WHERE SAL <> 2000;
```

DISPLAY THE DETAILS OF EMPLOYEES WHOSE SALARY IS GREATER THAN 1000 AND ALSO WHOSE SALARY IS LESS THAN 2000.

```
   SQL> SELECT *FROM EMP WHERE SAL > 1000 AND SAL <2000;

SQL> SELECT ENAME,SAL,JOB FROM EMP

       WHERE (SAL>=1500 AND SAL<=5000) AND

       JOB='MANAGER';
```

DISPLAY THE DETAILS OF EMPLOYEES WHOSE SALARY IS GREATER THAN 1000 OR ALSO WHOSE SALARY IS LESS THAN 2000.

```
SQL> SELECT *FROM EMP WHERE SAL> 1000 OR SAL < 2000;
```

EXPLINATION:-WHOSE SALARIES MORE THAN 1000 OR LESS THAN 2000 THAT ALL EMP TABLE DISPLAY.

```
SQL> SELECT EMPNO,ENAME,JOB,HIREDATE  FROM EMP

         WHERE JOB='MANAGER' OR DEPTNO=20;

SQL> SELECT EMPNO,ENAME,JOB,HIREDATE  FROM EMP

          WHERE (JOB='MANAGER' OR DEPTNO=10);

SQL> SELECT EMPNO,ENAME,JOB,HIREDATE  FROM EMP

                WHERE (JOB='CLERK' OR JOB='SALESMAN' OR JOB='ANALYST');

SQL> SELECT EMPNO,ENAME,JOB,HIREDATE  FROM EMP

               WHERE (SAL<=2500 OR SAL>=5000) OR JOB='MANAGER';

SQL> SELECT ENAME,JOB ,SAL FROM EMP

               WHERE JOB='CLERK' OR JOB='MANAGER' AND SAL>1500;
```

DISPLAY THE DETAILS OF EMPLOYEES WHOSE SALARY IS GREATER THAN OR EQUALS TO 3000.

```
SQL> SELECT * FROM EMP WHERE SAL < 3000;
```

EXPLINATION:-WHOSE SALARY LESS THAN 3000 THAT SALARIES ALL ARE COMMING.

```
SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP

              WHERE NOT ENAME='SMITH';

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP

         WHERE NOT SAL>=5000;

SQL> SELECT EMPNO,ENAME,JOB,SAL,DEPTNO FROM EMP

          WHERE NOT JOB='CLERK' AND DEPTNO=20;

         SQL> SELECT *FROM EMP WHERE EMPNO IN (7125, 7369, 7782);

SQL> UPDATE EMP SET SAL=SAL+200 WHERE ENAME IN ('SMITH','ALLEN','WARD');

SQL> DELETE FROM EMP WHERE HIREDATE IN ('22-DEC-82','17-NOV-81');

SQL> UPDATE EMP SET SAL=SAL+200 WHERE ENAME NOT IN

                  ('SMITH','ALLEN','WARD');

         SQL> DELETE FROM EMP WHERE HIREDATE NOT IN ('22-DEC-82',' 17-NOV-81');

SQL> SELECT ENAME,SAL,JOB FROM EMP

        WHERE JOB BETWEEN 'MANAGER' AND 'SALESMAN';

SQL> SELECT ENAME,SAL,JOB,HIREDATE FROM EMP

        WHERE HIREDATE    BETWEEN '17-DEC-81' AND '20-JUN-83';

SQL> SELECT ENAME,SAL,JOB FROM EMP
```

```
   WHERE JOB  NOT  BETWEEN 'MANAGER' AND 'SALESMAN';

 SQL> SELECT ENAME,SAL,JOB,HIREDATE FROM EMP

      WHERE HIREDATE NOT   BETWEEN '17-DEC-81' AND '20-JUN-83';
```

EXAMPLE:- DISPLAY THE EMPLOYEES WHOSE NAME IS STARTING WITH ‘S’ IN EMP TABLE.

```
SQL> SELECT * FROM EMP WHERE ENAME LIKE   'S%'

EMPNO ENAME    JOB          MGR HIREDATE    SAL  COMM DEPTNO

-----------------------------------------------------------------------------------------------

 7369 SMITH             CLERK        7902 17-DEC-80   800            20

 7788 SCOTT             ANALYST      7566 19-APR-87  3000           20
```

DISPLAY THE EMPLOYEES WHOSE NAME ENDS WITH ‘S’ IN EMP TABLE

```
SQL> SELECT * FROM EMP WHERE ENAME LIKE '%S'

EMPNO ENAME    JOB          MGR HIREDATE           SAL  COMM DEPTNO

----------------------------------------------------------------------------------------------------

 7566 JONES              MANAGER           7839 02-APR-81    2975  20

 7876 ADAMS             CLERK        7788 23-MAY-87           1100    20

 7900 JAMES              CLERK       7698 03-DEC-81           950     30
```

DISPLAY THE EMPLOYEES WHOSE NAMES ARE HAVING SECOND LETTER AS ‘L’ IN EMP TABLE

```
SQL> SELECT * FROM EMP WHERE ENAME LIKE '_L%'


EMPNO ENAME                  JOB       MGR HIREDATE    SAL  COMM DEPTNO

-----------------------------------------------------------------------------------------------
```

```
 7499 ALLEN                  SALESMAN   7698 20-FEB-81  1600   300            30

 7698 BLAKE                  MANAGER    7839 01-MAY-81  2850           30

 7782 CLARK                  MANAGER    7839 09-JUN-81  2450           10
```

```
SQL> SELECT ENAME ,HIREDATE FROM  EMP WHERE HIREDATE  LIKE '%JAN%';

SQL> SELECT EMPNO,ENAME,JOB FROM EMP WHERE JOB LIKE '_____';


SQL> SELECT EMPNO,ENAME,JOB ,HIREDATE FROM EMP WHERE HIREDATE LIKE '%-FEB-81';

  SQL> SELECT *FROM DEPT WHERE DNAME LIKE '__/_%' ESCAPE '/';

(UPDATE DEPT SET DNAME='SO_FT_WARE'  WHERE DEPTNO=50;)
```

DISPLAY THE EMPLOYEES WHOSE NAMES ARE NOT HAVING SECOND LETTER AS 'L' IN EMP TABLE?

```
SQL> SELECT *FROM EMP WHERE ENAME NOT LIKE '_L%';
```

DISPLAY THE EMPLOYEES WHOSE NAMES ARE NOT START WITH 'S' IN EMP TABLE.?

```
SQL> SELECT *FROM EMP WHERE ENAME NOT LIKE 'S%';

SQL> SELECT ENAME ,HIREDATE FROM  EMP WHERE HIREDATE NOT LIKE '%JAN%';

SQL> SELECT EMPNO,ENAME,JOB FROM EMP WHERE ENAME NOT LIKE '_O%';
```

DISPLAY THE EMPLOYEES WHOSE NAMES ARE SECOND LETTER START WITH 'R' FROM ENDING.?

```
SQL> SELECT *FROM EMP WHERE ENAME LIKE '%R_';

EMPNO ENAME    JOB           MGR HIREDATE    SAL  COMM DEPTNO

-----------------------------------------------------------------------------------------------

 7521 WARD               SALESMAN    7698 22-FEB-81    1250   500     30
 7782 CLARK              MANAGER    7839 09-JUN-81     2450                  10

 7902 FORD               ANALYST    7566 03-DEC-81     3000            20
```

DISPLAY THE NAMES IN EMP TABLE WHOSE NAMES HAVING ‘LL’.?

```
SQL> SELECT *FROM  EMP WHERE ENAME LIKE ‘%LL%’;

EMPNO ENAME         JOB                   MGR HIREDATE    SAL  COMM DEPTNO

------------------------------------------------------------------------------------------------

 7499 ALLEN            SALESMAN    7698 20-FEB-81  1600   300     30

 7934 MILLER           CLERK        7782 23-JAN-82  1300              10
```

```
SQL> SELECT *FROM EMP WHERE COMM IS NULL;

SQL> SELECT EMPNO,ENAME,JOB,SAL ,DEPTNO FROM EMP WHERE MGR IS NULL;

SQL> SELECT *FROM EMP WHERE COMM IS NOT NULL;

        SQL> SELECT EMPNO EMPNUMBER, ENAME EMPNAME,SAL "EMPSALARY",

                  JOB DESIGNATION FROM EMP;

        SQL>SELECT GRADE AS "SALGRADE", HISAL "HIGH SALARY RANGE",LOSAL "LOW SALARY RANGE"

                        FROM SALGRADE;

          SQL> SELECT EMPNO "EMPNUBER",SAL "BASIC",SAL*0.25 HRA,SAL*0.20 DA ,SAL*0.15 "PF",

                         SAL+SAL*0.25+SAL*0.20+SAL*0.15 "GROSS" FROM EMP;
```

```
SQL>SELECT *FROM EMP WHERE JOB<>'CLERK';

SQL>SELECT *FROM EMP WHERE SAL>2500;

SQL> SELECT *FROM EMP WHERE JOB='SALESMAN' OR SAL>3000;

SQL>SELECT *FROM EMP WHERE DEPTNO=10 OR DEPTNO=20 OR DEPTNO=50
OR DEPTNO=60;

SQL> SELECT *FROM EMP WHERE ENAME IN('FORD','MILLER');

SQL>SELECT *FROM EMP WHERE SAL BETWEEN 3000 AND 5000;

SQL> SELECT *FROM EMP WHERE COMM IS NOT NULL;


SQL>SELECT *FROM EMP WHERE ENAME LIKE '_L%';

SQL> SELECT ENAME||'   '||JOB||'   '||SAL FROM EMP;

SQL>SELECT ENAME,SAL,COMM,SAL+NVL(COMM,0) FROM EMP;

SQL> SELECT ENAME,SAL,COMM,(SAL*12)+NVL(COMM,0)  FROM EMP;

SQL>SELECT ENAME,SAL,COMM,(SAL+500)+NVL(COMM,0) FROM EMP;

SQL>SELECT  COMM,NVL2(COMM,0,1000) FROM EMP;

       SQL>SELECT SAL,COMM,SAL+NVL2(COMM,100,1000)  FROM EMP;

       SQL>SELECT    ENAME,SAL,COMM,NVL2(COMM,SAL+COMM,SAL)    INCOME
FROM EMP

                    WHERE DEPTNO IN(10,30);

SQL> SELECT DEPTNO,COUNT(*) FROM EMP

 GROUP BY DEPTNO

 HAVING COUNT(DEPTNO)>3;

SQL> SELECT DEPTNO,AVG(SAL) FROM EMP

 GROUP BY DEPTNO

 HAVING AVG(SAL)>2500;
```

```
SQL> SELECT DEPTNO,MIN(SAL),MAX(SAL), SUM(SAL) FROM EMP

 WHERE JOB='MANAGER'

 GROUP BY DEPTNO

 HAVING AVG(SAL)<1000;
```

```
SQL> SELECT MAX(AVG(SAL)) FROM EMP GROUP BY DEPTNO;
```

```
SQL> SELECT MAX(SUM(SAL)),MIN(SUM(SAL)) FROM EMP

            GROUP BY DEPTNO;
```

```
SQL> SELECT MAX(AVG(SAL)),MIN(AVG(SAL)) FROM EMP

GROUP BY JOB;
```

EXAMPLE: DISPLAY EMPLOYEE RECORDS ORDER BY ENAME.

```
SQL>SELECT *  FROM EMP  ORDER BY ENAME;
```

HERE WE CAN SEE THAT 'ENAME' VALUES RESULTED IN 'ASCENDING ORDER'.

EXAMPLE: DISPLAY EMPLOYEE RECORDS ORDER BY 'ENAME' IN DESCENDING ORDER. HERE WE NEED TO EXPLICITLY SPECIFY THE KEYWORD 'DESC'.

```
SQL>SELECT * FROM EMP ORDER BY ENAME DESC;
```

ORDERING THE RESULT BASED ON A NUMERIC COLUMN:-

```
  SQL>SELECT * FROM EMP  ORDER BY SAL;
```

WE CAN AS WELL ORDER THE RESULT USING A COLUMN SEQUENCE:-

```
  SQL>SELECT EMPNO,ENAME,JOB,SAL  FROM EMP

             ORDER BY 2;
```

HERE 2 REPRESENTS SECOND COLUMN. SO, DATA WILL BE DISPLAYED WITH ENAME IN ASCENDING ORDER.

```
SQL> SELECT ENAME,JOB,SAL,DEPTNO FROM EMP WHERE SAL>=2000

                ORDER BY DEPTNO,ENAME DESC;
```

```
SQL> SELECT  ENAME,JOB,SAL,SAL*12 ANNUALSAL FROM EMP

                ORDER BY ANNUALSAL;
```

```
SQL> SELECT  ENAME,JOB,SAL,DEPTNO FROM EMP

          ORDER BY DEPTNO,JOB,SAL;

SQL> SELECT *FROM EMP ORDER BY 5 DESC;


      SQL> SELECT *FROM EMP WHERE JOB=(SELECT JOB FROM EMP WHERE
      ENAME='SMITH')

                AND ENAME<>'SMITH'

                AND ENAME LIKE '%A%';



      SQL> SELECT ENAME,SAL,DEPTNO FROM EMP

             WHERE SAL>2000 AND DEPTNO=(SELECT DEPTNO FROM DEPT

              WHERE LOC='DALLAS');

      SQL> SELECT ENAME,JOB,DEPTNO,SAL FROM EMP

              WHERE JOB='CLERK' AND DEPTNO=(SELECT DEPTNO FROM DEPT

               WHERE DNAME='SALES');

      SQL> SELECT ENAME,JOB ,DEPTNO FROM EMP

       WHERE DEPTNO IN(SELECT DEPTNO FROM EMP

       WHERE ENAME='FORD')

       AND

       JOB IN(SELECT JOB FROM EMP

       WHERE DEPTNO=(SELECT DEPTNO FROM DEPT

       WHERE DNAME='SALES'));

      SQL>SELECT EMPNO,ENAME,JOB,SAL,DEPTNO FROM EMP

            WHERE SAL<ANY(SELECT SAL FROM EMP
            WHERE DEPTNO=30);

      SQL> SELECT *FROM EMP

      WHERE SAL>ALL(SELECT AVG(SAL) FROM EMP

      GROUP BY DEPTNO);

      SQL>SELECT ENAME,JOB,SAL,DEPTNO FROM EMP

              WHERE SAL<ALL(SELECT MAX(SAL) FROM EMP  GROUP BY JOB);

      SQL> SELECT E.EMPCOUNT,D.DEPTCOUNT FROM

             (SELECT COUNT(*) EMPCOUNT FROM EMP)E,

             (SELECT COUNT(*) DEPTCOUNT FROM DEPT)D;
```

```
SQL> SELECT  E.EMPCOUNT,D.DEPTCOUNT,S.GRADECOUNT,

     E.EMPCOUNT+D.DEPTCOUNT+S.GRADECOUNT TOTALRECCOUNT FROM

     (SELECT COUNT(*) EMPCOUNT FROM EMP )E,

     (SELECT COUNT(*) DEPTCOUNT FROM DEPT)D,

     (SELECT COUNT(*) GRADECOUNT FROM SALGRADE)S

SQL> SELECT A.DEPTNO "DEPTMENT
NUMBER",(A.NUMEMP/B.TOTALCOUNT)*100 "%EMPLOYEES",

     (A.SALSUM/B.TOTALSAL)*100 "%SALARY" FROM(SELECT
DEPTNO,COUNT(*) NUMEMP,

     SUM(SAL) SALSUM FROM EMP GROUP BY DEPTNO)A,

     (SELECT COUNT(*) TOTALCOUNT ,SUM(SAL) TOTALSAL FROM EMP)B

SQL> SELECT *FROM DEPT D WHERE EXISTS(SELECT *FROM EMP E

         WHERE DEPTNO=D.DEPTNO

          AND DEPTNO=10);
```

```
SQL>SELECT P.ENAME FROM EMP P
            WHERE EXISTS (SELECT *FROM EMP C
            WHERE C.EMPNO=P.MGR);
 SQL> SELECT *FROM DEPT D WHERE EXISTS(SELECT *FROM EMP E
        WHERE DEPTNO=D.DEPTNO
         AND DEPTNO=10);
SQL>SELECT P.ENAME FROM EMP P
            WHERE EXISTS (SELECT *FROM EMP C
            WHERE C.EMPNO=P.MGR);
SQL>SELECT EMPNO,ENAME,JOB,SAL,NVL(COMM,0),SAL+NVL(COMM,0)
            FROM EMP
            WHERE DEPTNO=(SELECT DEPTNO FROM EMP WHERE
EMPNO=7654);


SQL>SELECT *FROM EMP WHERE DEPTNO=(SELECT DEPTNO FROM EMP
           WHERE ENAME='FORD')AND JOB IN(SELECT JOB FROM EMP
            WHERE DEPTNO=(SELECT DEPTNO FROM DEPT
             WHERE DNAME='SALES'));
SQL> SQL> SELECT EMPNO,ENAME,JOB,SAL,DEPTNO FROM EMP
             WHERE SAL>(SELECT MAX(SAL) FROM EMP
             WHERE JOB='SALESMAN');
```

```
1. SQL> SELECT EMPNO, ENAME FROM EMP WHERE DEPTNO=(SELECT DEPTNO
FROM DEPT WHERE DNAME='RESEARCH');
```

2. SQL> SELECT EMPNO, ENAME FROM EMP WHERE DEPTNO IN (SELECT DEPTNO FROM DEPT WHERE LOC IN ('NEW YORK','CHICAGO'));

3. SQL> SELECT DNAME FROM DEPT WHERE DEPTNO IN (SELECT DEPTNO FROM EMP WHERE JOB ='ANALYST');

4. SQL> SELECT EMPNO, ENAME, MGR FROM EMP WHERE MGR = (SELECT EMPNO FROM EMP WHERE ENAME='JONES');

5. SQL> SELECT EMPNO, ENAME, MGR FROM EMP WHERE MGR = (SELECT MGR FROM EMP WHERE ENAME='JONES')

6. SQL> SELECT EMPNO, ENAME, JOB FROM EMP WHERE DEPTNO IN (SELECT DEPTNO FROM DEPT WHERE DNAME IN ('SALES','ACCOUNTING'))

7. SQL> SELECT EMPNO, ENAME, JOB FROM EMP WHERE DEPTNO IN (SELECT DEPTNO FROM DEPT WHERE DNAME IN ('SALES','RESEARCH')) AND EMPNO IN (SELECT MGR FROM EMP)

8. SQL> SELECT EMPNO, ENAME FROM EMP WHERE EMPNO NOT IN (SELECT MGR FROM EMP WHERE MGR IS NOT NULL)

9. SELECT EMPNO, ENAME FROM EMP WHERE EMPNO IN (SELECT MGR FROM EMP GROUP BY MGR

HAVING COUNT(*) >= 2)

10. SQL> SELECT DNAME FROM DEPT WHERE DEPTNO IN (SELECT DEPTNO FROM EMP GROUP BY DEPTNO HAVING COUNT(*) >=5)

11. SQL> SELECT DEPTNO, JOB, COUNT(*) FROM EMP WHERE JOB = 'SALESMAN' GROUP BY DEPTNO, JOB HAVING COUNT(*) >=3

12. SQL> SELECT EMPNO, ENAME, DEPTNO FROM EMP WHERE EMPNO IN (SELECT MGR FROM EMP GROUP BY MGR

HAVING COUNT(*) >= 2) AND DEPTNO IN (SELECT DEPTNO FROM DEPT WHERE DNAME='RESEARCH' OR DNAME='ACCOUNTING')

13. SQL>SELECT MAX(SAL) FROM EMP WHERE SAL < (SELECT MAX(SAL) FROM EMP);

14. SQL> SELECT MAX(SAL) FROM EMP WHERE SAL < (SELECT MAX(SAL) FROM EMP WHERE SAL < (SELECT MAX(SAL) FROM EMP WHERE SAL < (SELECT MAX(SAL) FROM EMP)))

15.SQL> SELECT * FROM EMP R1

WHERE &N=(SELECT COUNT(DISTINCT(SAL))

FROM EMP WHERE SAL>=R1.SAL);

1. SQL> SELECT * FROM EMP WHERE ENAME LIKE 'S%';

2. SQL> SELECT * FROM EMP WHERE ENAME LIKE '_L%';

3. SQL> SELECT * FROM EMP WHERE ENAME LIKE '%E_';

```
4. SQL> SELECT * FROM EMP WHERE ENAME LIKE '____';

5. SQL> SELECT * FROM EMP WHERE ENAME LIKE '%L%';

6. SQL> SELECT * FROM EMP WHERE ENAME LIKE '_____%';

7. SQL> SELECT * FROM EMP WHERE SAL BETWEEN 2000 AND 3000;

9. SQL> SELECT * FROM EMP WHERE MGR IS NULL OR COMM IS NULL;

10. SQL> SELECT * FROM EMP WHERE MGR IS NULL AND COMM IS NULL;

11. SQL> SELECT * FROM EMP WHERE JOB = 'MANAGER';

12. SQL> SELECT * FROM EMP WHERE JOB = 'MANAGER' AND DEPTNO IN
(10,20);

13. SQL> SELECT * FROM EMP WHERE JOB IN ('CLERK','ANALYST') AND SAL >=
1000 AND DEPTNO IN

    (20,30);

14. SQL> SELECT * FROM EMP WHERE DEPTNO IN (20,30) AND COMM IS NULL;

15. SQL> SELECT * FROM EMP WHERE ENAME LIKE ('A%') OR ENAME LIKE
('S%');
```

16. SQL>SELECT * FROM EMP WHERE ENAME NOT LIKE ('%S') AND DEPTNO IN (20,30);

17. SQL> SELECT * FROM EMP WHERE COMM IS NOT NULL AND SAL > 1500 AND DEPTNO = 30;

18. SQL> SELECT * FROM EMP WHERE COMM IS NOT NULL AND SAL > 1500 AND JOB = 'MANAGER'

19. SQL> SELECT * FROM EMP WHERE JOB = 'MANAGER' OR JOB = 'CLERK' AND SAL >=2000 AND DEPTNO NOT IN (10,20);

20. SQL> SELECT * FROM EMP WHERE COMM IS NOT NULL;

21. SQL> SELECT * FROM EMP WHERE SAL NOT BETWEEN 2000 AND 3000 AND JOB LIKE ('%MAN%');

1).DISPLAY THE EMPLOYEE WHO GOT THE MAXIMUM SALARY

```
   SQL> SELECT *FROM EMP WHERE SAL=(SELECT MAX(SAL) FROM EMP);
```

2) DISPLAY THE EMPLOYEE WHO ARE WORKING IN SALES DEPARTMENT

```
SQL> SELECT *FROM EMP WHERE DEPTNO IN(SELECT DEPTNO FROM DEPT

           WHERE DNAME='SALES');
```

3) DISPLAY THE EMPLOYEE WHO ARE WORKING ARE WORKING AS “CLERK”

```
SQL> SELECT *FROM EMP WHERE JOB IN(SELECT JOB FROM EMP

         WHERE JOB='CLERK');
```

4) DISPLAY THE EMPLOYEE WHO ARE WORKING ARE WORKING IN “NEW WORK”

```
SQL> SELECT *FROM EMP WHERE DEPTNO=(SELECT DEPTNO FROM DEPT
          WHERE LOC=(SELECT LOC FROM DEPT
          WHERE LOC='NEW YORK'));
```

5)FIND OUT NUMBER OF EMPLOYEES WORKING IN SALES DEPARTMENT

```
SQL> SELECT *FROM EMP WHERE DEPTNO=(SELECT DEPTNO FROM DEPT
        WHERE DNAME='SALES'
        GROUP BY DEPTNO);
```

6) DELETING THE EMPLOYEES WHO ARE WORKING ARE WORKING IN ACCOUNTING

DEPARTMENT

```
SQL> DELETE FROM DEPT WHERE DEPTNO=(SELECT DEPTNO FROM DEPT
        WHERE DNAME='ACCOUNTING');
```

SUB QUERY OPERATORS :(ALL,ANY,EXISTS,SOME,NOT EXISTS)

1)LIST OUT THE EMPLOYEES WHO EARN MORE THAN EVERY EMPLOYEE IN DEPARTMENT 30

```
SQL> SELECT *FROM EMP WHERE SAL>ALL(SELECT SAL FROM EMP
       WHERE  DEPTNO=30);
```

2) LIST OUT THE EMPLOYEES WHO EARN MORE THAN THE LOWEST SALARY IN DEPTNO 30

```
SQL> SELECT *FROM EMP WHERE SAL>ANY(SELECT SAL FROM EMP

          WHERE DEPTNO=30);
```

3)FIND OUT WHICH DEPARTMENT DOES NOT HAVE ANY EMPLOYEES

```
SQL> SELECT DEPTNO FROM DEPT D

        WHERE NOT EXISTS (SELECT DEPTNO FROM EMP E

        WHERE E.DEPTNO=D.DEPTNO);
```

4)FIND OUT EMPLOYEES WHO EARN GREATER THAN THE AVERAGE SALARY FOR THEIR DEPARTMENT

```
SQL> SELECT EMPNO,ENAME,JOB,SAL,DEPTNO FROM EMP E

  WHERE SAL>(SELECT AVG(SAL) FROM EMP  WHERE DEPTNO=E.DEPTNO);
```

5)LIST THE EMPNO, ENAME, SAL, DNAME OF ALL THE 'MGRS' AND

'ANALYST'WORKING IN NEWYORK, DALLAS WITH AN EXP MORE THAN 7 YEARS WITHOUT

RECEIVING THE COMMA ASC ORDER OF LOC

```
SQL> SELECT EMPNO, ENAME,SAL, DNAME FROM EMP, DEPT

            WHERE LOC IN ('NEW YORK','DALLAS') AND

             JOB IN('MANAGER','ANALYST')AND

               MONTHS_BETWEEN(SYSDATE,HIREDATE)/12 > 7




   AND COMM IS NULL

                AND EMP.DEPTNO = DEPT.DEPTNO;

SQL> SQL> SELECT ENAME,JOB,SAL,D.DEPTNO,D.DNAME FROM EMP E,DEPT D
```

```
              WHERE E.DEPTNO=D.DEPTNO AND E.ENAME IN('SMITH','FORD');

SQL>SELECT EMPNO,JOB,SAL,E.DEPTNO ,D.DNAME,D.LOC FROM EMP E,DEPT D

               WHERE E.DEPTNO=D.DEPTNO AND D.DEPTNO<>20;

SQL> SELECT E.EMPNO,E.ENAME,E.SAL ,S.GRADE FROM EMP E,SALGRADE S

              WHERE (E.SAL>=S.LOSAL AND E.SAL<=S.HISAL) AND S.GRADE=1;

 SQL>SELECT E.ENAME,E.JOB,M.ENAME FROM EMP E,EMP M

              WHERE E.MGR=M.EMPNO

              AND E.JOB  NOT IN('MANAGER','CLERK','ANALYST');

SQL>SELECT ENAME,JOB,DNAME,LOC FROM EMP ,DEPT

                  WHERE ENAME ='MARTIN';

SQL> SELECT E.ENAME,D.DEPTNO,D.DNAME FROM EMP E,DEPT D

        WHERE E.DEPTNO=D.DEPTNO(+)

        AND E.DEPTNO(+)=10

        ORDER BY E.DEPTNO;

SQL> SELECT EMP.ENAME FROM EMP LEFT OUTER JOIN DEPT ON
(EMP.DEPTNO =

                  DEPT.DEPTNO) AND EMP.ENAME LIKE 'A%';

                        SQL>SELECT DNAME, ENAME FROM DEPT NATURAL JOIN EMP;


SQL> SELECT ENAME,LOC FROM EMP E JOIN DEPT D

             ON E.DEPTNO=D.DEPTNO

             WHERE LOC='CHICAGO';


SQL> SELECT E.ENAME,E.SAL,D.DEPTNO,D.DNAME,S.GRADE FROM EMP E JOIN

               DEPT D ON E.DEPTNO=D.DEPTNO JOIN SALGRADE S
```

```
            ON E.SAL BETWEEN S.LOSAL AND S.HISAL;
```

```
SQL>ALTER TABLE DEPT DROP PRIMARY KEY CASCADE;
```

*** TO DROP A NOT NULL CONSTRAINT TO EXISTING TABLE**

SYNTAX:-ALTER TABLE TABLENAME CONSTRAINT CONSTRAINTNAME;

```
SQL>ALTER TABLE EMP DROP CONSTRAINT EMP_MGR-FK;

SQL>ALTER TABLE DEPT DROP UNIQUE(DNAME);
```

*** IF WE WANT TO SEE THE INDEXNAME THAT TIME WE WRITY THE BELOW EXAMPLE**

EXAMPLE:-

```
SQL> SELECT INDEX_NAME FROM USER_CONSTRAINTS

       WHERE TABLE_NAME='EMP';
```

ENABLING CONSTRAINTS:-

- **THE CONSTRAINT CAN BE ENABLED WITHOUT DROPPING IT OR RECREATING IT.**
- **THE ALTER TABLE STATEMENT WITH THE ENABLE CLAUSE IS USED FOR THE PURPOSE.**

SYNTAX:-

```
    ALTER TABLE <TABLENAME> ENABLE CONSTRAINT

         <CONSTRAINTNAME>;
```

```
SQL> CREATE TABLE   CUSTOMER

             (STATUS CHAR(3) NOT NULL,

              SAL    NUMBER  NOT NULL);

SQL> CREATE TABLE STU_CLASS(

        STU_ID NUMBER(2) NOT NULL,

        STU_NAME VARCHAR2(15) NOT NULL,
          STU_CLASS  VARCHAR2(10) NOT NULL);

SQL>CREATE TABLE SUPPLIER

( SUPPLIER_ID NUMERIC(10) NOT NULL,

  SUPPLIER_NAME VARCHAR2(50) NOT NULL,

  CONTACT_NAME VARCHAR2(50),

  CONSTRAINT SUPPLIER_UNIQUE UNIQUE (SUPPLIER_ID)

);


SQL> CREATE TABLE SUPPLIER

( SUPPLIER_ID NUMERIC(10) NOT NULL,

  SUPPLIER_NAME VARCHAR2(50) NOT NULL,

  CONTACT_NAME VARCHAR2(50),

  CONSTRAINT SUPPLIER_UNIQUE UNIQUE (SUPPLIER_ID, SUPPLIER_NAME)

);

SQL>CREATE TABLE T1 (

        C1 NUMBER,

        C2 VARCHAR2(30),

        C3 VARCHAR2(30),

        PRIMARY KEY (C1,C2));

SQL>CREATE TABLE T1 (

        C1 NUMBER,

        C2 VARCHAR2(30),

        C3 VARCHAR2(30),


CONSTRAINT T1_PK PRIMARY KEY (C1,C2));
```

```
SQL> CREATE TABLE B1(NAME VARCHAR2(20)  CONSTRAINT CHE_NAME
CHECK(NAME=UPPER(NAME)));

SQL> CREATE TABLE DEPT(DEPTNO NUMBER(2) CONSTRAINT DNO_PK
PRIMARY KEY

          CONSTRAINT DEPTNO_CHK CHECK(DEPTNO BETWEEN 10 AND 99),

          DNAME VARCHAR2(20) CONSTRAINT DNAME_NN NOT NULL
CONSTRAINT DNAME_CHK

          CHECK(DNAME=UPPER(DNAME)),LOC VARCHAR2(20) DEFAULT
'NEW YORK'

          CONSTRAINT LOC_CHK CHECK(LOC IN('NEW
YORK','DALLAS','BOSTON','CHICAGO')));

SQL> CREATE TABLE EMP(EMPNO NUMBER(4) CONSTRAINT EMPNO_PK
PRIMARY KEY,

     ENAME VARCHAR2(20) CONSTRAINT ENAME_NN NOT NULL

     CHECK(SUBSTR(ENAME,1,1) BETWEEN 'A' AND 'Z' AND

    ENAME=UPPER(ENAME)),JOB VARCHAR2(20) CONSTRAINT JOB_CHK

   CHECK(JOB IN('ANALYST','CLERK','MANAGER','PRESIDENT','SALESMAN')),

    HIREDATE DATE DEFAULT SYSDATE,

    SAL NUMBER(8,2) CONSTRAINT SAL_NN NOT NULL

    CONSTRAINT CHK_SAL CHECK(SAL BETWEEN 1000 AND 10000),

    COMM NUMBER(8,2),

   DEPTNO NUMBER(2), CONSTRAINT TOT_SAL_CHK

   CHECK(SAL+COMM<=100000));
```

```
 SQL>SELECT JOB FROM EMP WHERE DEPTNO=20
•                   UNION
•                   SELECT JOB FROM EMP WHERE DEPTNO=30;
•          SQL> SELECT * FROM
•                   (SELECT ENAME FROM EMP WHERE JOB = 'CLERK'
•                     UNION
•                   SELECT ENAME FROM EMP WHERE JOB = 'ANALYST');
SQL> SELECT * FROM
             (SELECT SAL FROM EMP WHERE JOB = 'CLERK'
                  UNION ALL
            SELECT SAL FROM EMP WHERE JOB = 'ANALYST');
      SQL>  SELECT ENAME FROM
              (SELECT ENAME FROM EMP WHERE JOB = 'CLERK'
                  INTERSECT
              SELECT ENAME FROM EMP WHERE JOB = 'ANALYST');
     SQL> SELECT * FROM (SELECT SAL FROM EMP
                WHERE JOB = 'PRESIDENT'
                    EXCEPT
            SELECT SAL FROM EMP WHERE JOB =
                    'MANAGER');
   SQL>SELECT ROWNUM ,ENAME FROM EMP WHERE ROWNUM<7
                             EXCEPT
           SELECT ROWNUM,ENAME FROM EMP WHERE ROWNUM<6;
          SQL> CREATE OR ALTER VIEW EDEPT30 AS
                   SELECT *FROM EMP WHERE DEPTNO=30
          WITH CHECK OPTION CONSTRAINT EDEPT30CHKVIEW;

     SQL> CREATE OR ALTER VIEW EMANAGE AS


     SELECT *FROM EMP WHERE JOB='MANAGER'

         WITH CHECK OPTION CONSTRAINT EMPMANAGERVIEW;

      SQL> CREATE OR ALTER VIEW EDEPTREAD(EMPID,NAME,DESIGNATION)

             AS

             SELECT EMPNO,ENAME,JOB FROM EMP

             WHERE DEPTNO=20 WITH READ ONLY;

       SQL> CREATE OR ALTER FORCE VIEW V5

              AS

            SELECT *FROM MASTER;

        SQL> CREATE VIEW PAYINFO AS SELECT EMPNO ECODE,SAL
BASIC,SAL*0.25 DA,SAL*0.35 HRA,

            SAL*0.12 PF ,SAL+SAL*0.25+SAL*0.35+SAL*0.12 GROSS FROM
EMP;

        SQL> SQL>CREATE OR ALTER VIEW EMPMANAGERS AS

          SELECT ROWNUM SERIALNO,INITCAP(E.ENAME)||'WORKS
UNDER'||M.ENAME "EMPLOYEE AND

           MANAGERS" FROM EMP E,EMP M WHERE E.MGR=M.EMPNO;

         SQL>CREATE OR ALTER VIEW EMPACCOUNTS AS

          SELECT ENAME,DEPTNO,SAL MONTHLY,SAL*12 ANNUAL FROM EMP

          WHERE DEPTNO=(SELECT DEPTNO FROM DEPT WHERE
DNAME='ACCOUNTING')

          ORDER BY ANNUAL;

      SQL> CREATE OR ALTER VIEW CUMSUM AS  SELECT
```

```
          B.SAL,SUM(A.SAL)CUM_SAL

          FROM EMP A,EMP B WHERE A.ROWID<=B.ROWID

          GROUP BY B.ROWID,B.SAL;


      SQL> CREATE OR ALTER VIEW ORGDESIGNATIONS AS

           SELECT JOB FROM EMP

           WHERE DEPTNO=10 UNION

           SELECT JOB FROM EMP WHERE DEPTNO IN(20,30);

        SQL>        CREATE        OR        ALTER        VIEW
EMPVIEW(EMPNUMBER,EMPNAME,EMSSAL,EMPDEPTNO)

                AS

               SELECT EMPNO,ENAME,SAL,DEPTNO FROM EMP

                 WHERE DEPTNO=20;

      SQL> CREATE MATERIALIZED VIEW EMPLOYEE_VIEW

           REFRESH ON COMMIT

           ENABLE QUERY REWRITE

                AS

           SELECT JOB,SUM(SAL) FROM EMP

              GROUP BY JOB

            /

         SQL> CREATE MATERIALIZED VIEW  EMP_DEPT_SUM

              ENABLE QUERY REWRITE

              AS

              SELECT DNAME,D.DEPTNO,SUM(SAL) FROM EMP E,DEPT D

              WHERE E.DEPTNO=D.DEPTNO
```

```
                GROUP BY DNAME,D.DEPTNO;

        SQL> CREATE INDEX JOBINDEX ON EMP(JOB);

        SQL> CREATE UNIQUE INDEX ENO_UNIQ_INDEX ON EMP(EMPNO);

        SQL> CREATE INDEX DNOINDEX ON DEPT(DEPTNO);


        SQL> CREATE INDEX UPPER_DEPT_DNAME_INDEX ON DEPT(UPPER(LOC));

        SQL> CREATE BITMAP INDEX INDEXEMPBITMAPDEPTNO ON
EMP(DEPTNO);

        SQL> CREATE UNIQUE INDEX ENO_ENAME_INDEX ON
EMP(EMPNO,ENAME);

SQL> SELECT JOB,AVG(SAL) FROM EMP GROUP BY ROLLUP(JOB);

SQL> SELECT  JOB, DEPTNO,AVG(SAL) SALARY FROM EMP

        GROUP BY ROLLUP(JOB,DEPTNO);

SQL> SELECT DEPTNO,JOB,SUM(SAL) SALARY FROM EMP

        GROUP BY CUBE(DEPTNO,JOB);

SQL> SELECT DEPTNO,GROUPING(DEPTNO),JOB,GROUPING(JOB),SUM(SAL)
FROM EMP

        GROUP BY CUBE(DEPTNO,JOB);

   SQL> SELECT DECODE(GROUPING(JOB),1,'ALL
DESIGNATIONS',JOB)DEPARTMENTS,SUM(SAL)

          FROM EMP GROUP BY ROLLUP(JOB);

SQL> SELECT ENAME,MGR ,SUM(SAL) FROM EMP GROUP BY GROUPING
SETS(ENAME,MGR)

SQL> SELECT EMPNO,ENAME,JOB,SAL,CASE JOB WHEN 'MANAGER' THEN 'MAN'

          WHEN 'CLERK' THEN 'CLK'

          WHEN 'SALESMAN' THEN 'SMAN'

          ELSE 'OTHER JOB'
```

```
            END

            FROM EMP;

SQL> SELECT ENAME,SAL,CASE WHEN SAL>=800 AND

            SAL<=2000 THEN 'LOWEST PAY'

                  WHEN SAL>=2001 AND SAL<=4000 THEN 'MODERATE PAY'


 ELSE 'HIGH PAY' END

            FROM EMP;

SQL> SELECT EMPNO,ENAME,JOB,SAL,

         CASE WHEN JOB='MANAGER' THEN  'MAG'

            WHEN SAL=3000  THEN 'MPAY'

            WHEN JOB='SALESMAN' THEN 'SMAN'

         ELSE 'NOT SPECIFIED' END

            CJOB FROM EMP;

SQL>  SELECT MGR,DEPTNO ,SUM(SAL) FROM EMP

                  GROUP BY MGR ,CUBE(MGR,DEPTNO);

SQL>     SQL> SELECT DEPTNO,JOB,GROUP_ID(),SUM(SAL) FROM EMP

            GROUP BY DEPTNO,ROLLUP(DEPTNO,JOB)

            HAVING GROUP_ID()=0;

  SQL>   SELECT   *FROM(SELECT   EMPNO,ENAME,SAL,DEPTNO,RANK()   OVER(
PARTITION BY DEPTNO

            ORDER BY SAL DESC)R FROM EMP)

            WHERE R<=5

            ORDER BY DEPTNO,SAL DESC;

SQL>   SELECT   *FROM(SELECT   EMPNO,ENAME,SAL,DEPTNO,RANK()   OVER(
PARTITION BY DEPTNO
```

```
 ORDER BY SAL DESC)RANK FROM EMP)

            WHERE RANK BETWEEN 3 AND 8

            ORDER BY DEPTNO,SAL DESC

SQL> SQL> SELECT ENAME,SAL,SUM(SAL) OVER(ORDER BY SAL ASC RANGE
5000

            PRECEDING )SALTOL FROM EMP;


SQL> SELECT ENAME,HIREDATE,SAL,LEAD(SAL,1,0) OVER(ORDER BY HIREDATE
)PRESAL FROM EMP;

SQL>CREATE TABLE PET_INFO( PET VARCHAR2(20),CITY
VARCHAR2(20),PCOUNT NUMBER(4));

SQL>INSERT INTO PET_INFO VALUES('DOG','HYD',3500);

SQL>INSERT INTO PET_INFO VALUES('CAT','HYD',300);

SQL> INSERT INTO PET_INFO VALUES('DOG','SECUNDERABAD',250);

SQL>INSERT INTO PET_INFO VALUES('CAT','SECUNDERABAD',200);
```