**JOINS:**

* Joins are used to retrieving the required data from multiple tables.
* Join statements can be written in two ways,

1. Non-ansi joins(oracle8i version)

> equi join

> non-equi join

> self join

2. Ansi joins(oracle9i version)

> inner join

> outer joins

* left outer join
* right outer join
* full outer join

> cross join

> natural join

|  |  |
| --- | --- |
| **NON-ANSI JOINS** | **ANSI JOINS** |
| 1. not a portability | 1. portability |
| 2. Join tables based on "where clause" condition. | 2. Join tables with "on clause" condition. |

**Syntax For Non-Ansi Joins:**

SELECT \* FROM <TN1>,<TN2> WHERE <JOINING CONDITION>;

**Syntax For Ansi Joins:**

SELECT \* FROM <TN1> <JOIN KEY> <TN2> ON <JOINING CONDITION>;

**Equi Join / Inner Join:**

* Retrieving data from multiple tables based on an "=" operator.
* We should have a common column in both tables and those columns datatypes must match.
* Having relationship between tables are optional.
* It always retrieves matching rows from tables.

**Syntax:**

WHERE <TN1>.<COMMON COLUMN NAME> = <TN2>.<COMMON COLUMN NAME>

(OR)

ON <TN1>.<COMMON COLUMN NAME> = <TN2>.<COMMON COLUMN NAME>

**EX:1**

SQL> SELECT \* FROM COURSE;

CID CNAME CFEE

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

1 JAVA 5000

2 ORACLE 2000

3 .NET 4500

SQL> SELECT \* FROM STUDENT;

STID SNAME CID

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

101 SMITH 1

102 ALLEN 2

103 JAMES 2

104 WARD NULL

**Waq to retrieve student along with the corresponding course details from tables by equi join?**

**NON-ANSI:**

SQL> SELECT \* FROM STUDENT,COURSE WHERE STUDENT.CID=COURSE.CID; (OR)

SQL> SELECT \* FROM STUDENT S,COURSE C WHERE S.CID=C.CID; (OR)

SQL> SELECT SNAME,CNAME FROM STUDENT S,COURSE C WHERE S.CID=C.CID;

**ANSI:**

SQL> SELECT \* FROM STUDENT INNER JOIN COURSE ON STUDENT.CID=COURSE.CID; (OR)

SQL> SELECT \* FROM STUDENT S INNER JOIN COURSE C ON S.CID=C.CID; (OR)

SQL> SELECT SNAME,CNAME FROM STUDENT S INNER JOIN COURSE C ON S.CID=C.CID;

**Rule Of Joins:**

* A row in a table is comparing with all rows of another table.

**EX:2**

**Waq to retrieve students who are selected "oracle" course from student, course tables by using equi join?**

**ANSI**:

SQL> SELECT \* FROM STUDENT INNER JOIN COURSE ON STUDENT.CID=COURSE.CID AND CNAME='ORACLE';

**NON-ANSI:**

SQL> SELECT \* FROM STUDENT,COURSE WHERE STUDENT.CID=COURSE.CID AND CNAME='ORACLE';

**EX:3**

**Waq to display employees who are working in Chicago from emp, dept tables by using equi join?**

**ANSI**:

SQL> SELECT \* FROM EMP E INNER JOIN DEPT D ON E. DEPTNO=D.DEPTNO AND LOC = 'CHICAGO';

**NON-ANSI:**

SQL> SELECT \* FROM EMP E , DEPT D WHERE E.DEPTNO=D.DEPTNO AND LOC = 'CHICAGO';

**EX:4**

**Waq to display sum of salaries of each department from emp, dept tables by using equi join?**

**ANSI:**

SQL> SELECT DNAME,SUM(SAL) FROM EMP E INNER JOIN DEPT D ON E.DEPTNO = D.DEPTNO GROUP BY DNAME;

**NON-ANSI**

SQL> SELECT DNAME,SUM(SAL) FROM EMP E,DEPT D WHERE E.DEPTNO = D.DEPTNO GROUP BY DNAME;

**EX:5**

**Waq to display departments in which department sum of salary is more**

**then 10000 from emp, dept table by using equi join?**

**ANSI:**

SQL> SELECT DNAME, SUM(SAL) FROM EMP E INNER JOIN DEPT D ON E.DEPTNO = D.DEPTNO GROUP BY DNAME HAVING SUM(SAL)>10000;

**NON-ANSI:**

SQL> SELECT DNAME, SUM(SAL) FROM EMP E, DEPT D WHERE E.DEPTNO = D.DEPTNO GROUP BY DNAME HAVING SUM(SAL)>10000;

**OUTER JOINS:**

1. **LEFT OUTER JOIN:**

* Retrieving matching rows from both tables and unmatching rows from left side table only.

**ANSI:**

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

**NON-ANSI:**

* When we implement outer joins in non-ansi format then we should use a join operator is called as "(+)".

SQL> SELECT \* FROM STUDENT S,COURSE C WHERE S.CID=C.CID(+);

1. **RIGHT OUTER JOIN:**

* Retrieving matching rows from both tables and unmatching rows from right side table only.

**ANSI:**

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

**NON-ANSI:**

SQL> SELECT \* FROM STUDENT S,COURSE C WHERE S.CID(+)=C.CID;

1. **FULL OUTER JOIN:**

* Retrieving matching and also unmatching rows from both tables.
* It is combination of left outer and right outer.

**ANSI:**

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

**NON-ANSI:**

SQL> SELECT \* FROM STUDENT S,COURSE C WHERE S.CID=C.CID(+) UNION SELECT \* FROM STUDENT S,COURSE C WHERE S.CID(+)=C.CID;

NON-EQUI JOIN:

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

- retrieving data from multiple tables based on any

operator except an "=" operator.

- we can use the following operators are:

<,>,<=,>=,!=,between,and,....etc

EX:

SQL> SELECT \* FROM TEST1;

SNO NAME

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

1 SMITH

2 MILLER

SQL> SELECT \* FROM TEST2;

SNO SAL

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

1 25000

3 15000

EX:

ANSI:

SQL> SELECT \* FROM TEST1 T1 JOIN TEST2 T2

2 ON T1.SNO>T2.SNO;

NON-ANSI:

SQL> SELECT \* FROM TEST1 T1,TEST2 T2 WHERE T1.SNO>T2.SNO;

EX:

waq to display employees who salary is between low salary and

high salary?

ANSI:

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

2 ON SAL BETWEEN LOSAL AND HISAL;

(OR)

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

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

NON-ANSI:

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

2 WHERE SAL BETWEEN LOSAL AND HISAL;

(OR)

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

2 WHERE (SAL>=LOSAL) AND (SAL<=HISAL);

CROSS JOIN:

===========

-joining two or more than two tables without any condition.

- a row in a table is comparing with all rows of the second

table.for example a table is having (m) no.of rows and another

table is having (n) no.of rows then the result is mxn rows.

Ex:

non-ansi:

=========

SQL> SELECT \* FROM STUDENT,COURSE;

ansi:

=====

SQL> SELECT \* FROM STUDENT CROSS JOIN COURSE;

Ex:

SQL> SELECT \* FROM ITEMS1;

SNO INAME PRICE

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

1 PIZZA 160

2 BURGER 80

SQL> SELECT \* FROM ITEMS2;

SNO INAME PRICE

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

101 COCACOLA 25

102 PEPSI 20

ANSI:

SQL> SELECT I1.INAME,I1.PRICE,I2.INAME,I2.PRICE,

2 I1.PRICE+I2.PRICE TOTAL\_AMOUNT FROM

3 ITEMS1 I1 CROSS JOIN ITEMS2 I2;

(OR)

NON-ANSI:

SQL> SELECT I1.INAME,I1.PRICE,I2.INAME,I2.PRICE,

2 I1.PRICE+I2.PRICE TOTAL\_AMOUNT FROM

3 ITEMS1 I1,ITEMS2 I2;

INAME PRICE INAME PRICE TOTAL\_AMOUNT

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

PIZZA 160 COCACOLA 25 185

NATURAL JOIN:

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

- it is similar to equi join but avoiding duplicate columns

from the result set.

- when we use natural join there is no need to write a

join condition by explicitly(user) because internall oracle

db server will preparing a join condition based on common

column along with "=" operator.

- when we use natural join on tables then we should have

a common column name in both tables it is mandatory.

ansi:

SQL> SELECT \* FROM STUDENT S NATURAL JOIN COURSE C;

non-ansi:

SQL> SELECT c.cid,stid,sname,cname,cfee FROM STUDENT S,

COURSE C WHERE S.CID=C.CID;

SELF JOIN:

==========

- joining a table by itself is called as "self join".

(or)

- comparing a table data by itself is called as "self join".

- it can perform on a single table only.

- self join can be implemented with the help of "alias names"

without alias names we cannot implement self join mechanism.

- we can create any no.of alias names on a single table

but each alias name should be different.

EX:

waq to display manager and their employees from emp table?

SQL> SELECT M.ENAME MANAGERS,E.ENAME EMPLOYEES

FROM EMP E,EMP M WHERE M.EMPNO=E.MGR;

MANAGERS EMPLOYEES

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

JONES SCOTT

JONES FORD

EX:

waq to display employees who are joined before their manager?

SQL> SELECT E.ENAME EMPLOYEE,E.HIREDATE E\_DOJ,

2 M.ENAME MANAGER,M.HIREDATE M\_DOJ FROM

3 EMP E,EMP M WHERE M.EMPNO=E.MGR AND E.HIREDATE<M.HIREDATE;

EX:

waq to display employees whose salary is more than their manager salary?

SQL> SELECT E.ENAME EMPLOYEE,E.SAL EMP\_SAL,

2 M.ENAME MANAGER,M.SAL MGR\_SAL FROM

3 EMP E,EMP M WHERE M.EMPNO=E.MGR AND E.SAL>M.SAL;

Ex:

waq to display employees who are working under "BLAKE" manager?

SQL> SELECT E.ENAME EMPLOYEES FROM EMP E,EMP M

2 WHERE M.EMPNO=E.MGR AND M.ENAME='BLAKE';

Ex:

waq to find out the manager of BLAKE employee?

SQL> SELECT M.ENAME MANAGER FROM EMP E,EMP M

2 WHERE M.EMPNO=E.MGR AND E.ENAME='BLAKE';

MANAGER

----------

KING

HOW TO JOIN MORE THAN TWO TABLES:

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

SYNTAX FOR NON-ANSI JOINS:

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

SELECT \* FROM <TN1>,<TN2>,<TN3>,........WHERE <JOIN COND1> AND <JOIN COND2>

AND <JOIN COND3> AND ......;

EQUI JOIN:

==========

SQL> SELECT \* FROM STUDENT;

SQL> SELECT \* FROM COURSE;

SQL> SELECT \* FROM REGISTER;

REGNO REGDATE CID

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

1 08-AUG-22 1

1 09-AUG-22 2

3 10-AUG-22 5

SQL> SELECT \* FROM STUDENT S,COURSE C,REGISTER R

2 WHERE S.CID=C.CID AND C.CID=R.CID;

SYNTAX FOR ANSI JOINS:

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

SELECT \* FROM <TN1> <JOIN KEY> <TN2> ON <JOIN COND1>

<JOIN KEY> <TN3> ON <JOIN COND2>

<JOIN KEY> <TN4> ON <JOIN COND3>

..............................

..............................

<JOIN KEY> <TNn> ON <JOIN CONn-1>;

INNER JOIN:

===========

SQL> SELECT \* FROM STUDENT S INNER JOIN COURSE C ON S.CID=C.CID

2 INNER JOIN REGISTER R ON C.CID=R.CID;

Day:26 09/08/2020

1. Waq to display employee who are joined before their manager?

SELECT E.ENAME EMPLOYEE,E.HIREDATE E\_DOJ,

M.ENAME MANAGER, M.HIREDATE M\_DOJ FROM EMP E, EMP M,

WHERE M.EMPNO=E.MGR AND E.HIREDATE<M.HIREDATE;

1. Waq to display employees whose salary is more than their managers salary?

SELECT E.ENAME EMPLOYEE, E.SAL EMP\_SAL,

M.ENAME MANAGER, M.SAL MGR\_SAL, FROM EMP E, EMP M

WHERE M.EMPNO = E.MGR AND E.SAL>M.SAL;

1. Waq to display employees who are working under ‘Blake’ manager?

SELECT E.ENAME EMPLOYEES FROM EMP E, EMP M

WHERE M.EMPNO=E.MGR AND M.ENMAE=’BLAKE’;

1. Waq to find out the manager of ‘Blake ‘ employee ?

SELECT M.ENAME MANAGER FROM EMP E, EMP M,

WHERE M.EMPNO=E.MGR AND ENAME=’BLAKE’;

**How to join More than 2 Tables?**

**Syntax for non-Ansi joins:**

SELECT \* FROM <TN1>,<TN2>,<TN3>………..

WHERE <JOIN COND1> AND <JOIN COND2> AND <JOIN COND3> AND….;

**EQUI JOIN:**

SELECT \* FROM STUDENT;

SELECT \* FROM COURSE;

SELECT \* FROM REGISTER;

SELECT \* FROM STUDENT S, COURSE C, REGISTER R

WHERE S.CID=C.CID AND C.CID=R.CID;

**Syntax for Ansi joins:**

SELECT \* FROM <TN1> <JOIN KEY> <TN2> ON <JOIN COND1>

<JOIN KEY> <TN3> ON <JOIN COND2>

<JOIN KEY> <TN4> ON <JOIN COND3>

…………………………………………………………

<JOIN KEY> <TNn> ON <JOIN CONDn-1>;

Inner join:

SELECT \* FROM STUDENTS INNER JOIN COURSE C ON S

**DATA INTEGRITY:**

* To maintain accurate and consistency data in database tables.

1. Declarative Integrity
   1. By using “Constraints” (SQL)
2. Procedural Integrity
   1. By Using “tiggers” (PL/SQL)

**Constraints:**

* To restricted/enforce unwanted data into a table.
  + - Unique
    - Not null
    - Check
    - Primary Key
    - Foreign Key/ References
    - Default
  + Can be defined at two levels on a table.

1. Column level:
   1. Can be defined to each column wise
   2. Syntax:

CREATE TABLE <TN> (<CN1> <dt>[SIZE] <CONSTRAINT TYPE>,

<CN2> < dt >[SIZE] <CONSTRAINT TYPE>,

<CN3> < dt >[SIZE] <CONSTRAINT TYPE>,

…………………………………………);

1. Table Level:
   1. Can be defined after all columns declared. (end of the table)
   2. Syntax:

CREATE TABLE <TN> ( <CN1> <dt>[SIZE],

<CN2> < dt >[SIZE],

<CN3> < dt >[SIZE],

………………………………<CONSTRAINT TYPE>(<CN1>,<CN2>,….));

Note:

Table level constraints are also called as “composite constraint”.

**UNIQUE:**

* Not allowed duplicate values but allowed NULLs.
* Column level: CREATE TABLE TEST1 ( SNO INT UNIQUE, NAME VARCHAR2(10) UNIQUE ) ;

INSERT INTO TEST1 VALUES(1,’A’); 🡺 Allowed

INSERT INTO TEST1 VALUES(1,’B’); 🡺Not Allowed

INSERT INTO TEST1 VALUES(null, null);🡺 Allowed

* Table Level: CREATE TABLE TEST2 ( SNO INT, NAME VARCHAR2(10), UNIQUE(SNO,NAME)) ;

INSERT INTO TEST2 VALUES(1,’IND’); 🡺Allowed

INSERT INTO TEST2 VALUES(1,’AUS’); 🡺Allowed

INSERT INTO TEST2 VALUES(1,’IND’); 🡺Not Allowed

**NOT NULL:**

* Restricted NULLs but allows duplicates.
* Can defined at column level only.

CREATE TABLE TEST3 (SNO INT NOT NULL, NAME VARCHAR2(10) NOT NULL);

INSERT INTO TEST3 VALUES (NULL, NULL); 🡺 Not allowed.

Day:27 10/08/2020

**PRIMARY KEY:**

* It is a combination of unique and not null constraint.
* Restricted duplicates and nulls.
* A table is having only one primary key.

Column Level:

CREATE TABLE TEST3(EID INT PRIMARY KEY, ENAME VARCHAR2(10));

SQL> INSERT INTO TEST3 VALUES(1,'SMITH'); 🡺 Allowed

SQL> INSERT INTO TEST3 VALUES(2,'ALLEN'); 🡺 Allowed

SQL> INSERT INTO TEST3 VALUES(NULL,'ALLEN'); 🡺 Not Allowed (cannot insert NULL)

Table Level:

CREATE TABLE TEST4 (SNO INT, CNAME VARCHAR2(10), PRIMARY KEY (SNO, CNAME));

SQL> INSERT INTO TEST4 VALUES(1,'C'); 🡺Allowed

SQL> INSERT INTO TEST4 VALUES(1,'C');🡺Not Allowed

SQL> INSERT INTO TEST4 VALUES(1,'C++');🡺Allowed

**Check:**

* To check values with user defined condition.

Column Level:

SQL> INSERT INTO TEST6 VALUES('A',5000); 🡺 Not allowed (check constraint violated)

SQL> INSERT INTO TEST6 VALUES('A',6000); 🡺 allowed

Table Level:

SQL> CREATE TABLE TEST7(ENAME VARCHAR2(10), SAL NUMBER (10), CHECK(ENAME=UPPER(ENAME) AND SAL>=8000));

SQL> INSERT INTO TEST7 VALUES ('SAI', 7500);🡺 check constraint violated

SQL> INSERT INTO TEST7 VALUES('Sai', 8000);🡺 check constraint violated

SQL> INSERT INTO TEST7 VALUES('SAI', 8000);🡺 Allowed

**Foreign Key/Reference:**

* It is used to create relationships between tables.
* Basic rules:
  + We have a common column in both tables and those column’s datatype must be match.
  + The common column is primary key of 1st tables and foreign key of 2nd table.
  + A primary key table is called as parent and the foreign key table is called as child.
  + A foreign key column must be a primary key column value only.
  + By default, a foreign key column is accepting duplicates and null values.

SQL> CREATE TABLE DEPT1(DEPTNO INT PRIMARY KEY, DNAME VARCHAR2(10));

SQL> INSERT INTO DEPT1 VALUES(2,'TESTING');

SQL> INSERT INTO DEPT1 VALUES(1,'DB');

SQL> SELECT \* FROM DEPT1;

DEPTNO DNAME

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

2 TESTING

1 DB

SQL> SELECT \* FROM EMP1;

EID ENAME DEPTNO

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

101 A 1

102 B 1

103 C 2

104 D

SQL> CREATE TABLE EMP1(EID INT, ENAME VARCHAR2(10), DEPTNO INT REFERENCES

DEPT1(DEPTNO));

SQL> INSERT INTO EMP1 VALUES(101,'A', 1);🡺Allowed

SQL> INSERT INTO EMP1 VALUES(102,'B', 1); 🡺Allowed

SQL> INSERT INTO EMP1 VALUES(103,'C', 2); 🡺Allowed

SQL> INSERT INTO EMP1 VALUES(104,'D', NULL); 🡺Allowed

SQL> INSERT INTO EMP1 VALUES(105,'D', 3);🡺 Not Allowed( integrity constraint violated - parent key not found.

Note:

* Once we created relationship between tables there are two rules come into picture.
* Rule-1(Insertion):
  + We cannot insert values into a foreign key column which was not found in primary key column of parent table.
* Rule-2(Deletion):
  + We cannot delete a row from parent table that parent is having child rows in child table without addressing to child.
  + EX: SQL> DELETE FROM DEPT1 WHERE DEPTNO=1; integrity constraint violated - child record found

Day:28 11/08/2020

* How to address child table?
  + When we address to a child table then use the following rules are called as ‘cascade rules”.
    - * 1. On delete cascade

Once we delete a row from a parent then the corresponding child record also deleted from child table automatically.

SQL> CREATE TABLE DEPT2(DEPTNO INT PRIMARY KEY, DNAME VARCHAR2(10));

SQL> INSERT INTO DEPT2 VALUES(1,'DB');

SQL> INSERT INTO DEPT2 VALUES(2,'TESTING');

SQL> SELECT \* FROM DEPT1;

SQL> CREATE TABLE EMP2 (EID INT, ENAME VARCHAR2(10), DEPTNO INT REFERENCES

DEPT1(DEPTNO) ON DELETE CASCADE);

SQL> INSERT INTO EMP2 VALUES (101,'A', 1);

SQL> INSERT INTO EMP2 VALUES (102,'B', 1);

SQL> INSERT INTO EMP2 VALUES (103,'C', 2);

SQL> DELETE FROM DEPT2 WHERE DEPTNO=1; 🡺Allowed

* + - * 1. On delete set null

Once we delete a row from a parent then the corresponding child record reference key column values are converted into nulls from child table automatically.

SQL> CREATE TABLE DEPT3(DEPTNO INT PRIMARY KEY, DNAME VARCHAR2(10));

SQL> INSERT INTO DEPT3 VALUES(1,'DB');

SQL> INSERT INTO DEPT3 VALUES(2,'TESTING');

SQL> SELECT \* FROM DEPT3;

SQL> CREATE TABLE EMP3 (EID INT, ENAME VARCHAR2(10), DEPTNO INT REFERENCES

DEPT3(DEPTNO) ON DELETE SET NULL);

SQL> INSERT INTO EMP3 VALUES (101,'A', 1);

SQL> INSERT INTO EMP3 VALUES (102,'B', 1);

SQL> INSERT INTO EMP3 VALUES (103,'C', 2);

SQL> DELETE FROM DEPT3 WHERE DEPTNO=1;

**DATADICTIONARIES (OR) READ ONLY TABLES**

* When we install oracle software internally system creates some pre-defined tables are called as datadictionaries.
* These datadictionaries are stored the information which related to db objects are tables, synonyms, views, sequences, index, constraints, …. Etc
* If we want to view all datadictionaries in oracle DB then we following,

**Syntax:**

SELECT \* FROM DICT;

**Note:**

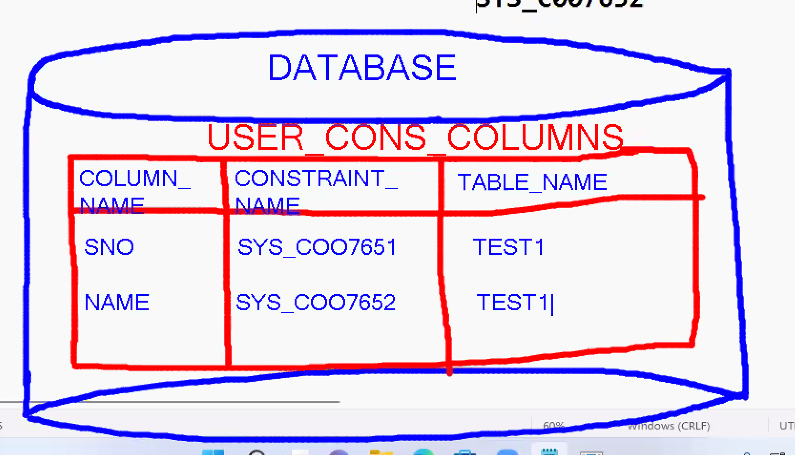
* If we want to view constraint name along with column name of a particular table in oracle then we use a datadictionary is called as ”user\_cons\_column”.

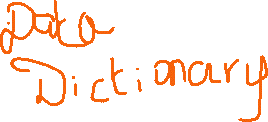
SQL> DESC USER\_CONS\_COLUMN;

SQL> SELECT COLUMN\_NAME,CONSTRAINT\_NAME FROM USER\_CONS\_COLUMNS

WHERE TABLE\_NAME='EMP1';

|  |  |
| --- | --- |
| **COLUMN\_NAME** | **CONSTRAINT\_NAME** |
| DEPTNO | SYS\_C007547 |





**USER DEFINED CONSTRAINT KEY NAMES:**

**Syntax:**

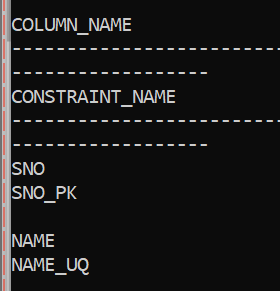
<COLUMN NAME> <DT>[SIZE] CONSTRAINT <USER DEFINED CONSTRAINT KEY NAME> <CONSTRAINT TYPE>

SQL> CREATE TABLE TEST8(SNO INT CONSTRAINT SNO\_PK PRIMARY KEY,

NAME VARCHAR2(10) CONSTRAINT NAME\_UQ UNIQUE);

SQL> SELECT COLUMN\_NAME, CONSTRAINT\_NAME FROM USER\_CONS\_COLUMNS

WHERE TABLE\_NAME='TEST8';



SQL> INSERT INTO TEST8 VALUES(1,'A');

SQL> INSERT INTO TEST8 VALUES(1,'B'); 🡺 unique constraint (DEEPI.SNO\_PK) violated

SQL> INSERT INTO TEST8 VALUES(2,'B');

SQL> INSERT INTO TEST8 VALUES(3,'B'); 🡺 unique constraint (DEEPI.NAME\_UQ) violated

Day:29 12/08/2020

**How to add constraint to an existing table?**

**Syntax:**

ALTER TABLE <TN> ADD CONSTRAINT <CONSTRAINT KEY NAME> <CONSTRAINT TYPE> (COLUMN NAME);

* + - * 1. **Adding Primary Key:**

SQL> CREATE TABLE PARENT (EID INT, ENAME VARCHAR2(10), SAL NUMBER (10));

SQL> ALTER TABLE PARENT ADD CONSTRAINT EID\_PK PRIMARY KEY(EID);

* + - * 1. **Adding unique, check constraint:**

SQL> ALTER TABLE PARENT ADD CONSTRAINT ENAME\_UQ UNIQUE(ENAME);

SQL> ALTER TABLE PARENT ADD CONSTRAINT SAL\_CHK CHECK(SAL>6000);

* + - * 1. **Adding Not Null constraint:**

**Syntax:**

ALTER TABLE <TN> MODIFY <CN> CONSTRAINT <CONSTRAINT KEY NAME> NOT NULL;

SQL> ALTER TABLE PARENT MODIFY ENAME CONSTRAINT ENAME\_NN NOT NULL;

* + - * 1. **Adding Foreign Key:**

**Syntax:**

ALTER TABLE <TN> MODIFY <CN> CONSTRAINT <CONSTRAINT KEY NAME> FOREIGN KEY (COMMON COLUMN OF CHILD) REFERENCES <PARENT TABLE NAME> (COMMON COLUMN OF PARENT) ON DELETE CASCADE / ON DELETE SET NULL;

SQL> CREATE TABLE CHILD (DNAME VARCHAR2(10), EID INT);

SQL> ALTER TABLE CHILD ADD CONSTRAINT EID\_FK FOREIGN KEY(EID) REFERENCES

PARENT(EID) ON DELETE CASCADE;

**How to Drop constraint to an existing table?**

**Syntax:**

ALTER TABLE <TN> DROP CONSTRAINT <CONSTRAINT KEY NAME> ;

1. **Drop a Primary Key:**

**Case 1: Without Relationship**

SQL> ALTER TABLE PARENT DROP CONSTRAINT EID\_PK;

**Case 1: With Relationship**

SQL> ALTER TABLE PARENT DROP CONSTRAINT EID\_PK CASCADE;

1. **Drop Unique, Check, Not Null Constraints:**

SQL> ALTER TABLE PARENT DROP CONSTRAINT ENAME\_UQ;

SQL> ALTER TABLE PARENT DROP CONSTRAINT SAL\_CHK;

SQL> ALTER TABLE PARENT DROP CONSTRAINT ENAME\_NN;

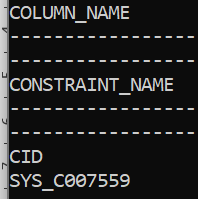
**How to rename a constraint Name?**

**Syntax:**

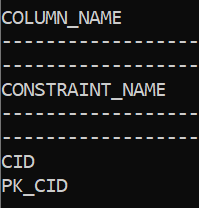
ALTER TABEL <TN> RENAME <CONSTRAINT> <OLD CONSTRAINT KEY NAME> TO <NEW CONSTRAINT KEY NAME>;

SQL> CREATE TABLE TEST9(CID INT PRIMARY KEY);

SQL> SELECT COLUMN\_NAME, CONSTRAINT\_NAME FROM USER\_CONS\_COLUMNS WHERE TABLE\_NAME='PARENT';



SQL> ALTER TABLE TEST9 RENAME CONSTRAINT SYS\_C007559 TO PK\_CID;



**How to disable/enable constraint on a table:**

**Syntax:**

ALTER TABLE<TN> DISABLE CONSTRAINT <CONSTRAINT KEY NAME>;

**Note:**

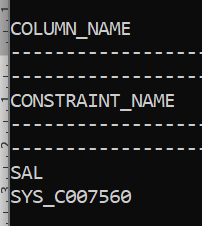
* By default, all the constraint are working mode(enable).
* If we want to disable a constraint then we use disable keyword.

SQL> CREATE TABLE TEST10(ENAME VARCHAR2(10), SAL NUMBER(10) CHECK(SAL<10000));

SQL> INSERT INTO TEST10 VALUES('A', 1001);

SQL> INSERT INTO TEST10 VALUES('B', 2001);

SQL> SELECT COLUMN\_NAME, CONSTRAINT\_NAME FROM USER\_CONS\_COLUMNS WHERE TABLE\_NAME='TEST10;



**How to disable a constraint:**

SQL> ALTER TABLE TEST10 DISABLE CONSTRAINT SYS\_C007560;

SQL> INSERT INTO TEST10 VALUES ('B', 11000);

**How to enable a constraint:**

**Syntax:**

ALTER TABLE<TN> ENABLE NOVALIDATE CONSTRAINT <CONSTRAINT KEY NAME>;

* If we want to enable a check constraint on a column then we use a pre-defined statement is “NOVALIDATE”.
* NOVALIDATE statement is not comparing existing values in a column but comparing newly inserting values.

SQL> ALTER TABLE TEST10 ENABLE NOVALIDATE CONSTRAINT SYS\_C007560;

SQL> INSERT INTO TEST10 VALUES ('B', 11000); 🡺 Not Allowed now.

**Default Constraint:**

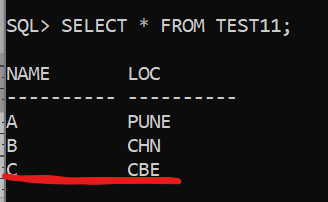
* It is used to assign a user defined default value to a column.

SQL> CREATE TABLE TEST11 (NAME VARCHAR2(10), LOC VARCHAR2(10) DEFAULT 'CBE');

SQL> INSERT INTO TEST11 VALUES('A','PUNE');

SQL> INSERT INTO TEST11 VALUES('B','CHN');

SQL> INSERT INTO TEST11(NAME) VALUES('C');



**How to add a default value to an existing table column:**

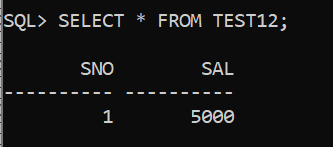
**Syntax:**

ALTER TABLE<TN> MODIFY <CN> DEFAULT <VALUE>;

SQL> CREATE TABLE TEST12(SNO INT, SAL NUMBER(10));

SQL> ALTER TABLE TEST12 MODIFY SAL DEFAULT 5000;

SQL> INSERT INTO TEST12(SNO) VALUES(1);



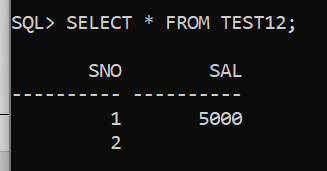
**How to remove a default value to an existing table column:**

**Syntax:**

ALTER TABLE<TN> MODIFY <CN> DEFAULT NULL;

SQL> ALTER TABLE TEST12 MODIFY SAL DEFAULT NULL;

SQL> INSERT INTO TEST12(SNO) VALUES (2);



Day:30 13/08/2020

**TRANSACTIONAL CONTROL LANGUAGE (TCL)**

* Transaction is nothing but to perform some operation over database.
* To manage Transactions over database then we use the following 3 commands
  + COMMIT
  + ROLLBACK
  + SAVEPOINT

**COMMIT:**

* This command is used to make a transaction is permanent.
  + Implicit Commit Transaction
  + Explicit Commit Transaction.
* **Implicit Commit Transaction:** 
  + These transactions are committed by system.
  + Ex: DDL Commands
* **Explicit Commit Transaction:** 
  + These Transactions are committed by user as per our need.
  + Ex: DML commands
* **Syntax:**
  + COMMIT;

EX:

SQL> CREATE TABLE BRANCH (BCODE INT, BNAME VARCHAR2(10), BLOC VARCHAR2(10));

SQL> INSERT INTO BRANCH VALUES (101,'HDFC','HYD');

SQL> INSERT INTO BRANCH VALUES (102,'SBI','PUNE');

SQL> COMMIT;

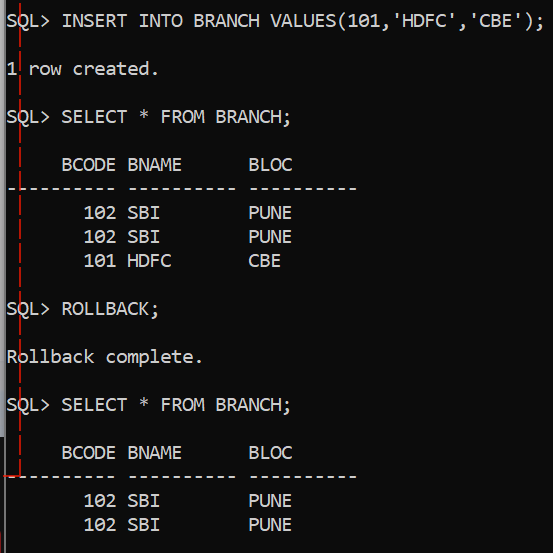
**ROLLBACK:**

* To cancel a transaction.
* But once a transaction is committed then we cannot rollback.
* Syntax: ROOLBACK;

EX:

SQL> DELETE FROM BRANCH WHERE BCODE=101;

SQL> ROLLBACK; 🡺 Rollback complete.



**SAVEPOINT:**

* When we created a savepoint internally system is allocating a special memory to pointer in instance and will store row(s)which we rollback(cancel).
* **Syntax to create a savepoint**: SAVEPOINT<POINTER NAME>
* **Syntax to Rollback a savepoint**: ROLLBACK TO <POINTER NAME>

Day:31 16/08/2020

**ACID PROPERTIES**

* By default, all databases are having ACID Properties to maintain accurate and consistency data.

**A-Atomicity:** Atomic=single

**C-consistency:**

**I-Isolation:** Every transaction is independent.

**D-durability:**

**SUBQUERY**

* A query inside another query is called as “Subquery”.

**SubQuery = Outer Query + Inner Query**

* **Syntax**:

SELECT \* FROM <TN> WHERE <CONDITION> (SELECT\* FROM-----( SELECT\* FROM---));

* A subquery statement is having two more queries those are
  + Outer Query/Parent Query/Main Query
  + Inner Query/Child Query/Subquery
* As per the execution process of a subquery statement it again two types
  + Non-correlated Subquery
  + Co-related Subquery

**Non-correlated Subquery:**

* In this mechanism first inner query is executed and later outer query will execute.
  1. Single row Subquery
  2. Multiple row Subquery
  3. Multiple column Subquery
  4. Inline view

1. **Single row Subquery:**

* When a subquery returns a single value is called as “SRSQ”.
* We can use the following operators,
* **= , < , > , <= , >= , !=**

**Ex:** Waq to display employee details who are earning 1st highest salary?

SQL> SELECT \* FROM EMP WHERE SAL = (SELECT MAX(SAL) FROM EMP);

**Ex:** Waq to display senior most employee details from emp?

SQL> SELECT \* FROM EMP WHERE HIREDATE= (SELECT MIN(HIREDATE) FROM EMP);

**Ex:** Waq to display employee whose salary is more than the maximum salary of the job salesman?

SQL> SELECT \* FROM EMP WHERE SAL>(SELECT MAX(SAL) FROM EMP WHERE JOB='SALESMAN');

Day:32 17/08/2020

**Ex:** Waq to display employees who are earning 2nd highest salary?

SQL> SELECT \* FROM EMP WHERE SAL=(SELECT MAX(SAL) FROM EMP WHERE SAL<(SELECT MAX(SAL) FROM EMP));

**Ex:** Waq to display employees who are earning 3rd highest salary?

SQL> SELECT \* FROM EMP WHERE SAL =(SELECT MAX(SAL) FROM EMP WHERE SAL<(SELECT MAX(SAL) FROM EMP WHERE SAL<(SELECT MAX(SAL) FROM EMP)));

|  |  |
| --- | --- |
| **Nth** | **N+1** |
| 1st | 2Q |
| 2nd | 3Q |
| 3rd | 4Q |
| 30th | 31Q |
| 150th | 151Q |

**HOW TO OVERCOME THE ABOVE PROBLEM?**

1. **Multiple row subquery:**

* when a subquery returns more than one value.
* Can use the following operators "IN, ANY, ALL".

**IN:**

* The IN operator allows you to specify multiple values in a WHERE clause.
* The IN operator is a shorthand for multiple OR conditions.

**Ex:** Waq to display employees whose job is same as the job of the employee

"smith”, “martin"?

SQL> SELECT \* FROM EMP WHERE JOB IN (SELECT JOB FROM EMP WHERE

ENAME='SMITH' OR ENAME='MARTIN');

(OR)

SQL> SELECT \* FROM EMP WHERE JOB IN (SELECT JOB FROM EMP WHERE

ENAME IN ('SMITH','MARTIN'));

**Ex:** Waq to display employees who are getting minimum and maximum salary

from emp table?

SQL> SELECT \* FROM EMP WHERE SAL IN (SELECT MIN(SAL) FROM EMP UNION SELECT MAX(SAL) FROM EMP);

**Ex:** Waq to display employees who are getting maximum salary from

each job wise?

SQL> SELECT \* FROM EMP WHERE SAL IN(SELECT MAX(SAL) FROM EMP GROUP BY JOB);

"**ANY" OPERATOR:**

* It satisfies any one value in the given group of list values with

user defined condition value then it returns true otherwise false.

Ex: X (45) >ANY (10,20,30)

X=45 ------> TRUE

X=09 ------> FALSE

X=25 ------> TRUE

**"ALL" OPERATOR:**

* It satisfied all values in the given group of list values with user defined condition value then it returns true otherwise false.

Ex: X (45) >All (10,20,30)

X=45 ------> TRUE

X=09 ------> FALSE

X=25 ------> false

**Ex:** Waq to display employees whose salary is more than any one salesman?

SQL> SELECT \* FROM EMP WHERE SAL >ANY(SELECT SAL FROM EMP WHERE JOB='SALESMAN');

**Ex:** Waq to display employees whose salary is more than all salesman?

SQL> SELECT \* FROM EMP WHERE SAL >ALL(SELECT SAL FROM EMP WHERE JOB='SALESMAN');

1. **Multiple column subquery:**

* Comparing multiple columns values of inner query with multiple

columns values of outer query are called as "MCSQ".

**Syntax:**

SELECT \* FROM <TN> WHERE (<COLUMN NAME1>,<COLUMN NAME2>,........)

IN(SELECT <COLUMN NAME1>,<COLUMN NAME2>,.....................);

**Ex:** Waq to display employees whose job, salary is same as the job, salary

of the employee Scott?

SQL> SELECT \* FROM EMP WHERE (JOB, SAL) IN (SELECT JOB, SAL FROM EMP WHERE ENAME='SCOTT');

**Ex:** Waq to display employees whose job, mgr is same as the job, mgr

of the employee Allen?

SQL> SELECT \* FROM EMP WHERE (JOB, MGR) IN (SELECT JOB,MGR FROM EMP WHERE ENAME='ALLEN');

Day:33 18/08/2020

**PSEUDO COLUMNS:**

These are working just like a table column.

**Two Types:**

1. ROWID
2. ROWNUM

**ROWID:**

* Whenever we insert a new row data into a table internally system will generate ROWIDs to each row wise.
* These ROWID’s are saved in database automatically so that ROWIDs are permanent id’s.

**Ex:** SQL> SELECT ROWID, ENAME FROM EMP;

ROWID ENAME

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

AAASH0AAHAAAAFkAAA SMITH

AAASH0AAHAAAAFkAAB ALLEN

AAASH0AAHAAAAFkAAC WARD

SQL> SELECT ROWID, ENAME FROM EMP WHERE DEPTNO=10;

ROWID ENAME

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

AAASH0AAHAAAAFkAAG CLARK

AAASH0AAHAAAAFkAAI KING

AAASH0AAHAAAAFkAAN MILLER

SQL> SELECT MAX(ROWID) FROM EMP;

MAX(ROWID)

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

AAASH0AAHAAAAFkAAN

SQL> SELECT MIN(ROWID) FROM EMP;

MIN(ROWID)

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

AAASH0AAHAAAAFkAAA

**HOW TO DELETE MULTIPLE DUPLICATE ROWS EXCEPT THE ORIGINAL ROW FROM A TABLE:**

**EX:**

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

SQL> SELECT \* FROM TEST;

SNO NAME

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

1 A

1 A

1 A

2 B

3 C

3 C

4 D

4 D

4 D

**EX:**

SQL> DELETE FROM TEST WHERE ROWID NOT IN (SELECT MAX(ROWID)

FROM TEST GROUP BY SNO);

SQL> SELECT \* FROM TEST;

SNO NAME

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

1 A

2 B

3 C

4 D

**II) ROWNUM:**

* To generate row numbers to each row wise / to each group of

rows wise automatically. These numbers are not saved in DB so

that these are temporary numbers.

**EX:**

SQL> SELECT ROWNUM, ENAME FROM EMP;

SQL> SELECT ROWNUM, ENAME, DEPTNO FROM EMP WHERE DEPTNO=10;

**Note:**

* By using this pseudo column, we can perform "nth" and

"Top n" operations over a table.

**Ex:** Waq to fetch the first row from emp table by using rownum?

SQL> SELECT \* FROM EMP WHERE ROWNUM=1;

**Ex:** Waq to fetch the 2ND row from emp table by using rownum?

SQL> SELECT \* FROM EMP WHERE ROWNUM=2;

no rows selected

**NOTE:**

* Generally, rownum is always starts with 1 for each selected row from a table so that to overcome this problem we need to use the following two operators are "<, <= ".

**SOLUTION:**

SQL> SELECT \* FROM EMP WHERE ROWNUM<=2 MINUS SELECT \* FROM EMP WHERE ROWNUM=1;

**EX:** Waq to fetch 5th position row from emp by using rownum?

SQL> SELECT \* FROM EMP WHERE ROWNUM<=5 minus SELECT \* FROM EMP WHERE ROWNUM<=4;

**EX:** Waq to fetch from 5th row to 10th row from emp by using rownum?

SQL> SELECT \* FROM EMP WHERE ROWNUM<=10 minus SELECT \* FROM EMP WHERE ROWNUM<5;

**EX:** Waq to fetch top 5 rows from emp by using rownum?

SQL> SELECT \* FROM EMP WHERE ROWNUM<=5;

**EX:** Waq to fetch the last two rows from emp by using rownum?

SQL> SELECT \* FROM EMP WHERE ROWNUM<=14 minus SELECT \* FROM EMP WHERE ROWNUM<=12;

(OR)

SQL> SELECT \* FROM EMP MINUS SELECT \* FROM EMP WHERE ROWNUM<= (SELECT COUNT(\*)-2 FROM EMP);

Day:34 19/08/2020

1. **INLINE VIEW**:

* Providing a select query in place of table name in select statement is called as "inline view".

(or)

* Providing a select query in the "from clause" is also called "inline view".

**Syntax:**

SELECT \* FROM (<SELECT QUERY>);

**Note:**

* In Inline view mechanism the result of inner query will act as table for outer query.

**Purpose:**

* Generally, Subqueries are not allowed to use “ order by “ clause if we want to use “order by ” clause in subquery that should be inline view only.
* Generally, non-alias names are not allowed under where clause condition. If we want to use column alias names in where clause condition then that should be inline view only.

**EX:** Waq to display the first 5 highest salary row from emp table by using rownum along with inline view?

SQL> SELECT \* FROM EMP ORDER BY SAL DESC;

SQL> SELECT \* FROM (SELECT \* FROM EMP ORDER BY SAL DESC) WHERE ROWNUM<=5;

**Ex:** Waq to display the 5th highest salary row from emp table by using rownum along with inline view?

SQL> SELECT \* FROM (SELECT \* FROM EMP ORDER BY SAL DESC) WHERE ROWNUM<=5 MINUS SELECT \* FROM (SELECT \* FROM EMP ORDER BY SAL DESC) WHERE ROWNUM<5;

**Ex:** Waq to display annual salary of employees whose salary is more than 25000?

SQL> SELECT \* FROM (SELECT EMPNO, ENAME, SAL, SAL\*12 ANNSAL FROM EMP) WHERE ANNSAL>25000;

**BY USING ROWNUM ALIAS NAME:**

**Ex:** Waq to fetch 3rd, 5th, 9tn, 13th rows from emp table by using rownum alias name along with inline view?

SQL> SELECT \* FROM (SELECT EMPNO, ENAME, ROWNUM R FROM EMP) WHERE R IN (3,5,9,13);

SQL> SELECT \* FROM (SELECT ROWNUM R, EMP.\* FROM EMP) WHERE R IN (3,5,9,13);

**Ex:** Waq to fetch 10th row from emp table by using rownum alias name along with inline view?

SQL> SELECT \* FROM (SELECT ROWNUM R, EMP.\* FROM EMP) WHERE R=10;

**Ex:** Waq to fetch Even rows from emp table by using rownum alias name along with inline view?

SQL> SELECT \* FROM (SELECT ROWNUM R, EMP.\* FROM EMP) where MOD(R,2) =0;

**Ex:** Waq to fetch first and last rows from emp table by using rownum alias name along with inline view?

SQL> SELECT \* FROM (SELECT ROWNUM R, EMP.\* FROM EMP) WHERE R IN (1,14);

SQL> SELECT \* FROM (SELECT ROWNUM R, EMP.\* FROM EMP) WHERE R=1 OR R=14;

SQL> SELECT \* FROM (SELECT ROWNUM R, EMP.\* FROM EMP) WHERE R IN (1, (SELECT COUNT (\*) FROM EMP));

SQL> SELECT \* FROM (SELECT ROWNUM R, EMP.\* FROM EMP) WHERE R=1 OR R=(SELECT COUNT(\*)FROM EMP);

Day:35 20/08/2020

**ANALYTICAL FUNCTIONS:**

* To assigning rank numbers to each row wise / to each group of rows wise.
  1. RANK ()
  2. DENSE\_RANK ()

ENAME SALARY RANK () DENSE\_RANK ()

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

A 85000 1 1

B 72000 2 2

C 72000 2 2

D 68000 4 3

E 54000 5 4

F 42000 6 5

* The above both functions are used to assign a rank number to each row wise but rank () will skip the next rank number in the order whereas dense\_rank () will not skip the next rank number in the order.

**Syntax:**

ANALYTICAL FUNTION NAME () OVER ([PARTITION BY <CN>] ORDER BY <CN> < DESC/ASC>)

**Note:**

* Partition by Clause is Optional
* Order by is Mandatory

**Without Partition by Clause:**

SQL> SELECT EMPNO, ENAME, SAL, RANK () OVER (ORDER BY SAL DESC) RANKS FROM EMP;

SQL> SELECT EMPNO, ENAME, SAL, DENSE\_RANK () OVER (ORDER BY SAL DESC) RANKS FROM EMP;

**With Partition by Clause:**

SQL> SELECT EMPNO, ENAME, SAL, DEPTNO, RANK () OVER (PARTITION BY DEPTNO ORDER BY SAL DESC) RANKS FROM EMP;

SQL> SELECT EMPNO, ENAME, SAL, DEPTNO, DENSE\_RANK () OVER (PARTITION BY DEPTNO ORDER BY SAL DESC) RANKS FROM EMP;

**Ex:** Waq to display employees who are earning the 3rd highest salary from each deptno wise along with dense\_rank () by using inline view?

SQL> SELECT EMPNO, ENAME, SAL, DEPTNO FROM (SELECT EMPNO, ENAME, SAL, DEPTNO, DENSE\_RANK () OVER (PARTITION BY DEPTNO ORDER BY SAL DESC) RANKS FROM EMP) WHERE RANKS = 3;

**Ex:** Waq to display THE 4TH senior most employee from each job wise along with dense\_rank () by using inline view?

SQL> SELECT \* FROM (SELECT EMPNO, ENAME, JOB, HIREDATE, DENSE\_RANK () OVER (PARTITION BY JOB ORDER BY HIREDATE) RANKS FROM EMP) WHERE RANKS = 4;

1. **CO-RELATED SUBQUERY:**

* In this mechanism first outer query is executed and later inner query will be executed.

**Syntax to find out ‘Nth’ highest/lowest salary:**

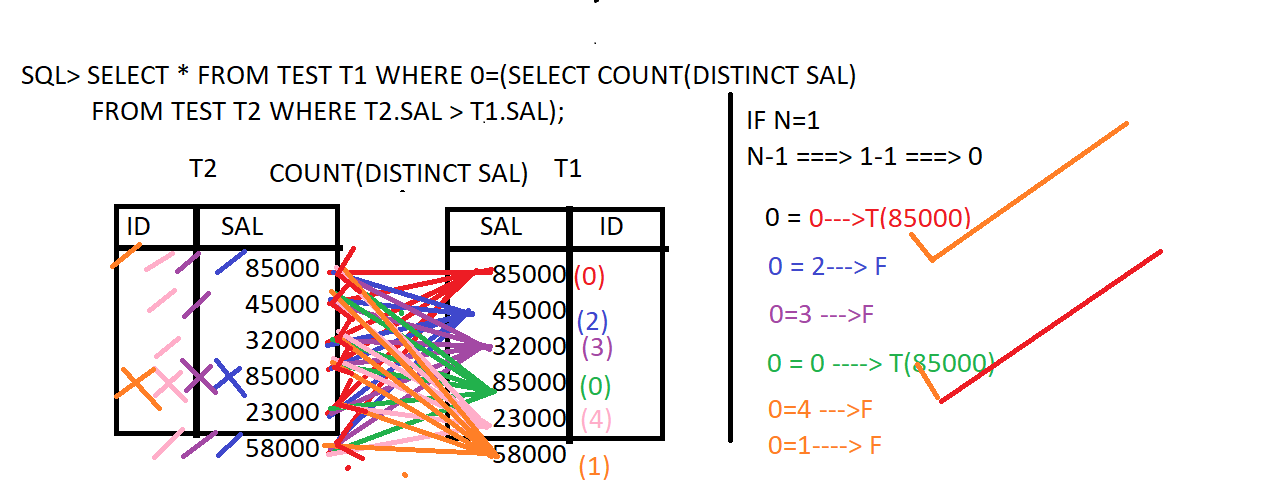
SELECT \* FROM <TN> <TAN1> WHERE N-1 = (SELECT COUNT (DISTINCT <CN>) FROM <TN> <TAN2> WHERE <TAN2>. <CN> </> <TAN1>. <CN>);

Here,

< ------- Low salary > -------------High salary

**Ex:** Waq to find out the 1st highest salary employee details?

SQL> select \* from emp e1 where 0 = (select COUNT (distinct sal) from emp e2 where e2.sal > e1.sal);



**Ex:** Waq to find out the 9th highest salary employee details?

SQL> select \* from emp e1 where 8= (select COUNT (distinct sal) from emp e2 where e2.sal > e1.sal);

Day:36 22/08/2020

**Ex:** Waq to find out the 1st LOWEST salary employee details?

SQL>SELECT \* FROM TEST T1 WHERE 0= (SELECT COUNT (DISTINCT SAL)

FROM TEST T2 WHERE T2.SAL < T1.SAL);

**Syntax to Display "Top N" High / Low Salaries:**

SELECT \* FROM <TN> <TAN1> WHERE N> (SELECT COUNT (DISTINCT <COLUMN NAME>)

FROM <TN > <TAN2> WHERE <TAN2>. <CN> < / > <TAN1>. <CN>);

**EX:** Waq to display top 3 highest salaries employee’s details?

SQL> SELECT \* FROM EMP E1 WHERE 2 = (SELECT COUNT (DISTINCT SAL) FROM EMP E2 WHERE E2.SAL>E1.SAL);

**EX:** Waq to display top 3 lowest salaries employee’s details?

SQL> SELECT \* FROM EMP E1 WHERE 2 = (SELECT COUNT (DISTINCT SAL) FROM EMP E2 WHERE E2.SAL<E1.SAL);

**EXITS OPERATOR:**

* It is a special operator which is used to check a row is existing in a table or not.
* If a row is existed in a table -🡪TRUE
* If a row is not existed in a table -🡪FLASE

**Syntax:**

WHERE EXISTS (<SELECT QUERY>);

**Ex:** Waq to display department details in which department employees are working?

SQL> SELECT \* FROM DEPT D WHERE EXISTS (SELECT DEPTNO FROM EMP E WHERE E. DEPTNO=D.DEPTNO);

**Ex:** Waq to display department details in which department employees are not working?

SQL> SELECT \* FROM DEPT D WHERE NOT EXISTS (SELECT DEPTNO FROM EMP E WHERE E. DEPTNO=D.DEPTNO);

**SCALAR SUBQUERY**

* Providing select query in place of columns in select statement.

(Or)

* Providing “Select Query” in select clause.
* The result of subquery (Inner Query) will act as a column.

**Syntax:**

SELECT (SELECT QUERY1), (SELECT QUERY1), (SELECT QUERY1), ……. FROM<TN>;

**Ex:**

SQL> SELECT (SELECT COUNT (\*) FROM DEPT), (SELECT COUNT (\*) FROM EMP) FROM DUAL;

**Ex:**

SQL> SELECT (SELECT SUM(SAL) FROM EMP WHERE DEPTNO =10) "10”,

(SELECT SUM(SAL) FROM EMP WHERE DEPTNO =20) "20",

(SELECT SUM(SAL) FROM EMP WHERE DEPTNO =30) "30" FROM DUAL;

10 20 30

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

8750 10875 9400

Day:37 23/08/2020

**DB SECURITY**

1. Authentication
2. Authorization

**AUTHENTICATION**

* To check / verify user credentials (username & password) before login / connect to db server.
* These credentials are created by DBA only.
* **Syntax:**

CREATE USER <USERNAME> IDENTIFIED BY <PASSWORD>;

**Ex:** SQL> CREATE USER U1 IDENTIFIED BY U1;

**Authorization**

1. To perform some operation over database.
2. Giving by DBA only.
3. By Using DCL commands.

**DCL Language:**

1. GRANT
2. REVOKE

**GRANT:**

* Granting permissions to user.

**Syntax:**

GRANT <PRIVILEGE NAME> TO <USERNAME>;

**REVOKE:**

* To cancel permissions of user.

**Syntax:**

REVOKE <PRIVILEGE NAME> FROM < USERNAME>;

**PRIVILEGES:**

* Privilege is nothing but permission
* These privileges are two types:
  1. System privileges
  2. Object Privileges.

**SYSTEM PRIVILEGES:**

* These privileges are giving by DBA only
* **Ex:** 
  + Connect
  + Create table
  + unlimited tablespace
  + create view
  + create synonym
  + create sequence
  + create index
  + materialized view……. etc,.

**Syntax:**

GRANT <SYSTEM PRIVILEGE NAME> TO <USERNAME>;

**Ex:** sql> conn system/tiger

sql> create user u1 identified by u1;

sql> grant connect, create TABLE, UNLIMITED tablespace to u1;

sql> conn u1/u1

sql> create table test1(sno int, name varchar2(10));

sql> insert into test1 values(1,'smith');

sql> update test1 set sno=1021 where sno=1;

sql> delete from test1 where sno=1021;

**HOW TO CANCEL A PERMISSION OF USER:**

sql> conn system/tiger

sql> revoke connect from u1;

SQL> CONN

Enter user-name: U1/U1

ERROR.

**OBJECT PRIVILEGES:**

* These permissions are giving by DBA or user to perform some operations over a table/DB object.
* Ex:
  + select
  + Insert
  + Update
  + Delete
  + “all” keyword.

**Syntax:**

GRANT <OBJECT PRIVILEGE NAME> ON <TN / DB OBJECT NAME> TO <USERNAME>;

**case-1: DBA to USER:**

**EX:** sql> conn system/tiger

sql> create user u2 identified by u2;

sql> grant connect to u2;

sql> conn u2/u2

sql> select \* from dept; ----🡪error.

sql> select \* from system. dept; ---🡪error.

SQL> INSERT INTO SYSTEM.DEPT VALUES (50,'DBA','HYD'); ---🡪error.

SQL> UPDATE SYSTEM.DEPT SET LOC='PUNE' WHERE DEPTNO=50; ---🡪error.

SQL> DELETE FROM SYSTEM.DEPT WHERE DEPTNO=10; ---🡪error.

**Granting object privileges to user u2:**

**EX:** sql> conn system /tiger

sql> grant select, INSERT, UPDATE, DELETE on dept to u2;

(or)

sql> grant all on dept to u2;

sql> select \* from system. DEPT; ---allowed

SQL> INSERT INTO SYSTEM.DEPT VALUES (50,'DBA','HYD’) --allowed

SQL> UPDATE SYSTEM.DEPT SET LOC='PUNE' WHERE DEPTNO=50--allowed

SQL> DELETE FROM SYSTEM.DEPT WHERE DEPTNO=10; ---allowed

**Cancel object privileges of user u2:**

**EX:** sql> conn system /tiger

sql> revoke select, insert, update, delete on dept from u2;

(or)

sql> revoke all on dept from u2;

sql> select \* from system. dept; -🡪 error.

Day 38 24/08/22

**Case-2: User to User:**

* If a user wants to give permission to another user then the first user should get permission from DBA with "with grant option" statement then only a user can give permission to another user otherwise not possible.

**EX:** sql> conn system/tiger

sql> create user u22 identified by u22;

sql> create user u33 identified by u33;

sql> grant connect to u22;

sql> grant connect to u33;

sql> grant select on dept to u22 with grant option;

sql> conn u22/u22

sql> select \* from system.dept;----allowed

sql> grant select on system.dept to u33;

Grant succeeded.

SQL> CONN

Enter user-name: U33/U33

Connected.

SQL> SELECT \* FROM SYSTEM.DEPT;---allowed

**ROLE:**

* A role is nothing but to assign same privileges to group of users who are working on same project.
* These roles are created by DBA only.

**Ex:**

X-PROJECT

|

MOD1 MOD2 MOD3

| | |

3E 5E 3E

(A,B,C) (P,Q,R,S,T) (X,Y,Z)

|

R1

|

CONNECT privilege

* To create a role, we need to follow the following 3 steps

those are,

* **Step1: create a role:**

**Syntax:** create role <role name>;

* **Step2: Assign privileges to a role:**

**Syntax:** GRANT <PRIVILEGE NAME> TO <ROLE NAME>;

* **Step3: Assign role to users:**

**Syntax:** GRANT <ROLE NAME> TO <USERS>;

**EX:**

SQL> CREATE ROLE R1;

Role created.

SQL> GRANT CONNECT TO R1;

Grant succeeded.

SQL> GRANT R1 TO A,B,C;

Grant succeeded.

SQL> CONN

Enter user-name: A/A

Connected.

SQL> CONN

Enter user-name: B/B

Connected.

SQL> CONN

Enter user-name: C/C

Connected.

**SQL:**

====

> **DDL**

**> DML BY DEVELOPER**

**> DRL / DQL**

**> TCL**

**===========================**

**> DCL BY DBA ONLY**

**SYNONYM**

* It is database object which is used to create a permanent alias name or alternate name for the db object/table.
* **Purpose of Synonym:**
  + To reduce the lengthy table name.

Ex: COLLEGE\_ENROLLMENT\_DETAILS 🡪 Table Name

CREATE SYNONYM S1 FOR COLLEGE\_ENROLLMENT\_DETAILS;

SELECT \* FROM S1;

INSERT INTO S1 VALUES (1,2,3);

* + Hiding Owner Name and Object Name (Table)

EX: SELECT \* FROM SYSTEM. DEPT;

CREATE SYNONYM S2 FOR SYSTEM.DEPT;

GRANT SELECT ON S2 TO U22;

CONN U22/U22

SELECT \* FROM S2;

**TYPES OF SYNONYMS:**

1. Private Synonyms
2. Public Synonyms

**Private Synonyms:**

* These Synonyms are created by user who are having permission.
* **Syntax:**

CREATE SYNONYM <SYNONYM NAME> FOR [OWNERNAME]. OBJECT NAME;

**EX:**

sql> conn system/tiger

sql> create user a identified by a;

sql> grant connect, create TABLE, UNLIMITED tablespace to a;

sql> conn a/a

SQL> CREATE TABLE COLLEGE\_ENROLLMENT\_DETAILS (STID INT,SNAME VARCHAR2(10),BNAME VARCHAR2(10));

Table created.

SQL> INSERT INTO COLLEGE\_ENROLLMENT\_DETAILS VALUES (1021,'SMITH','EEE');

SQL> CREATE SYNONYM S1 FOR COLLEGE\_ENROLLMENT\_DETAILS

ERROR at line 1:

ORA-01031: insufficient privileges

SQL> CONN

Enter user-name: system/tiger

Connected.

SQL> grant create synonym to a;

Grant succeeded.

SQL> CONN

Enter user-name: A/A

Connected.

SQL> CREATE SYNONYM S1 FOR COLLEGE\_ENROLLMENT\_DETAILS;

Synonym created.

SQL> SELECT \* FROM S1;

Day 39 25/08/22

**NOTE:**

* If we want to view synonym name, table name and table owner name on a particular table in oracle db then we use the following datadictionary is called as "user\_synonyms".

**EX:**

sql> desc user\_synonyms;

SQL> SELECT SYNONYM\_NAME, TABLE\_NAME, TABLE\_OWNER FROM USER\_SYNONYMS;

SYNONYM\_NAME TABLE\_NAME TABLE\_OWNER

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

S1 COLLEGE\_ENROLLMENT\_DETAILS A

**How to drop a private synonym:**

**Syntax:** DROP SYNONYM <SYNONYM NAME>;

**EX:** DROP SYNONYM S1;

**2. Public synonyms:**

- these synonyms are created by DBA to hide ownername and

table name from users.

**Syntax:**

CREATE PUBLIC SYNONYM <SYNONYM NAME> FOR [OWNERNAME]. <TABLE NAME>;

**EX:**

SQL> conn

Enter user-name: system/tiger

SQL> CREATE PUBLIC SYNONYM PS1 FOR SYSTEM.DEPT;

Synonym created.

dd3

SQL> GRANT SELECT ON PS1 TO A, B, C;

Grant succeeded.

SQL> CONN

Enter user-name: A/A

Connected.

SQL> SELECT \* FROM PS1;

SQL> CONN

Enter user-name: B/B

Connected.

SQL> SELECT \* FROM PS1;

SQL> CONN

Enter user-name: C/C

Connected.

SQL> SELECT \* FROM PS1;

**NOTE:**

* If we want view all synonyms in oracle database then we use a datadictionary is "all\_synonyms".

**EX:**

SQL> DESC ALL\_SYNONYMS;

SQL> SELECT SYNONYM\_NAME,TABLE\_NAME,TABLE\_OWNER FROM ALL\_SYNONYMS WHERE TABLE\_NAME='DEPT';

SYNONYM\_NAME TABLE\_NAME TABLE\_OWNER

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

PS1 DEPT SYSTEM

**HOW TO DROP A PUBLIC SYNONYM:**

**Syntax:** DROP PUBLIC SYNONYM <SYNONYM NAME>;

**EX:** DROP PUBLIC SYNONYM PS1;

**VIEWS**

* It is a virtual / logical / sub set of a base table.
* It is not stored data but it saved query.
* It is created with the help of "select command".
* By using select query we can extract the required data from base table and show to user through a view object.
* Whenever we perform DML operations through a view those operations are internally performed on base table.

**Syntax:**

CREATE VIEW <VIEW NAME> AS <SELECT QUERY>;

**Advantages:**

1. Security:

> column level security

> row level security

2. Converting complex query into simple query.

**Ex:**

create view v1 as

select s. STID, S. SNAME, C.name, C. CFEE, C.CID, R. REGNO,

r. regdate from student s inner join course c

on s. cid= c.cid inner join register r

on c.cid=r.cid; ---🡪complex query

**Ex:**

select \* from v1; --🡪simple query

Day 40 27/08/22

1. To check Data Integrity Rules.

**TYPES OF VIEWS:**

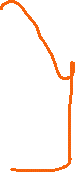
1. Simple Views
2. Complex Views
3. **SIMPLE VIEWS:**
   * When we created a view to access the required data from a single base table is called as simple view.
   * By default, simple views are supporting DML operations to perform over the base table.

**Ex: Create a view to access the data from dept table?**

SQL> CREATE VIEW V1 AS SELECT \* FROM DEPT;



SQL> INSERT INTO V1 VALUES (50,'DBA',' HYD');



SQL> UPDATE V1 SET LOC = 'ATLANTA' WHERE DEPTNO =50;

SQL> DELETE FROM V1 WHERE DEPTNO = 50;

SQL> SELECT \* FROM V1;

**Ex: Create a view to display empno, ename, sal from emp table?**

SQL> CREATE VIEW V2 AS SELECT EMPNO, ENAME, SAL FROM EMP;

SQL> INSERT INTO V2 VALUES (1122, 'GEORGIA', 3450);

SQL> ALTER TABLE EMP ADD CONSTRAINT ENO\_PK PRIMARY KEY(EMPNO);

SQL> INSERT INTO V2 VALUES (1122, 'MARK', 4350);

🡺 ERROR: unique constraint (DEEPI.ENO\_PK) violated

SQL> INSERT INTO V2 VALUES (1132, 'MARK', 4350, 'CLERK');

🡺 ERROR: too many values

**Ex: Create a view to display employee’s details who are working under working under deptno is 20.**

SQL> CREATE VIEW V3 AS SELECT \* FROM EMP WHERE DEPTNO=20;

**VIEW OPTIONS:**

1. With Check Option
2. With Read Only.
3. **With Check Option**

* To check values with user defined condition in select query at the time view creation.
* By using “with check option” statement we can restrict unwanted data on base table through a view object.

**Ex: Create a view to display and accept employee’s details whose salary is 3000**

SQL> CREATE VIEW V5 AS SELECT \* FROM EMP WHERE SAL =3000 WITH CHECK OPTION;

SQL> INSERT INTO V5 (EMPNO, ENAME, SAL) VALUES (1132, 'MARK', 4350);

ERROR 🡺view WITH CHECK OPTION where-clause violation

SQL> INSERT INTO V5 (EMPNO, ENAME, SAL) VALUES (1132, 'MARK', 3000);

SQL> SELECT \* FROM V5;

EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO

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

7788 SCOTT ANALYST 7566 09-DEC-82 3000 20

7902 FORD ANALYST 7566 03-DEC-81 3000 20

1132 MARK 3000

1. **With Read Only:**

* Whenever an user wants to restrict DML operations on a base table through a view object then we use “with Read Only” Statement as follows.

SQL> CREATE VIEW V6 AS SELECT \* FROM DEPT WITH READ ONLY;

SQL> INSERT INTO V6 VALUES (50, 'SAP’, 'HYD');

ERROR 🡺 cannot perform a DML operation on a read-only view

Day 41 29/08/22

1. **COMPLEX VIEWS:**

* When we called a view is complex:
  + Based on Multiple Tables.
  + By Using Group By
  + By using Having
  + By Using Aggregative / Grouping Functions
  + By using Distinct Keyword.
  + By using Set Operators
  + By using Joins.
  + By using Subquery
* These complex views are not allowed DML operations by default.
* **Syntax:**

CREATE VIEW <VIEWNAME> AS <SELECT QUERY>;

**Ex:**

SQL>CREATE VIEW V7 AS SELECT\* FROM EMP\_HYD UNION SELECT \* FROM EMP\_CHN;

SQL>INSERT INTO V7 VALUES (6, ‘YUVAN’, 56000);



SQL>UPDATE V7 SET SAL = 11000 WHERE EID=1;

SQL>DELETE FROM V7 WHERE EID = 5;

ERROR: dATA MANIPULATION OPERATION NOT LEGAL ON THIS VIEW.

**Ex:**

SQL> CREATE VIEW V8 AS SELECT DEPTNO, SUM(SAL) TOTAL\_SAL FROM EMP GROUP BY DEPTNO;

SQL> UPDATE V8 SET TOTAL\_SAL= 1000 WHERE DEPTNO=10;

ERROR: data manipulation operation not legal on this view

**FORCE VIEW:**

* When we create a view without base table is called as force view.
* **Syntax:**

CREATE FORCEVIEW <FVNAME> AS <SELECT QUERY>;

**Ex:**

SQL> CREATE FORCE VIEW FV1 AS SELECT \* FROM TEST100;

Warning: View created with compilation errors.

SQL> SELECT \* FROM FV1; -----------------------view "DEEPI.FV1" has errors

* To activate a force view then we should create a Base table.

SQL> CREATE TABLE TEST100 (SNO INT , NAME VARCHAR2(10));

SQL> INSERT INTO TEST100 VALUES (1, 'SMITH');

SQL> SELECT \* FROM FV1;

SNO NAME

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

1 SMITH

**MATERIALIZED VIEWS:**

* Materialized views are created based on a base table just like a view.

|  |  |
| --- | --- |
| **VIEW** | **MATERIALIZED VIEW** |
| 1. Not Storing Data | 1.Storing Data |
| 1. Dependent Object | 2.Independent Object |
| 1. Once drop a Base Table then view cannot be accessible. | 3. Even drop a Base Table then MV can be accessible. |
| 4. Supporting DML Operations | 4. Not Supporting DML Operations |

**Syntax:**

CREATE MATERIALIZED VIEW <VIEW NAME> AS <SELECT QUERY>;

**EX:**

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

Table created.

SQL> CREATE VIEW V11 AS SELECT \* FROM TEST1;

View created.

SQL> CREATE MATERIALIZED VIEW MV11 AS SELECT \* FROM TEST1;

Materialized view created.

SQL> INSERT INTO TEST1 VALUES (101,'ALLEN');

* Once we insert data into a base table(test1) then the data can see in view(v11) also but we cannot see data in materialized view directly.
* If we want to see data in materialized view then we should refresh a materialized view by using the following two methods.

1. **ON DEMAND:**

* It is a default refreshing method of mview.
* **Syntax:**

EXECUTE DBMS\_MVIEW.REFRESH('MVIEW NAME');

**EX:**

EXECUTE DBMS\_MVIEW.REFRESH('MV11');

**EX:**

SELECT \* FROM MV11;

Day 42 30/08/22

1. **ON COMMIT:**

* When we want to see data in mview after performing DML operations on a base table then we should commit those operations on a base table otherwise we cannot see data in mview.
* **Syntax:**

CREATE MATERIALIZED VIEW <VIEW NAME> REFRESH ON COMMIT AS <SELECT QUERY>;

**EX:**

SQL> CREATE TABLE EMP77 (EID INT, ENAME VARCHAR2(10));

SQL> CREATE VIEW V77 AS SELECT \* FROM EMP77;

SQL> CREATE MATERIALIZED VIEW MV77 REFRESH ON COMMIT AS SELECT \* FROM EMP77; ------------- ERROR: cannot set the ON COMMIT refresh attribute for the materialized view.

* To overcome above error then we should apply primary key constraint on a base table.

SQL> ALTER TABLE EMP77 ADD CONSTRAINT P\_EID PRIMARY KEY(EID);

SQL> CREATE MATERIALIZED VIEW MV77 REFRESH ON COMMIT AS SELECT \* FROM EMP77; ------------------- Materialized view created

.

SQL> INSERT INTO EMP77 VALUES (1, 'WARD');

SQL> SELECT \* FROM MV77; -------------no rows selected

SQL> COMMIT;

SQL> SELECT \* FROM MV77;

EID ENAME

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

1 WARD

**Note:**

* To see all view in oracle Oracle DB then we use the following data dictionary is “USER\_VIEWS”.

**Ex:**

SQL> DESC USER\_VIEWS;

SQL> SELECT VIEW\_NAME FROM USER\_VIEWS;

**Note:**

* To view the text of a particular view in oracle DB.
* Ex:

SQL> SELECT TEXT FROM USER\_VIEWS WHERE VIEW\_NAME = 'V1';

**Syntax to drop View:**

DROP VIEW <VIEW NAME>;

**Ex:**

SQL> DROP VIEW V1;

**Note**:

* If we want to see all materialized views in Oracle DB then we use a Data dictionary is “USER\_MVIEWS”

**Ex:**

SQL> DESC USER\_MVIEWS;

SQL> SELECT MVIEW\_NAME FROM USER\_MVIEWS;

**Syntax to drop Materialized View:**

DROP MATERIALIZED VIEW <VIEW NAME>;

**Ex:**

SQL> DROP MATERIALIZED VIEW MV11;

Day 43 01/09/22

**SEQUENCE:**

* It is db object which is used to generate sequence numbers on a particular column automatically.
* Providing "auto incremental value" facility.

**Syntax:**

CREATE SEQUENCE <SEQUENCE NAME>

[START WITH N]

[MINVALUE N]

[INCREMENT BY N]

[MAXVALUE N]

[NO CYCLE / CYCLE]

[NO CACHE / CACHE N];

**start with n:**

* It specifies the starting value of a sequence.

Here "n" is number.

**minvalue n:**

* It shows minimum value in the sequence.
* Here "n" number.

**increment by n:**

* It specifies incremental value between sequence

numbers.

* Here "n" number.

**maxvalue n:**

* It shows maximum value in the sequence.
* Here "n" number.

**Ex:**

create sequence s1

start with 1

minvalue 1

increment by 1

maxvalue 3;

**output:**

1

2

3----seq.stop

**No Cycle:**

* It is a default parameter of sequence.
* When we create a sequence object with "no cycle" parameter then the set sequence numbers are not repeated again and again.

**Ex:**

create sequence s1

start with 1

minvalue 1

increment by 1

maxvalue 5;

**output:**

1

2

3

4

5------seq.stop

**Cycle:**

* When we create a sequence object with "cycle" parameter then the set sequence numbers are repeated again and again.

**Ex:**

create sequence s1

start with 1

minvalue 1

increment by 1

maxvalue 3

cycle;

**output:**

1

2

3

1

2

3

1

2

3

1

2

3

**No Cache:**

* It is a default parameter.
* Cache is a temporary memory file.
* When we create a sequence object with "no cache" parameter then the set sequence numbers are saved in DB directly.so that whenever user want access the data based on sequence number then each and every request will to go database.
* So that the number of users request will increase the burden on database and reduce the performance of database.

**Cache n:**

* When we create a sequence object with "cache" parameter then the set sequence numbers are saved in DB and the copy of sequence numbers are also saved in cache file memory.so that whenever user want access the data based on sequence number then each and every request will to go cache file instead database.
* So that it reduced number of users requests and also the burden on database and improve the performance of database.
* Here cache file size is min 2kb.

**Note:**

* When we work on sequence object, we should use the following

two pseudo columns:

1. **nextval :** To generate next by next sequence number.

2. **currval :** It shows the current sequence number.

**Ex:**

1

2

3

4 -----> nextvalue = 5,currvalue = 4

**EX:**

SQL> CREATE SEQUENCE SQ1

START WITH 1

MINVALUE 1

INCREMENT BY 1

MAXVALUE 3;

Sequence created.

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

Table created.

SQL> INSERT INTO TEST1 VALUES (SQ1.NEXTVAL,'&NAME');

Enter value for name: A

SQL> /

Enter value for name: B

SQL> /

Enter value for name: C

SQL> /

Enter value for name: D

ERROR at line 1:

ORA-08004: sequence SQ1.NEXTVAL exceeds MAXVALUE and

cannot be instantiated

**ALTERING A SEQUENCE:**

**Syntax:**

ALTER SEQUENCE <SEQUENCE NAME> <PARAMETER NAME> n;

**EX:**

SQL> ALTER SEQUENCE SQ1 MAXVALUE 5;

Sequence altered.

SQL> INSERT INTO TEST1 VALUES(SQ1.NEXTVAL,'&NAME');

Enter value for name: D

SQL> /

Enter value for name: E

SQL> /

**EX:**

SQL> CREATE SEQUENCE SQ2

2 START WITH 1

3 MINVALUE 1

4 INCREMENT BY 1

5 MAXVALUE 3

6 CYCLE

7 CACHE 2;

Sequence created.

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

Table created.

SQL> INSERT INTO TEST2 VALUES(SQ2.NEXTVAL,'&NAME');

Enter value for name: M

SQL> /

.......................

SQL> /

.......................

**EX:**

SQL> CREATE SEQUENCE SQ3

2 START WITH 3

3 MINVALUE 1

4 INCREMENT BY 1

5 MAXVALUE 5

6 CYCLE

7 CACHE 2;

Sequence created.

SQL> CREATE TABLE TEST3(SNO INT,NAME VARCHAR2(10));

Table created.

SQL> INSERT INTO TEST3 VALUES(SQ3.NEXTVAL,'&NAME');

Enter value for name: A

SQL> /

Enter value for name: B

SQL> /

.......................

SQL> /

.......................

**Note:**

* If we want to view all sequence objects in oracle db then we use a datadictionary is called as "user\_sequences".

**EX:**

SQL> DESC USER\_SEQUENCES;

SQL> SELECT SEQUENCE\_NAME FROM USER\_SEQUENCES;

**HOW TO DROP A SEQUENCE:**

**Syntax:**

DROP SEQUENCE <SEQUENCE NAME>;

**EX:** DROP SEQUENCE SQ1;

**PARTITION TABLE**

1. Range Partition
2. List Partition
3. Hash Partition
4. **Range Partition**
   * Created a partition table based on a particular range value.
   * **Syntax:**

CREATE TABLE <TN> (<CN1><DT> [SIZE], ………)

PARTITION BY RANGE (COLUMN NAME)

(PARTITION <PARTITION NAME1> VALUES LESS THAN (VALUES),

PARTITION <PARTITION NAME2> VALUES LESS THAN (VALUES),

…………………..);

**Ex:**

SQL> CREATE TABLE EMP11(EID INT, ENAME VARCHAR2(10), SAL NUMBER (10)) PARTITION BY RANGE(SAL) (PARTITION P1 VALUES LESS THAN (500),PARTITION P2 VALUES LESS THAN (1000),PARTITION P3 VALUES LESS THAN (2000));

SQL> INSERT INTO EMP11 VALUES (1,'SMITH', 200);

SQL> INSERT INTO EMP11 VALUES (2,'WARD', 500);

SQL> INSERT INTO EMP11 VALUES (3,'ALLEN', 1500);

SQL> INSERT INTO EMP11 VALUES (4,'MARTIN', 400);

SQL> SELECT \* FROM EMP11;

EID ENAME SAL

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

1 SMITH 200

4 MARTIN 400

2 WARD 500

3 ALLEN 1500

SQL> SELECT \* FROM EMP11 PARTITION(P1);

EID ENAME SAL

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

1 SMITH 200

4 MARTIN 400

SQL> SELECT \* FROM EMP11 PARTITION(P2);

EID ENAME SAL

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

2 WARD 500

SQL> SELECT \* FROM EMP11 PARTITION(P3);

EID ENAME SAL

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

3 ALLEN 1500

Day 44 02/09/22

1. **List Partition**

* Created partition table based on the list of values.
  + **Syntax**:

CREATE TABLE<TN> (<CN1> <DT>[SIZE], …….)

PARTITION BY LIST (CN)

(PARTITION <PN1> VALUES (V1,V2,V3…….),

PARTITION <PN2> VALUES (V1,V2,V3…….),

………… PARTITION OTHER VALUES (DEFAULT));

**Ex:**

SQL> CREATE TABLE PART1(NAME VARCHAR2(10), AGE INT) PARTITION BY LIST (AGE)

2 (PARTITION P1 VALUES (1 ,2, 3),

3 PARTITION P2 VALUES (4,5,6,7,8,9,10),

4 PARTITION P3 VALUES (11,12,13,14,15),

5 PARTITION OTHERS VALUES (DEFAULT));

SQL> INSERT INTO PART1 VALUES ('AARA', 2);

SQL> INSERT INTO PART1 VALUES ('RAGHAV', 5);

SQL> INSERT INTO PART1 VALUES ('AAPU',15);

SQL> INSERT INTO PART1 VALUES ('PUPPY',27);

SQL> SELECT \* FROM PART1;

NAME AGE

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

AARA 2

RAGHAV 5

AAPU 15

PUPPY 27

SQL> SELECT \* FROM PART1 PARTITION(P1);

NAME AGE

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

AARA 2

SQL> SELECT \* FROM PART1 PARTITION(P2);

NAME AGE

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

RAGHAV 5

SQL> SELECT \* FROM PART1 PARTITION(P3);

NAME AGE

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

AAPU 15

SQL> SELECT \* FROM PART1 PARTITION(OTHERS);

NAME AGE

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

PUPPY 27

1. **Hash Partition**
   * Created Partition Table based on hash algorithm by system by default.
   * Here system will create partition based on user request.
   * **Syntax**:

CREATE TABLE <TN> (<CN1 <DT>[SIZE],….)

PARTITION BY HASH (CN) PARTITION <NUMBER>;

**Ex:**

SQL> CREATE TABLE EMP12(NAME VARCHAR2(10), SAL NUMBER(10)) PARTITION BY HASH (SAL) PARTITIONS 5;

**Note:**

* If we want to view partitions of a particular table in oracle db then we use a datadictionary is called ‘USER\_TAB\_PARTITIONS”.

**Ex:**

SQL> DESC USER\_TAB\_PARTITIONS;

SQL> SELECT PARTITION\_NAME FROM USER\_TAB\_PARTITIONS WHERE TABLE\_NAME ='EMP12';

**How to add a new partition to an existing table:**

1. **Syntax for Range Partition:**

ALTER TABLE<TN> ADD PARTITION <PN> VALUES LESS THAN (VALUE);

1. **Syntax for List Partition:**

ALTER TABLE<TN> ADD PARTITION <PN> VALUES (V1, V2, V3……..);

**EX:**

SQL> ALTER TABLE EMP11 ADD PARTITION P4 VALUES LESS THAN (3000);

**How to drop a partition from an existing table:**

**Syntax:**

ALTER TABLE<TN> DROP PARTITION <PN>;

**EX:**

SQL> ALTER TABLE EMP11 DROP PARTITION P1;

**Note:**

* If we want to know the table has partition or not, then we use a datadictionary known as “USER\_TABLES”.

**EX:**

SQL> DESC USER\_TABLES;

SQL> SELECT PARTITIONED FROM USER\_TABLES WHERE TABLE\_NAME ='EMP';

PAR

---

NO

SQL> SELECT PARTITIONED FROM USER\_TABLES WHERE TABLE\_NAME ='EMP11';

PAR

---

YES

Day 45 03/09/22

**LOCKS:**

* To prevent our resource
  + - 1. Row Level Locking
      2. Table Level locking

1. **Row Level Locking:**
   * 1. **Single row Lock**
        + Oracle DB server is locking a single row in a table.
        + Ex:

|  |  |
| --- | --- |
| User 1 | User 2 |
| SQL> CONN SYSTEM/TIGER | SQL> CONN DEEPI/123 |
| SQL> UPDATE DEEPI.EMP SET SAL = 1100 WHERE EMPNO = 7369;  [ Row is Locked by DB Server] | SQL> UPDATE EMP SET SAL = 1100 WHERE EMPNO = 7369;  [We cannot perform update] |
| SQL> COMMIT | 1 Row Updated |

* + 1. **Multiple Rows Lock**
* When we lock multiple rows in a table then we use “for update” clause in select Query.

|  |  |
| --- | --- |
| User 1 | User 2 |
| SQL> CONN SYSTEM/TIGER | SQL> CONN DEEPI/123 |
| SQL> SELECT \* FROM EMP WHERE DEPTNO =10;  [All Rows of selected Query are Locked by Db server} | SQL> UPDATE EMP SET SAL = 1100 WHERE EMPNO = 7369;  [We cannot perform update] |
| SQL>COMMIT/ ROLLBACK; | 3 rows Updated. |

1. **Dead Lock**
   * + Both users are waiting for their resources

|  |  |
| --- | --- |
| User 1 | User 2 |
| SQL> CONN SYSTEM/TIGER | SQL> CONN DEEPI/123 |
| SQL> UPDATE DEEPI.EMP SET SAL = 1100 WHERE EMPNO = 7369;  [ Row is Locked by DB Server] | SQL> UPDATE EMP SET SAL = 1100 WHERE EMPNO = 7900;  [ Row is Locked by DB Server] |
| SQL> UPDATE DEEPI.EMP SET SAL = 2100 WHERE EMPNO = 7900;  [We cannot perform update] | SQL> UPDATE DEEPI.EMP SET SAL = 3400 WHERE EMPNO = 7369;  [We cannot perform update] |
| Error: Dead lock Detected while waiting for resource |  |
| SQL>COMMIT/ ROLLBACK;  [Lock is Releasing] |  |

Day 46 05/09/22

1. **Table Level Locking:**

* In this level we can lock the entire table.

1. Share Lock
2. Exclusive Lock
3. **Share Lock:**

* Both Users can lock a table
* **Syntax:**

LOCK TABLE <TN> IN SHARE MODE.

|  |  |
| --- | --- |
| User 1 | User 2 |
| SQL> CONN SYSTEM/TIGER | SQL> CONN DEEPI/123 |
| SQL> LOCK TABLE DEEPI.EMP IN SHARE MODE;  [WE CANNOT PERFORM DML OPERATIONS] | SQL> LOCK TABLE EMP IN SHARE MODE;  [WE CANNOT PERFORM DML OPERATIONS] |
| SQL> COMMIT/ROLLBACK;  [FOR LOCK RELEASING] | SQL> COMMIT/ROLLBACK;  [FOR LOCK RELEASING] |

1. **Exclusive Lock:**

* In one user can lock a table.
* **Syntax:**

LOCK TABLE <TN> IN EXCLUSIVE MODEL;

|  |  |
| --- | --- |
| User 1 | User 2 |
| SQL> CONN SYSTEM/TIGER | SQL> CONN DEEPI/123 |
| SQL> LOCK TABLE DEEPI.EMP IN EXCLUSIVE MODE;  [TABLE LOCKED] | SQL> LOCK TABLE EMP IN EXCLUSIVE MODE;  [WE CANNOT LOCK A TABLE] |
| SQL> COMMIT/ROLLBACK;  [FOR LOCK RELEASING] | [TABLE LOCKED] |

**INDEXES**

* Indexes are used to retrieving the required row data from a table fast.
* Database index object is similar to index page in textbook. However, by using index page we can access the required topic from a text book quickly same as by using index object we can retrieve the required row data from a table quickly.
* A DB index object can be created on column/(s) and this column is called as "indexed key column".
* whenever we want to access a row data from a table then we should use an "indexed column" in where clause condition otherwise indexes are not working.

eid int(index) ename varchar2(10) (no index)

**Ex:**

select \* from t where ename='smith’; ---- indexes are not working

select \* from t where eid=1021; ---- indexes are working

* All databases are supporting the following two types of searching

mechanisms those are,

1. Table scan

2. Index scan

**1. Table scan:**

* It is a default searching mechanism in database.
* It will searching the entire table(top-bottom) for required data purpose.

**Ex:**

SQL> SELECT \* FROM EMP WHERE SAL=3000; ---performing table scan by default.

SAL

----------

800

1600

1250

2975

1250

2850 --------> sal=3000 (comparing 14 rows in a table)

2450 (14 rows x 1 sec = 14sec)

3000

5000

1500

1100

950

3000

1300

Day 47 06/09/22

**2. INDEX SCAN**

* In this scan oracle DB server is searching based on indexed column for required data.
* We can create an index object on particular columns in a table in two ways

1. Implicit Indexes
2. Explicit Indexes
3. **Implicit Indexes:**

* These indexes are created by system automatically when we create a table along with unique/ primary key constraints.

**Ex:**

SQL> CREATE TABLE TEST22(SNO INT UNIQUE, SNAME VARCHAR2(10));

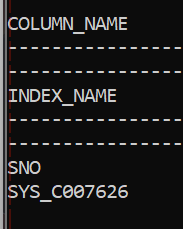
**Note**:

* To view index name along with column name of a particular table in oracle DB then we use a data dictionary is called as “user\_ind\_columns”.

**Ex:**

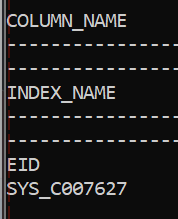
DESC USER\_IND\_COLUMNS;

SQL> SELECT COLUMN\_NAME, INDEX\_NAME FROM USER\_IND\_COLUMNS WHERE TABLE\_NAME ='TEST22';



SQL> CREATE TABLE TEST33 (EID INT PRIMARY KEY, SAL NUMBER(10));

SQL> SELECT COLUMN\_NAME, INDEX\_NAME FROM USER\_IND\_COLUMNS WHERE TABLE\_NAME ='TEST33';



1. **Explicit Indexes:**

* These indexes are created by user.
* **Syntax**:

CRETAE INDEX <IN> ON <TN> (<COLUMN NAME>);

* There are two types of indexes are created on a table.
  + - 1. B-Tree Index
         1. Simple Index
         2. Composite Index
         3. Unique Index
         4. Functional Based Index
      2. Bitmap Index

1. **Simple Index:**

* When we create an index object based on a single column.
* **Syntax**:

CRETAE INDEX <INDEX NAME> ON <TN> (COLUMN NAME);

**EX:**

SQL> CREATE INDEX I1 ON EMP(SAL);

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

b-TREE INDEX

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

(LP)<|3000|>=(RP) ----------ROOT LEVEL

|

(LP) <|2975|>=(RP) (LP) < |5000| >=(RP) ---- PARENT LEVEL

| |

2850|\*,2459|\*,1600|\*, 3000|\*,\*------CHILD LEVEL (HERE "\*" IS NOTHING BUT ROWID)

1500|\*,1300|\*,1250|\*,\*

1100|\*,950|\*,800|\*

1. **Composite Index:**

* When we created ai index object based on multiple columns.
* **Syntax:**

CRETAE INDEX<IN> ON <TN> (<C1>, <C2>, …...);

* Index scan is performed based on leading column.

**Ex:**

SQL> CREATE INDEX I2 ON EMP (DEPTNO, JOB);

SQL> SELECT \* FROM EMP WHERE DEPTNO =10; 🡺 Index scan

SQL> SELECT \* FROM EMP WHERE DEPTNO =10 AND JOB = 'SALESMAN'; 🡺index scan

SQL> SELECT \* FROM EMP WHERE JOB = 'MANAGER'; 🡺 table Scan

1. **Unique Index:**

* When we created an index object based on unique constraint.
* Unique indexed column is not allowed duplicate values. So that searching time is reduced and improve the performance of database.
* **Syntax**:

CREATE UNIQUE INDEX <INDEX NAME> ON <TN> (<CN>);

**Ex:**

SQL> CREATE UNIQUE INDEX UI ON DEPT(DNAME);

SQL> INSERT INTO DEPT VALUES (50. 'SALES'.'HYD');

Error: unique constraint (DEEPI.UI) violated

Day 48 12/09/22

1. **Functional Based Index:**

* When we create an index object based on function name.
* **Syntax:**

CREATE INDEX<INDEX NAME> ON <TN> (<FUNCTION NAME> (CN));

**Ex:**

SQL> CREATE INDEX F1 ON EMP (UPPER(ENAME));

SQL> SELECT \* FROM EMP WHERE ENAME = 'smith'; 🡺 no rows selected

SQL> SELECT \* FROM EMP WHERE ENAME = 'SMITH'; 🡺 Index Activated

**BITMAP INDEX:**

* Create based on ‘’low cardinality of column”.

**Cardinality:**

* It refers uniqueness of data values in a column.

**Syntax to find out cardinality of column:**

CARDINALITY OF COLUMN = NO.OF DISTINCT VALUES IN A COLUMN

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

NO.OF VALUES IN THE COLUMN

**Ex:**

Cardinality of empno => 14/14 ==== 1; (High Cardinality)

Cardinality of job => 5/14 ==== 0.35; (Low Cardinality)

**Syntax:**

CREATE BITMAP INDEX <INDEX NAME> ON <TN> (COLUMN NAME);

**Ex:**

SQL> CREATE BITMAP INDEX BI ON EMP(JOB);

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

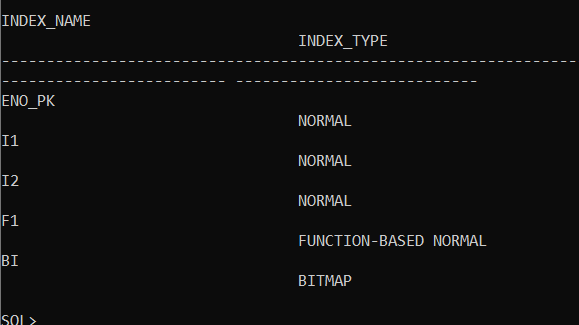
**Note:**

* When we created a bitmap index object on a particular column internally system will generate a bitmap index table with bit number 1 (or) 0.
* Here 1 represents condition is true and 0 represents condition is false.

**Note:**

* **I**f we want to see all the indexes IN ORACLE db then we use a data dictionary called as “user\_indexes”.

SQL> SELECT INDEX\_NAME , INDEX\_TYPE FROM USER\_INDEXES WHERE TABLE\_NAME = 'EMP';



**Syntax to drop indexes:**

DROP INDEX <INDEX NAME>;

**Ex:**

SQL> DROP INDEX Bi;

Day 49 13/09/22

**CLUSTER**

* It is a collection of tables together saved in same data block memory.
* To improve the performance of joins.
* When we create cluster tables then those tables should a common column name.
* Cluster can create at the time of table creation.
* When we create a cluster object then we follow the following 3 steps,
* **Step 1: Create A Cluster**

Syntax: CREATE CLUSTER <CLUSTER NAME> (COMMON CN <DT> );

* **Step 2: Create An Index Object On A Cluster**

Syntax: CREATE INDEX <INDEX NAME> ON CLUSTER <CLUSTER NAME>;

* **Step3: Create Cluster Tables:**

syntax: CREATE TABLE <TN>(<COL1> <DT>[SIZE],..............)

CLUSTER <CLUSTER NAME>(COMMON COLUMN NAME);

**EX:** SQL> CREATE CLUSTER EMP\_DEPT(DEPTNO INT);

SQL> CREATE INDEX CI ON CLUSTER EMP\_DEPT;

SQL> CREATE TABLE EMP1(EID INT,ENAME VARCHAR2(10),DEPTNO INT) CLUSTER EMP\_DEPT(DEPTNO);.

SQL> CREATE TABLE DEPT1(DEPTNO INT,DNAME VARCHAR2(10)) CLUSTER EMP\_DEPT(DEPTNO);

SQL> INSERT INTO EMP1 VALUES(1,'A',10);

SQL> INSERT INTO EMP1 VALUES(2,'B',20);

SQL> COMMIT;

SQL> INSERT INTO DEPT1 VALUES(10,'D1');

SQL> INSERT INTO DEPT1 VALUES(20,'D2');

SQL> COMMIT;

SQL> SELECT \* FROM EMP1;

EID ENAME DEPTNO

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

2 B 20

1 A 10

SQL> SELECT \* FROM DEPT1 ;

DEPTNO DNAME

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

20 D2

10 D1

SQL> SELECT ROWID FROM EMP1;

ROWID

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

AAAR2hAAHAAAAGbAAA

AAAR2hAAHAAAAGfAAA

SQL> SELECT ROWID FROM DEPT1;

ROWID

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

AAAR2hAAHAAAAGbAAA

AAAR2hAAHAAAAGfAAA

**Note:**

* If we want to view clusters in oracle db then we use

a datadictionary is called as "user\_clusters".

**EX:**

SQL> DESC USER\_CLUSTERS;

SQL> SELECT CLUSTER\_NAME FROM USER\_CLUSTERS;

CLUSTER\_NAME

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

EMP\_DEPT

**Note:**

* To view tables in a cluster memory then we use a

datadictionary is called as "user\_tables".

**EX:**

SQL> DESC USER\_TABLES;

SQL> SELECT TABLE\_NAME FROM USER\_TABLES WHERE CLUSTER\_NAME='EMP\_DEPT';

TABLE\_NAME

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

EMP1

DEPT1

**How To Drop A Cluster:**

**Syntax** :

DROP CLUSTER <CLUSTER NAME>;

**EX:**

SQL> DROP CLUSTER EMP\_DEPT; 🡺 cluster not empty

* To drop Cluster when it has tables then use “including tables”

**Syntax:**

SQL> DROP CLUSTER <CLUSTER NAME> INCLUDING TABLES;

**Ex:**

SQL> DROP CLUSTER EMP\_DEPT INCLUDING TABLES; 🡺 Cluster Dropped

Day 50 14/09/22

**USER-DEFINED DATA TYPES**

* We create our own data types as requirement are called user defined data types.
* Advantages:
  1. Reusability that means we can creste user defined datatype and reuse in multiple tables.
* Three Types:
  1. Object type (or) composite type
  2. Varray
  3. Nested table

1. **Object Type(or) composite type**

**To Select:**

SQL> SELECT S.SID, S.SNAME, S.COURSE.CID, S.COURSE.CNAME, S.COURSE.FEE FROM STUDENTS S;

**OR**

SQL> SELECT S.SID, S.SNAME, S.COURSE.CID CID, S.COURSE.CNAME CNAME, S.COURSE.FEE FEE FROM STUDENTS S;

**To Update:**

1. **Varray**
2. **Nested table**

Day 51 15/09/22

**NORMALIZATION**

* Normalization is a technique of organizing the data into multiple tables.
* Normalization process automatically eliminates data redundancy (repetition) and also avoiding Insertion, Update and Deletion problems.
* Types of Normal Forms:
  + 1. First Normal Form
    2. Second Normal Form
    3. Third Normal Form
    4. BCNF
    5. Fourth Normal Form
    6. Fifth Normal form

**First Normal Form (1NF):**

1. Each column should contain single value.
2. A column should contain values that are same datatype.
3. All the columns in a table should have unique names.
4. The order in which data is stored, does not matter.

**Second Normal Form (2NF):**

1. No Partial Dependency

In a table if non-key column depends on part of the key column, then it is called as partial dependency

**Third Normal Form (3NF):**

1. No Transitive Dependency

In table if non-key column depends on non-key column, then it is called as transitive dependency.

**Boyce- Codd Normal Form (BCNF):**

**Fourth Normal Form (4NF):**

**Fifth Normal Form (5NF):**