

**ORACLE (12c)**

An Oracle database is a collection of data treated as a unit. The purpose of a database is to store and retrieve related information. A database server is the key to solving the problems of information management. In general, a server reliably manages a large amount of data in a multiuser environment so that many users can concurrently access the same data. All this is accomplished while delivering high performance. A database server also prevents unauthorized access and provides efficient solutions for failure recovery.

Oracle Database is the first database designed for enterprise grid computing, the most flexible and cost effective way to manage information and applications. Enterprise grid computing creates large pools of industry-standard, modular storage and servers. With this architecture, each new system can be rapidly provisioned from the pool of components. There is no need for peak workloads, because capacity can be easily added or reallocated from the resource pools as needed.

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

**CONTENT**

* Fundamentals of Database
* Introduction to Database
* Database Models
* Introduction to Oracle Database
* Oracle Database History
* Oracle 11g Server and DB Architecture

**SQL**

* Introduction to SQL
* Introduction to SQL \*Plus
* Role of SQL in Oracle 11g
* Classification of SQL Commands
* Data Definition Languages (DDL) commands
* Oracle database 11g Schema Objects
* Oracle Data Dictionary
* Oracle Naming conventions
* Oracle Data types
* Alternation of Table Definition and it’s options
* Pseudo columns Introduction
* Table Truncation and it’s advantages
* Data Manipulation Language (DML) Commands
* Insertion of Data (Value , Address and Select method )
* Insertion of Nulls and Overriding the Nulls with User defined Values
* Insertion of Data in required formats
* Data Loading methods in Oracle 11g
* Data Updation
* Techniques of updation
* Complex Data Updation
* Correlated Query mechanism in Update
* Data Deletion
* Simple Data Deletion
* Critical Data Deletion
* Table Delete Vs Table Truncation
* Transaction Control Language commands
* Data Retrieving Language(DRL) command SELECT
* Conditions
* Expressions
* Restricting ans Sorting data
* SELECT command and it’s clauses
* Operators Types of Operators in Oracle 11g & Filters
* Functions
* Single row functions,
* Multiple row functions
* Null functions
* Analytical functions
* Search functions
* Hierarchical functions
* Error functions
* Regular expressions(10g);
* Types of functions in Oracle 11g
* Pseudo Columns of Oracle 11g
* Displaying data from multiple tables(joins)
* Introduction to Table Join
* Types of Joins
* Inner & Outer Join/partition outer join(10g)
* Equi / Non Equi / Self Join / Cartesian Join/natural joins
* Vertical joins(set operators)
* Sub Queries and Simple sub query
* Complex Sub Query on multiple data source and Co-related Sub Query
* Top-n queries, hierarchical queries, scalar queries, flashback queries,version queries.
* Intigrity Constraints
* Oracle Database Objects
* Index and Introduction to Index
* Clusters and Introduction to Cluster
* Type of Clusters and Their Usage
* Explain plan Command Usage and Oracle Scripts
* Views and Introduction to Views
* Type of Views and usage
* Performance issues with Views
* Background process of Views
* DML restrictions on Views
* Materialized View and Usage
* Synonym and Introduction to Synonyms and Usage
* Sequence  and Introduction to Sequence and Usage
* Pseudo Columns Usage in Sequence
* Data Control Language commands
* What is privilege
* What is role
* Grating Privilege
* Removing Privilege
* Cascading Privilege
* Sql \* Plus Commands
* Environment setting commands
* Alter session language
* Alter system language
* Sql developer

**PL/SQL**

* Introduction to Programming Language
* Procedures Vs Non-Procedures Language
* Limitation of ANSI SQL and Oracle SQL
* Introduction to Oracle PL/SQL
* PL/SQL Usage in Production Database
* Key benefits of PL/SQL over SQL
* Anchor datatypes or Attributes
* Composite datatypes
* Collections
* PL/SQL block structure & Designing
* Scope and Visibility
* Constructs of PL/SQL
* Assignment operations
* Debugging statement
* Flow Control Statement
* IF / NESTED IF / EXIT / GOTO /
* Iterative statements
* Simple Loop / While Loop / For Loop
* Continue statement(11g)
* Embedded SQL
* Introduction to Embedded SQL
* Role of Embedded SQL in PL/SQL
* Constructs of Embedded SQL
* Transaction Mngmt Using Embedded SQL
* Dynamic SQL
* Introduction to Dynamic SQL
* Usage of Dynamic SQL in PL/SQL
* Introduction to Exceptions
* Importance of Exceptions in PL/SQL
* Type of Exceptions
* Exception handling
* Save exceptions
* Introduction and STANDARD Package
* Introduction to User Defined Exceptions
* Non predefined exceptions
* Exception cases
* Usage of PRAGMA EXCEPTION\_INIT()
* Cursor Management in PL/SQL
* Introduction to cursor management
* Pictorial presentation of cursor mechanism
* Introduction and usage of implicit cursor
* Introduction and usage of Explicit cursors
* Cursor attributes
* Cursor using simple loop
* Cursor using while loop
* Cursor using for loop
* Cursor exceptions
* Cursor expression
* Data Locking
* Data Manipulation through Cursor
* REF cursor(strong and weak)
* Using ref cursor variable as parameter
* Bulk Fetch and Bulk Data Retrieval in PL/SQL
* Bulk Collection
* Bulk Binding mechanism of Cursor (for all statement)
* Dynamic behavior of cursor mnt.
* SUB PROGRAMS
* Types of PL/SQL blocks
* Labelled blocks
* Anonymous PL/SQL blocks
* Named PL/SQL blocks
* Forward Declaration of Local Block
* Introduction to storage PL/SQL block
* Stored Procedures
* Stored Functions
* Nocopy(9i)
* Autonomous Transaction Management of PL/SQL
* PACKAGES
* Introduction to Package
* Stand-alone schema Vs Packaged Object
* Encapsulation mechanism of Package
* Data security
* Function overloading mechanism of Package
* Introduction and Usage of Package
* Oracle supplied packages
* Package data
* Restrict Reference and Complex Hints
* Usage of Pragma Serially\_ reusable
* DB TRIGGERS
* Introduction to Database Trigger
* Types of Triggers
* Triggering events
* Usage of Old & New Reference
* Instead of  Trigger
* Enforcing the referential integrity constraint
* Compound Trigger(11g);
* Follows key word(11g);
* Defining a disable trigger(11g)
* Trigger Cascading
* Enabling/Disabling Trigger
* Schema Trigger
* Table Mutation Error
* Transaction Audit Trigger
* Advanced Pl/sql Topics
* User Defined Types (RECORDS)
* Subtypes of Pl/sql
* Automation Transaction
* Advantages of Autonomous Transaction
* Usage of Autonomous Transaction
* Scope of autonomous Transaction
* Usage of Autonomous Transaction in Trigger
* Suing FORALL Statement
* About % BULK\_ROWCOUNT
* FGA and FGAC(VPD)
* Table functions
* Managing database dependencies
* Designing pl/sql code
* Using collections
* Working with lobs
* Using secure file lobs
* C ompiling pl/sql code
* Tuning pl/sql code
* Pragma inline(11g)
* Caching to improve performance
* Analysing pl/sql code
* Profiling pl/sql code
* Tracing pl/sql code
* Safeguarding pl/sql code against sql injection
* Pl/sql Architecture.

**DATATYPES**

**DEFINITION:**

* Specifies what kind of Data We have to store in a Memory
* SQL won’t support Boolean datatypes.
* Boolean Data types (TRUE, FALSE, NULL)

1) NUMBER

2) CHAR

3) VARCHAR OR VARCHAR2

4) DATE

5) TIMESTAMP

6) TIMESTAMP WITH TIMEZONE

7) TIMESTAMP WITH LOCAL TIMEZONE

8) INTERVAL DATATYPES;

I) Interval Year to Month

II) Interval Day to Second.

9) LONG

a) LONG

b) RAW

c) LONG RAW

10) LOB

* CLOB
* BLOB
* BFILE
* NCLOB

11) ROWID

12) UROWID

13) BINARY\_FLOAT

14) BINARY\_DOUBLE

15) SIMPLE\_INTEGER

16) SUBTYPES

17) NCHAR

18) NVARCHAR2

19) INTERNET DATAYPES

20) XML DATATYPES.

21) OTHER DATATYPES

1. **NUMBER:**

**SYN**: X NUMBER (P, (S));

* + To store numeric values
  + Max size is 38 digits.

**2) CHAR:**

**SYN**: X CHAR(S [BYTES|CHAR]);

* + To store alphanumeric values
  + Max size 2000 bytes;
  + Memory allocation is constant (wastage of memory).

**3) VARCHAR (ANSI) OR VARCHAR2 (ORACLE).**

**SYN**: X VARCHAR2(S);

* + TO STORE ALPHANUMBERIC DATA
  + MAXSIZE 4000 BYTES;
  + MEMORY ALLOCATION IS DYNAMIC(NO WASTAGE OF MEMORY)

**4) DATE**

**SYN:** X DATE; (DD-MON-YY HH:MI:SS)

* + To store the date value.
  + Max size 7 bytes.

**Disadvantage:**

* + Not possible to store fraction of seconds.
  + Internally date value get stored in the form of julian day

number

5) **TIMESTAMP (9I)**

**SYN**: X TIMESTAMP [(P)]. (DD-MON-YY HH: MI:SS.FS)

* + It is derivative of data ,along with date it supports upto fraction

of seconds.

* + It stores fraction of seconds upto 9 digits,by default 6 digits.

**Disadvantage:**

Not possible to store timezones (tzh:tzm)

**6) TIMESTAMP WITH TIMEZONE (9I) (DD-MON-YY HH: MI: SS.FS THZ: TZM)**

**SYN**: X TIMESTAMP [(P)] WITH TIMEZONE;

* + In addition with timestamp values it stores time zone values.

7) **TIMESTAMP WITH LOCAL TIMEZONE**

**SYN**: X TIMESTAMP WITH LOCAL TIMEZONE.

* + Normalize the given time into ISO/GMT standard time

8) **INTERVAL DATATYPE**

I) Interval Year to Month

**SYN**: X INTERVAL YEAR TO MONTH.

* + To Store Year to Month Interval Data

II)Interval Day to Second:

**SYN**: X INTERVAL DAY TO SECOND.

* + To Store Day, Hour, Minute, Seconds Data.

9) **LONG:**

**a) LONG:**

**SYN:** X LONG;

* + To store information.
  + Max size 2 gb.

**Disadvantage:**

We have to use long datatype only once in an entire table.

So many disadvantages are there for long this is why notPreferable.

**b) RAW:**

**SYN**: X RAW(S).

* + To store the images.
  + Max size 2000 bytes.

**c) LONG RAW:**

**SYN**: X LONG RAW.

* + To store information+ images
  + Max sized 2gb
  + Not preferable, so many disadvantages.

10) **LOB: (LARGE OBJECTS (OOPS))**

1)CLOB :( CHARACTER LARGE OBJECTS);

**SYN**: X CLOB;

* + To store huge information.
  + Max size depends on sql and pl/sql.

**2) BLOB:**

**SYN**: X BLOB;

* + To store images.

3) **BFILE**: (EXTERNAL DATATYPE)

**SYN**: X BFILE;

* + TO store files.

11) **ROWID**:

**SYN**: X ROWID;

* + It is a datatype which is user to store the physical address of the records ‘rowid’ values.
  + Rowid values are usefull to identify the records in a table.

**12) UROWID (UNIVERAL ROWIDS):**

**SYN**: X UROWID;

* + We use it to store the logical address of index organized table (IOT).
  + UROWID even store RDBMS ROWID values.

13) BINARY\_FLOAT

14) BINARY\_DOUBLE

* + They introduced from 10 g.
  + They are used to store float values so to increase the performance.

15) SIMPLE\_INTEGER

16) SIMPLE\_FLOAT

17) SIMPLE\_DOUBLE

* + They are introduced from 11g.
  + Useful in performance.
  + Unlike other datatypes they won’t allow ‘null’ values.

18) NCHAR

19) NVARCHAR2

20) NCLOB

* + They are used to store the national database character set
  + They support multi languages.

**Null value:**

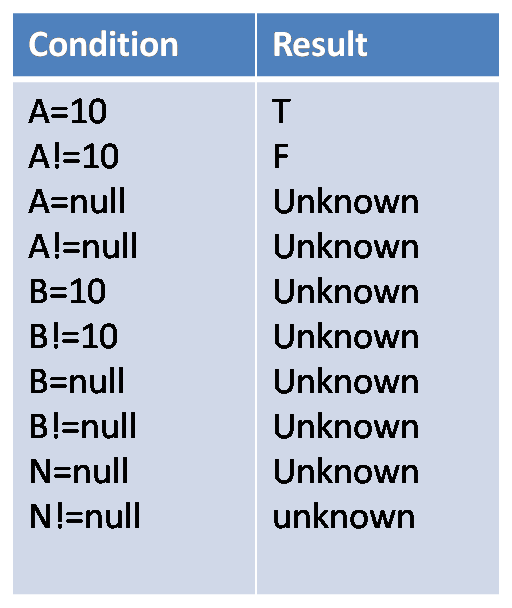
* + It is an unknown, undefined, insufficient, missing and junk values.
  + It is neither empty nor zero
  + Even system doesn’t know what exactly the null value is
  + Null is not applicable in a row concept
  + It will be displayed as a blank space on the screen
  + A special operator ‘IS NULL’ is used to handle the null values
  + A reserved word ‘NULL’ is used to assign null values
  + Every null is unique
  + Null values are taken higher values than number and character in order by clause
  + All the arithmetic operators with null values will result in null values
  + Decode function will treats the null values equal
  + Concatenation operator will ignore the null values
  + Codes without any space will also be treated as null value
  + Oracle database allocates one byte for null value when it fall between column values
  + A concept of null is not applicable to row or record

**Truth Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **AND**  **O R** | **T** | **F** | **NULL** |
| **T** | T  T | F  T | N  T |
| **F** | F  T | F  F | F  N |

**T& F**condition1 & condition2 & condition3

**Tor F**condition1 & condition2 & condition3



& F N T OR T N F

F T T N F

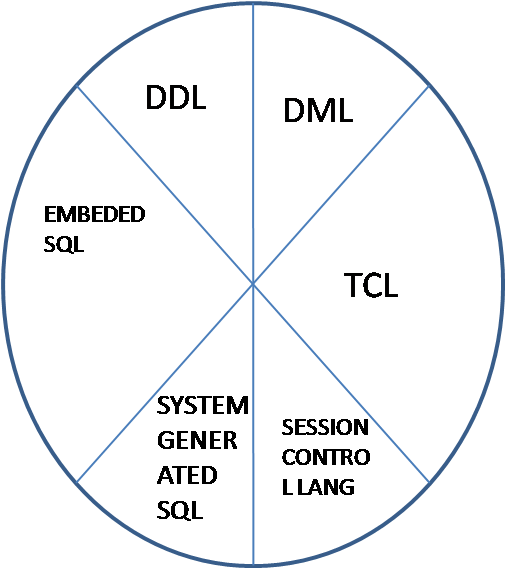
F T T F

**SQL STATEMENTS**

* It’s a structured query language(SQL) pronounced as SEQUEL
* It’s a set oriented language which means handles the bunch of records at a time that’s way it’s more faster than the PL/SQL
* SQL is a sub-language
* Case-insensitive
* Gateway to the RDBMS
* It’s the only language that directly interacts with databse
* Fourth generation language[PL/SQL is 3 rd generation]
* SQL won’t supports the control structures and Boolean data type
* You can embed this language with other languages like java, PL/SQL….[embed SQL]
* It’s a database standard language

**SQL - Sub divided into 6 sub languages**

|  |  |  |
| --- | --- | --- |
| DDL |  | DML |
| DCL | DRL |



**SEE TO DIAGRAM**

1) DDL (DCL)

2) DML (DRL OR DQL)

3) TCL

4) ALTER SESSION

5) ALTER SYSTEM

6) EMBEDDED SQL

1. **DDL (DATA DEFINATION LANGUAGE):**

* To handle the database object we use DDL commands.
* They are auto commit commands.
* They are session independent.
* CREATE
* ALTER
* DROP
* RENAME(9I)
* PURGE(10G)
* FLASHBACK(10G)
* COMMENT
* TRUNCATE……
* **CREATE**
* Which is used to define database
* Objects (tables, view, sequences…)
* Creating a table is our main concern for us here.

**SYN**: CREATE TABLE TABLENAME (COL DTPS(S), COL DTPS(S),….);

**EG**: CREATE TABLE NEWTAB (SNO NUMBER (5),SNAME VARCHAR2(10));

Note: A table can contain max of 1000 columns.

* **ALTER:** We use alter to modify the structure of database objects with the help of keywords.

KEY WORDS:

a)ADD - To add columns

b)MIDIFY - To modify column datatypes and size

c)RENAME -To rename column and table names

d)DROP- To drop columns.

1. **ADD:**

SYN: ALTER TABLE TABLENAME ADD(COL DTPS(S),COL1 DTPS(S),…);

**EG**: ALTER TABLE NEWTAB ADD(ID NUMBER(5),LOC VARCHAR2(10));

1. **MODIFY:**To modify the column datatype & even size also

SYN: ALTER TBALE TABLENAME MODIFY (COL NEWDATATYPE(S), COL

NEWDATATYPE(S));

EG: ALTER TABLE NEWTAB MODIFY (ID NUMBER (10), LOC NUMBE R(5));

1. **RENAME:** To rename a column name

SYN: ALTER TABLE TABLENAME RENAME COLUMN OLDCOLUMNNAME TO

NEW COLUMNAME.

**d) DROP:** To drop columns

SYN: ALTER TABLE TABLENAME DROP COLUMN COLUMNAME;

ALTER TABLE TABLENAME DROP (COL1, COL2, COL3,..);

EG: ALTER TABLE NEWTAB DROP COLUMN LOC;

ALTER TABLE NEWTAB DROP (ID, BALANCE);

* **RENAME:** To rename a database object name

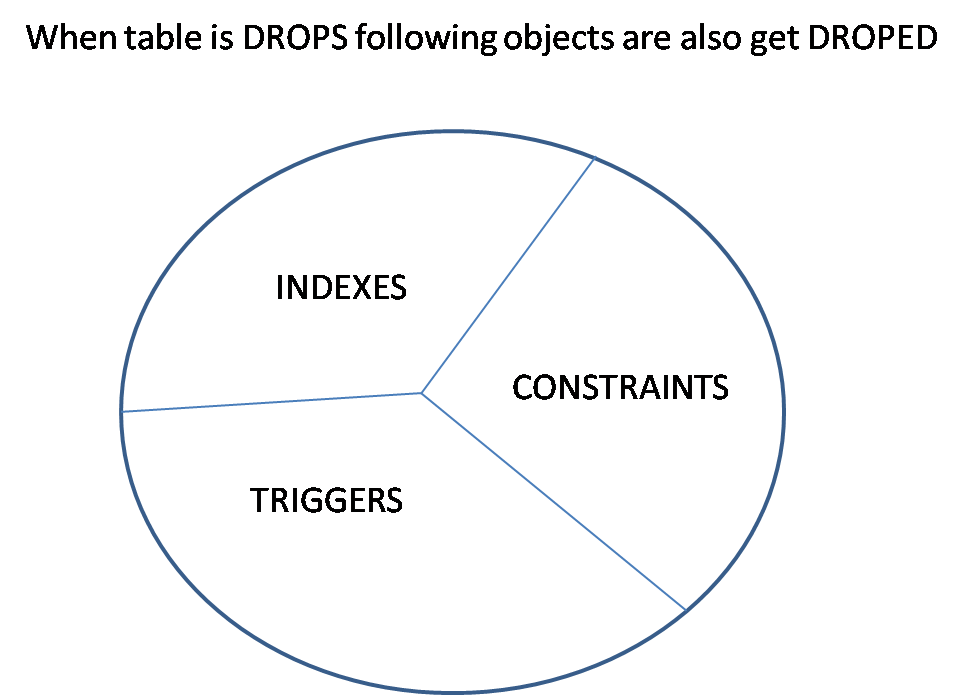
SYN: RENAME OLDNAME TO NEWNAME;

EG: RENAME NEWTAB TO NEWTAB1;

* **DROP:** To drop database objects

SYN: DROP TABLE TABLENAME;

EG : DROP TABLE NEWTAB1;



* **TRUNCATE**(DELETE+COMMIT):

SYN: TRUNCATE TABLE TABLENAME;

EG:TRUNCATE TABLE NEWTAB;

* **PURGE:**TO DROP THE TABLE FROM RECYCLEBIN;

SYN: PURGE TABLE TABLENAME;

TO BIPASS THE TABLE FROM RECYCLEBIN;

SYN: DROP TABLE TABLENAME PURGE;

* **FLASHBACK:**
* To retrieve the drop table
* From 10g onwards we have a concept called recycle in
* If you drop the table or database object for that matter
* They store in the recyclebin.to get back the object from
* Recycle in we user flashback (10g)

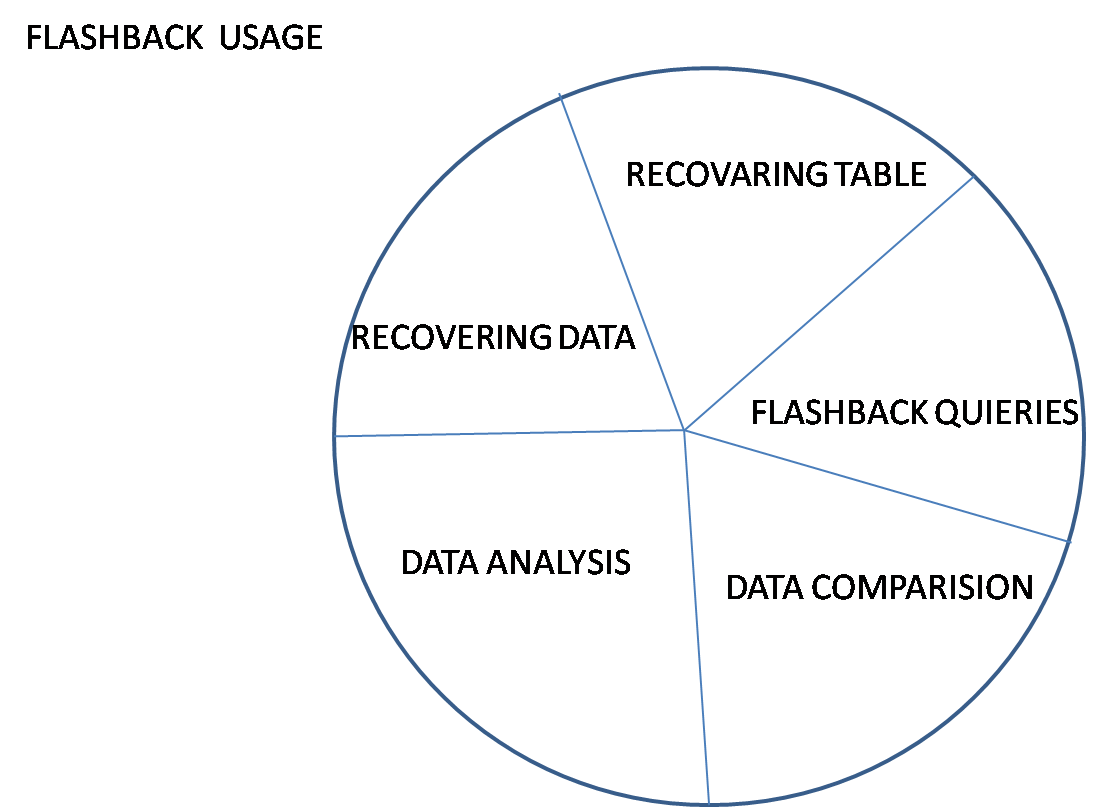
SYN: FLASHBACK TABLE TABLENAME TO BEFORE DROP;

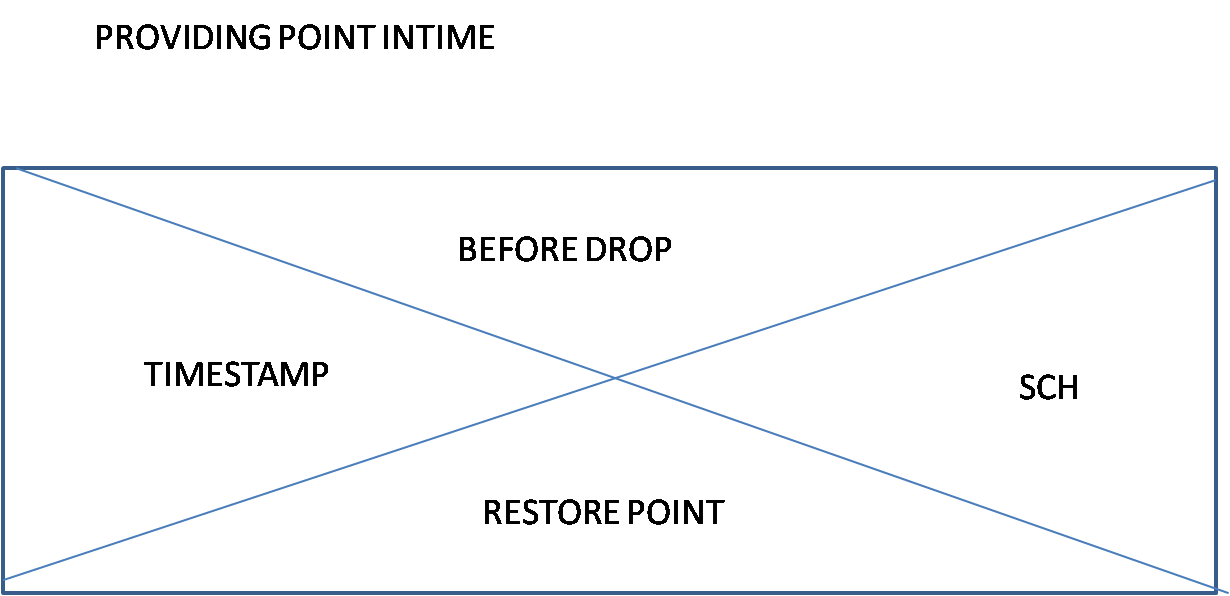
EG: DROP TABLE TL;

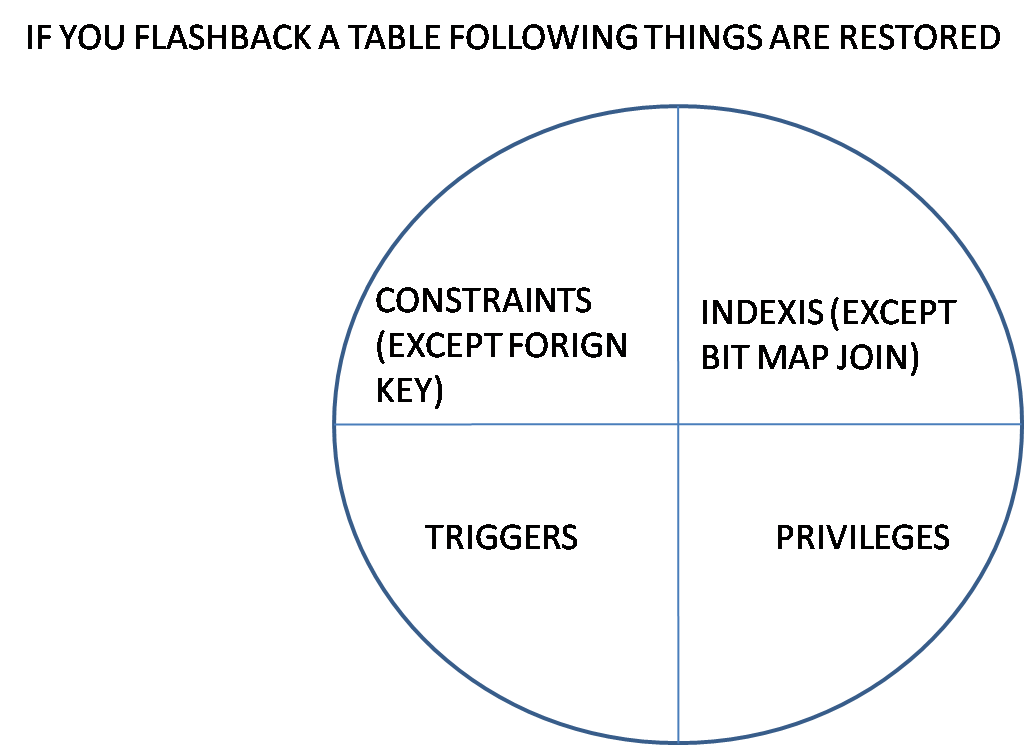
SELECT \* FROM TL;

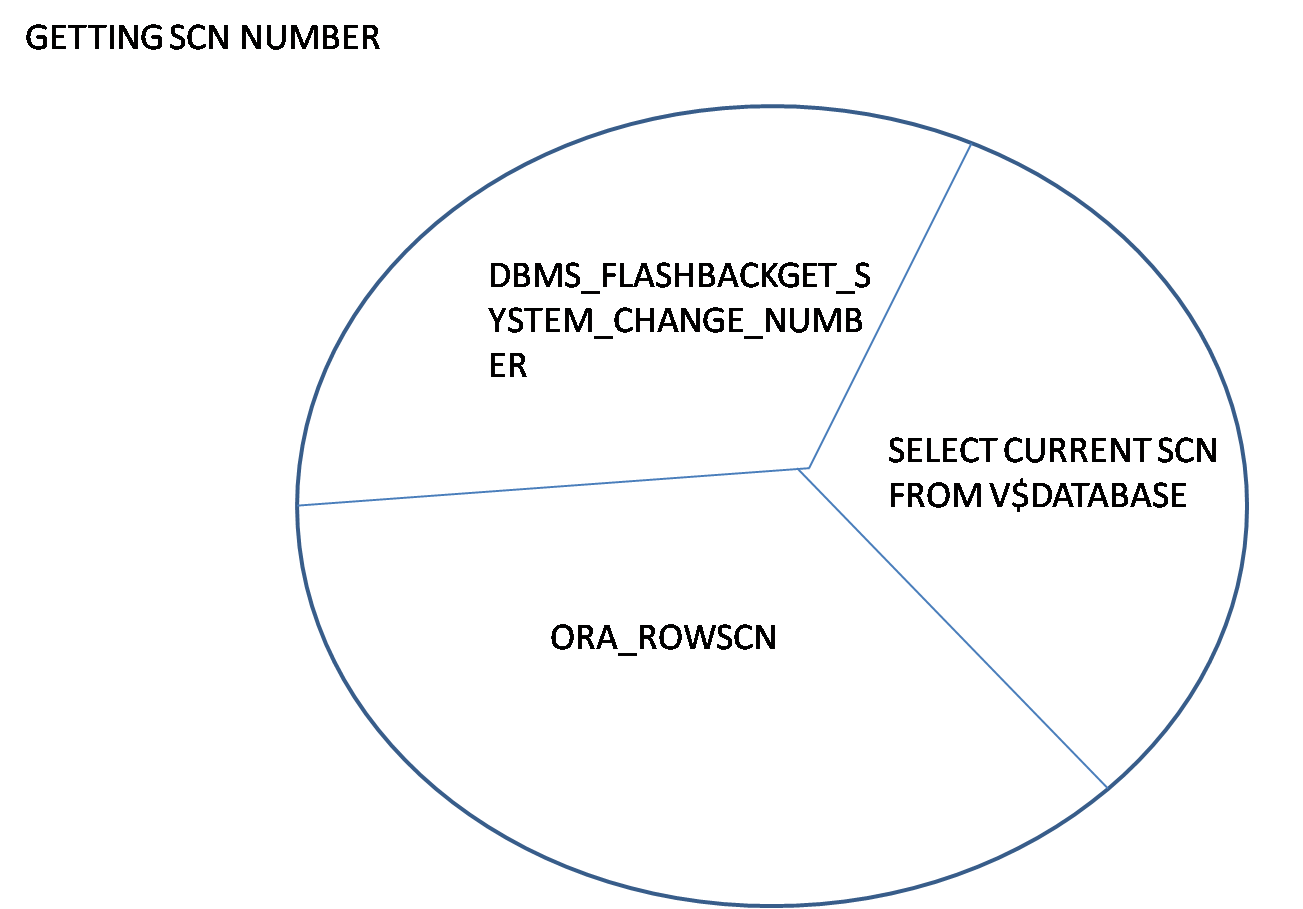
FLASHBACK TABLE TL TO BEFORE DROP;

SELECT \* FROM TL;









* **COMMENT:**

SYN: COMMENT ON TABLE TABLENAME|COLUMN TABLENAME.COLUMNAME IS

‘TEXT’

EG:COMMENT ON TABLE TL IS ‘HELLO’;

SELECT \* FROM USER\_TAB\_COMMENTS WHERE TABLE\_NAME=’TL’;



1. **DML(DATA MANUPULATION OR MODIFICATION LANGUAGE):**

* We use it to manipulate the data
* They are session dependent
* They are the temporary transactions
* Explicitly we have to make this transactions permanently
* They are non-auto commit commands
* INSERT
* UPDATE
* DELETE
* MERGE(INSERT+UPDATE+DELETE)
* SELECT(DRL OR DQL)
* INSERT
* VALUE METHOD
* REFRENCE METHOD
* SELECT METHOD
* **VALUE METHOD**

**SYN**: INSERT INTO TABLENAME [(COL1, COL2,..)] VALUES (VAL1, VAL2,)

EG:INSERT INTO NEWTAB VALUES (10,’A’);

INSERT INTO NEWTAB (SNO) VALUES (11);

* **REFRENCE METHOD:** Ampersand (&)

Which allows us to provide values from client to database?

SYN:INSERT INTO TABLENAME [(COL1, COL2)] VALUES (&N, &M);

EG: INSERT INTO NEWTAB (SNO, SNAME) VALUES (&N, &M);

* **SELECT METHOD:**

**SYN**: INSERT INTO TABLENAME SELECT …..

EG: CREATE TABLE TB1 AS SELECT \* FROM TB;

CREATE TABLE TAB2 AS SELECT \* FROM TB WHERE 1=2;

SELECT \* FROM TAB2;

DESC TB2;

INSERT INTO TB2 SELECT \* FROM TB;

INSERT INTO T1 VALUES SELECT \* FROM T: ERROR;

INSERT INTO T SELECT \* FROM T;

INSERT INTO T1 (C3) SELECT C2 FROM T;

INSERT INTO T1 VALUES (10, SELECT C2 FROM T): ERROR;

INSERT INTO T1 VALUES (10, (SELECT C2 FROM T WHERE C1=1));

* **UPDATE**:

**SYN**: UPDATE TABLENAME SET COL=VAL [, COL1=VALUES,…]

[WHERE CONDITION];

EG: UPDATE NEWTAB SET SNAME=’B’ WHERE SNO=10;

UPDATE NEWTAB SET SNAME=NULL;

UPDATE T SET C2= (SELECT C3 FROM T1);

UPDATE T SET C2= (SELECT C2 FROM T WHERE C1=2);

* **DELETE**: To delete the records.

SYN: DELETE FROM TABLENAME [WHERE CONDITION];

EG:DELETE FROM NEWTAB;

DELETE FROM NEWTAB WHERE SNO=10;

* **TRUNCATE:**

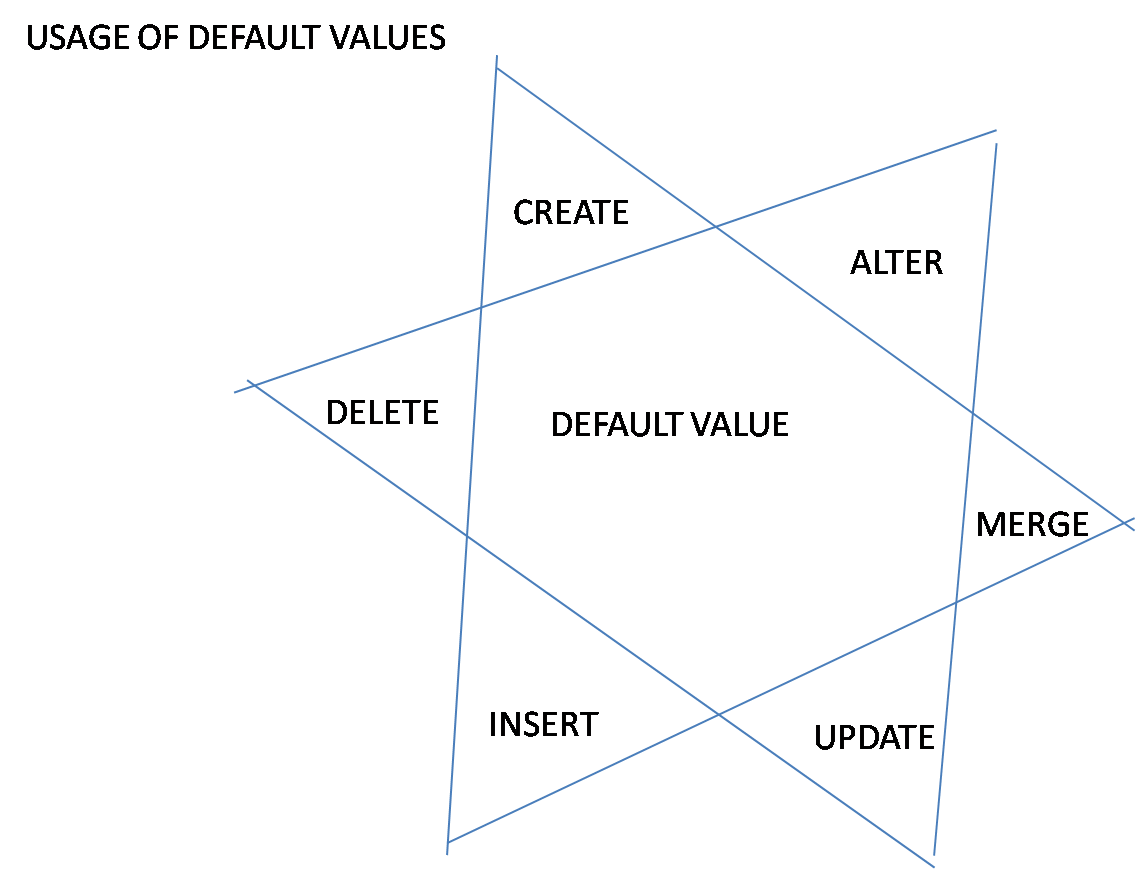
SYN: TRUNCATE TABLE TABLENAME;

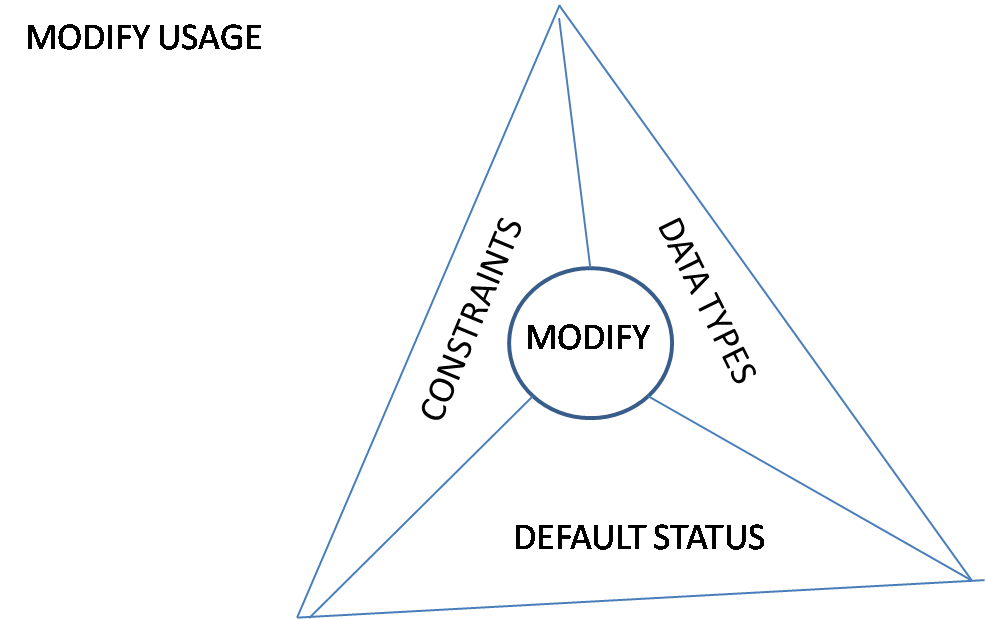
* It is used to delete the records permanently unlike deletion.
* It is auto commit command.
* It is handling only data not the structure.
* Not possible to delete the records specifically.

**DELETION:**

Delete the records temporarily.

Possible to delete specific records.





1. **DRL OR DQL (DATA QUERY/RETRIEVE LANGUAGE)**

**SELECT:**

**SYN**:

[WITH CLAUSE]

SELECT [DISTINCT|ALL] \*|COLUMNS|EXP|FUNCTIONS|LITERAL|SUBQUERIES

FROM TABLENAME|VIEWS|SUBQUERIES|TABLE FUNCTIONS

[WHERE CONDITION]

[START WITH CONDITON]

[CONNECT BY CONDITION]

[GROUP BY COLUMNS|EXP]

HAVING CONDITIONS (COLUMNS|FUNCTIONS)

ORDER BY COLUMNS|EXP|VALUES [ASC|DESC] [NULLS FIRST|NULLS LAST]|ANALYTICAL FUNCTIONS)

**EG**: SELECT \* FROM EMP;

SELECT 1 FROM EMP;

SELECT ALL \* FROM EMP;

SELECT DISTINCT \* FROM EMP;

SELECT DISTINCT DEPTNO FROM EMP;

SELECT 2\*2 FROM EMP;

**DUAL:**

* It is a dummy table.
* Dual table is having single record.
* To display the required result only for once we use dual table.

**MULTIPLE INSERT:**

TABLES: MULTAB,MULTAB1,MULTAB2;

**SYN**: INSERT ALL

INTO TABLENAME VALUES (VALUES)

INTO TABLENAME VALUES (VALUES)

INTO TABLENAME VALUES (VALUES)

SELECT \* FROM TABLENAME;

**EG**: INSERTING MULTIPLE VALUES INTO SAME TABLE;

INSERT ALL

INTO MULTAB VALUES (DEPTNO, DNAME, LOC)

INTO MULTAB VALUES (1,’X’,’HYD’)

INTO MULTAB VALUES (10,’Y’,’BANG’)

SELECT \* FROM DEPT;

**EG**: INSERTING MULTIPLE VALUES INTO SPECIFIED COLUMNS

INSERT ALL

INTO MULTAB (DEPTNO, LOC) VALUES (DEPTNO, LOC)

INTO MULTAB VALUES (1,’X’,’HYD’)

INTO MULTAB VALUES (10,’Y’,’BANG’)

SELECT \* FROM DEPT WHERE DEPTNO=10;

**EG**: INSERTING INTO MULTIPLE TABLES

INSERT ALL

INTO MULTAB VALUES (DEPTNO, DNAME, LOC)

INTO MULTAB VALUES (DEPTNO, DNAME, LOC)

INTO MULTAB VALUES (DEPTNO, DNAME, LOC)

SELECT \* FROM DEPT WHERE DEPTNO=10;

**EG**: INSERTING BASED ON WHEN CONDITION WITH OUT ELSE

INSERT ALL

WHEN DEPTNO=10 THEN

INTO MULTAB VALUES (DEPTNO, DNAME, LOC)

WHEN DEPTNO=20 THEN

INTO MULTAB VALUES (DEPTNO, DNAME, LOC)

WHEN DEPTNO=30 THEN

INTO MULTAB VALUES (DEPTNO, DNAME, LOC)

**SELECT \* FROM DEPT;**

**EG**: INSERTING WITH ELSE PART;

INSERT ALL

WHEN DEPTNO>50 THEN

INTO MULTAB VALUES (DEPTNO, DNAME, LOC)

WHEN DEPTNO>60 THEN

INTO MULTAB VALUES (DEPTNO, DNAME, LOC)

ELSE

INTO MULTAB VALUES (DEPTNO, DNAME, LOC)

SELECT \* FROM DEPT;

1. **TCL COMMANDS**

**TCL**: Transaction control language.

* Commit
* Roll Back
* Save Point
* **Commit:**
* It makes temporary transaction permanent
* It empties the buffer memory area
* A SCN number get generated for each of the transaction (or) for every commit
* It makes temporary piece of work as permanent
* **Roll Back**
* It cancels the transaction or piece of work
* It also empties the buffer memory area permanently
* Partial roll backing is also possible with the help of save point.
* **Save Point:**
* It is a mark which specifies teset of statements or piece of work.

**Note:**

* It is not possible to nest the transactions directly but by using some other means (program autonomous transaction) it is possible.
* A DDL command also makes the above temporary work permanent while becoming permanent.

**LITERALS**

Literals are predefined values or constants identified by oracle server

Types of literals

1) Number literals:

USAGE: 1, 10,

EG: SELECT 1 FROM DUAL;

2) String literals: anything enclosed in quotes is considered string.

USAGE: ‘A’,’10’,’13-SEP-13’

EG: SELECT ‘A’,’10’,’13-SEP-13’ FROM DUAL;

3) Date literals

USAGE: DATE ‘YYYY-MM-DD’

EG:SELECT DATE ’2013-09-13’ FROM DUAL;

EG:SELECT DATE ’2013-09-13’+1 FROM DUAL

4) Timestamp literal:

USAGE: TIMESTAMP ‘YYYY-MM-DD HH: MI: SS.FF’

EG: SELECT TIMESTAMP ‘2013-06-10 10:10:10.10

FROM DUAL;

5) Timestamp with time zone literal:

USAGE: TIMESTAMP ‘YYYY-MM-DD HH: MI: SS.FF +TZH: TZM’;

EG: SELECT TIMESTAMP '2012-09-10 10:10:10.000 +05:30'

FROM DUAL;

6) **Interval Literals:**

a) Interval year to month literals:

USAGE: INTERVAL ‘YY-MM’ YEAR TO MONTH;

EG: SELECT INTERVAL '10-10' YEAR TO MONTH FROM DUAL;

b) Interval day to second:

USAGE: INTERVAL ‘DD HH: MI: SS.FF DAY TO SECOND;

EG: SELECT INTERVAL '10 10:10:10.1000' DAY TO SECOND FROM

DUAL

**OPERATORES**

* Athematic
* Concatenation
* Relational or Comparison
* Special
* Logical
* Row Operators
* Hierarchical Operators;
* Set Operators
* Table Operators

**ORDER OF PRECEDENCE:**

**UNIARY OPERATORES**>1>2>3>4>5

**1) ARITHEMATIC:** \*, / , + , -

**2) CONCATINATION:** || (JOIN STRINGS)

**3) RELATIONAL:** =,>, <,>=, <=,!=,<>,^=,~=

**4) SPECIAL** IS,IN,LIKE( \_ ,%),BETWEEN(AND),ANY/SOME,ALL,EXISTS,

**5) ROW:** DISTINCT, ALL, PRIOR

**6) LOGICAL:** NOT, AND, OR;

**7) HIERARCHICAL:** CONNECT\_BY\_ROOT, PIROR

**8) TABLE:** THE;

**NOTE:** We can override the order of precedence or we can divert the order of

precedence by interpreting parantasis ( )

* **ARTHIMATIC OPERATORES:**

EG: SELECT 2\*100+10, 2\*(100+10) FROM DUAL;

SELECT SAL, SAL+SAL/10 FROM EMP;

SELECT \* FROM EMP WHERE DEPTNO>-1;

* **CONCATINATION OPERATOR:** To merge up or join the strings we use

Concatenation operators.

EG: SELECT ‘WELL’||’COME’ FROM DUAL;

SELECT EMPNO||’EMPLOYEE NAME’||ENAME FROM EMP;

SELECT ‘HELLO’|| NULL FROM DUAL;

SELECT ‘HELLO’||NULL||’WORLD’ FROM DUAL;

SELECT ‘HELLO’||’NULL’||’WORLD’ FROM DUAL;

SELECT ‘HELLO’||’ WELCOME’ FROM DUAL;

SELECT ‘123’+4||5 FROM DUAL;

SELECT ‘123’+5||-5 FROM DUAL;

* **RELATIONAL OR COMPARISON :**

EG: SELECT \* FROM EMP WHERE DPETNO=10 AND SAL>1000;

SELECT \* FROM EMP WHERE DEPTNO<>10;

SELECT \* FROM EMP WHERE DEPTNO!=10;

SELECT \* FROM EMP WHERE ENAME =’KING’;

SELECT \* FROM EMP WHERE ENAME >’KING’;

* **SPECIAL OPERATORES**:

**1)IS:**TO HANDLE NULL VALUES

**EG**: SELECT \* FROM EMP WHERE COMM=NULL; NO ROWS;

SELECT \* FROM EMP WHERE COMM IS NULL;

SELECT \* FROM EMP WHERE COMM IS NOT NULL;

**2)IN:** TO PROVIDE THE LIST OF VALUES

SYN: [NOT] IN (LIST VALUES) , IN=ANY; NOT IN !=ALL;

**EG: SELECT \* FROM EMP WHERE DEPTNO IN(10,20);**

SELECT \* FROM EMP WHERE DEPTNO NOT IN (10,20);

SELECT \* FROM EMP WHERE DEPETNO IN(10,NULL,20);

SELECT \* FROM EMP WHERE DEPTNO NOT IN(10,NULL,20);

**3) LIKE:**To search the patterns we use ‘like’ operator with the help of wild card

Characters

**%**  ANY OR NO NUMBER OF CHARACTERS

\_ SINGLE CHARACTERS COMPARISION

**EG**: SELECT \* FROM EMP WHERE ENAME LIKE ‘S%’;

SELECT \* FROM EMP WHERE ENAME LIKE ‘\_A%’;

SELECT \* FROM EMP WHERE ENAME LIKE ‘%E\_’;

SELECT \* FROM EMP WHERE ENAME LIKE ‘%A%’;

SELECT \* FROM EMP WHERE ENAME LIKE ‘S%’ OR ENAME LIKE ‘A%’;

SELECT \* FROM EMP WHERE ENAME REGEXP\_LIKE (ENAME,’^[SA]’);

SELECT \* FROM EMP WHERE ENAME NOT LIKE ‘S%’;

SELECT \* FROM EMP WHERE ENAME LIKE ‘KING’;

**NOTE:**like works as equal operator in the absence of wild card characters;

**4) BETWEEN**:

* To provide range of values.
* Always lower limit must be less than upper limit if not Condition becomes false.
* Range includes the boundary values also.

**EG:**SELECT \* FROM EMP WHERE SAL BETWEEN 1000 AND 2000;

SELECT \* FROM EMP WHERE SAL NOT BETWEEN 1000 AND 3000;

SELECT \* FROM EMP WHERE SAL BETWEEN 5000 AND 1000; NO ROWS

SELECT \* FROM EMP WHERE SAL BOT BETWEEN 5000 AND 1000;

(ALL THE ROWS ;)

**5) ANY/SOME:**In any, the given value has to become true with any of

the list of values

**6) ALL:**In all, the given value hat to become true with all of the list values.

**7) EXISTS:**returns true if record is found else false.

**NOTE:**we have to use any & all along with relational operators.

They can’t exist individually.

SELECT \* FROM EMP WHERE DEPTNO>ANY(10,20);

SELECT \* FROM EMP WHERE DEPTNO<ANY(10,20);

SELECT \* FROM EMP WHERE DEPTNO!=ANY(10,20);

SELECT \* FROM EMP WHERE DEPTNO!=ANY(10,NULL,20);

SELECT \* FROM EMP WHERE DEPTNO!=ALL(10,NULL,20);

SELECT \* FROM EMP WHERE DEPTNO=ANY(10,NULL,20);

SELECT \* FROM EMP WHERE DEPTNO=ALL(10,NULL,20);

**Arithmetic operators & order of** precedence**:**

**Arithmetic operators:**

|  |  |
| --- | --- |
| Level | Operator |
| High | ( ) |
| Mediumn | / |
| Medium | \* |
| Low | + |
| Low | - |

**Ex**:-=> 3 \* 100/5 + 20/10 – 5

=> 300/5 + 20/10 – 5

=> 60 + 20/10 – 5

=> 60+2-5

=> 62-5

=> 57

**Order of presidence:**

|  |  |
| --- | --- |
| Level | Operators |
| 1 | Unary (+, -, ~, connect\_by\_root) |
| 2 | Arithmetic |
| 3 | || |
| 4 | =, >, <, >=, <= |
| 5 | [not] like, is [not], [not] in |
| 6 | [not] between |
| 7 | !=, <>, ^= |
| 8 | Not |
| 9 | And |
| 10 | Or |
|  |  |

**COLUMN AND TABLE ALIASES**

* They are the alternate names to the table and columns
* **They are the temporary names only to that query**
* Which area used to identify the columns
* We use table aliases to avoid uncertainty and also to increase performance

**TABLE ALIASES**

* To qualify the columns
* We use it to increase the performance

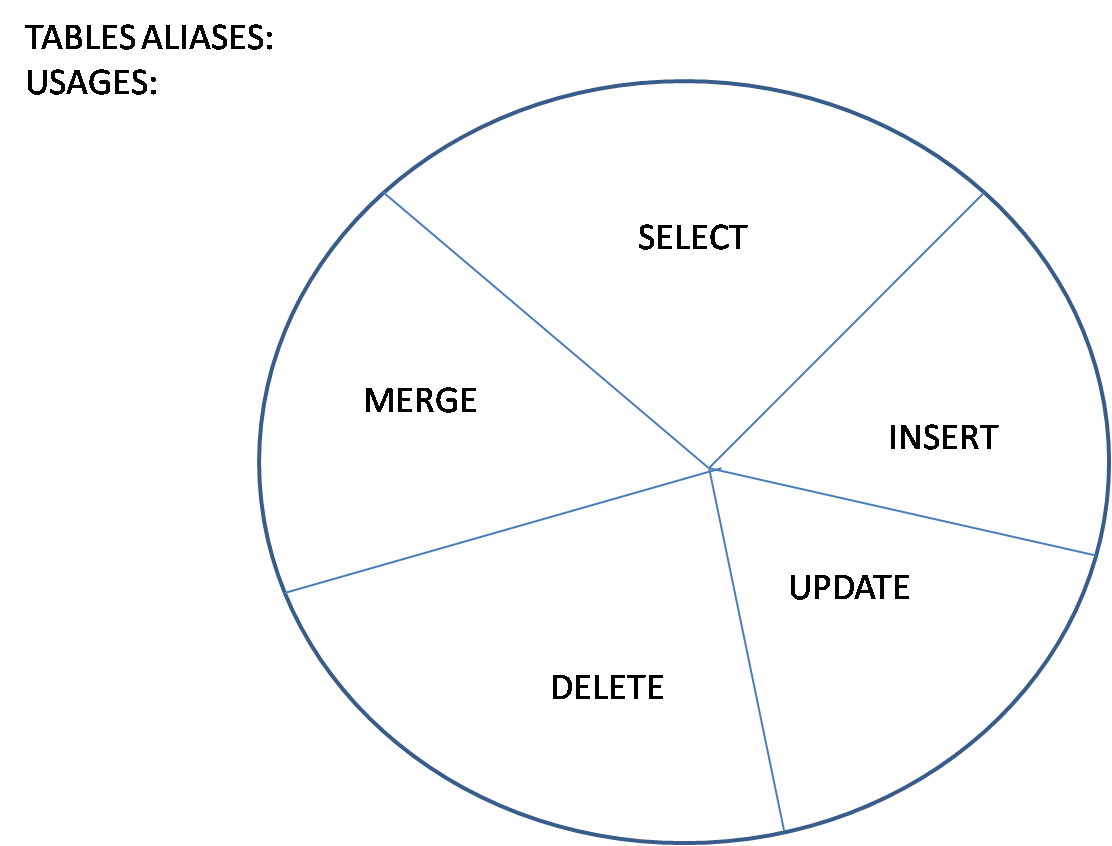
**EG**: SELECT DEPTNO FROM EMP, DEPT; ERROR (AMBIGUOUS)

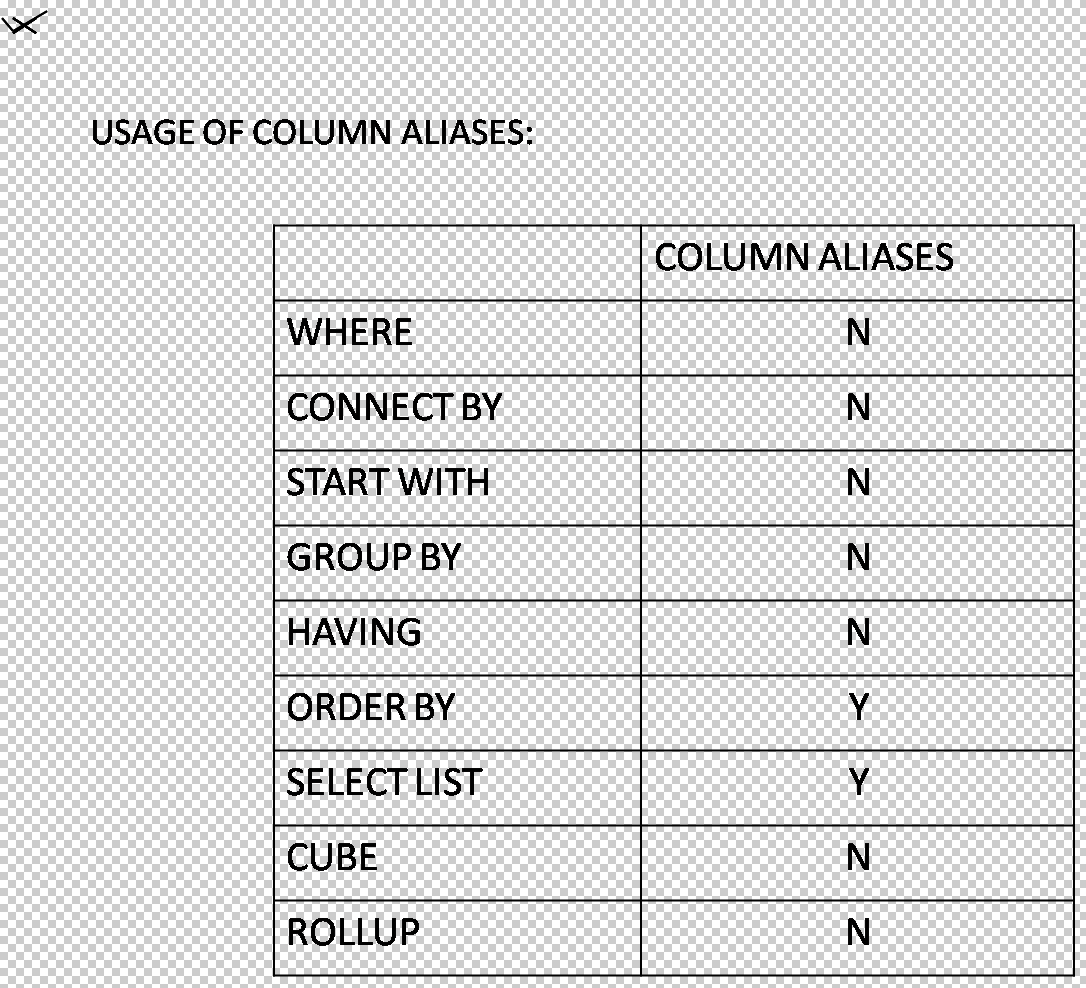
SELECT E.DEPTNO, D.LOC FROM EMP E,DEPT D

WHERE E.DEPTNO=D.DEPTNO;

**EG**: SELECT SAL “SALARY OF” FROM EMP;

SELECT 2\*2 EXP FROM DUAL;





**PSEUDO COLUMNS:**

They are the false columns or dummy columns which behaves as same to that of table columns. They are actually functions.

* SYSDATE;
* ROWNUM
* ROWID
* USER
* UID
* LEVEL
* NEXTVAL
* CURRVAL
* CONNECT\_BY\_ISLEAF
* CONNECT\_BY\_ISCYCLE
* XML PSEUDO COLUMNS
* COLUMN\_VALUE
* OBJECT\_VALUE
* **SYSDATE**: Displays the system date or server date

EG: SELECT SYSDATEL, ENAME FROM EMP;

SELECT SYSDATE FROM EMP;

SELECT SYSDATE FROM DUAL;

* **ROWNUM:**
  + ROWNUM provides the sequential number to the rows.
  + They are temporary numbers only to that query.
* **ROWID:**
* It is the physical address of the row
* It is in hexadecimal ,generated for each of the row useful to identify the records and to increase the performance and they are permanent
* They differ even for duplicate records also

**EG**: SELECT ROWID, EMP.\* FROM EMP;

DELETE FROM EMP WHERE ROWID=’ROWIDVALUE’;

SELECT \* FROM EMP WHERE ROWID=’ROWIDVALUE;

**Note**: Indexes will make use of ROWID to increase the performance

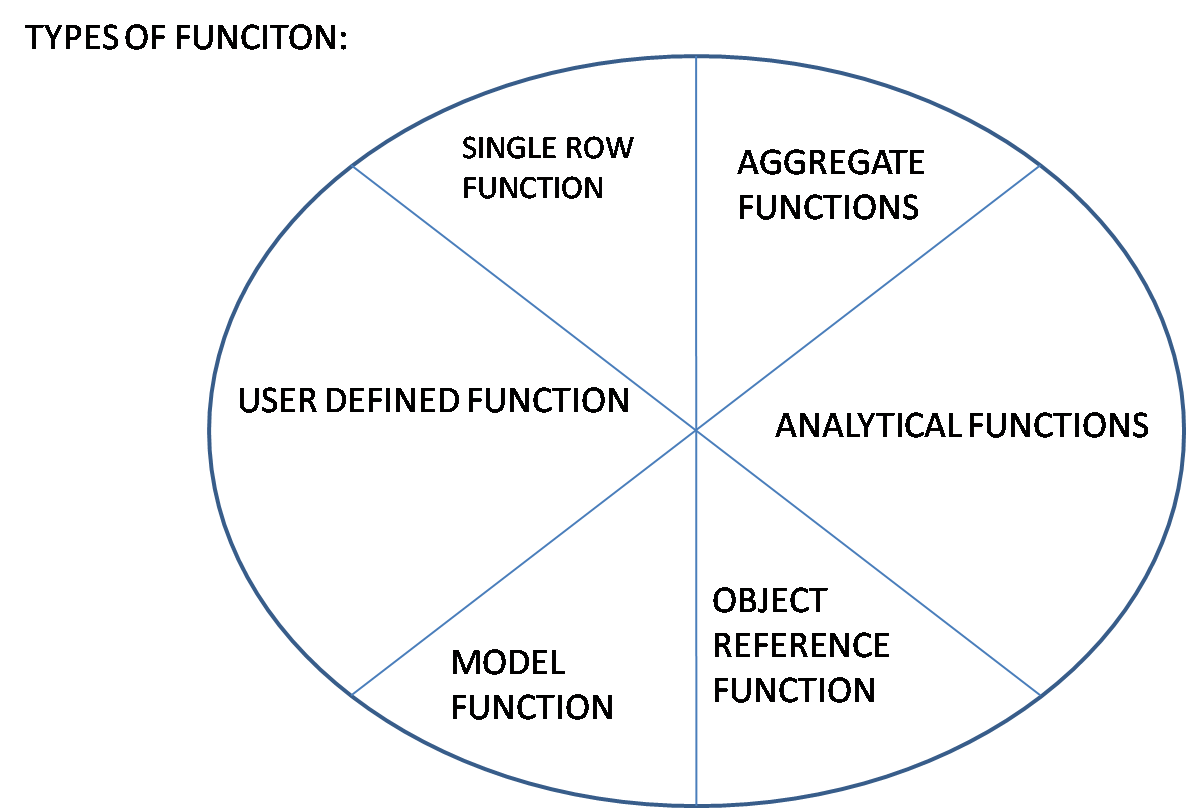
* **USER AND UID**: They specify username and USERID

SELECT USER,UID FROM DUAL;

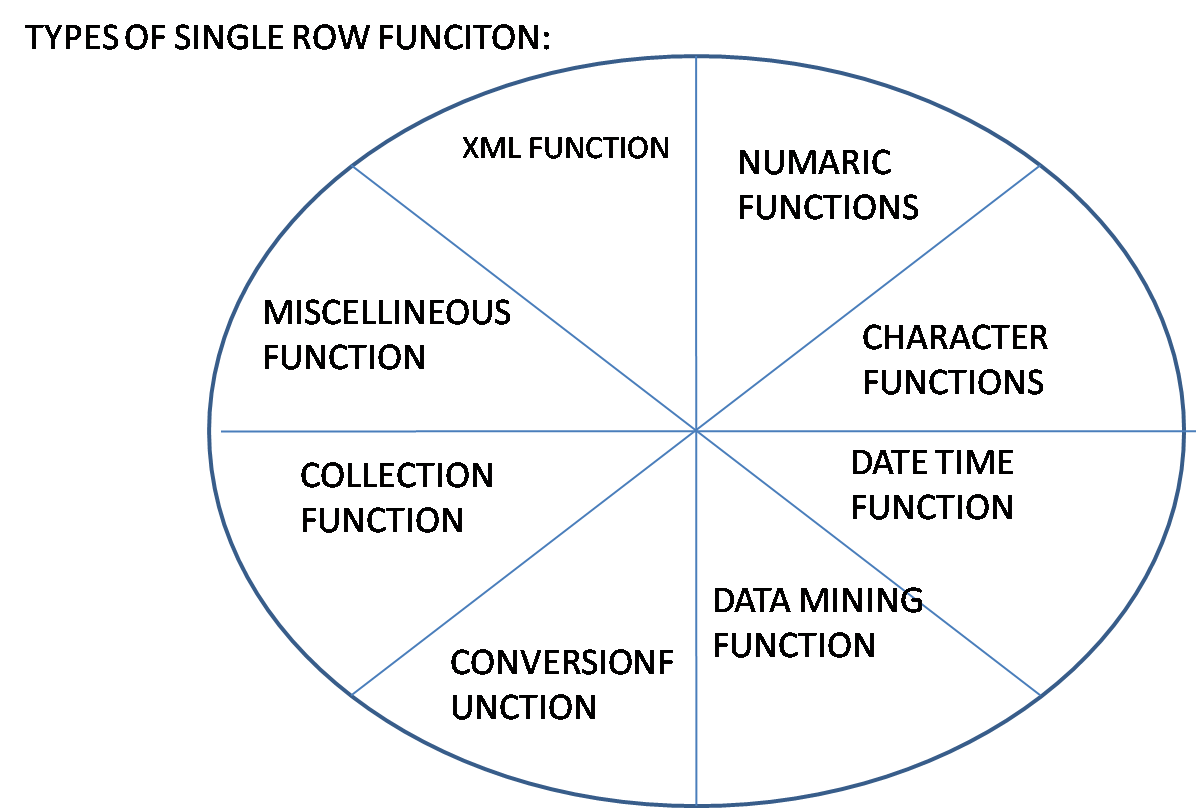
**FUNCTIONS**

They are built in programs which are used to modify the existing date or for calculations so as to full fill the business requirements followingare function types.

* SINGLE ROW FUNCTIONS
* MULTIPLE ROW FUNCTIONS (OR) GROUP FUNCTIONS (OR)
* AGGREGATE FUNCTIONS (OR) SUMMARY FUNCTIONS
* WINDOW FUNCTIONS
* ANALYTICAL (RANK FUNCTIONS)



* **SINGLE ROW FUNCTIONS OR SCALAR FUNCTIONS:**



* They get execute for each of the row and return a value
* Based on data we can classify the functions in following ways
  + - Number (or) numerical functions or

arthimatic functions(old versions)

* + - String functions or character functions or text functions
    - Date functions
    - Conversion functions
    - Null functions
    - Search functions
    - Conditional functions
    - Hierarchical functions
    - General functions
* **NUMBER FUNCTIONS:**
  + POWER
  + SQRT
  + MOD
  + REMINDER
  + SIN, COS,
  + SIGN
  + ABS
  + SINH, COSH
  + EXP
  + LOG
  + LN
  + CEIL
  + FLOOR
  + TRUNC
* **POWER:**To find out the power values

**SYN**: POWER (M, N) (M TO THE POWER OF N)

* **SQRT:**To find out the square root value

SYN: SQRT (VALUE)

* **MOD:**To find out the remainder

**SYN**: MOD (M, N)

* **REMAINDER:**To find out the remainder.

**SYN**: REMAINDER (M,N);

**NOTE:** MOD AND REMAINDER FUNCTIONS work with different mechanisms.

* **SIN,COS,…:**Tringometric functions
* **SIGN:**RETURNS -1 FOR ALL OF THE –VE VALUES

+1 FOR ALL OF THE +VE VALUES

0 FOR ZERO VALUE

**SYN**: SIGN (V);

* **ABS:**Returns absolute value irrespective of sign

**SYN**: ABS (V);

* **SINH,COSH,…:**HEPERBOLIC VALUES OF SINE,COS,…
* **EXP:** EXPONENTIAL VALUES
* **LOG:**LOG VALUES
* **LN :**NATURAL VALUES
* **CEIL:**Returns same value or next highest value

**SYN**: CEIL (V);

* **FLOOR:** RETURNS SAME VALUE OR NEXT LOWEST VALUE

**SYN**: FLOOR (V);

* **ROUND:**ROUND ROUNDS THE GIVEN VALUE TO GIVEN POSITION

**SYN**: ROUND (V);

* **TRUNC:**TRUNCS THE VALUE

**SYN:** TRUNC (V);

**EG:**SELECT POWER(5,2),POWER(0,0),POWER(1,0),POWER(0,1) FROM DUAL;

SELECT MOD(2,5) ,MOD(1,2),MOD(4,2) FROM DUAL;

SELECT REMAINDER(5,2) FROM DUAL;

SELECT MOD(3,-2),REMAINDER(3,-2) FROM DUAL;

SELECT SIN(45),SINH(50) FROM DUAL;

SELECT SIGN(-30),SIGN(0),SIGN(+30),ABS(-30),ABS(0),ABS(+30)FROM DUAL;

SELECT CEIL(-12.45),FLOOR(-14.56),CEIL(0.56),FLOOR(-0.567) FROM DUAL;

**ROUND:**Round rounds the value to given position and it also checks the position i.e. if the last eliminating value is greater than are equal to 5 or >5 then it simply add one value to the left adjacent value

**EG**: SELECT ROUND (12.567, 2) FROM DUAL;

SELECT ROUND (12.563999, 2) FROM DUAL;

SELECT ROUND (12.56, 3) FROM DUAL;

SELECT ROUND (13.56, 1) FROM DUAL;

SELECT ROUND (13.56) FROM DUAL;

SELECT ROUND (15.56,-1) FROM DUAL;

SELECT ROUND (-16.99,-1) FROM DUAL;

SELECT ROUND ( -56.99,-2) FROM DUAL;

SELECT ROUND ( -56.99,-3) FROM DUAL;

SELECT ROUND (12.56, 1 ) ,TRUNC(12.56,1) FROM DUAL;

SELECT ROUND ((TRUNC (98.56),-2) FROM DUAL;

* **STRING FUNCTIONS:**
* LENGTH
* VSIZE
* DUMP
* REVERSE
* SOUNDEX
* UPPER
* LOWER
* INITCAP
* LTRIM
* RTRIM
* LPAD
* RPAD
* TRANSLATE
* REPLACE
* DECODE
* SUBSTR
* INSTR
* SUBSTRB
* SUBSTRC
* SUBSTR2
* SUBSTR4
* CONCAT

**----FROM 10G ON WARDS THEY INTRODUCED REGUAL EXPRESSIONS----**

* REGEXP\_LIKE
* REGEXP\_COUNT(11G)
* REGEXP\_SUBSTR
* REGEXP\_SUBSTR
* REGEXP\_REPLACE
* **LENGTH**: TO FIND OUT THE NO OF CHARACTERS IN A GIVEN STRING

**EG:** SELECT LENGTH (‘ABC’) FROM DUAL;

* **VSIZE:**NO OF BYTES ACCOUPIED BY A GIVEN STRING

**EG**: SELECT LENGTH (‘ABC’), VSIZE (‘ABC’) FROM DUAL;

SELECT LENGTH(SYSDATE),VSIZE(SYSDATE) FROM DUAL;(DIFFERS);

* **DUMP**: TO DISPLAY THE GIVEN CHARACTER IN DATABASE CHARACTER SET

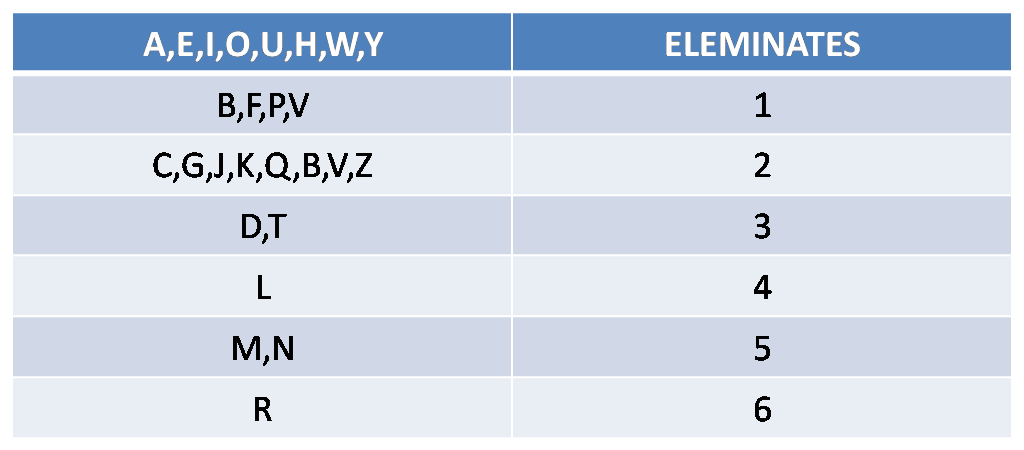
**EG**: SELECT DUMP (‘ABC’) FROM DUAL;

* **REVERSE**: TO DISPLAY THE GIVEN STRING IN REVERSE MANNER;

**EG**: SELECT REVERSE (‘ABC’) FROM DUAL;

SELECT REVERSE (ENAME) FROM DUAL;

* **SOUNDEX:**WORKS BASE ON PHONITICAL SOUNDS. INTERNALLY IT CONVERTS THE STRING INTO CODE VALUES.



**EG**: SELECT \* FROM EMP WHERE SOUNDEX(ENAME)=SOUNDEX(‘SMITHE’);

SELECT SOUNDEX (ENAME) FROM EMP;

* **CONCAT(||):**TO APPEND THE STRINGS;

**SYN**: CONCAT (S1,S2);

EG: SELECT CONCAT (‘WEL’,’COME’) FROM DUAL;

SELECT ‘WEL’||’COME’ FROM DUAL;

**DISADV**: concat join only two strings unlike concatenation operators

**NOTE:** both will ignore the null values

* **CERT**: SELECT ‘WEL’||NULL||’COME’ FROM DUAL;

SELECT CONCAT(‘WEL’,NULL) FROM DUAL;

SELECT CONCAT(NULL,’WELL’) FROM DUAL;

* **UPPER**: To convert the string into upper case or capitalize
* **LOWER**: Reverse to that of upper

**EG**: SELECT UPPER (‘abs’), LOWER (‘ABC’) FROM DUAL;

* **INITCAP:**CONVERTS THE INITIAL LETTER INTO UPPER CASE AND REMAINING LETTERS INTO LOWER CASE
* **CERT:**SEECT INITCAP(‘abc’) ,INITCAP(‘abc abc’),INITCAP(‘abc\_efg%hij’) FROM DUAL;
* **LTRIM AND RTRIM**: TO TRIM THE CHARACTERS FROM LEFT OR FROM RIGHT ENDS
* **TRIM**: TO TRIM FROM BOTH ENDS.

**SYN**: TRIM([LEADING|BOTH|TRAILING [‘C’ FROM]],’TEXT’);

**NOTE:** DEFAULT TRIMS SPACES;

**EG**: SELECT LTRIM(‘WELCOME’,’W’) FROM DUAL;

SELECT LTRIM(‘WELCOME,’E’) FROM DUAL;

SELECT RTRIM(‘WELCOMEEE’,’E’) FROM DUAL;

SELECT LTRIM(‘EEEWELCOMEE’,’E’) FROM DUAL;

SELECT LTRIM(‘EEEWELCOME’,’WE’) FROM DUAL;

SELECT LTRIM(RTRIM(‘EWEWELCOME’,’EW’),’WE’) FROM DUAL;

SELECT TRIM(‘E’ FROM ‘EEEWELCOMEEE’) FROM DUAL;

SELECT TRIM(‘ WELCOME ‘) FROM DUAL;

SELECT TRIM(LEADING ‘E’ FROM ‘EEEWELCOME’) FROM DUAL;

* LPAD
* **RPAD**: To append the character from left or from right end of a given string to a given position

**SYN**:LPAD(S,N,’C’);

RPAD(S,N,’C’);

**EG:**SELECT LPAD(‘WELCOME’,10,’\*’) FROM DUAL;

SELECT RPAD(‘WELCOME’,9,’\*@’) FROM DUAL;

SELECT RPAD(‘WELCOME’,10,’\*@’) FROM DUAL;

SELECT LPAD(‘WELCOME’,7,’\*’) FROM DUAL;

SELECT LPAD(‘WELCOME’,6,’\*’) FROM DUAL;

SELECT RPAD(‘WELCOME’,6,’\*’) FROM DUAL;

* **TRANSLATE**: To translate character wise

**SYN**: TRANSLATE(S,’C’,’C’): It takes strings only individually

SELECT TRANSLATE(‘WELCOME’,’E’,’A’) FROM DUAL;

SELECT TRANSLATE(‘WELCOME’, ‘EL’,’A’) FROM DUAL;

SELECT TRANSLATE(‘WELCOME ‘EL’,’AB’) FROM DUAL;

SELECT TRANSLATE(‘WELCOME’,’EL’,’A\_’) FROM DUAL;

**DISADV:**NOT POSSIBLE TO REPLACE STRING WISE

* **REPLACE:**To replace string wise

**SYN**: REPLACE(S,S,S);

EG: SELECT TRANSLATE(‘INDIA’,’IN’,’XY’),REPLACE(‘INDIA’,’IN’,’XY’)

FROM DUAL;

SELECT JOB,REPLACE(JOB,’MANAGER’,’MGR’) FROM EMP;

**DISADV**: NOT POSSIBLE TO REPLACE MORE THAN ONE STRING

* **DECODE**: (ORACLE) It works as same to that of ‘IF CONDITION’ IN SQL;

**SYN:**DECODE (COLUMN|EXP|VALUE|NULL, COND1, DO1,COND2,DO2,…[ELSE]);

**NOTE:** It considers the null values. In decode nulls are equal

MAX 255 ARGUMENTS ARE ALLOWED;

MIN 3 ARGUMENTS MANDATOREY;

**EG:**SELECT DECODE(1,2,3) FROM DUAL;

SELECT DECODE(1,2,3,1,2,3) FROM DUAL;

SELECT DECODE(NULL,1,NULL,2) FROM DUAL;

SELECT DECODE(1,DECODE(1,2,3),2,NULL) FROM DUAL;

SELECT DECODE(JOB,’MANAGER’,’MGR’,’SALESMAN’,’SLS’,JOB)

FROM DUAL;

**DISADV:**Won’t allow relational operator

* **SUBSTR:**TO DISPLAY THE SET OF CHARACTERS FROM A GIVEN POSITION

**SYN**: SUBSTR(S,M,(N));

S=STRING,

M=POSITION,

N=NO OF CHARACTERS

**EG**: SELECT SUBSTR(‘WELCOME’,2,2) FROM DUAL;

SELECT SUBSTR(‘WELCOME’,1,2) FROM DUAL;

SELECT SUBSTR(‘WELCOME’,7,2) FROM DUAL;

SELECT SUBSTR(‘WELCOME’,2,-2) FROM DUAL;

SELECT SUBSTR(‘WELCOME’,2) FROM DUAL;

SELECT SUBSTR(‘WELCOME’,-2,2) FROM DUAL;

SELECT SUBSTR(‘WELCOME’,-8,2) FROM DUAL;

SELECT SUBSTR(‘WELCOME,1) FROM DUAL;

SELECT SUBSTR(‘WELCOME’,0) FROM DUAL;

* **INSTR:**

SYN: INSTR (S,’C’,[P [,O]]);

S=STRING;

C=CHARACTER

P=POSITION

O=OCCURANCE

**EG:**SELECT INSTR(‘WELCOME’,’E’,1,1) FROM DUAL;

SELECT INSTR(‘WELCOME’,’E’,1,2) FROM DUAL;

SELECT INSTR(‘WELCOME’,’E’,2,1) FROM DUAL;

SELECT INSTR(‘WELCOME’,’E’,3,2) FROM DUAL;

SELECT INSTR(‘WELCOME’,’E’,3,-1) FROM DUAL;

SELECT INSTR(‘WELCOME’,’E’) FROM DUAL;

SELECT INSTR(‘WELCOME’,’E’,1) FROM DUAL;

SELECT INSTR(‘WELCOME’,’E’,-3,1) FROM DUAL;

* **DATE FUNCTIONS:**

DATE FORMATS

D(1,2,..) DAY OF THE WEEK(1-7)

DD(1 TO 31) DAY OF THE MONTH(1-31)

DDD(1 TO 365) DAY OF THE YEAR

DY(SUN,MON,..) FIRST THREE CHARACTERS OF THE WEEK

Dy

DAY(SUNDAY,…) COMPLETE CHARACTERS OF THE WEEK

Day

MM(1-12) MONTH OF THE YEAR

MON(JAN,FEB,..) FIRST THREE CHARACTERS OF THE MONTH

MONTH(JANUARY,..) COMPLETE CHARACTERS OF THE MONTH

Month

Y(3)

YY(13) LAST ONE,TWO,THREE DIGITS OF THE YEAR

YYY(013)

YYYY(2013) COMPLETE DIGITS OF THE YEAR

SYYYY (AD AND BC) YEAR WITH SIGN FOR BC –VE AND +VE FOR AD

RR(13) LAST TWO DIGITS OF THE YEAR

RRRR(2013) YEAR COMPLETE DIGITS

I

IY

IYY

IYYY DIGITS OF THE YEAR IN ISO FORMATS

W(1-5) WEEK OF THE MONTH

WW(1-52) WEEK OF THE YEAR

IW WEEK OF THE YEAR IN ISO STANDARDS

FM FILL MODE( ELEMENATES SPACES AND ZEROES)

FF FRACTION OF SECONDS

XF EXACT FORMAT

HH HOUR FORMAT(DEFAULT 12 HOUR)

HH12 12 HOUR FORMAT

HH24 24 HOUR FORMAT

MI MINUTES

SS SECONDS

SP TO SPELL THE DIGITS

TH ORDINALS(TH,RD,ST,..)

Q QUARTER OF THE YEAR

J JULIAN DAY(A/C TO JULIAN CALENDRE)

RM ROMAN NUMBERICAL LETTERS WHICH REPRESENTS NO OF……… MONTHS

DL LONG DATE

DS SHORT DATE

TZH TIMEZONE HOUR

TZM TIMEZONE MINUTE

TZR TIMEZONE REGION

TZA TIMEZONE ABBAR

AM/PM

**.** Period

**:**

-

**/**

“TEXT”

**DATE FUNCTIONS**

* SYSDATE
* CURRENT\_DATE
* SYSTIMESTAMP
* CURRENT\_TIMESTAMP
* LOCAL TIMESTAMP
* DBTIMEZONE
* ADD\_MONTHS
* MONTHS\_BETWEEN
* NEXT\_DAY
* LAST\_DAY
* EXTRACT
* ROUND
* TRUNC
* NEW\_TIME

**EG:**SELECT SYSDATE,CURRENT\_DATE FROM DUAL;

SELECT SYSTIMESTAMP,CURRENT\_TIMESTAMP FROM DUAL;

SELECT LOCALTIMESTAMP FROM DUAL;

SELECT DBTIMEZONE FROM DUAL;

* **ADD\_MONTHS:**TO ADD OR SUBSTRACT NUMBER OF MONTHS TO A GIVEN DATE;

**SYN**: ADD\_MONTHS (DATE, N);

**EG:**SELECT SYSDATE,ADD\_MONTHS(SYSDATE,1),SYSDATE+30 FROM DUAL;

SELECT SYSDATE,ADD\_MONTHS (SYSDATE,-1) FROM DUAL;

* **MONTHS\_BEWTWEEN**: To display the no of months between two given date but always date1>date2

**SYN**: MONTHS\_BETWEEN (DATE1, DATE2);

**EG:** SELECT SYSDATE, HIREDATE, MONTHS\_BETWEEN (SYSDATE, HIREDATE)

FROM EMP;

SELECT SYSDATE,HIREDATE,ROUND(MONTHS\_BETWEEN

(SYSDATE,HIREDATE)) FROM EMP

* **NEXT\_DAY:**Based on the format it display the next day of the week

**SYN**: NEXT\_DAY (DATE,’FORMAT’);

**EG:**SELECT NEXT\_DAY (SYSDATE,’MON’) FROM DUAL;

SELECT NEXT-DAY (SYSDATE,’TUE’) FROM DUAL;

* **LAST\_DAY:**Based on the given date.it displays the last day date of the month

**SYN**: LAST\_DAY (DATE);

**EG:** SELECT LAST-DAY (SYSDATE) FROM DUAL;

* **EXTRACT:**

**SYN**: EXTRACT (YEAR|MONTH|DAY|HOUR|MINUTE|SECOND FROM DATE);

EG:SELECT EXTRACT (YEAR FROM SYSDATE) FROM DUAL;

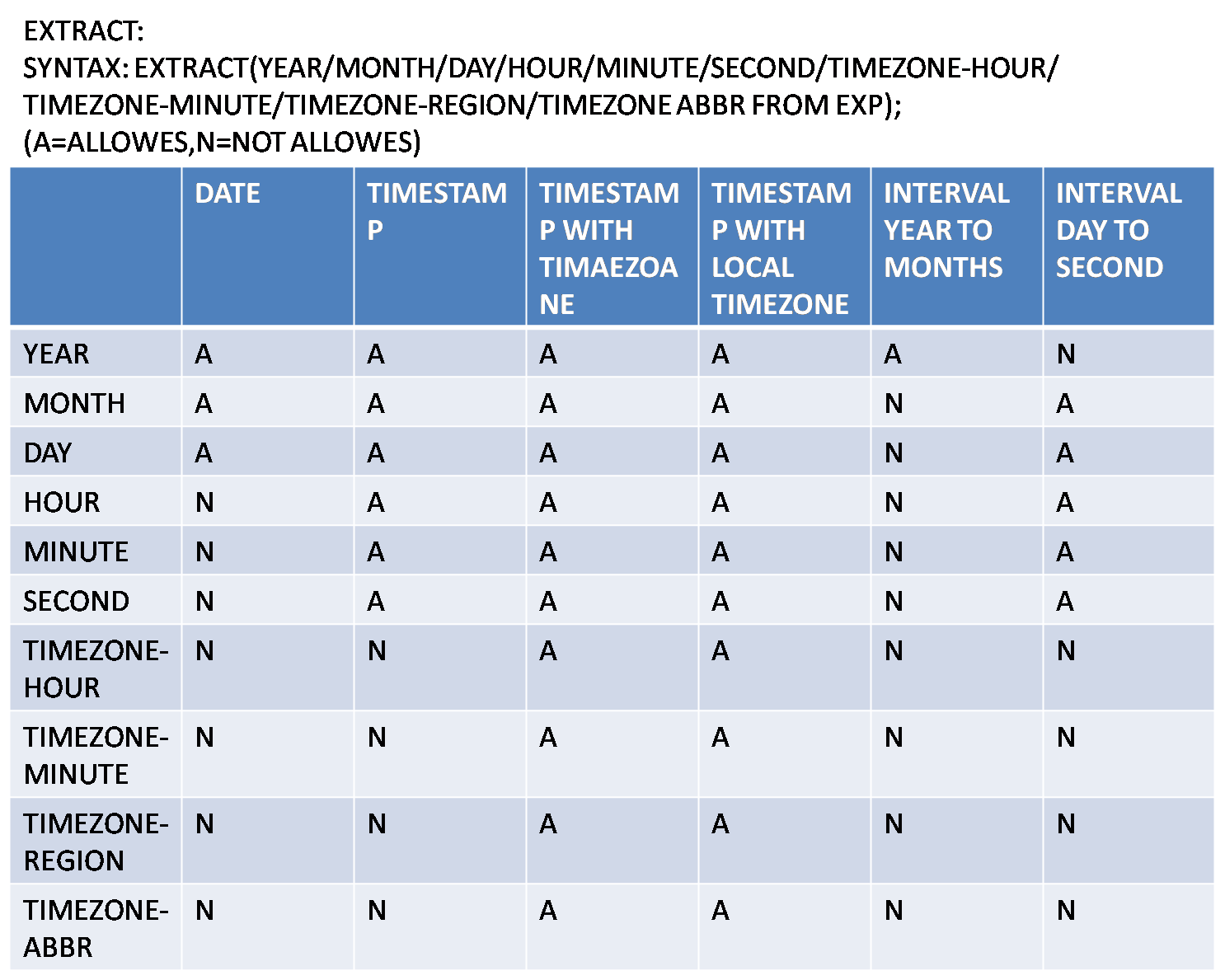
SELECT EXTRACT (MONTH FROM SYSDATE) FROM DUAL;

SELECT EXTRACT ( DAY FROM SYSDATE) FROM DUAL;

SELECT EXTRACT (HOUR FROM SYSTIMESTAMP) FROM DUAL;

SELECT EXTRACT (MINUTE FROM SYSTIMESTAMP) FROM DUAL;

SELECT EXTRACT (SECONDS FROM SYSTIMESTAMP) FROM DUAL;



* **ROUND:**Rounds the date based on formats

**SYN**: ROUND (DATE,’YEAR|MONTH|DAY’);

**EG:**SELECT ROUND (SYSDATE,’YEAR’) FROM DUAL;

SELECT ROUND (SYSDATE,’MONTH’) FROM DUAL;

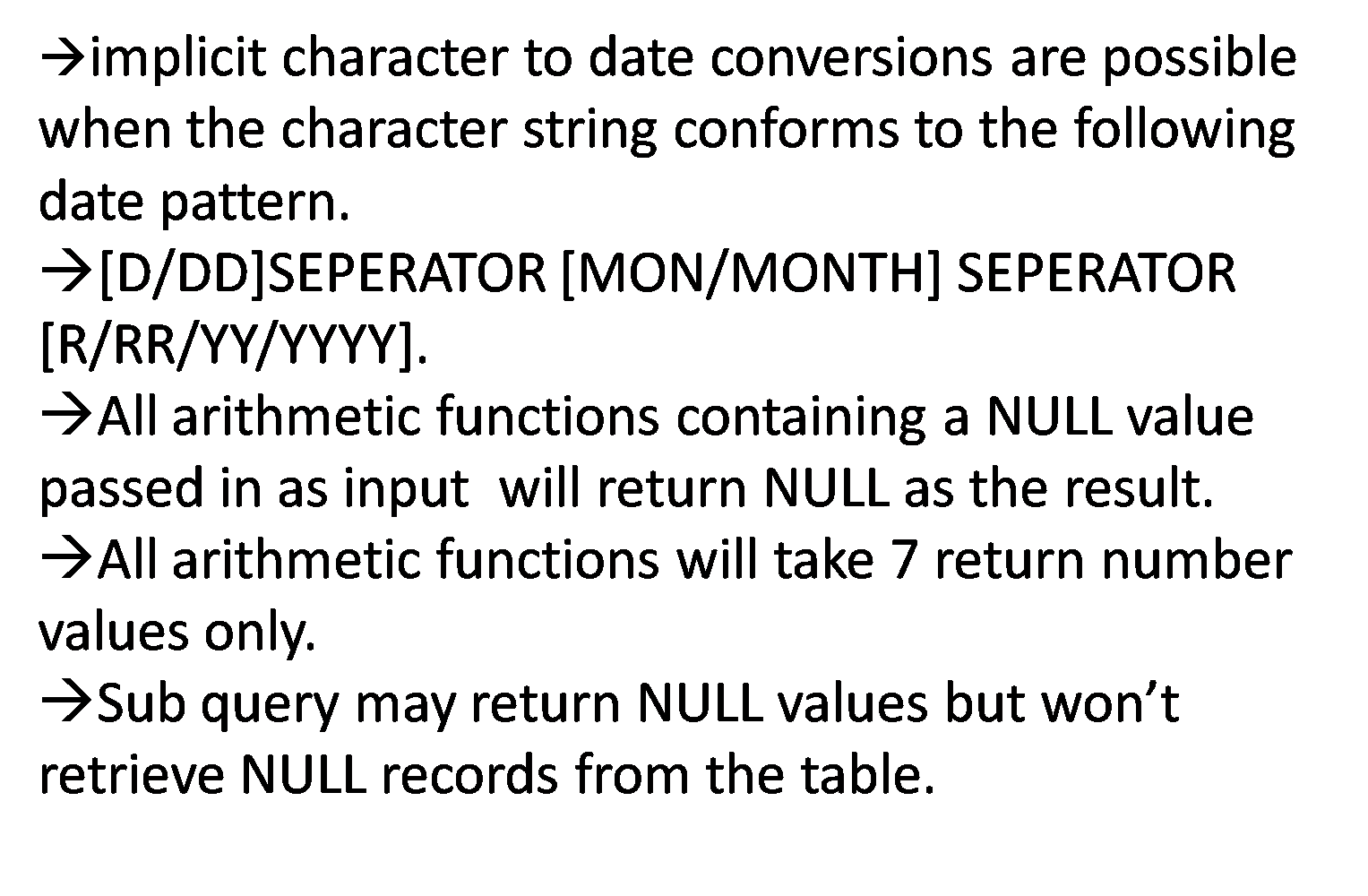
SELECT ROUND (SYSDATE,’DAY’) FROM DUAL;

* **TRUNC:**TRUNC THE DATES

**SYN**:TRUNC (DATE,’FORMAT’);

**EX**: SELECT TRUNC (SYSDATE,’YEAR’) FROM DUAL;

SELECT TRUNC(SYSDATE,’MONTH’) FROM DUAL;

* **CONVERSION FUNCTIONS:**
  + - TO\_CHAR;
    - TO\_DATE;
    - TO\_NUMBER;
    - TO\_TIMESTAMP;
    - TO\_TIMESTAMP\_TZ;
    - TO\_YMINTERVAL;
    - TO\_DSINTERVAL;
    - TO\_BINARY\_FLOAT;
    - TO\_BINARY\_DOUBLE;
    - TO\_BLOB;
    - TO\_CLOB;
    - TO\_LOB
    - BIN\_TO\_NUM;
    - NUMTOYMINTERVAL;
    - NUMTODSINTERVAL;
    - RAWTOHEX
    - TIMESTAMP\_TO\_SCN
    - SCN\_TO\_TIMESTAMP;
    - TO\_NCHAR;
    - TO\_NCLOB;
* **TO\_CHAR:**Converts date and numbers into character format;**SYN**: TO\_CHAR (DATE,’FORMAT’);

**EG**: SELECT TO\_CHAR(SYSDATE,’D’) FROM DUAL;

SELECT TO\_CHAR(SYSDATE,’DD’) FROM DUAL;

SELECT TO\_CHAR(SYSDATE,’DY’) FROM DUAL;

SELECT TO\_CHAR(SYSDATE,’CC’) FROM DUAL;

SELECT TO\_CHAR(SYSDATE,’FM MM/DD/YY’) FROM DUAL;

SELECT \* FROM EMP WHERE TO\_CHAR(HIREDATE,’MON’)=’FEB’;

SELECT \* FROM EMP WHEE TO\_CHAR(HIREDATE,’MM’)=02;

SELECT \* FROM EMP WHERE TO\_CHAR(HIREDATE,’YY’)=81;

SELECT \* FROM EMP WHERE TO\_CHAR(HIREDATE,’D’)=1;

SELECT \* FROM EMP WHERE TO\_CHAR(HIREDATE,’W’)=1;

* **TO\_DATE**: Converts character format to date format

**SYN**: TO\_DATE (‘C’,’FORMAT’);

**EG:** SELECT TO\_DATE (‘10’,’DD’) FROM DUAL;

SELECT TO\_DATE (‘10’,’MM’) FROM DUAL;

SELECT TO\_DATE (‘10’,’YY’) FROM DUAL;

SELECT TO\_DATE (‘FEB’,’MON’) FROM DUAL;

SELECT TO\_CHAR (ROUND(TO\_DATE(‘11’,’YY’),’YEAR’),’MM’)

FROM DUAL;

* TO\_NUMBER: Converts character into NUMBER FORMAT
* TO\_YMINTERVAL: Converts the character into YEAR AND MONTH INTERVAL
* TO\_DISINTERVAL: Converts the character into DAY,HOUR,MINUTE AND SECONDS INTERVAL
* BIN\_TO\_NUM: BINARY TO NUMBER
* NUMTOYMINTERVAL: NUMBER TO YEAR AND MONTH INTERVALS
* NUMTODSINTERVAL: NUMBER TO DAY ,HOUR,MINUTE AND SECONDS INTERVAL

**EG:**

SELECT TO\_TIMESTAMP(‘11’,’FF’) FROM DUAL;

SELECT TO\_TIMESTAMP(‘11’,’HH’) FROM DUAL;

SELECT TO\_TIMESTAMP\_TZ(‘05’,’TZH’) FROM DUAL;

SELECT TO\_YMINTERVAL (’10-06’) FROM DUAL;

SELECT SYSDATE,SYSDATE+TO\_YMINTERVAL(’10-06’) FROM DUAL;

SELECT SYSTIMETAMP, SYSTIMESTAMP+TO\_DSINTERVAL (’10 10:10:10’)

FROM DUAL;

SELECT BIN\_TO\_NUM(1,1,1) FROM DUAL;

SELECT NUMTOYMINTERVAL(11,’YEAR’) FROM DUAL;

SELECT SYSDATE,SYSDATE+NUMTOYMINTERVAL(11,’YEAR’) FROM DUAL;

SELECT NUMTOYMINTERVAL(11,’MONTH’) FROM DUAL;

SELECT NUMTOYMINTERVAL(9999,’YEAR’) FROM DUAL;

SELECT NUMTODSIINTERVAL(10,’DAY’) FROM DUAL;

SELECT NUMTODSINTERVAL(10,’HOUR’) FROM DUAL;

SELECT NUMTODSINTERVAL(10,’SECONDS’) FROM DUAL;

SELECT ORA\_ROWSCN FROM DUAL;

SELECT SCN\_TO\_TIMESTAMP(ORA-ROWSCN) FROM DUAL;

SELECT TIMESTAMP\_TO\_SCN(SCN\_TO\_TIMESTAMP(ORA\_ROWSCN))FROM DUAL;

SELECT TO\_NUMBER (‘99’) FROM DUAL;

SELECT SYSDATE+TO\_NUMBER(‘10’) FROM DUAL;

* **NUMBER FORAMTS:**

$

9 DIGIT REPRESENTATION

0

**.** SPECIFIES THE DECIMAL

**,**

PR ENCLOSE THE –VE VALUES IN ANGLE BRACKETS

MI REPRESENTS THE –VE SIGN

S SIGN

L LOCAL CURRENT SYMBOL

B BLANK SPACE

C CURRENCY CODE

D DECIMAL POINT

EEEE SPECIFIES THE EXPONENTIAL

G GROUPING

U

V

X

N

**EG:**SELECT SAL,TO\_CHAR(SAL,’$9999’) FROM EMP;

SELECT SAL,TO\_CHAR(SAL,’$9999.99’) FROM EMP;

SELECT SAL,TO\_CHAR(SAL,’$9999D99’) FROM EMP;

SELECT TO\_CHAR(-10,’99PR’) FROM DUAL;

SELECT TO\_CHAR(-10,’99MI’) FROM DUAL;

SELECT TO\_CHAR(10,’99U9’) FROM DUAL;

SELECT TO\_CHAR(10,’L99’) FROM DUAL;

SELECT TO\_CHAR(10,’C99’) FROM DUAL;

* **GENRAL FUNCTIONS**
* GREATEST
* LEAST
* USER
* UID
* DECODE
* CASE
* NVL
* NVL2
* NULLIF
* COALESCE
* **GREATEST**

SYS\_CONNECT\_BY\_PATH (HIERARCHIAL FUNCTION);

**EG**:SELECT GREATEST (1, 2, 3), LEAST (1, 2, 3) FROM DUAL;

SELECT GREATEST (‘A’,’B’,’C’), LEASTA (‘A’,’B’,’C’) FROM DUAL;

SELECT GREATEST (SYSDATE, HIREDATE) FROM EMP;

* **NVL:** We user NVL to handle null values.

**SYN**: NVL (EXP1, EXP2); IF EXP1 IS NULL RETURNS EXP2 ELSE EXP1;

SELECT SAL,COMM,SAL+COMM FROM EMP;

SELECT SAL,COMM,SAL+NVL(COMM,0) FROM EMP;

* **NVL2:** ADVANCE OF NVL

**SYN:** NVL2 (EXP1, EXP2, EXP3);

**SYN:** IF EXP1 IS NULL RETURNS EXP3 ELSE EXP2;

**EG**: SELECT NVL2 (COMM, SAL+COMM, SAL) FROM EMP;

* **NULLIF:**

**SYN**: NULLIF (EXP1,EXP2); IF EXP1=EXP2 THEN RETURNS NULL ELSE EXP1;

**EG:**SELECT NULLIF(1,1),NULLIF(1,2) FROM DUAL;

* **COALESE:** Returns first not null value.

EG: SELECT COALESE (1, 2, NULL, 3, NULL,4) FROM DUAL;

**GROUP FUNCTIONS OR AGGREGATE FUNCTIONS OR MULTIPLE**

**ROW FUNCTIONS**:They execute only once for list of values or group of values

1. **SUM**

**SYN**:SUM ([ALL|DISTINCT] VALUE|EXP|COLUMN);

**2) AVG**

**SYN:** AVG ([ALL|DISTINCT] VALUE|EXP|COLUMN);

1. **MIN**

**SYN**: MIN (([ALL|DISTINCT] VALUE|EXP|COLUMN);

1. **MAX**

**SYN**: MAX ([ALL|DISTINCT] VALUE|EXP|COLUMN);

1. **COUNT:**

**SYN**: COUNT (\*|[ALL|DISTINCT] VALUE|EXP|COLUMN);

1. **STDDEV**
2. **VARIANCE**

**EG:**SELECT SUM(SAL),AVG(SAL),MIN(SAL),MAX(SAL) FROM EMP;

SELECT MIN(HIREDATE),MAX(HIREDATE) FROM EMP;

SELECT MIN(ENAME),MAX(ENAME) FROM EMP;

SELECT STDDEV(SAL),VARIANCE(SAL) FROM EMP;

**COUNT (\*):**Count Counts the records.it also consider the null values whereas count column ignore the null values

**EG:** SELECT COUNT (\*) FROM EMP;

SELECT COUNT (COMM) FROM EMP;

**SOME MORE EXAMPLES**

SELECT COUNT(0) FROM EMP;

SELECT COUNT(NULL) FROM EMP;

SELECT SUM(0) FROM EMP;

SELECT SUM(1) FROM EMP;

NOTE: A concept of null doesn’t work for a row as a whole

Count never returns null.

**CLAUSES**

* WHERE
* CONNECT BY
* START WITH
* GROUP BY
* HAVING
* ORDER BY
* **WHERE**: We use where clause to filter the table records.

This phenomena is called selection.

**EG:** SELECT EMPNO, ENAME FROM EMP WHERE DEPTNO=10;

SELECT \* FROM EMP WHERE ENAME=’KING’;

SELECT \* FROM EMP WHERE 1=1;

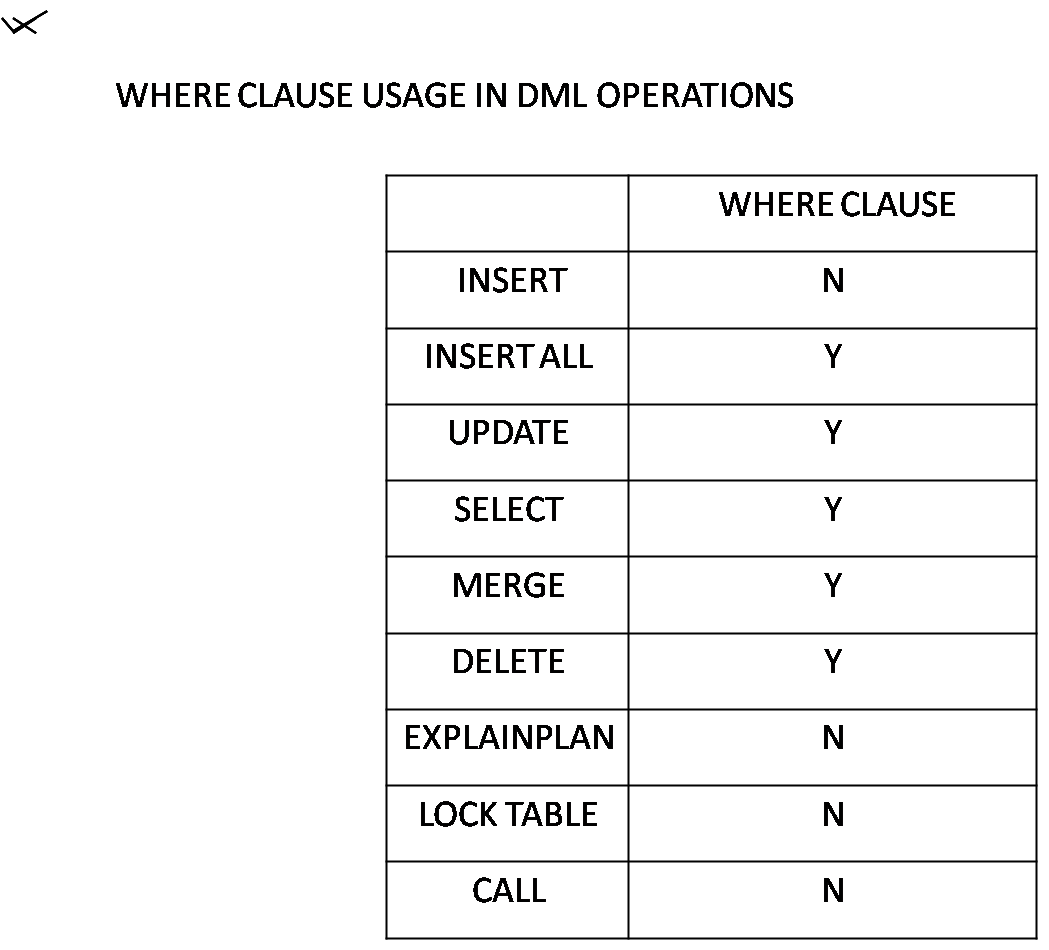
SELECT \* FROM EMP WHERE LENGTH (ENAME)=4;

SELECT \* FROM EMP WHERE 10=DEPTNO;

SELECT \* FROM EMP WHERE DEPTNO=DEPTNO;

SELECT \* FROM EMP WHERE ENAME LIKE ‘A%’;

* In where clause we can provide the conditions
* In where clause we can provide the ‘n’ no of conditions
* Separating with logical operator (and, or)
* Where clause won’t allow column aliases and group functions and lob columns.



* **GROUP BY CLAUSE:**
  + It groups the same kind of data into segments
  + All the select list normal columns, single row functions must be in group by clause but reverse is not so.

**EG:**

SELECT C1,C2 FROM EMP GROUP BY C1,C2;

SELECT C1 FROM EMP GROUP BY C1,C2;

SELECT C1 FROM EMP GROUP BY C1,C2(WONT ALLOWS);

* In the presence of group by clause group functions are forced to execute for each of the grouping sets

**EG:**SELECT DEPTNO,COUNT(\*) FROM EMP GROUP BY DEPTNO;

SELECT DEPTNO,COUNT(\*) ,SUM(SAL),MAX(SAL),MIN(SAL) FROM EMP

GROUP BY DEPTNO;

SELECT TO\_CHAR(HIREDATE,’YY’),COUNT(\*) FROM EMP

GROUP BY TO\_CHAR (HIREDATE,’YY’);

SELECT JOB,COUNT(\*) FROM EMP GROUP BY JOB;

SELECT DEPTNO,JOB,COUNT(\*) FROM EMP GROUP BY DEPTNO,JOB;

SELECT 1,JOB FROM EMP GROUP BY JOB;

SELECT 1,JOB FROM EMP GROUP BY JOB;

SELECT HIREDATE,COUNT(\*) FROM EMP GROUP BY HIREDATE;

SELECT DEPTNO||SAL FROM EMP GROUP BY DEPTNO,SAL;

SELECT DEPTNO||SAL FROM EMP GROUP BY DEPTNO||SAL;

**NOTE**: Without group by clause we can’t nest group functions.

Maximum we can nest two group functions in one another.

SELECT MIN(SAL) FROM EMP GROUP BY DEPTNO;

SELECT MAX(MIN(SAL)) FROM EMP GROUP BY DEPTNO;

* **HAVING CLAUSE:**
* It filters the group by data.
* Generally we user having clause to provide group function condition.
* Normally we user having clause along with group by clause so to

get meaning full data.

* We can also use the having clause very individually but it is not preferable.
* It won’t allow column aliases and analytical functions.
* We can also provide normal columns conditions in having clause but we have to see that all the normal columns in having clause must be in group by clause and select list.

SELECT SAL FROM EMP GROUP BY SAL HAVING SAL>1000;

SELECT SAL FROM EMP GROUP BY SAL HAVING DEPTNO=10;(\*);

SELECT SAL FROM EMP GROUP BY SAL,DEPTNO HAVING DPETNO=10;

SELECT COUNT(\*) FROM EMP GROUP BY DEPTNO HAVING

MAX(COUNT(\*))>1;

* We can user the having clause very individually as shown in following examples

SELECT COUNT(\*) FROM EMP HAVING COUNT(\*)>10;

* **ORDER BY CLAUSE:**
* It displays the table data in one proper order that is either ascending or descending.
* Order by clause comes last in the list of clauses.
* Order by clause allows column aliases and analytical functions unlike other clauses.

**SYN:** SELECT ….FROM … ORDER BY COLUMNS|EXP|FUNCTIONS|COLUMN

POSITION [ASC|DESC] [NULLS FIRST|NULLS LAST];

**EG:**SELECT \* FROM EMP ORDER BY DEPTNO;

SELECT \* FROM EMP ORDER BY DEPTNO DESC;

SELECT \* FROM EMP ORDER BY DEPTNO ASC;(DEFAULT);

SELECT \* FROM EMP ORDER BY ENAME(ASCII)

SELECT \* FROM EMP ORDER BY HIREDATE;

SELECT \* FROM EMP ORDER BY DEPTNO,SAL;

SELECT \* FROM EMP ORDER BY DEPTNO ASC, SAL DESC;

SELECT \* FROM EMP ORDER BY 6

(COLUMN POSITION IN SELECT LIST);

SELECT \* FROM EMP ORDER BY 6,2;

SELECT \* FROM EMP ORDER BY COMM;

**NOTE: NULLS ARE CONSIDERD AS HIGHER VALUES**

SELECT \* FROM EMP ORDER BY COMM NULLS FIRST;

SELECT \* FROM EMP ORDER BY COMM DESC NULLS LAST;

SELECT SAL, JOB FROM EMP ORDER BY 2;

SELECT SAL SALARY FROM EMP ORDER BY SALARY; (ALIASE);

**INTIGRITY CONSTRAINTS**

**DEFINITION**: They are the rules or restrictions, which are imposed on a table column to restrict the invalid data.

CONSTRAINT KEYS:

* PRIMARY KEY(UNIQUE+NOT NULL)
* UNIQUE
* NOT NULL
* CHECK
* FOREIGN KEY
* REF
* DEFAULT(KEY WORD)
* **PRIMARY KEY:**
* It acts has both UNIQUE+NOT null
* There must be only one primary key for an entire table
* Table which are having primary key constraint are called as

Master tables (0r) parent tables

* A primary key can contain n number of columns;such keys are called as composite keys
* Implicitly an unique index get defined on a primary key column
* A primary key can hold maximum of 32 columns
* **UNIQUE:**
* Won’t allow duplicate values, for even unique constraints also an unique index get defined.
* It allows null values.
* **NOT NULL:**Won’t allow null values, but allows duplicate values.
* **DEFAULT:**To provide default values
* **FOREIGN KEY:**
* It is a referential constraint, which refers to the primary key or unique key.
* It allows nulls, duplicates.
* **CHECK:**
* We use check constraints to fulfill the user requirements.
* To enforce business rules.

**WE CAN PROVIDE CONSTRAINTS IN TWO WAYS**

* WHILE DEFING TABLES(CREATE)
* ON EXISTING TABLE (ALTER)
* WHILE DEFINING TABLES:
* COLUMN LEVEL(INLINE)
* TABLE LEVEL(OUT OF LINE)

**WE CAN PROVIDE NAMES TO CONSTRAINTS**.

* USER DEFINED
* SYSTEM DEFINED

By taking all the above things into consideration, we can provide the

constraints in following wayss

* SYSTEM DEFINED COLUMN LEVEL
* USER DEFINED COLUMN LEVEL
* SYSTEM DEFINED TABLE LEVEL
* USER DEFINED TABLE LEVEL
* **SYSTEM DEFINED COLUM LEVEL:**

CREATE TABLE CONTAB(SNO NUMBER(5) PRIMARY KEY,SNAME VARCHAR2(10) UNIQUE,LOC VARCHAR2(10) NOT NULL,BAL NUMBER(5) CHECK(BAL>1000),BANK VARCHAR2(10) DEFAULT ‘SBI’);

CREATE TABLE CONTAB1 (SNO NUMBER (5) REFERENCES CONTAB (SNO),

LOC VARCHAR2 (10));

* **USER DEFINED COLUMN LEVEL:**

CREATE TABLE CONTAB2 (SNO NUMBER (5) CONSTRAINT PY PRIMARY KEY, SNAMEVARCHAR2(10) CONSTRAINT UN UNIQUE, LOC VARCHAR2(10) CONSTRAINT NN NOT NULL, BAL NUMBER(5) CONSTRAINT CHK CHECK(BAL>1000));

CREATE TABLE CONTAB3(SNO NUMBER(5) CONSTRAINT FY REFERENCES

CONTAB1(SNO),LOC VARCHAR2(10));

* **SYSTEM DEFINED TABLE LEVEL:**

**NOTE:**Not possible to provide ‘NOT NULL’ constraints in

table level

CREATE TABLE CONTAB4 (SNO NUMBER(5),SNAME VARCHAR2(10),LOC

VARCHAR2 (10),BAL NUMBER(5),PRIMARY KEY(SNO),UNIQUE(SNAME),

CHECK (BAL BETWEEN 1000 AND 2000));

CREATE TABLE CONTAB5(SNO NUMBER(5),LOC VARCHAR2(10),

FOREIGN KEY(SNO) REFERENCES CONTAB4(SNO));

* **USER DEFINED TABLE LEVEL:**

CREATE TABLE CONTAB6(SNO NUMBER(5),SNAME VARCHAR2(10),

LOC VARCHAR2(10),BAL NUMBER(5),

CONSTRAINT PY1 PRIMARY KEY(SNO),

CONSTRAINT UN1 UNIQUE(SNAME),

CONSTRAINT CHK1 CHECK(SNAME IN (‘A’,’B’));

CREATE TABLE CONTAB7(SNO NUMBER(5),LOC VARCHAR2(10),

CONSTRAINT FY1 FOREIGN KEY(SNO) REFERENCES CONTAB6(SNO));

* **MIXED METHOD:**

CREATE TABLE CONTAB8(SNO NUMBER(5) PRIMARY KEY,SNAME

VARCHAR2(10),LOC VARCHAR2(10),BAL NUMBER(5),CHECK(BAL>2000));

**ADV OF TABLE LEVEL:**Column comparison is possible unlike column level;

**EG**: CREATE TABLE TL(X NUMBER (5),Y NUMBER(5),CHECK(X>Y));

* **COMPOSITE PRIMARY KEY:**When a primary key holds more than one column, those are said to be composite primary keys.

**EG:** CREATE TABLE CONTABX(X NUMBER(5),Y VARCHAR2(10),LOC VARCHAR2(10),CONSTRAINT CPY PRIMARY KEY(X,Y));

CREATE TABLE CONTABY(S NUMBER(5),Y VARCHAR2(10),

CONSTRAINT CFY FOREIGN KEY(X, Y) REFERENCES CONTABX(X,Y));

**NOTE:** When you refer a foreign key to composite primary key,

Foreign key also has to contain that many no of columns.

* **SELF KEY**:Referring a foreign key with in a same table primary key column.

**EG:** CREATE TABLE SELFTAB (SNO NUMBER (5) PRIMARY KEY, LOCVARCHAR2 (10), IDNUMBER (5) REFERENCES SELFTAB (SNO));

**CASCADE CONSTRAINT:**

* Dropping master table directly is not possible when it is having child tables. We have to drop child tables before dropping master tables, but by using ‘cascade constraint’.
* It is possible.

**EG:** DROP TABLE CONTAB; (ERROR);

DROP TABLE CONTAB CASCADE CONSTRAINT;

ON DELETE CASCADE

ON DELETE SET NULL

ON DELETE RESTICT (DEFAULT)

**ON DELETE CASCADE:**

* It is not possible to delete the parent records when they are having dependent child records very directly, but by using on delete cascade it is possible.
* We have to mention on delete cascade while defining foreign key.
* It is not possible to provide on existing foreign key.

**EG**: CREATE TABLE SAM(SNO NUMBER(5) PRIMARY KEY);

INSERT INTO SAM VALUES(10);

INSERT INTO SAM VALUES(20);

INSERT INTO SAM VALUES(30);

CREATE TABLE SAMC(SNO NUMBER(5) REFERENCES SAM(SNO) ON

DELETE CASCADE);

INSERT INTO SAMC VALUES(10);

INSERT INTO SAMC VALUES(20);

DELETE FROM SAM WHERE SNO=10;

(SAMC TABLE 10 VALUES ALSO GET DELETED);

**ON DELETE SET NULL:**

* Instead of deleting dependent records ,on delete set null sets the null value.
* DEFINING CONSTRAINTS ON EXISTING TABLE:

**GENERIC SYNTAX:**

ALTER TABLE TABLENAME ADD|MODIFY|DISABLE|ENABLE(VALIDATE|INVALIDATE)|ENFORCE|DROP

|RENAME CONSTRAINT CONSTRAINTNAME;

**EG:**

CREATE TABLE CON21(SNO NUMBER(5),SNAME VARCHAR2(10),BAL NUMBER(5),LOC VARCHAR2(10));

**ADD:**

ALTER TABLE CON21 ADD CONSTRAINT KP10 PRIMARY KEY(SNO);

ALTER TABLE CON21 ADD CONSTRAINT CHK10 CHECK(BAL>100)

ADD UNIQUE(LOC);

NOTE: NOT POSSIBLE TO ADD ‘NOT NULL ‘CONSTRAINT;

**MODIFY:**We will modify not null to null and vice versa.

**SYN**: ALTER TABLE TABLENAME MODIFY COLUMNNAME NOT NULL|NULL;

**EG:**ALTER TABLE CON21 MODIFY SNAME NOT NULL;

INSERT INTO CON21 VALUES (10, NULL, 8000,’HYD’); ERROR;

ALTER TABLE CON21 MODIFY SNAME NULL;

INSERT INTO CON21 VALUES (10, NULL, 8000,’HYD’); ROW CREATED;

ALTER TABLE CON21 MODIFY SNAME NOT NULL; (ERROR, NULL FOUND);

**DISABLE:**

**EG**: ALTER TABLE CON21 DISABLE CONSTRAINT KP10;

INSERT INTO CON21 VALUES (10, NULL, 4000,’BAN’);

INSERT INTO CON21 VALUES (10, NULL, 4000,’C’); DUPLICATE VALUES;

SELECT \* FROM CON21;

**ENABLE:**

**NOTE**: BEFORE ENABLING A CONSTRAINT OR IMPOSING CONSTRAINT ON A COLUMN, COLUM DATA MUST NOT VOILATE THE RULES;

**EG**: ALTER TABLE CON21 ENABLE CONSTRAINT KP10;

**DROP:**

**EG**: ALTER TABLE CON21 DROP CONSTRAINT KP10;

**NOTE:**WHEN YOU DROP TABLE AUTOMATICAL CONSTRAINTS GET DROPED;

**RENAME:**

**SYN**: ALTER TABLE TABLENAME RENAME CONSTRAINT CONSTRAINTNAME TO

NEWNAME;

**Note:** CHECK CONSTRAINT WON’T ALLOW FOLLOWING THINGS.

* SUBQUERY
* SCALAR SUBQUERY
* NEXTVAL
* CURRVAL
* LEVEL
* ROWNUM
* SYSDATE
* USER
* UID
* USERENV
* DBTIMEZONE
* SESSION TIMEZONE
* TIMESTAMP WITH TIMEZONE

**JOINS**

* Join is a query which is used to retrieve data from more than one table by providing join condition.
* We provide the join condition in “where clause” and even in “from clause”.
* Join condition columns must be compatible data types or same data types.

**Oracle Traditional or Native joins: (prior to 9i)**

* Inner join:
  + Equi join
  + Non-Equi join
* Self-join
* Outer join
  + Left outer join
  + Right outer join
  + Full outer join

**9i joins:**

* + Cross Join
  + Natural join
  + Join on
  + Join using
  + Outer join
    - * Left outer join
      * Right outer join
      * Full outer join

Q: To find out the table which we have in database

A: desc user\_object

desc user\_table

**Cartesian Product:**

**NOTE:** In the absence of join condition, if you combine more than one table than the result will be **‘Cartesian Product’** results. Which means each record of one table will combine with multiple records of another table.

**SQL**> select \* from emp,dept;

* **Equi joins:** In this join we will provide the join condition with equal to (=) operator.

**SQL**> select \* from emp, dept where emp.deptno=dept.detpno;

**SQL**> select \* from emp e, dept d where e.detpno=d.detpno;

**Note:** We can provide ‘n’ no.of join conditions by separated by and (or) or.

* **Non-equi join:** In non-equi join we provide the join condition between the columns with other than equal to (=) operator.

**SQL>** select \* from emp e, dept d where e.deptno!=d.deptno;

**SQL**> select \* from emp e, dept d where e.detpno<=d.deptno;

**Note:** Inner join will skip null record values.

* **Self-join:** Join in the same table columns is called self join.

**SQL**> select e.empno, e.ename, m.empno, m.ename from emp e , emp m where e.mgr=m.emp no;

Emp e Emp m

|  |  |  |
| --- | --- | --- |
| Empno | Ename | Mgr |
| 1 | X | 2 |
| 2 | Y | 3 |
| 3 | Z | 4 |
| 4 | A |  |
| 5 | B | 3 |

|  |  |  |
| --- | --- | --- |
| Empno | Ename | Mgr |
| 1 | X | 2 |
| 2 | Y | 3 |
| 3 | Z | 4 |
| 4 | A |  |
| 5 | B | 3 |

**Output:** e.empno e.ename m.empno m.ename

1 x 2 y

2 y 3 z

3 z 4 a

5 b 3 z

**NOTE:** here emp table is splitting as 2 emp tables i.e. copy of emp.

**SQL**> select e.empno employno, e.ename employname, m.empno managerno, m.ename managername from emp e , emp m where e.mgr = e.empno;

**NOTE :** we have to split one table to ‘n’ no.of tables(virtual tables) physically the table not available but logically it takes values from virtual tables.

Virtual tabler ---> no data (logically it takes data).

* **Outer join:-**

Along with matched records further if we want to get additional records from either of the table we use outer joins.

We will perform outer joins with outer join operator (+).

* **Left outer join:**
* In this join we will get full details from the left table and matched record of right table.
* In this join condition (+) symbol should has to be placed at right side of equal operator.
* Right outer join:

**Which is reverse to that of left outer join?**

ltab rtab

|  |
| --- |
| Sno |
| 10 |
| 20 |
| 50 |
| 60 |
| 100 |

|  |
| --- |
| Sno |
| 10 |
| 20 |
| 30 |
| 40 |

**SQL**>select rtab.sno, ltab.sno from ltab, rtab where rtab.sno(+) = ltab.sno;

**Output:**

rtab.sno ltab.sno

10 10

20 20

50

60

100

**SQL**> select rtab.sno, ltab.sno from ltab, rtab where rtab.sno(+) > ltab.sno;

**Output:**

rtab.sno ltab.sno

20 10

30 10

40 10

30 20

40 20

50

60

100

**Left outer join example:**

**SQL**> select rtab.sno, ltab.sno from rtab, ltab where rtab.sno = ltab.sno(+);

Rtab.sno ltab.sno

10 10

20 20

30 null

40 null

* **Full outer join:**

Concept of full outer join is from 9i onwards. But still if we want to

achieve full outer join result even prior to 9i.

**We use following example:**

In full outer join we will combine both left & right outer joins with union operator.

It display the all records from both of the rables.

**SQL**> select \* from rtab,ltab where ltab.sno = rtab.sno(+) **union** select \* from rtab, ltab where ltab.sno(+) = rtab.sno;

**Set Operators :(vertical joins)**

They are used to combine the queries so to get the compound query. They are also called vertical joins. We can join ‘n’ no.of queries with set operators. In set operators individual query results will be combine to get a final result set.

**Compound query**

Select \* from t**set operator**select \* from t1**set operator**select \* from t2;

component query component query (component query)

* In compound query all the component queries must contain same no.of columns with compatible data types. This rule we call it as

‘**Union Combination Condition**’:

* In compound queries order by clause will be allowed at the end.
* Providing order by clause for individual component queries will not be allowed.
* In compound queries result will be displayed with the first component query select list columns.
* Order by clause in compound query allows only the first component query select list columns.
* All the set operators has equal priority ,except union all , all the set operators will sort & suppress duplicate values.

**Set Operators:**

1. Union
2. Union all
3. Intersect
4. Minus
5. Multiset (11g)

In set operators default execution takes place from left to right but we can alter default execution by using parenthesis.

**Union :**

It displays the records from both tables by suppressing the duplicate records and also sort data.

**Note (for restrictions) :** Elimination of duplicate records becomes a problem when we use order by clause for component query.

**Union all :**It displays all the records from both tables regardless of duplicating and it doesn’t sort data.

Note : Union all is more faster than union.

**Intersect :**Display the common records between tables. It also suppresses duplicate values.

**Minus :**We will get records from one table which are not matching with other table. Result won’t get effect or varies. When you change the order of component query except in minus operator.

Sql> select sno from rtab union select sno from ltab;

Sql> select sno from rtab unionall select sno from ltab;

Sql> select sno from rtab intersect select sno from ltab;

Sql> select sno from rtab minus select sno from ltab;

Sql> select sno from rtab minus select sno from ltab union select \* from rtab;

Sql> select 1 from dual union select 2 from dual;

Sql> select sno from rtab union select sno from ltab order by sno;

**Joins (9i) : (ISO/sql 1999)**

1. Cross join
2. Natural (pure nj)
3. Join.....using
4. Join.....on
5. Outer..join
6. Left outer join
7. Right
8. Full

Log join (ISQ/SQL 2003)

1. Partition outer join (log)

9i joins won’t give any guarantee for the performance.

9i syn:

Select \* from t

[cross join t1] |

[natural join t1] |(alternate)

[[inner] join t1 {using (col [col1,....]|on con d1,....)}] |

[{left / right/full } [ outer ] join t1

{ using (col [col1,...] | on (cond1,....)}]

* **Cross joins :**It works as same to that of Cartesian product (or) display the Cartesian result.

**Sql**> select \* from emp cross join dept; (56 records..Cartesian product results)

* **Natural join :**Natural join works between table names.

It works as same to that of equi join (or) with natural join we will achieve equi join result.

* Natural join automatically provides the equi join conditions between the columns of tables whose column names are same.
* In natural join at least one set of columns must exist with same name.
* In natural join column name should be same and it should be same/ compatible data type.

**IMP: When a natural join becomes cross join?**

**Ans)**Natural join becomes cross join when column names are differing.

It’s not possible to qualify the columns which are used up by natural join.

|  |
| --- |
| Rtab |
| Sno |
| 10  20  30  40 |

|  |
| --- |
| Ltab |
| Sno |
| 10  20  30  40  60  100 |

**Sql**>select \* from rtab;

**Sql**>select \* from ltab;

**Sql**>select \* from rtab,ltab where rtab.sno=ltab.sno;

|  |
| --- |
| Sno |
| 10  20 |

|  |
| --- |
| Sno |
| 10  20 |

Snosno

10 10

20 20

Sql>select \* from rtab natural join ltab;

Sql>select sno from rtabnatural join ltab;

Sql>select rtab.sno from rtab natural join ltab;

**Error**:- column used in natural join can’t have qulifier.

Sql>alter table ltab rename column sno to sn;

|  |
| --- |
| Ltab |
| Sn |
| 10  20  30 |

|  |
| --- |
| Rtab |
| Sno |
| 10  20  50  60 |

Sql>select \* from ltab;

Sql>select \* from rtab;

**Sql**>select \* from ltab natural join rtab;

(which is a Cartesian product ) i.e 4\*5=20 records.

|  |  |
| --- | --- |
| Sno | loc |
| 10  20  50  60  100 | A  B  C  X |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
| Sno | loc |
| 50  10  20  30  40 | X  A  B  C |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**Sql**>alter table rtab rename column sno to sn;

**Sql**>select \* from ltab;

**Sql**>select \* from rtab;

**Sql**>select \* from rtab natural join ltab;

Reslt is same to the following example.

**Sql**>select \* from ltab,rtab where ltab.sno=rtab.sno and ltab.loc=rtab.loc;

**Disadvantage:**

In pure natural join it is not possible to specify join condition only for particular columns. When there is more no. of same columns, same sets across the tables.

* **Join using:-**In join using we use using clause to specify the columns.

So we provide equal join condition between mentioned columns.

Using clause columns can’t be qualified in an entire select statement. Using clause column must be there in both of tables.

**Sql**>select \* from ltab join rtab using (sno);

Result: snolocloc

10 A A

20 B B

50 C x

**Sql**>select sno,ltab.loc,rtab.loc from ltab join rtab using (sno);

Result: snolocloc

10 A A

20 B B

50 C x

**Sql**>select sno,loc,loc from ltab join rtab using (ltab.sno);

**Error:** only simple column names allowed here.

**Sql**>select \* from ltab join rtab using (sno,loc);

snolocloc

10 A A

20 B B

**Disadvantages:** In using clause at least one set of column names must exist with same name, if not using clause is not applicable.

**Note:** Join...... using clause will works like natural joins also but natural joins will not work like using clause as same join ......on clause will.

* **Join.....on(cond/.....):**(it is preferable) We use join on to provide condition.

**Sql**>select \* from rtab join ltab on (rtab.sno1=ltab.sno);

|  |  |
| --- | --- |
| Sno | loc |
|  |  |

|  |  |
| --- | --- |
| Sno1 | Loc1 |
|  |  |

**Result:**Sno1 loc1 sno loc

10 A 10 A

20 B 20 B

50 X 50 C

**Sql>**select \* from ratb join ltab on (rtab.loc1=ltab.loc);

**Sql**> select \* from ratb join ltab on(rtab.loc1=ltab.loc and rtab.sno1=ltab.sno);

**Sql**> select \* from ratb join ltab on(rtab.sno1=ltab.sno and rtab.loc1=ltab.loc);

**Note:** Natural join , join on & join using are mutually exclusive.

**Sql**> select \* from ratb join ltab on (rtab.sno=ltab.sno and ltab.sno>rtab.sno);

(Using normal condition)

**Sql**>select \* from rtab joins ltab on .ltab.sno=rtab.sno and ltab.lov=rtab.loc;

(using without parenthesis)

**NOTE:**Join on will works as join ...using and additionally works on conditions.

Join on, using clause will work on alter joins also.

**Outer joins:-**

1) left outer,

2) right outer,

3) full outer.

Sql>select \* from rtab left outer join ltab on (rtab.sno1=ltab.sno and rtab.loc1=ltab.loc);

|  |  |
| --- | --- |
| Sno1 | Loc1 |
| 10  20  50  30  40 | A  B  C  X  D |

|  |  |
| --- | --- |
| Sno | Loc |
| 10  20 | A  B |

|  |  |
| --- | --- |
| Sno | Loc |
| 10  20  50  30  40 | A  B  C  X  D |

|  |  |
| --- | --- |
| Sno | Loc |
| 10  20  50  60  100  --  --  -- | A  B  C  X  --  --  --  -- |

**Sql>** select \* from rtab right outer join ltab on (rtab.sno1=ltab.sno and rtab.loc1=ltab.loc);

|  |  |
| --- | --- |
| Sno1 | Loc1 |
| 10  20 | A  B |

**Sql>** select \* from rtab full outer join ltab on (rtab.sno1=ltab.sno and rtab.loc1=ltab.loc);

|  |  |
| --- | --- |
| Sno1 | Loc1 |
| 10  20  --  --  --  50  40  30 | A  B  --  --  --  X  D  C |

**QUERIES**

Sub queries:

* Simple Sub queries
* Correlated Sub queries
* Single row Sub queries
* Multiple row Sub queries
* Inline Sub queries
* Hierarchical Sub queries
* Flashback Sub queries
* Scalar Sub queries
* Simple Sub queries
* Query with in another Query is called Sub query.
* Always we have to enclosed Sub queries in parenthesis ‘( )’.
* We can nest the Sub queries up to 255.
* In Simple Sub query at first inner Query get Executes based on the inner query value
* Outer query get executes, outer query may contain update or even delete.
* We provide sub queries in where clause, having clause and in from clause.
* In simple sub queries outer query depends on inner query whereas inner query not depends on outer query.

**Examples:**

* Select \* from emp where sal=(select max(sal) from emp);
* Select \* from emp where hiredate=(select min(hiredate) from emp);
* Select \* from emp where sal=(select max(sal) from emp where deptno=10);
* Select \* from emp where sal=(select max(sal) from emp where sal<(select max(sal) from emp));
* Select \* from emp where hiredate=(select min(hiredate) from emp where hiredate<(select max(hiredate) from emp));
* Select \* from emp where hiredate=(select min(hiredate) from emp where hiredate>(select min(hiredate) from emp));
* Select \* from emp where empno in(select mgr from emp); 🡪6 rows selected.
* Select \* from emp where empno not in (select mgr from emp); 🡪no rows selected.
* Select \* from emp where empno not in (select mgr from emp where mgr is notnull);
* Select \* from emp where sal>(select avg(sal) from emp where deptno=10);
* Select \* from emp where sal>(select avg(sal) from emp where deptno=10 and deptno<>10);
* Select \* from emp where sal>=(select max(sal) from emp where sal<(select max(sal) from emp));
* Select \* from emp where sal=(select max(sal) from emp where sal>(select max(sal) from emp));🡪no rows selected.
* Select \* from emp where sal<(select min(sal) from emp where sal>(select min(sal)from emp));
* Select \* from emp where hiredate=(select max(hiredate) from emp where hiredate<(select max(hiredate) from emp));
* Select job from emp where deptno=10 and job not in(select job from emp where deptno in (30,20));
* **Single row Operators:-**

=,>,<,<>,……..,etc.

* **Multiple row operators:-**

In, any/some, all, exit.

**Note:** If a sub query returns more than one value, we have to

make use of multiple row operators.

* Select sal from emp where sal in(select sal from emp); 🡪14 rows selected.
* Select sal from emp where sal=(select sal from emp); 🡪error.
* **Correlated sub queries:- (synchronization)**
* In Correlated sbu queries at first outer query get executes and pass a value into inner query by making using of outer query value. Inner query get executes and return a value to the outer query condition, finally based on inner query returned value outer query display the result. This phenomena or property is said to be correlation.
* In Correlation at first inner Query depends on the outer query and then outer query depending on inner query.
* Inner query executes once for each of the outer query record.

**Examples:**

* Select \* from emp e where e.sal=(select max(sal) from emp where e.deptno=deptno);

300 10

500 20

* Select \* from emp where sal in(select max(sal) from emp group by deptno);

10 300

20 500

Empemp

|  |  |  |  |
| --- | --- | --- | --- |
| empno | ename | sal | deptno |
| 1  2  3  4  5  6 | x  y  z  a  b  c | 100  200  300  500  400  300 | 10  10  10  20  20  20 |

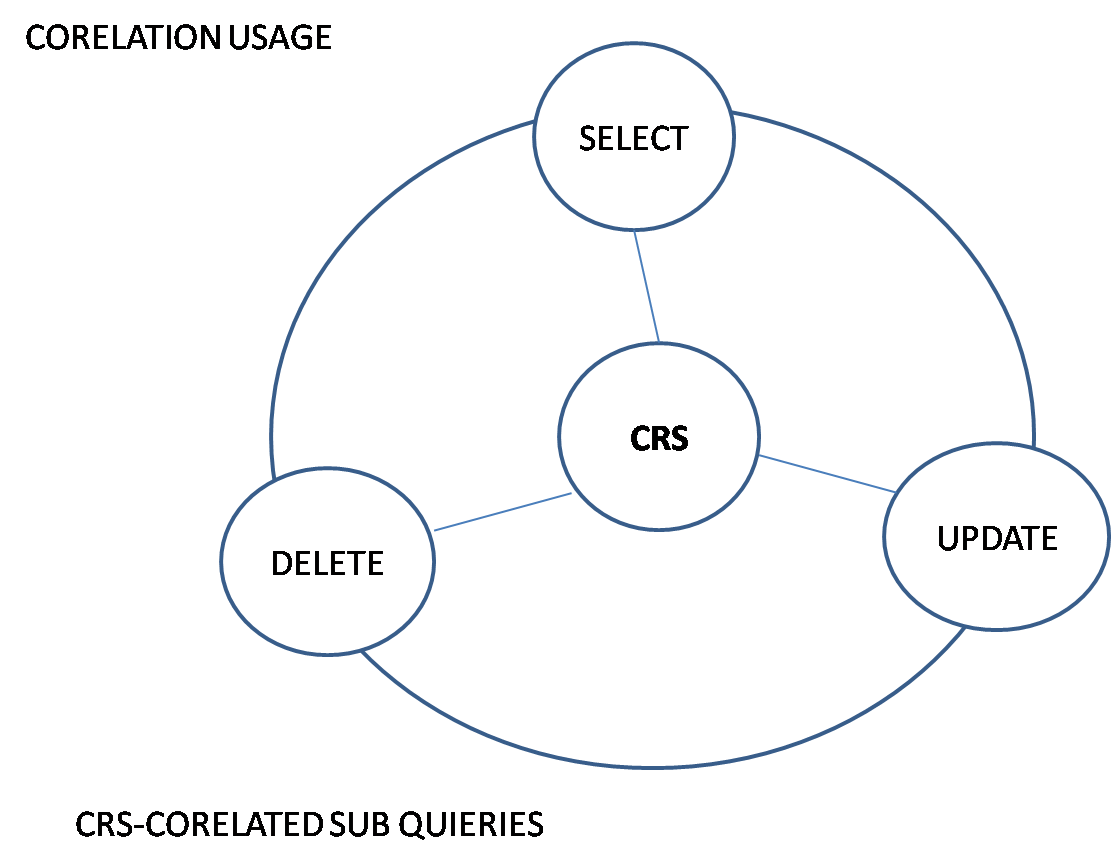
|  |  |  |  |
| --- | --- | --- | --- |
| deptno | Sal | ename | empno |
| 10  10  10  20  20  20 | 100  200  300  500  400  300 | x  y  z  a  b  c | 1  2  3  4  5  6 |

* Select \* from emp e where sal >(select sal from emp where e.mgr=empno);

|  |  |  |  |
| --- | --- | --- | --- |
| empno | ename | sal | Mgr |
| 1  2  3  4 | x  y  a  z | 300  100  500  100 | 2  4  4  1 |

|  |  |  |  |
| --- | --- | --- | --- |
| mpno | ename | sal | mgr |
| 1  2  3  4 | x  y  a  z | 300  100  500  100 | 2  4  4  1 |

* Select \* from emp e where deptno=(select deptno from emp where e.mgr=empno);
* Select \* from emp e where sal in (select max(sal) from emp group by deptno union select min(sal) from emp group by deptno);**🡪7 rows selected.**



**Queries:**

* **Display the employee numbers and names working as clerks and earning highest salary among clerk?**

Select \* from emp where job=’CLERKS’ and sal=(select max(sal  
) from emp where job=’CLERKS’;

Select \* from emp where job=’SALESMAN’ and sal>(select max(sal) from emp where job=’CLERKS’;

* **Display the name of clerks who earn salary more than that of James and lesser than that of the Scott?**

Select \* from emp where job=’CLERK’ and sal>(select sal from emp where ename=’JAMES’) and sal<(select sal from emp where ename=’SCOTT’);

* **Display the names of the employees who earn highest salary in the respective job groups?**

Select \* from emp e where sal=(select max(sal) from emp where e.job=job);

* **Display the employee names who are working in Chicago?**

Select \* from emp where deptno=(select deptno from dept where loc=’CHICAGO’);

Select \* from emp where mgr in(select empno from emp where ename=’JONES’);

* **Delete these employees who joined the company before 31st dec ’82 while their location is New York or Chicago?**
* **Find out the top five earners of the company?**

Select \* from emp where 1= (select count(\*) from emp where e.sal<=sal);

Emp e Emp

|  |  |
| --- | --- |
|  | sal |
|  | 400  300  200  500  100 |

|  |  |
| --- | --- |
| sal |  |
| 400  300  200  500  100 |  |

**Inline sub queries (INLINE VIEWS OR TOP-N QUERIES)**

* If you write a query in from clause or Instead of table name or in front of from clause such queries are said to be ‘Inline queries’.
* We can provide unlimited number of queries in from clause.
* From clause query provides the data to the outer query as same to that of table data.

Once outer query get executes inner query data evaporates or vanishes on the fly.

**Example:**

* Select \* from (select \* from table name);
* Select \* from (select \* from emp);
* Select \* from (select \* from emp) where deptno=10;
* Select \* from (select \* from emp where deptno in(10,20))

where deptno=10;

* Select \* from (select \* from emp) where deptno in(10,20)

where deptno=10;

**Queries:**

* Write a question to display first five records, last five records , random records, nth record, range of records, last two records, last but one record, first and last record, except first five records, except last five records, except random records, except nth record, except range of records, except last two records, except last but one record, except first and last record, salary wise first five records, salary wise last five records, salary wise random wise records, salary wise nth record, salary wise range of records, salary wise last two records, salary wise last but one record, salary wise first and last record?
* To display unique records, to display duplicate records?
* To delete first records, last five records?
* Select \* from emp where hiredate>to\_date(’01-feb-80’,dd-mon-yy)+30;
* Select \* from emp where ename>’KING’;

**ROW NUMBER:**

* Select \* from emp where rownum=1;🡪1 record. N and N 🡪N
* Select \* from emp where rownum>1;🡪no rows. N and F 🡪F
* Select \* from emp where rownum<5;🡪5 rows. N and T 🡪N
* Select \* from emp where rownum!=5;🡪4 rows. N or N 🡪N
* Select \* from emp where rownum=5;🡪no rows. N or F 🡪N
* Select rownum,emp.\* from emp;🡪14 rows. N or T 🡪T
* Select \* from emp where rownum<=5;🡪5 rows. NULL is NULL 🡪T
* Select \* from emp where rownum>5;🡪no rows. NULL = NULL🡪F
* Select \* from emp where rownum in(1,2,3);🡪3 rows.
* Select \* from emp where rownum in (3,1,2);🡪3 rows.
* Select \* from emp where rownum in (5,4,1);🡪3 rows.
* Select \* from emp where rownum not in (5,4,1);🡪no rows.
* Select \* from emp where rownum +3<-1;🡪no rows.
* Select \* from emp where decode(rownum,rownum,1)>=0;🡪14 rows.
* Select \* from (select rownum r,emp.\* from emp) where r=(select count(\*) from emp);🡪(n-1) rows.
* Select \* from (select rownum r,emp.\* from emp) where r between 1 and 6;
* Select \* from (select rownum r,emp.\* from emp) where r in (1,(select count(\*) from emp));🡪1,14 rows.
* Select \* from (select rownum r,emp.\* from emp) where r=2;🡪1 row.
* Select \* from (select rownum r,emp.\* from emp) where r not in(1,(select count(\*) from emp));🡪12 rows.
* Select \* from (select rownum r,emp.\* from emp) where rownum=2;

🡪no rows.

* Select \* from (select rownum r,emp.\* from emp) where r>5;🡪 5 rows.
* Select \* from (select rownum r,emp.\* from emp) where r=&n;
* Select \* from (select rownum r,emp.\* from emp) where r!=&n;
* Select \* from (select rownum r,emp.\* from emp) where r=(select count(\*) from emp);🡪1 row.
* Select \* from (select rownum r,emp.\* from emp) where r=(select count(\*)-1 from emp);🡪1 row.
* Select \* from (select rownum r,emp.\* from emp) where r not between 6 and 10;🡪9 rows.
* Select \* from (select rownum r,emp.\* from emp) where r in(1,14);🡪2 rows.

**INSERT TOPIC:** Ampersand (&) is used to prompt the user to enter

the value. It takes values from user.

**Example:**

* Select &col from emp;
* Select \* from &n;
* Enter value for n:emp
* We get 14 rows.
* How come the SQL \* PLUS allows to provide our own values as a (by using ampersand).
* By using double (&&) ampersand we can provide a value only once for N of occurrences throw out the session.
* All of the SQL elements can replace with &.

**Examples:**

* Select \* from emp where &col;🡪enter value for col:sal>3000.
* Select &&col from emp order by &col;🡪enter value for col:deptno.

**Define:**It is used to list out the variables and also to define the variables

this defined variables last for the session.

**Example:**

* Select \* from (&n);

Enter value for n: select \* from emp

* Select &n;

Enter value for n:\* from emp

* Select \* from emp where deptno=&x;

Enter value for x:10

* MULTI ROW OPERATORS: - IN, SOME/ANY, ALL,EXISTS.
* IN: Search the list of values.
* ANY: In any the given value become true any one of the value.
* ALL: The given value has to be the true with all of the listed values.
* ANY & ALL: It will always come with relational operators. It won’t exist individually.

**Example:**

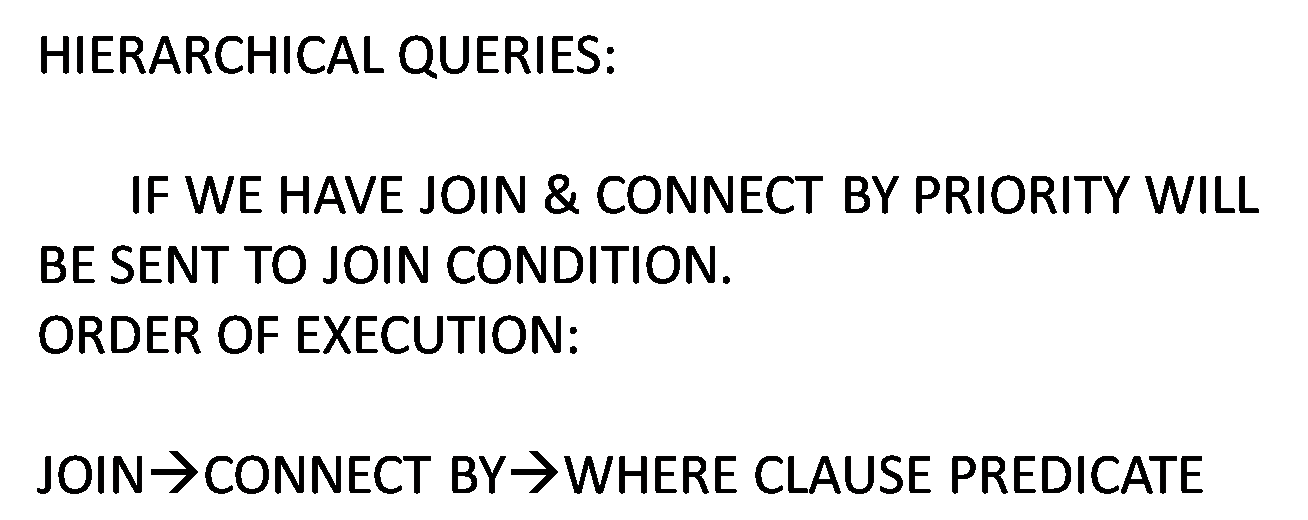
* Select \* from emp where deptno=(10,20,30);🡪error.
* Select \* from emp where deptno in(10,20,30);
* Select \* from emp where deptno >any(10,20);🡪11 rows.
* Select \* from emp where deptno !=any(10,20);🡪14 rows.
* Select \* from emp where deptno =any(10,20);
* Select \* from emp where deptno <any(10,20);
* Select \* from emp where deptno !=all(10,20);
* Select \* from emp where deptno >all(10,20);
* Select \* from emp where deptno =all(10,20);
* Select \* from emp where deptno =all(10);
* Select \* from emp where deptno =all(10,null);
* Select \* from emp where deptno >=all(10,20);
* Select \* from emp where deptno >=all(20,20);🡪11 rows.
* Select \* from emp where deptno>=all((20,30);

**EXISTS:**If records are found returns to else false.

* We use exist for record existence.

**Examples:**

* Select \* from emp where deptno=10 and exists (select \* from emp where deptno=20 and job=’MANAGER’);🡪3 rows.
* Select \* from emp where deptno=10 and not exists (select \* from emp where deptno=20 and job=’MANAGER’);
* HIERARCHICAL SUB QUERIES:



* We use hierarchical sub queries to display data in hierarchical.

1) Start with

2) Connect by

3) Prior

4) Level

5) Sys\_connect\_by\_path

6) Connect\_by\_iscycle

7) Connect\_by\_isleaf 10g

8) Connect\_by\_root

9) Sibillings 9i

1) Start with:

* Start with specifies root record, in the absence of start with each and every record will be treated as a root record.
* To the root record level provides 1 and for the subsequent child records provides 2,3,4….and so on.

2) Connect by:

* This clause specifies relation between parent and child records.

3) Prior:

* Represents the prior record with respect to current record.

4) Level:

* Level is pseudo column which provides number values to that root subsequent child records. It supports maximum of 255 values.

5) Sys\_connect\_by\_path:

* It is a hierarchical function. It results the path from root to current node.

6) Connect\_by\_isleaf:

* To the leaf record provides 1,the other provides 0.

Root/parent

**Examples:**

* Select level,empno,ename,prior empno,prior ename from emp

start with mgr is null connect by prior empno=mgr;

* Select level,max(sal) from emp where level <=3 connect by prior sal>sal;🡪error.
* Select level,max(sal) from emp where level <=3 connect by prior sal>sal group by level;

1. 400
2. 300
3. 200

* Select level,min(sal) from emp where level <=3 connect by prior sal>sal group by level;

1. 100
2. 100
3. 100

|  |
| --- |
| sal |
| 300  200  400  100 |

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| 400  300  200  100 | 300  200  100  200  100  100 | 200  100  100  100 | 100 |

* Select level,max(sal) from emp where level<=3,connect by prior sal<sal group by level;

Q) **Write a query to display all the managers to the smith?**

* Select level,empno,ename from emp

start with ename=’SMITH’

connect by prior mgr=empno;

* Select level,empno,ename,prior ename from emp

start with ename=’SMITH’

connect by prior mgr=emp[no;

* Select level,empno,ename,prior ename from emp

start with ename=’JONES’

connect by prior empno=mgr;

* Select level,empno,ename,prior ename,sys\_connect\_by\_path(ename,’>’) path from emp start with ename=’JONES’

connect by prior empno=mgr;

* Select level,empno,ename,prior ename,sys\_connect\_by\_path(ename,’>’) path connect\_by\_isleaf from emp

start with ename=’JONES’

connect by prior empno=mgr;

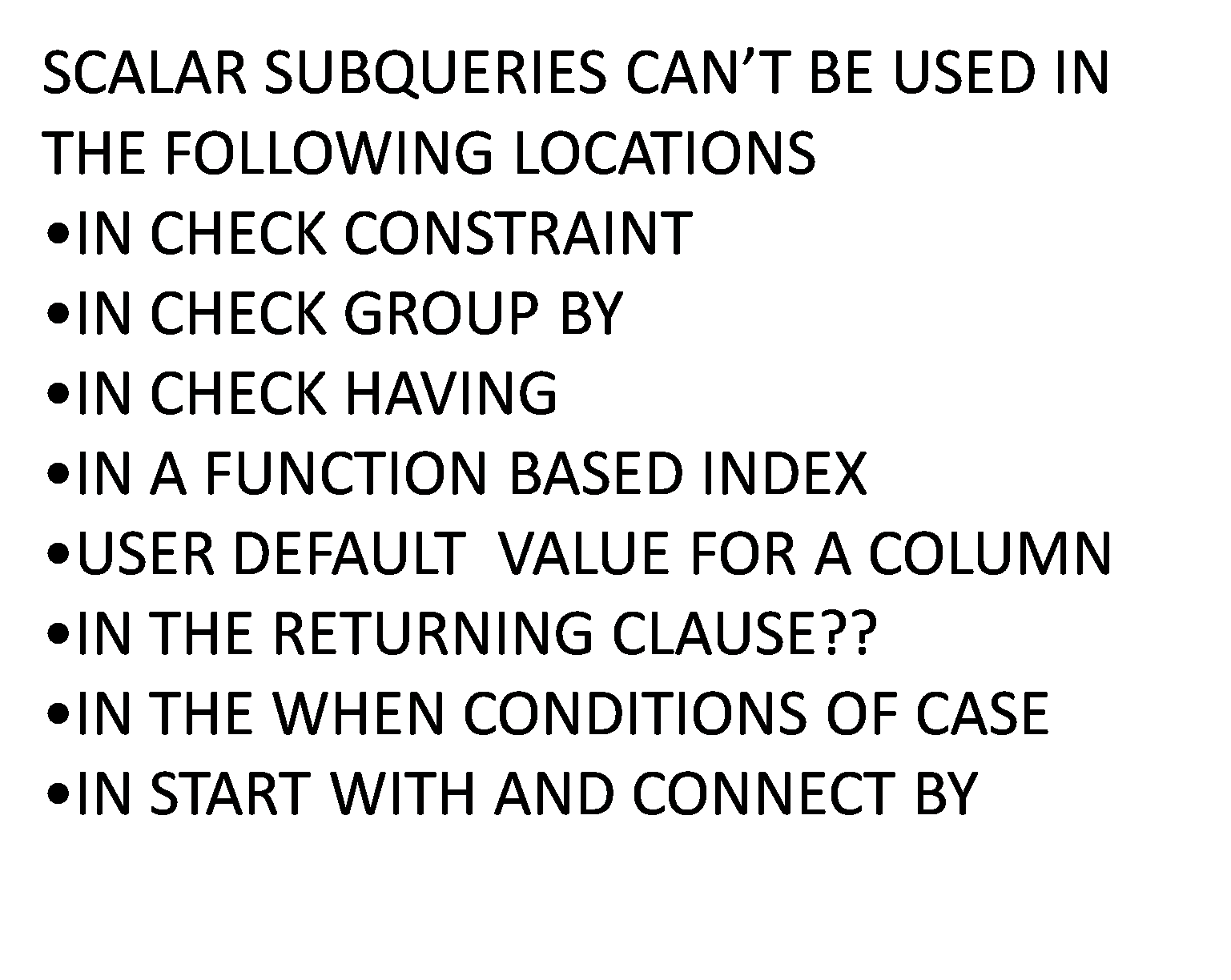
* Select level, empno, ename, connect\_by\_isleaf from emp where connect by\_isleaf=1 or 0 start with ename =’KING’

connect by prior empno=mgr;🡪8 rows.

* SCALAR SUBQUERIES:
* Using sub queries instead of column names is said to be scalar queries.
* Scalar queries have to return only one value for each of the outer query record.
* Scalar query contains only one column.

**Example:**

Select ename, (select loc from dept where dept.deptno=emp.deptno), job from emp;



**Note:**

* It is a special case of single row sub query.
* It is not possible to provide the order by clause, which are used in

where and having clauses of outer queries.

Select - SQ

From - SQ - order by

Where - SQ not possible to order by

Having - SQ clause

* We can use scalar sub queries as a correlated sub queries.
* We can use ANY and ALL an alternatives to a group functions particularly minimum and maximum.

**Examples:**

* Select \* from emp where sal>(select max(sal) from emp where deptno=10);
* Select \* from emp where sal>all(select sal from emp where deptno=10);

**Note:**

* =ALL
* <ALL(Smaller than the Lesser)
* >ALL(More than the Greater)
* >ANY(More than the Lesser)
* <ANY(Smaller than the Lesser)
* =ANY

Example:

* Create table ‘space tab’ (sno number(5));

#,/,-,$,char,number are allow in table name.

Table name space allow specifying with in double coats.

* Select \* from “space tab”;
* Sub query value cannot be used as a default value. Only we have to use literals or functions.

**USING INLINE VIEW DELETE CLAUSE:**

Sno Sname

10 A

20 20

30 X

40 Y

* Select \* from sam1
* Delete (select sno s from sam1) where s<30; 🡪3 rows.
* Update (select sname m from sam1) set m=’D’; 🡪2 rows.

**Example:**

Where [[start with condition1]

Group by connect by condition2]

Having

Order by

|  |  |  |
| --- | --- | --- |
| Empid | Ename | Mid |
| 1  2  3  4  5  6  7  8  9  10 | K  Blake  Jones  Clark  X  Y  Z  A  B  C | 1  1  1  2  2  2  3  3  4 |

* Delete from emp where empid in(select empid from emp mstart with ename=’Blake’ connect by prior empid=mid;

**Example:**

* Select level,lpad(‘ ‘,2\*(level-1))||ename xname from emp

Start with mgr is null

Connect by prior empno=mgr

Order by siblings by ename;

**SIBILINGS: (9i)**

* It place the child nodes there parent nodes while preserving hierarchy.
* In the absence of siblings hierarchy get disturbed.
* Example is given above.
* Select sum (sal) from emp start with ename=’BLAKE’ connect by prior empno=mgr;🡪9400.
* Select empno,ename,sal from emp

start with mgr is null

connect by prior empno=mgr;🡪6 rows.

* Select max(level) from emp

start with mgr is null

connect by prior empno=mgr;🡪4 rows.

* Select x.ename,x.sal,(select sum(sal) from emp y

start with y.ename=x.ename

connect by prior by y.empno=y.mgr), sum(sal) from emp x;

* Select level,ename,deptno from emp

start with mgr is null

connect by prior empno=mgr and

prior deptno!=deptno;🡪KING,JONES,BLAKE.

* Select level,count(empno) from emp

start with mgr is null

connect by prior empno=mgr group by level;

* Select level,count(empno) from emp

start with mgr is null

connect by prior empno=mgr group by level

having level=(select max(level from emp

start with mgr is null

connect by prior empno=mgr);

* Select level,ename,hiredate,prior ename,prior hiredate from emp

start with hiredate=(select min(hiredate) from emp

connect by prior empno=mgr);

* Select level,ename,hiredate,prior ename,prior hiredate from emp

start with hiredate=(select max(hiredate) from emp

connect by prior empno=mgr);s

* Select \* from emp where ename=’BLAKE’

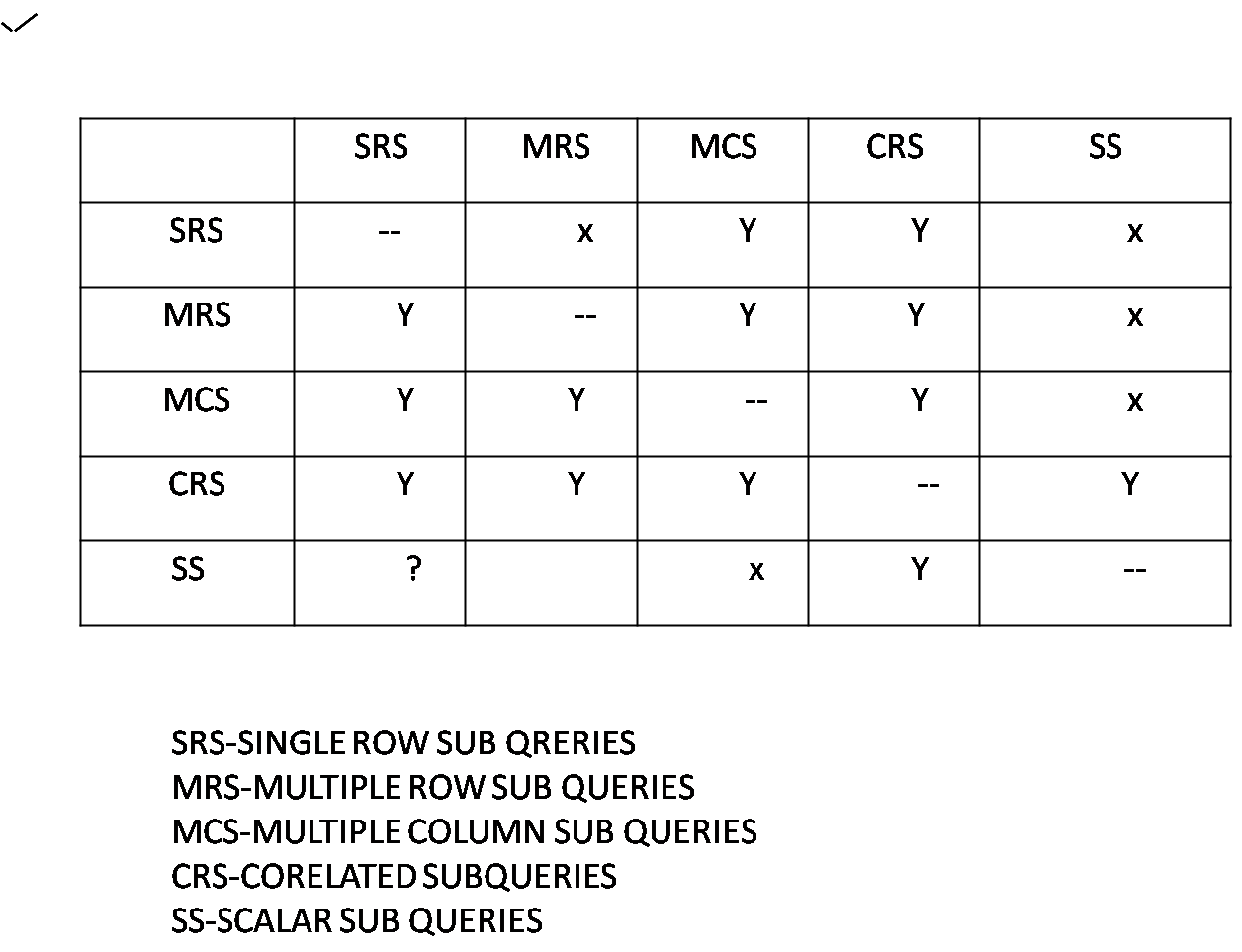
start with ename=’JONES’

connect by prior mgr=empno;

* Select \* from emp where ename=’JONES’

start with ename=’KING’

connect by prior empno=mgr group by level;



**CONSTRAINTS**

* They are the data integrity rules/restrictions which allows only valid data into tables.
* Through constraints we can fulfil the business requirements
* We provide constraints on tables and even on views.

**Constraint keys:**

1. **Primary key**
2. **Unique**
3. **Not null**
4. **Check**
5. **Ref(log)**
6. **Default**
7. **Foreign key**

**1. Primary key:**

* It acts as both (unique +not null) which means it won’t allow duplicate and null values
* Implicitly an unique index et defined on primary key columns
* These should be only one primary key for an entire table
* A P.K can hold maximum of 32 columns(i.e unique index limitation)
* Materialized view get defined only on tables, which are aving primary key’s
* Generally we call primary key table as master table/parent table.

**2. Unique key:**

* It allows only unique (null) and values (won’t allow duplicates)
* Not null values are allowed through unique constraint
* For unique key also an implicit unique index get defined
* An unique key can hold maximum of 32 columns

**3) Not Null:** It won’t allow null values, but allows duplicate values.

**Note:**

* Not null constraint are allowed only in columns levels
* Views won’t allow not null constraints

1. **Check:**

* To restrict/enforce other than standard integrity rules

(Primary key (unique + not null) we use check constraints.

* Check constraint throws an error only when condition becomes false, won’t throw for ‘true and null’
* It won’t allow sub queries as a condition
* It won’t allow user defined functions

**Note:** Check constraint allows regular expressions. Views won’t allow

check constraints.

1. **Default:**

* It takes default values (if user won’t provide any value to a columns then default provides default values to columns).
* It won’t allow sub queries and user defined functions.

1. **Foreign key:**

* It’s a reference integrity constraint (RIC), if ever you provide any value into the foreign key columns before begin inserted that value will be referred through F.K with primary/unique column
* F.K allows null values for flexibility and allows duplicates
* PK and FK columns names could be different but data types should be same/compatible, size should also be same.
* System defined column level (CL)
* System defined table level (TL)
* User defined column level
* User defined table level.
* **System defined column level:**

**SQL**>create table con(sno number(5) primary key, name varchar2(10) unique, bal number(5) check (bal>5000),id number(5) not null, bank varchar2(10) default ‘sbi’);

**SQL**>create table con1 (sno number (5) references con(sno), loc varchar2(10));

* **System defined table level:**

**SQL**>create table con2(sno number(5), name varchar2(10), bal number(5), primary key(sno), unique (name), check(bal between 1000 and 5000));

**SQL**>create table con3(sno number(5), loc varchar2(10), foreign key(sno) references con2(sno));

**Schema:**The logical collection of dat structures called ‘schema’.

**Schema objects and name spaces:-**

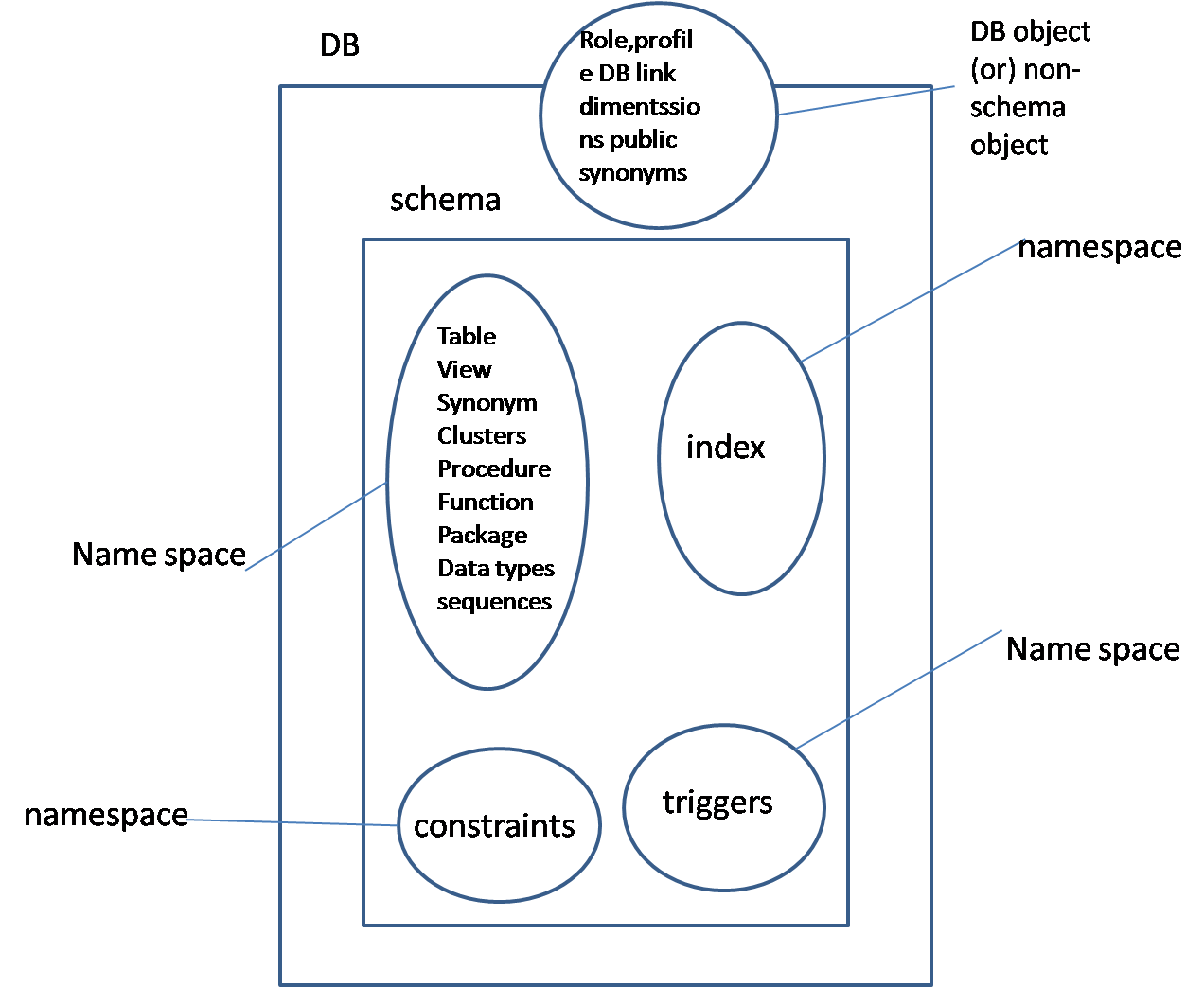


Table also DB object but additionally it acts as schema object

For eg in schema objects it table has emp name

* **User defined column level:**

**SQL**>create table con5(sno number(5) constraint key1 primary key, name varchar2(10) constraint un1 unique, bal number(10) constraint chk1 check(bal>1000), id number(5) constraint nn1 not null);

**SQL**>ceate table con6 (sno number (5) constraint keyfl references con5 (sno), loc varchar2(10));

* **User defined table level:**

**SQL**>create table con7(sno number(5), name varcar2(10), id number(5), bal number(5),constraint keyp2 primary key(sno),constraint un2 unique(name),constraint chk2 check(bal in(1000,2000,3000)));

**(or)**

**SQL**>create table con9 (constraint key3 primary key (sno), constraint un3 unique (name),sno number(5),name varchar2(10));

**SQL**>create table con8 (sno number (5),loc varchar2(10), constraint keyf2 foreign key(sno) references con7(sno));

* **Mixed method:**

**SQL**>create table con10 (sno number (5) primary key, loc varchar2 (10),name varchar2(10), constraint un5(loc));

**Advantage of table level than column level constraints:**

Column comparison is possible in table level which is not in column level

**SQL**>create table ctab(x number(5),y number(5), check (x>y));

**SQL**>create table ctab(x number(5),y number(5) check(x>y)); //invalid

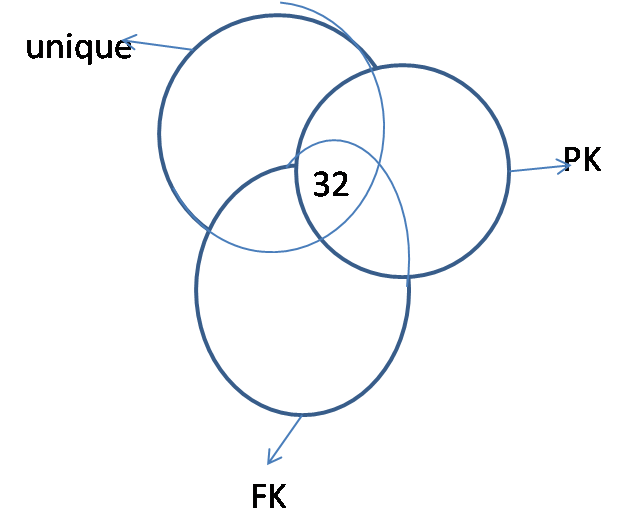
* **Self-key:**If you refer a FK with in the same table PK that FK is called as

‘self-key’.

**SQL**>create table selftab(sno number(5) primary key, id number(5) references selftab(sno));

* **Composite key:**

It holds more than one column (PK +FK +unique). Whichever the number of columns PK is having FK also s to contain same number of columns.



**Eg:-**

**SQL**>create table bt(sno number(5), loc varchar2(10),name varchar2(10), constraint comkey primary key(sno,loc));

**SQL**>create table ct(sno number(5),addr varcar2(10), constraint comfkey foreignkey(sno, addr) references bt(sno,loc));

**SQL**>insert into bt(sno,loc) values(10,’x’);

**SQL**>Insert into ct values (10,’x’);

**SQL**>Insert into ct values (10,’y’); //invlid

**SQL**>Insert into ct values (10, null);

**SQL**>Insert into ct values (20, null);

**SQL**>create table utab(sno number(5),name varcar2(10),unique(sno,name));

**SQL**>Insert into utab values (10,’x’);

**SQL**>Insert into utab values (10,’y’);

**SQL**>Insert into utab values (10,’x’); //invalid

**SQL**>Insert into utab values (10, null);

**SQL**>Insert into utab values (10,null); //invald

|  |  |
| --- | --- |
| C1 | C2 |
| 10 | X |
| 10 | X |
| 10 | Y |
| 10 | Null |
| 10 | Null |

In the above table last two rows are unique or same values then null also treat as same at that time it treats as duplicate record, so won’t allow.

In the above example null values become equal when all of the non-null values are same.

**Defining constraints on existing table by using alter command:**

By using alter command we can also provide constraints in two levels.

1. Columns level. (inline constraints)
2. Table level (dat\_of\_line constraints).

**Generic sysntax:** (not exact syntax)

**SQL**>alter table <table\_name> add|modify|disable|enable|validate|nonvalidate|rename|drop|enforce constraint <constraint\_name>;

**Note:** It is not possible to add Not Null constraint rather we modify it from Null to Not Null and Not Null to Null by using one alter we can use ‘n’ number of ‘adds’.

**Defining our own index name on a PK column:**

**SQL**>create table ctab(sno number(5) primary key using index

(create index dex on ctab(sno));

**if index is already exists**

**SQL**>create table ctab1 sno number(5) primary key using index <index\_nme>;

**Add:**

**SQL**>create table con15(sno number(5),nme varchar2(10),bal number(5));

**SQL**>alter table con15 add constraint PK15 primary key(sno);

**SQL**>alter table con15 add unique(name)

add constaint c15 check(bal>1000)

add check(loc in(‘hyd’));

**SQL**>alter table con15 add foreign key(bal) references con15(sno);

**Modify:**

SQL>create table con16(sno number(5) null); //it is not a constraint

SQL>alter table con16 modify sno constraint nn15 not null;

SQL>alter table con16 modify sno null;

SQL>alter tble con15 modify (sno not null, name unique);

**Rename:**

Syn: Alter table <table\_name> rename constraint oldname to newname;

SQL>create table ct(sno number(5) constraint kp10 primary key, name varchar2(10) constraint un10 unique);

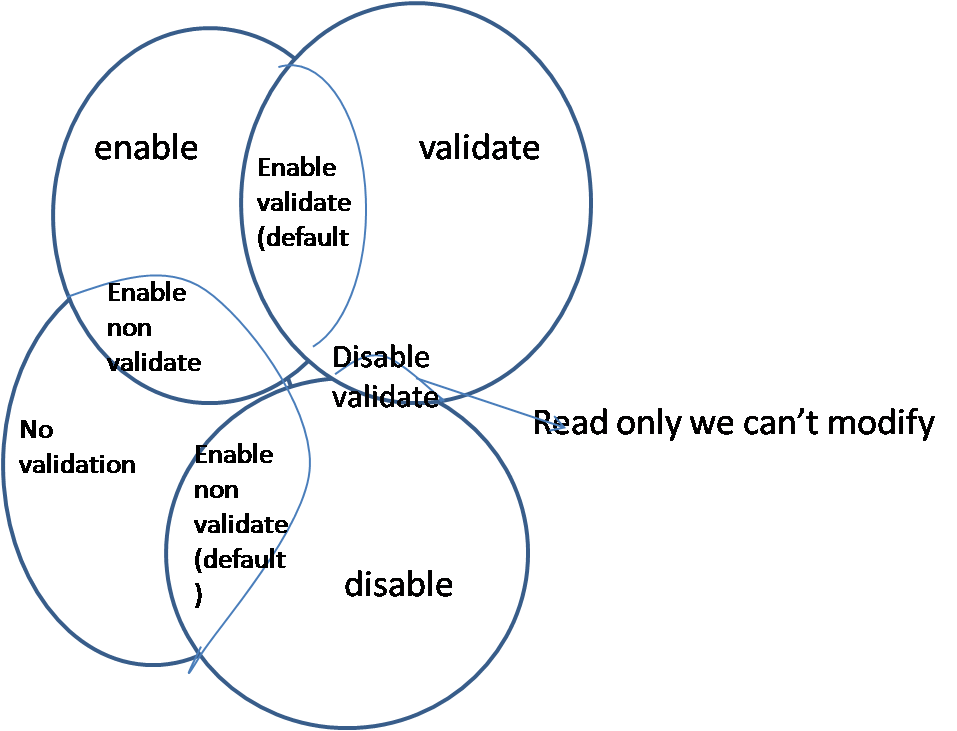
SQL>alter table ct rename constraint kp10 to kp11;

Note: we cannot alter more than one column at a time.

**Constraint states:**

Enable/disable: works for futex data.

Validate/novalidate: past data.



* But by default enable means enable-validate and disable means disable no validate.
* While disabling and enabling constraints indexes get dropped and re-created but by using ‘keep index’ we can prevent the index by dropping.

**Defining a disabled constraint:**

**SQL**>create table lb(sno number(5) constraint kp12 primary key disable);

**SQL**>alter table lb enable constraint kp12;

**SQL**>alter table lb disable constraint kp12;

**SQL**>alter table lb enable primary key;

**SQL**>alter table lb disable primary key;

**Note:** Without knowing the name of a PK unique constraint we can enable and disable and drop the constraints of a table.

**Eg for ‘keep index’:**

**SQL**>alter table lb disable primary key keep index;

**SQL**>alter table lb enable constraint uk1 unique;

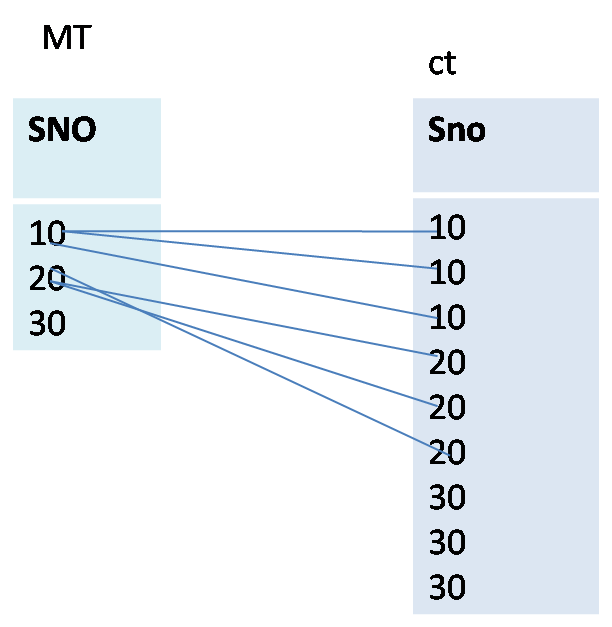
**SQL**>alter table lb disable constraint(sno,loc);

**SQL**>alter table lb drop constraint kp12;

**SQL**>alter table lb drop unique(sno,name);

**Delete rules:**

|  |  |
| --- | --- |
| Event | Action |
| * Deleting the parent records * Dropping parent column and table * Dropping/disabling PK when it is relation with FK | * On deleting (while defining FK) * Cascade constraint * Cascade |



* If you want to delete the PK record you can not delete because table is in relation with FK and you have child records so, you can not delete, first you need to delete child records, these is a chance to delete the records by using ‘on delete cascade’.
* If tables are in rlation (PK with FK) you can’t delete PK column and PK record and PK table and PK until unless deleting the CT, but we have the chance to drop and delete by using one table.

**On deleting cascade:**  Generally it is not possible to delete the parent records/master records directly when they are having child records but by providing ‘on delete cascade’ at the time of FK definition it is possible.

**On delete set null:** It provides null values for department child records while deleting parent record.

**Cascade constraint:** Vary directly dropping master/parent table and PK column is not possible when they are in relation with FK , but by using cascade constraint it is possible.

**Cascade:**  If you to drop/disable PK/unique key constraint when they are in relation with FK you have use cascade.

**SQL**>create table mtab(sno number(5)) constraint kp3 primary key, loc varchar2(10));

**SQL**>create table mtab1(sno number(5) constraint kp4 primary key, name varchar2(10));

**SQL**>create table ctab(sno number(5) conatrnt fk3 references matb(sno) on delete cascade);

**SQL**>create table ctab1(sno number(5) constraint fk4 references mtab(sno) on delete set null);

**SQL**>insert into mtab(sno) values(10);

**SQL**>insert into mtab(sno) values(11);

**SQL**>insert into ctab(sno) values(10);

**SQL**>insert into ctab(sno) values(10);

**SQL**>insert into ctab(sno) values(10);

**SQL**>delete from mtab where sno=10;

**SQL**>select \* from ctab;

**SQL**>delete from mtab1 where sno=10;

**SQL**>select \* from ctab1;

**SQL**>alter table matb drop column sno; //error

**SQL**>alter table mtb column sno cascade constraint;

**SQL**>drop table mtab; //valid

**Note:** Ere already we dropped PK column so that table is dropped in the below statement mtab1 is in relation so, table is not dropped it gives error.

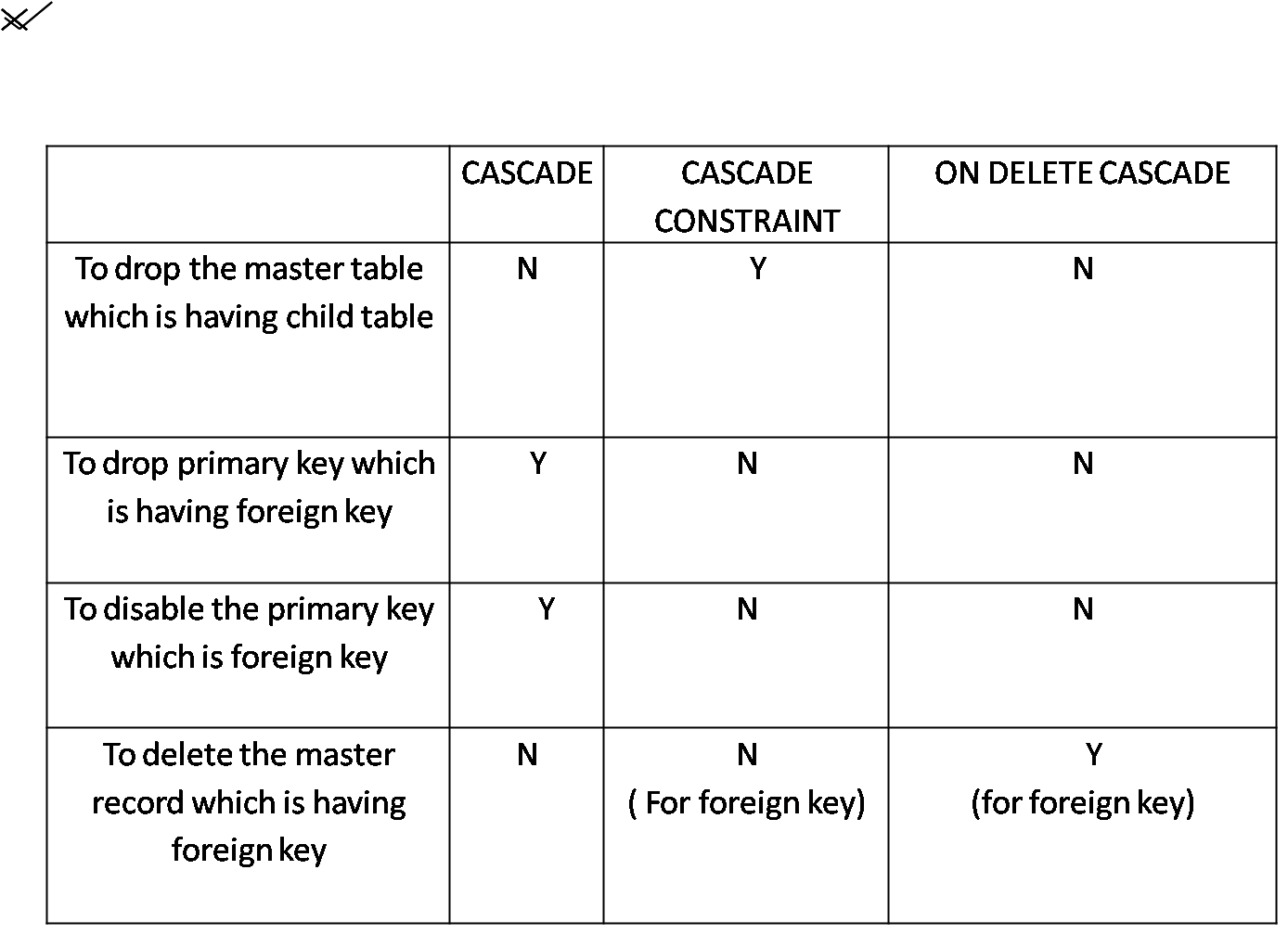
**SQL**>drop table mtab1; //invalid

**SQL**>drop table mtab1 cascade constraint; //valid

**SQL**>alter table mtab1 drop constraint pk4;

//error: this unique/PK is referenced by some FK’s

**SQL**>alter table mtab1 drop constraint PK4 cascade; //valid



**Constraint checking:**

Constraint checking is takes place in two ways:

1. Initially immediate(default)
2. Initially deferrable

* If constraint checking takes place at the individual statements i.e called ‘initially immediate’ which is default.
* But if constraint checking takes place at the time of transaction is called ‘transaction specific’ we do this with ‘initially deferrable’.

**Eg for initially immediate:**

**SQL**>create table mtab4(sno number(4) primary key initially immediate);

**SQL**>insert into mtab4 values(10);

**SQL**>insert into mtab4 values(10);//invalid

**Error:** unique constraint (apps-sys-c00209..) violated.

**SQL**>set constraint all deferrable.

**Eg for initially deferrable:**

**SQL**>create table mtab5(sno number(5) constraint keyp primary key deferrable);

**SQL**>set constraint keyp deferrable;

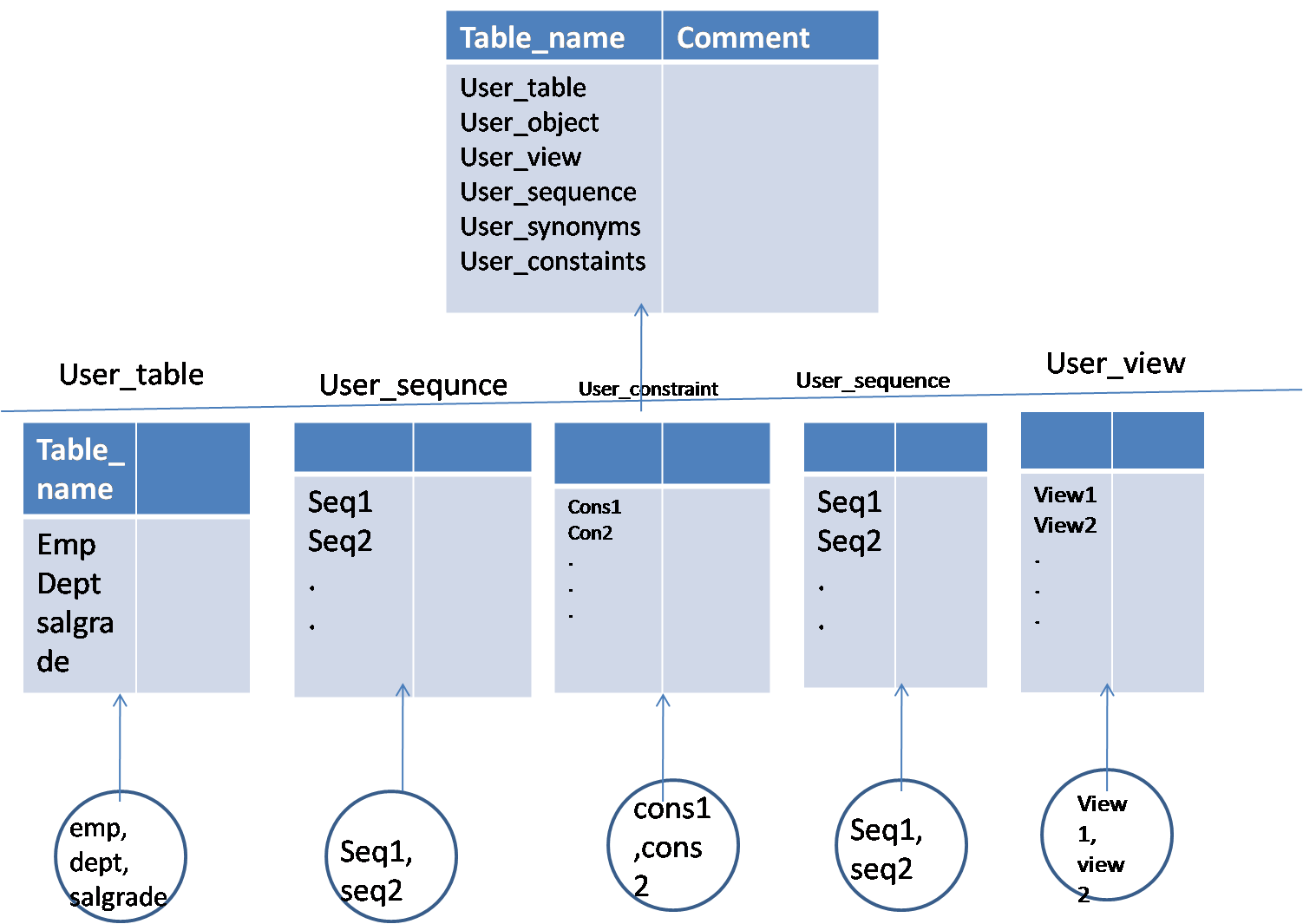
**Note:**here in the (create table) above eg we used deferrable, after that.

**Disadvantages of constraints:**

* Constraints can’t handle varying data, but by using trigger we can handle varying data.
* Constraint can check the old data but trigger’s can’t check the existing data.(it checks only incoming data)
* Constraints are more useful than trigger.
* Constraints can give guarantee for centralized data.

**Tables regarding with constraints:**

Data dictionary tables:



**SQL**>select table\_name, comment from dict where table\_name like ‘%constraint%’,‘%view’;

* In dict everything will be in uppercase
* Metadata: data to the data.
* Data dictionary tables will store in ‘dict’
* To find out the column and total no of column in a given table we use following queries

**SQL**>select column-id,column-name,table-name from user-tab-columns where table-name=’emp’;

**SQL**>select count(column\_id) from user\_tab\_columns where table\_name=’emp’;

->To find out the constraint\_name once we know the tablename;

**SQL**>select constraint\_name, constraint\_type, table\_name, status,index\_name from user\_constraints where table\_name=’emp’;

**we use following query to find out constraint name and table name once we know the column name.**

**SQL**>select column\_name,table\_name,constraint\_name from user\_cons\_columns where column\_name=’sno’;

**Constraint types:**

Primary key P

Unique U

Foreign key R

Check C

Not null C

**SQL**>select object\_name, created, timetamp, status from user\_object where object\_name=’conid’;

**SQL**>select object\_type, count(\*) from user\_objects roup by object\_type;

* In DICT we will get all table types information
* To know column identification, constraints, status, and everything we know by using data dictionary tables it’s vary use full.

**Synonyms**

They are the alternate names to tables, views, synonyms and so on……

* They are virtual/mirror objects
* Useful in frontend application and to hide the table owner name
* Useful in distribution databases
* They don’t have their own structure(it won’t occupy any memory area)
* They are simply logical names
* They again depends on base tables for SQL statements
* It is not possible to alter the structure of base table by using synonyms
* We can define synonym on another synonym
* A table can have ‘n’ no of synonyms but each synonym can be defined only on single table, which means a synonym can’t hold more than one table
* We can also define synonym on non\_exixting objects
* It is not possible to hide partial part of table data by using synonym like views
* They are two types of synonyms

1. Public synonym
2. Private synonym

* Private synonym share the same table name space
* Public synonym does not share same table name space, so we can define public synonym names with table names
* If you drop the synonym base table won’t get effect but, if you drop base table , synonym becomes invalid

**Syn:SQL**>create [public] synonym <synonym\_name> for <table\_name>;

* Synonym and views are subjected to table constraints
* We can’t create synonym for constraints
* Distribution database objects means more than one database

**SQL**>select \* from newtab;

**SQL**>create synonym s for newtab;

**SQL**>select \* from s;

**SQL**>insert into s values(20,’a’,’hyd’);

**SQL**>insert into s(sno,name) values(50,’x’);

**SQ**L>select \* from s;

**SQL**>create or replace synonym s1 for s;

**SQL**>select \* from s1;

**SQL**>drop synonym s;

**SQL**>select\* from s1;

**SQL**>create synonym s for newtab;

**SQL**>select \* from s1;

**SQL**>drop table newtab;

**SQL**>select \* from s1;

**SQL**>create public synonym emp for emp;

**SQL**>select \* from DICT where <table\_name> like ‘%synonym%’;

**SQL**>desc user\_synonym;

**SQL**>desc all\_synonym;

**SQL**>select table\_name from user\_synonyms where synonym\_name= ‘s’;

**SQL**>select synonym\_name from user\_synonym where table\_name=’emp’;

**User\_synonym:**

|  |  |
| --- | --- |
| Table name | Synonym name |
| T  S | S  S1 |

**Finding out the synonym for other synonym**

**SQL**>select table\_name from user\_synonym where synonym\_name=’1’;

**Creating synonym on non-existing object:**

**SQL**>select \* from dx; //there is no table

**SQL**>create synonym snd for dx; //synonym created

**Note:**Table/view is not existed for dx, but synonym creted we can do this.

**SQL**>select \* from sno; //synonym transaction is no longer vlid

**SQL**>create table dx(sno number(5));

**SQL**>select \* from sno;

**SQL**>insert into dx values(10);

**SQL**>select \* from sno;

**Views**

They are an advance of synonym

* They are mirror/logical names
* Views are stored queries
* They do not have their own structure, they again depends on base tables for SQL statements
* It is not possible to modify the structure of table by using views
* We can define views on synonyms and even on views
* We can define a view on more than one table
* We can also define views on non-existing objects
* It is possible to hide the partial part of data by using views so, to provide security
* Views makes the application design easy
* Views provide the location transparency
* Views makes the client work easy which means client no need to know about the technical things like join conditions, functions and so on..
* Views represents the summarized data
* Views display the table data in a client perspective
* Views are useful in like tool like data ware housing and so on..
* Materialized views are useful for performance and to store historical data
* We can provide constraints even on views
* Views won’t allow check and not null constraints

**Types of views:**

* **Simple view**
* **Complex/composite view**
* **Read only**
* **Inline**
* **Join**
* **Functional**
* **Force**
* **Partition**
* **Object**
* **Materialized**
* **Vertical**
* **Horizontal**
* **View wit check option**
* **Simple view:**This view is defined on single table

**SQL**>create or replace view <view\_name>[view column] as select statements;

**SQL**>create table new(sno number(5), loc varchar2(10));

**SQL**>create or replace view sview as select \* from new;

**SQL**>select \* from sview;

**SQL**>insert into sview values(10,’x’);

**SQL**>insert into sview(sno) values(20);

**SQL**>select \* from new;

**SQL**>select \* from sview;

* **Read only view:**These views are only read only purpose

**Syntax:**

**SQL**>create or replace view <viewname> as select statement with read only;

**SQL**>create or replace view rview as select \* from new wit read only;

**SQL**>select \* from rview;

**SQL**>insert into rview values(30,’y’); //invalid

**Note:** DML operations are not allowed on read only view.

* **Inline view:**
* Unlike other views they are not stored objects
* They are only temporary queries
* In inline views we will mention subsequeries in from clause of another query

**Eg**: **SQL**>select \* from (select \* from emp);

* **Force view:**We define force view on non-existing tables.

**Syntax:**

**SQL**>create or replace view <view\_name> as select \* from <table\_name>(which is not existing currently) i.e if table is not created also we can create view some times it is useful

**SQL**>create or replace forceview fview as select \* from new1;

**Warning:** view created with compilation error

**SQL**>select \* from new1; //table/view doesn’t exist

**SQL**>create table new1(sno number(5));

**SQL**>insert into new1 values(10);

**SQL**>select \* from fview;

* **Partial view:**These are the views which are defined on compound queries

**SQL**>create view pview as select \* from emp union select \* from emp

**Note:** DML operations are not allowed on partition views

* **Functional view:**In this we will make use of functions while defining a view in select statements

**Note:** In view select statements expressions and functions has to be defining with aliases, as shown in below

**Eg:**

**SQL**>create view asview as select 2\*3 from dual;

**Error:** must name this expression with column alias

**SQL**>create view sview as select min(sal),max(sal) from emp;

**SQL**>create view sview as select min(sal) minsal,max(sal) maxsal from emp;

**SQL**>ceate or replace view pview(minsal,maxsal) as select min(sal),max(sal) from emp;

**SQL**>ceate view svw as select sal from emp;

**SQL**>select \* from svw;

**SQL**>create or replace view svw1(minsal) as select min(sal) ename from emp groupby ename;

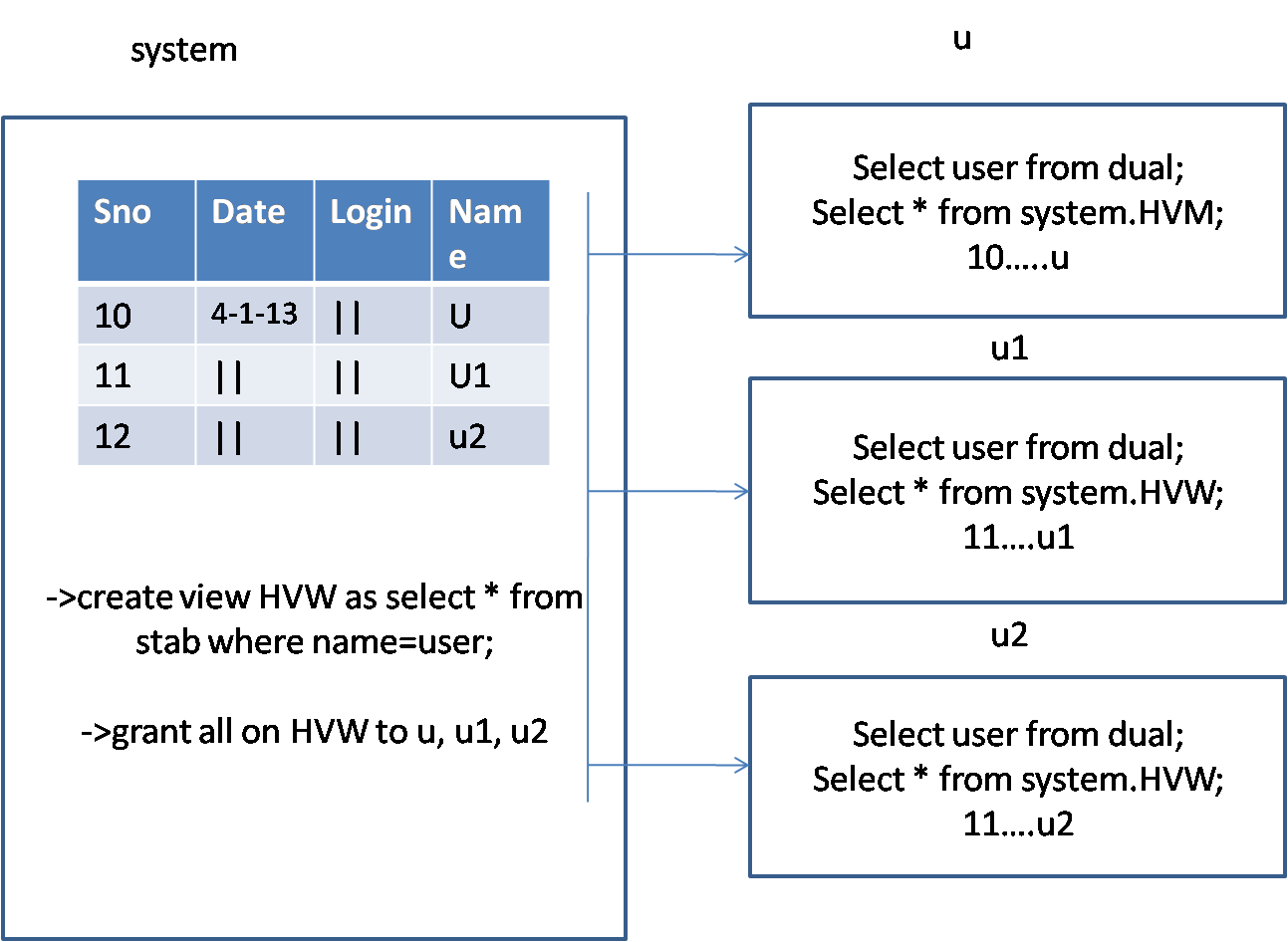
**SQL**>create or replace view svw1(minsal,vname) as select min(sal), ename from emp groupby ename;

* **Vertical View:**In this will create a view by selecting specific columns from a table so as to hide few of columns in a vertical manner

**SQL**>creae or replace view vview as select empno,ename,job from emp;

* **Horizontal view:**To hide rows, usage of horizantal view as mentioned below in diagram

Note: Vertical and horizontal views are useful in administration side



* **Complex view:**
* If you define a view by making use of more than one table those views are aid to be ‘complex views’
* Generally most of the complex views will have join conditions that views are considered as ‘join views’.

**SQL**>create or replace view comview as select empno, ename, emp.deptno, loc, dname from emp,dept where emp.deptno=dept.deptno;

* **Object view:**They are the views which are defined on object tables

Object table:A table which is defined by using object data type

**Object data type:**

* It is a oops concept
* It is user defined permanent data type wich is having fields to store homogenous data.
* Object data types are useful to full fill the real time applications and also alter the performance

**Syntax:**

**SQL**>create or replace type obj as object(eno number(5), ename varchar2(10). Mail varchar2(10));

**SQL**>create table lt(comp varchar2(10),empdet obj);

**SQL**>create table objtab of obj:

**SQL**>create view objview as select \* from objtab.

* **materialized view:**
* Unlike other views it has own structure, it is a replica
* Materialized views are useful to store historical data or summarized data
* These are useful to increase performance in tools like data ware housing and RAC mobile computing and so on….
* It is also useful for backup
* We need some special privileges to define materialized view
* Materialized views are get defined only on views which are having PK
* Before creating materialized view we have to define materialized view log for that table

**Syntax:**

Create materialized view viewname refresh on commit/demand fast/compile as **SQL>**select \* from tablename;

**Eg:**

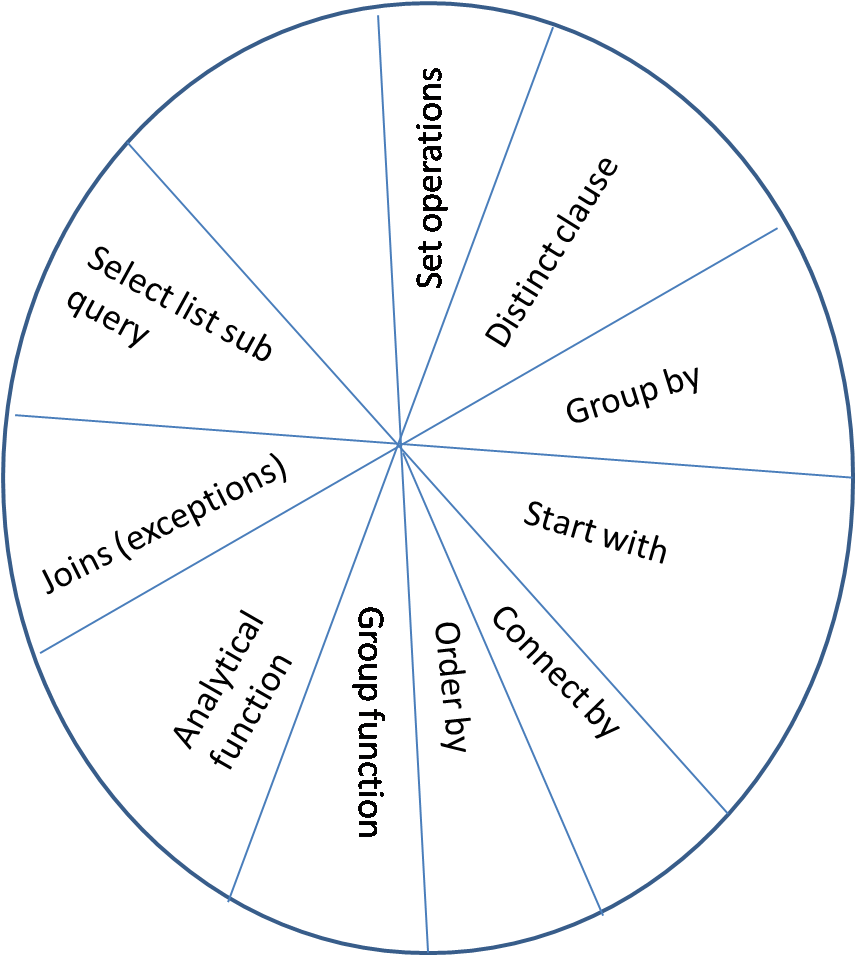
**SQL>**create materialized view log on emp;

**SQL>**create materialized view mview refresh on commit fast as select \* from emp;

we need to refresh for every n(5/10..) time, if any new record added. If we give ‘on demand’ we need to type in sql\*plus ‘DBMS\_mview’ then only the effect will be populated in replica(mview)

**Views without DML operations:**

1. Read only
2. Partition
3. Complex
4. Views which are having following tings



We cannot perform DML operations on these if anything is present in diagram in select list.

**Note:**View and synonym are subject to the table constraints. View and synonyms are transactions are also considered by the table constraints

**View with DML operations: View with not null column:**

|  |
| --- |
| C1 c2 |
|  |

|  |
| --- |
| C1 c2 c3 |
| 1o x –  20 -   * - |

**SQL>**insert into v values(10,’x’);

**SQL>**insert into v(c1) values(20);//invalid

**SQL>**insert into v(c2) values(2);

**SQL>**update v set c2=null;

* + Updating the not null column with null values through view is not possible in the above ex
  + Deleting is possible if we perform delete on c1,c2 also entire record will delete

**View without not null column but in table:**

* Insert is no way possible
* Update is very possible
* Deletion also possible

**View with check option:**

* When where clause condition matches the we can perform below operations
* It works as check constraint

**SQL>**create or replace view chview as select \* from newtab where sno=10 with check option

* As it is not possible to provide check constraint on views, we can use ‘view with check option’ to impose same restriction.

**SQL>select \* from chview;**

* Through this we can insert only values 10 and deletion also possible values 10

**SQL>**insert into chview values (10);

**SQL>**delete from chview where sno=10;

**Note:**If you drop a table department view will get individual if you recreate a table with the very same name now view becomes valid. If you alter the structure of a table without disturbing the columns which are used by view, in this case view won’t become invalid.

If you rename/drop the columns which are used by view then view becomes invalid. If you recreate the columns which are used by view their view automatically becomes valid.

**Sequences**

* Sequence is database object
* It is a shared object
* Sequence display the integer number
* Sequence eliminate serialized and improves concurrency
* Useful for multiple users across the DB
* Useful in frontend applications we can define synonyms on sequences

**Syntax:**Create sequence <seq\_name> [start with value]

[increment by value]

[minvalue value}nomin value]

[maxvalue value|nomax value]

[cycle|nocycle]

[cache value|no cache]

[order}no order]

[default]

**Start with value:**

* It specifies with which value sequence has to start by default it starts with ‘1’.
* Always start with values has to equal or greater than min value

Note: We can alter all other parameters except start with parameter

**Increment by value:**

* It specifies with which value sequence has to increment so, to get next value.
* By default it increment with ‘1’
* This may also have ‘-ve value’

Min value:Min value of the sequence

Max value: Max value of the sequence

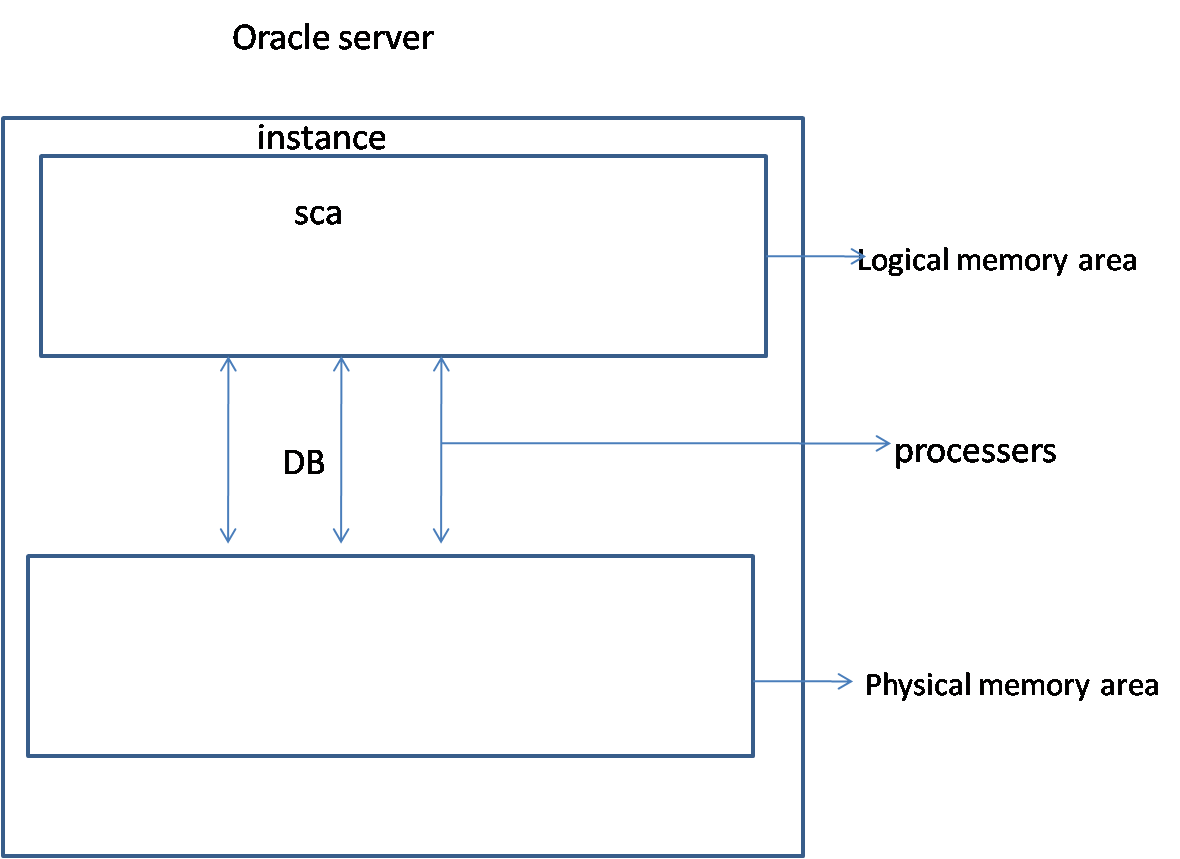
|  |  |  |
| --- | --- | --- |
|  | Minvalue | Maxvalue |
| +ve | 1 | 1\*1027 |
| -ve | -1\*1026 | -1 |

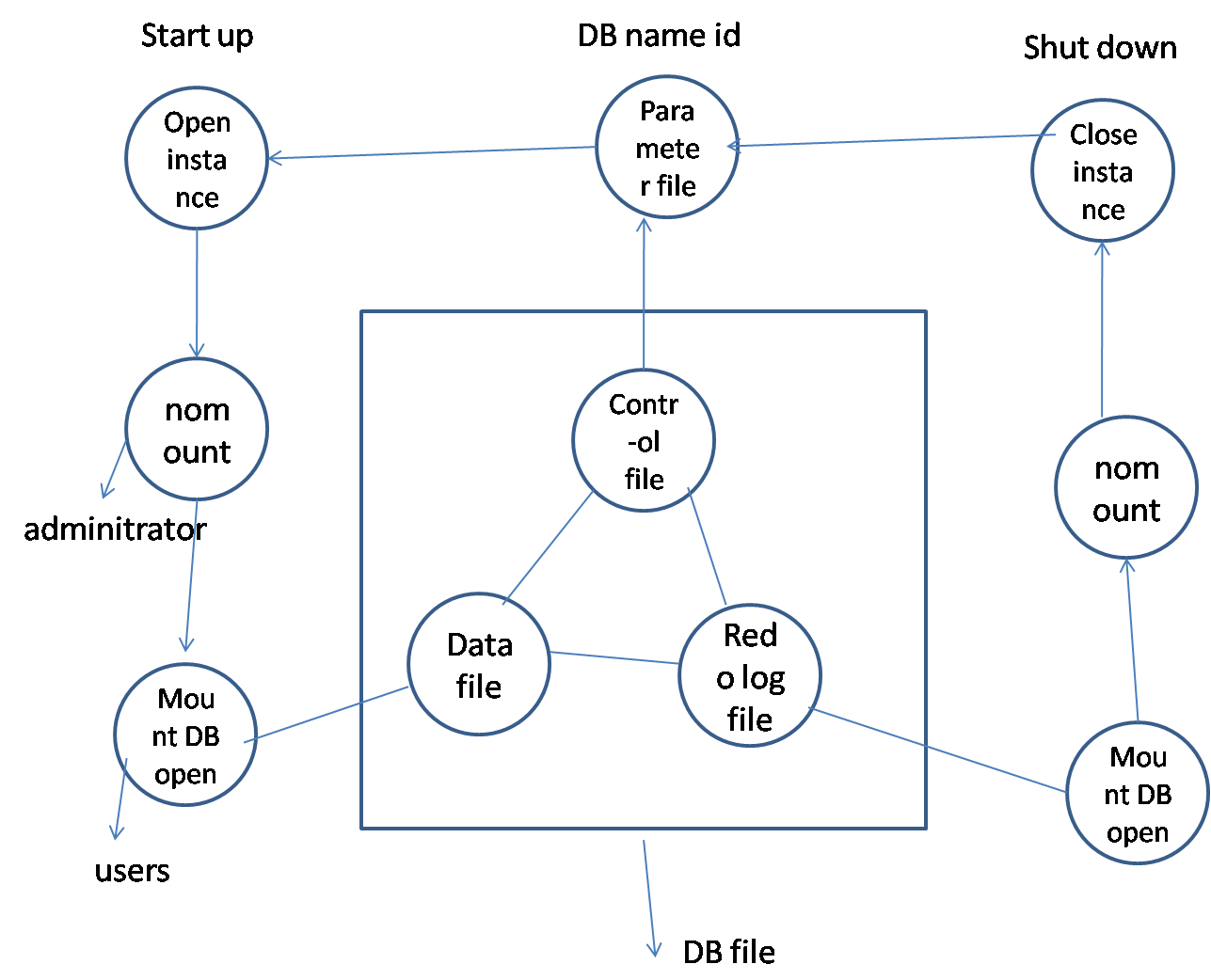
**Cycle:**

* To generate the sequence values in cyclic manner(looping/iterations) we use ‘cycle’
* Default ‘nocycle’
* For the second iteration sequence starts with min value

**Cache:**

* It is a memory area which is in saea(instance), which stores the pre-generated value, so has to increase the performance
* Cache values get vanished/erased when system abnormally shutdown
* By default it stores upto 20 values
* While exporting and importing the data we may have the chances of skipping sequence values
* By default min value of the cache is ‘2’





**Note:**Cache values has to fit into the cycle by using following formulae we have to specify the cache values

**Formula:**Ceil(maxvalue-minvalue)/abs(increment by value)

**Eg:** Ceil((10-1)/2)

(9/2)

Ceil(4.5)=5

**Order:-**We have to use tis order parameter only in RAC applications (real application clusters)

**Usage of sequence:** We use sequence with the help of sequence pseudo columns They are:

1. Next lavel
2. Curr level

**Next level:**displays the next value of sequence

**Curr value:**displays the current status of the sequence or currently

generated value

**Note:**We have to use currval only after nextval

**Syntax**: Sequencename.nextval;

Sequencename.currval;

**SQL>**create sequence sq;

**SQL>**select sq.nextval from dual;

**SQL>**select sq.nextval from dual;

**SQL>**select sq.currval from dual;

**SQL>**create table t(sno number(5));

**SQL>**insert into t values(sq.nextval);

**SQL>**select \* from t;

**SQL>**Update t set sno=sq.nextval;

**SQL>**create synonym st for t;

**SQL>**insert into st values(sq.nextval);

**SQL>**select \* from st;

**SQL>**create sequence sq1 start with 5 increment by 2 minvalue 1 maxvalue 10 cycle cache 5

**Output:**sequence created

**SQL>**alter sequence sq1 nocycle;

**SQL>**alter sequence sq1 maxvalue 20;

**Note:** We can alter all other parameters except start with value

**SQL>**alter sequence sq1 increment by 1 maxvalue 10;

**SQL>**select sq1.currval, sq.nextval from dual;

**Note:** If you use the currval just beside of nextval in the same select statement, firast nextvalue will be consider and also currval displays the nextval values or currently displayed value.

**SQL**> select \* from user\_squences where sequence\_name=’seq’;

**Usage of sequence:**

|  |  |
| --- | --- |
| Select list | Insert value clause |
| To assign pl/sql variables(11g) | Update set clause |

**Eg:**

Declare

V number(5):=sq.nextval;

Begin

DOPL(v);

End;

**We won’t use sequence in the following cases:**

* Sub queries
* View query (create or replace view sview as select \* from..) in this select list won’t use.
* Order by
* Group by
* Select list where clause and distinct clause
* Delete statement
* Set operators
* Materialized view
* Check constraints
* Default value(create and alter statement)
* Write a query to display to all child tables for a parent table
* How to get manager name along with employee name in a single column
* Why we use concatenation operator in DBMS\_output.

put\_line

|  |  |  |
| --- | --- | --- |
| Empno | Sal | Flag |
| 101  102  103 | 3000  2000  5000 |  |

|  |  |
| --- | --- |
| Empno | Bones |
| 102  101 | 1000  0 |

|  |  |  |
| --- | --- | --- |
| Empno | Sal | Flag |
| 101  102  103 | 3000  2000  5000 | Y  Y  Null |

**Analytical functions:**

* Rank()
* Dense-rank()
* Parent-rank()
* Cumu-dist()
* Row-num()
* Ntile()
* Log()
* Lead()

**Rank():** Rank provides the ranking values for each of the table records rank

skips the sequential ranking values when there is a duplicate records or values

**Dense-rank():**It works as same to that of rank but it won’t skip the sequential

ranking.

**Percent-rank():** Percent rank work based on rank values by using

following formulaeRecord rank-1/total rank-1

**Cumu-rank():**Calculate the cumulative distribution by using following

Formulae

**Row-num():**Provides the row numbers for each of the record this row

number allows partition unlike pseudo column row numbers

**Ntile():**Segments the records into given number of partitions

**(or)**

divides the records into n number of partitions

**Log(),lead():**

Displays logging and leading values with respect to current record

**Eg:**

Select deptno,sal ,

rank() over(partitions by deptno order by sal desc) rnk,

dense-rank()over(partitions by deptno order by sal desc) drnk,

perent-rank()over(partitions by deptno order by sal desc) prnk,

cumu-rank()over(partitions by deptno order by sal desc) cd,

row-number()over(partitions by deptno order by sal desc) rn,

ntile(2) over(partitions by deptno order by sal desc) nt,

log(sal,1) over(partitions by deptno order by sal desc) lg,

lead(sal,1) over(partitions by deptno order by sal desc) ld

From emp;

**Note:**Some of the group functions also works as analytical functions as in the following case

**SQL>**select deptno,sal,sum(sal) over(partition by deptno order by sal desc) ssal from emp;

**Flashback: (DDL commands 10g)**

Hear flashback won’t applicable to rollback because flashback is DDL command, commit and rollback are DML commands. And DDL are permanent and DML are temporary.

It retrives DB objects from ‘recycle-bin’

**Eg:**

**SQL**>select \* from ftab;

**SQL**>drop table ftab;

**Note:** From 10g onwards if drops an object that will be placed in recycle bin

**SQL**>select \* from ftab; //table/view doesn’t exist

**SQL**>flashback table ftab to before drop;//flashback completed

**SQL**>select \* from ftab;

|  |
| --- |
| Sno |
| 10  10 |

**Purge: (DDL command 10g)**

Purge eliminates the objects from recycle-bin permanently

Note: Purged objects are not possible for flashback

Purge table ftab;

While dropping we can bypass the flashback by usi g following

**EG:**

**SQL**>create table ptab(sno number);

**SQL**>drop table ptab purge;

**SQL**>flashback table ptab to before drop; //objects not in recycle bin

We can completely empty the recycle bin by using purge command as shown in blow.

**EG**: SQL>purge recycle bin;//total tables will be deleted permanently

SQL>desc recycle bin;

**Recycle bin metadata:**Here we can store all the dropped tables,

SCN, timestamp

**SQL>**create table stab(sno number(5));

**SQL>**drop table stab;

**SQL>**desc recyclebin;

**SQL>**select \* from recyclebin where original\_name=’stab’;

**Eg for SCN\_to\_timestamp and timestamp\_to\_SCN:**

**SQL>**Select original\_name,dropSCN,SCN\_to\_timestamp(DROPSCN) tmst,

timestamp\_to\_SCN(SCN\_to\_timestamp(DropSCAN)) SCN from recyclebin

where original\_name=’stab’;

**Merge:**Merge perform insert, update, delete, in a single statement

**Syntax:**

Merge into tablename using tablename|subquery|view on (condition)

When matched then update set col=val, col1=val1…[where condition]

[delete where condition]

When notmathed then

Insert[(col1,col2)] values(val1,val2);

[where condition];

**SQL>**select \* from dept;

**SQL>**create table mdept as select \* from dept where 1=2;

**SQL>**merge into mdept x using dept

on (x.deptno=y.deptno)

when matched then update set x.loc=’hyd’

when not matched then

insert values(y.deptno,y.dname,y.loc)

* which rows are involving in condition that column should not use in update
* In matched condition we have to write update and delete in table whatever the rows has updated that rows only will be affected for deletion. In not matched only we need to write insert.

**SQL>**select \* from mdept;

**SQL>**update set x.loc=’bang’;

**SQL>**delete where x.loc=’bang’

Deletion happens to the records which are affected by update statement.

**We can also specify merge with out matched condition:**

**SQL>**merge into mdept x using dept y on (x.deptno=y.deptno)

When not matched then

insert values(y.deptno,y.dname,y.loc);

**We can also specify merge without not matched condition:**

**SQL>**merge into mdept x using dept y on (x.deptno=y.deptno)

when matched then update set x.loc=’bang’ delete where x.loc=’bang’

* **We can write like this also**

**SQL>**merge into mdept x using dept y

**Indexes**

* Indexes are the database objects, defined on table columns, which makes the optimizer work easy
* Indexes are useful to enhances the performance oracle by default make use of index’s for data maintenance
* Defining an index won’t give any guarantee for its usage it all depends on how optimizer chooses
* Defining excess of indexes is not appropriate or preferable
* If you define an index on a column index stores that column data along with rowid’s
* If you provide a condition on index column now optimizer will search the index column data with the help of rowed it identifies the table record vary directly instead of scanning entire table data
* Define indexes on column which are frequently used in where and order by clauses
* Optimizer search the indexes based on column ‘selectivity’s’.

Selectivity:

More selectivity=less duplicity

Less selectivity=more duplicity

* Columns which are having more selectivity are good choice for indexes
* We can define n no of indexes on table columns
* More no of indexes are useful for select statement but for DML operations it is not useful since it hinders the performance.
* Equal operator readily invokes the column indexes whenever you use the index column in where clause with equal operator
* Not equal to(!=) will not invoke the indexes(performance degrades)
* Like operator will not invoke the indexes when ‘%’ is at leading position (or) starting character.

**Drawback:**

* Function won’t allow the indexes to make use of it unless it is a functional index

**optimizer**

**SQL**

Here all sql statements are make use of optimizer and optimizer is nothing but DBMS set of programs it choosing optimizer.

* LOBRAW and column won’t allow indexes
* Indexes will get automatically defined on a columns which are having unique and FK constraint. An index can hold maximum of 32 columns but in each bit map index it is 30
* Oracle by default make use of B+ tree index
* Indexes won’t store null values except bitmap index
* When you drop the table with it, indexes also get dropped
* Bitmap indexes are useful for flag columns(less selectivity, high duplicity)

**SQL>**select /\*+hint \*/

Here it providing hint to query for multiline comment

**SQL>**select count(\*) from emp;

**SQL>**select count(1) from emp;

**SQL>**select count(rowed) from emp;

**SQL>**select \* from t where c2=’balu’;

->based on rowed concept data will pick up directly and get displayed instead of total table scan

**Types of indexes:**

* B tree index(less duplicate)
* Bitmap index (more duplicate )
* Unique index
* Simple
* Complex/composite index
* Functional index
* Cluster index
* **Simple index:**
* If you define an index on single column those indexes are called simple indexes

SQL>create index indsql on emp(sal);

* To find out whether the optimizer has chosen the index (or) not we have a DML command called ‘explain plan’
* With the help of explain plan you can find out the path choosen by the optimizer
* Explain plan populates the plan table with optimizer information (or) explain plan will make use of plan table for optimizer information
* So, to find out whether the optimizer preferred index or not we use ‘explain plan’

**SQL>**explain plan for select \* from emp where sal>1000;

**Output:**explained

**SQL>**desc plan\_table;

select operation, options, object\_name, search\_columns from plan\_table where object\_name=’emp’;

Here optimizer choosen the index by making use of rowed, if index is dropped, then full table scan occurs

|  |  |  |
| --- | --- | --- |
| **Opration** | Options | Object\_nme |
| Table access | By index rowed | Emp |

Once we drop the index optimizer go for complete table scan as shown in the following

**EG:**

**SQL>**drop index indsal;

**SQL>**delete from plan\_table;

**SQL>**explain plan for select \* from emp where sal>1000;

|  |  |  |
| --- | --- | --- |
| **Operation** | Options | Object\_name |
| Table access | Full(complete table scan) | Emp |

* **Complex index:**

In this we define indexes on more than one column

**Note:** Defining an index on indexed column is not possible

**SQL>**create index indsql on emp(sal); //invalid

**Eg:**

**SQL>**create index comind on emp(sal,deptno);

**SQL>**select \* from emp where sal>2000 and deptno>10;

**SQL>**select \* from emp where deptno>10;

**Note:**here optimizer will make use of index even through you won’t

mention all the index column in where clause this is due to because of ‘skip

column’ mechanism

* **unique index:**

To define an unique index on a column that column should not contain

duplicate values.

**Note:** Without using PK and unique constraints we can restrict the user not to provide duplicate values on a column by providing unique index immediately after defining a table

Attempting on define unique index on a column which is having duplicate

data ends in error.

**SQL>**create unique index uniindex on emp(ename);

* **Non-unique index:** If index is creating on duplicate values i.e

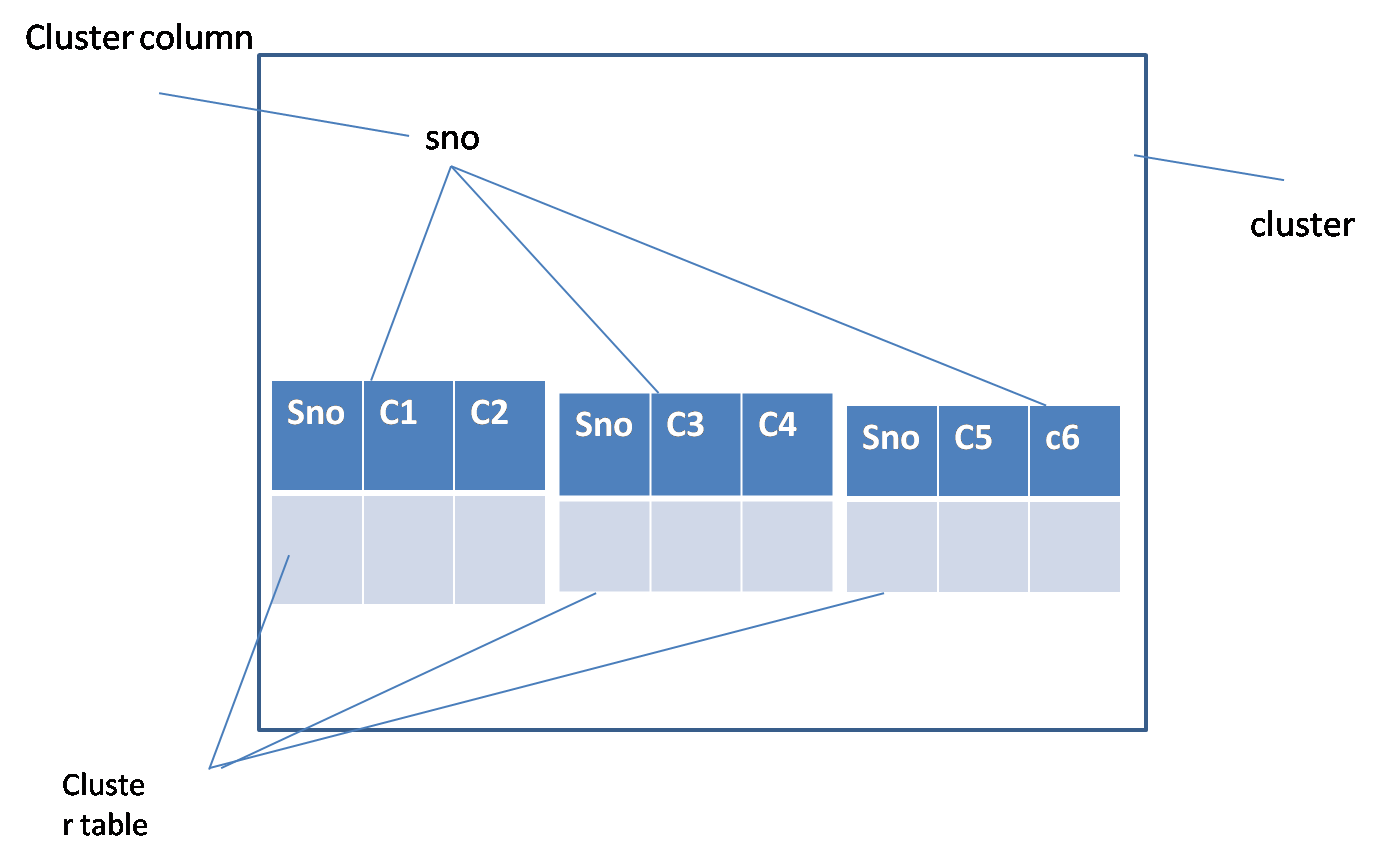
**SQL>**create unique index unidex2 on emp(deptno);

It is invalid statement because deptno is having duplicate data

* **Bitmap index:**

**SQL>**create bitmap index bindex on emp(deptno);

* **Cluster index:**
* It is logical memory area in which related and frequently tables are placed together binding with a cluster common column
* The concept of cluster and cluster indexes is useful to increase the performance.



**SQL>**create cluster cl(sno number(5));

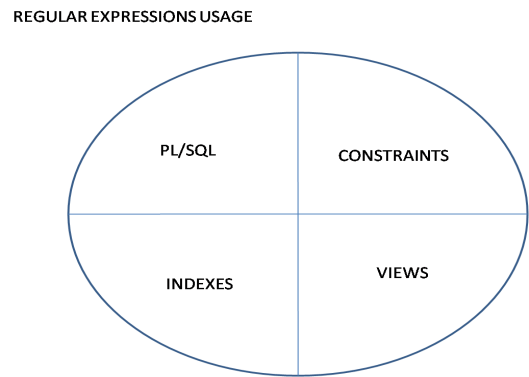
**SQL>**create table cltab(sno number(5), loc varchar2(10)) cluster cl(sno);

**SQL>**create table cltab1(sno number(5),name varchar2(10)) cluster cl(sno);

**SQL>**create index clindex on cluster cl;

**SQL>**desc user\_ind\_columns;

**Regular expressions(10g)**



It is simply called as pattern. They are the sequence of characters which

specifies the pattern in a given string. To make the complex search easy

we have regular expressions.

1. Regexp\_instr

->It displays the strating character (or) end character position based on the occurance (or) it is an extension to ‘instring’

1. Regexp substr
2. Regexp replace
3. Regexp like
4. Regexp count(11g)

* Displays the set of characters from a given string when characters are matched with pattern (or) extension to ‘substr’.
* Replace the text with another text when pattern is matched in given string
* It simply returns Boolean value after performing a match if match is found return true else null.

**Meta characters:**

* Any single character

? Zero or one character

+ One or more characters

* Zero or more characters

/ Alternate symbol

[…] To specify optional characters

() Grouping an expression

^ Anchors the starting character of line

$ Anchors the end character of line

+?

\*?

??

{m} Match exactly m characters

{m,} Has to match minimum m and more no of characters

{m.n} Has to match m to n no of characters

\n Specifies nth expression

\ Simply stands for backslash and also works as escape character

[:alnum:] Represents alpha numeric

[.:alpha:] Represents alpha characters only

[:digit:] Represents only digits

[:punct!] Represents punctuation

[:cntrl:] Represents control characters

[:upper:] Represents upper cases

[:lower:] Represents lower cases

[:print:] Represents printable character

[:space:] Represents space

[:graph] Represents graphical characters

[0-9] Range of digits

[1-9] Range of digits

[A-Z] Range of characters in upper case

[a-z] Range of characters in lower case

[^…..] Negation

[.coll.] Specifie collation

[=eq=] Equivalent

\d Digit

\D Non digit

\w Word

\W Non word

\s Space

\S No space

\A

\z

\Z

\x

**Match parameter:**

I Case insensitive

C Case sensitive(default)

M Multiline

N Newline

X Ignore space

**Syntax:**

Regexp-instr(str,patt[,pas[,oc[,ropt[,mp[,subexp]]]]])

Regexp-substr(str,patt[,pos[,oc[,mp[,subexp]]]])

Regexp-replace(str,patt[,reppatt[,pos[,oc[,mp]]]])

Regexp-like(str,patt[,mp]);

Regexp-count(str,patt[,pos]);

Str-source string

Patt-pattern

Pos-position

Oc-occurance

Ropt-return option(0,1) default 0

Mp-match parameter(I,c,m,n,x)

Subexp-subexpression(11g)(1-9)

Reppatt-replace pattern

**Regexp-substr:**

**SQL>**select regexp-substr(name,’(\(\d{3}\)[-/.])(\/’) from regtab;

**SQL>**select regexp-substr(name,’\(\d{3}\)[-1.]\(\d{3}\)[-/.]\(\d{3}\)’)

**Output:**(123)-(456)-(789)

(123)-(456)-(789)

(123)-(456)-(789)

**SQL>**select rgexp-replace(‘x@gmail.comy@gmail.com’,’@’,’#’,1,1) from dual;

**SQL>**select rgexp-replace(‘x@gmail.comy@gmail.com’,’@’,’#’,1,2) from dual;

**SQL>**select rgexp-replace(‘x@gmail.comy@gmail.com’,’@’,’#’) from dual;

**Note:**If you don’t mention position and occurance it replaces with #

wherever @ is there.

**Output:**

1. X # gmail.com [y@gmail.com](mailto:y@gmail.com)
2. [x@gmail.com](mailto:x@gmail.com) y#gmail.com
3. X#gmail.com y#gmail.com

**SQL>**select regexp\_instr(‘x@gmail.com [y@gmail.com’,’@’,1,1,0](mailto:y@gmail.com','@',1,1,0)) from dual;

**SQL>**select regexp-instr(‘x@gmail.com [y@gmail.com’,’@’,’1,1,1](mailto:y@gmail.com','@','1,1,1));

**SQL>**select \* from regtab where regexp-like(name,’a/x’);

**SQL>**select \* from regtab where regexp-like(name,’^a/^b’);

**Note**: We’ve to use regular expressions in where clause

**Regular expressions are useful in following things:**

|  |  |
| --- | --- |
| PL/SQL | Views |
| Indexes | Constraints |

* Regular expression always looks for true condition, else return null.

**SQL>**select ename from emp where ename like ‘A%’ or ename like ‘s%’;

**SQL>**select ename from emp where regexp-like(ename,’^A/^s’);

**SQL>**select \* from regtab where regexp-like(name,’\s’);

**SQL>**select \* from regtab where regexp-like(name,’\s’);

**SQL>**select \* from regtab where regexp-like(name,’\d’);

**SQL>**select regexp-count(‘welcome’,’E’,1) from dual;

* It counts the records in a given string and if there is no match it returns ‘0’

**SQL>**select regexp-substr(‘abc’,’a b c’,1,1,’y’) from dual;