**QUERY: 01**

**Q1:** Write a query to create a table employee with empno, ename, designation, and salary.

**Syntax:** It is used to create a table

SQL: CREATE <OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1 <DATATYPE> (SIZE),

COLUMN NAME.2 <DATATYPE> (SIZE) ………);

**Command:**

SQL>**CREATE TABLE** EMP (EMPNO **NUMBER (4),**ENAME **VARCHAR2 (10),** DESIGNATIN

**VARCHAR2 (10),**SALARY **NUMBER (8, 2));**

Table created.

**Constraints with Table Creation:**

Constraints are condition for the data item to be stored into a database. There are two types of Constraints viz., Column Constraints and Table Constraints.

**Syntax:**

[CONSTRAINT constraint name]

{[NOT] NULL / UNIQUE / PRIMARY

KEY}(Column[,column]..) FOREIGN KEY (column [, colum]…)

REFERENCES table

[ON DELETE CASCADE]

[CHECK (condition)]

**TABLE DESCRIPTION**

It is used to view the table structure to confirm whether the table was created correctly.

**QUERY: 02**

**Q2:** Write a query to display the column name and data type of the table employee.

**Syntax:** This is used to view the structure of the table. SQL: DESC <TABLE NAME>;

**Command:**

SQL> DESC EMP;

**Name Null? Type**

EMPNO NUMBER(4)

ENAME VARCHAR2(10)

DESIGNATIN VARCHAR2(10)

SALARY NUMBER(8,2)

#### QUERY: 03

**Q3:** Write a query for create a from an existing table with all the fields

**Syntax:** syntax for create a table from an existing table with all fields.

SQL> CREATE TABLE <TRAGET TABLE NAME> SELECT \* FROM<SOURCE TABLE NAME>;

#### Command:

SQL> CREATE TABLE **EMP1** AS SELECT \* FROM **EMP**;

Table created.

#### Command:

SQL> DESC EMP1

Name Null? Type

EMPNO NUMBER(4)

ENAME VARCHAR2(10) DESIGNATIN VARCHAR2(10) SALARY NUMBER(8,2)

#### QUERY: 04

**Q4:** Write a query for create a from an existing table with selected fields

**Syntax:** Syntax for create a from an existing table with selected fields.

SQL> CREATE TABLE <TRAGET TABLE NAME> AS SELECT EMPNO, ENAMEFROM <SOURCE TABLE NAME>;

#### Command:

SQL> CREATE TABLE **EMP2** AS SELECT **EMPNO**, **ENAME** FROM **EMP**;

Table created.

#### Command:

SQL> DESC EMP2

Name Null? Type

EMPNO NUMBER (4)

ENAME VARCHAR2 (10)

#### QUERY: 05

**Q5:** Write a query for create a new table from an existing table without any record:

**Syntax:** The syntax for create a new table from an existing table without any record.

SQL> CREATE TABLE <TRAGET TABLE NAME> AS SELECT \* FROM<SOURCE TABLE NAME> WHERE <FALSE CONDITION>;

#### Command:

SQL> CREATE TABLE EMP3 AS SELECT \* FROM EMP WHERE1>2;

Table created.

#### Command:

SQL> DESC EMP3;

Name Null? Type

EMPNO NUMBER(4)

ENAME VARCHAR2(10) DESIGNATIN VARCHAR2(10) SALARY NUMBER(8,2);

#### ALTER & MODIFICATION ON TABLE

To modify structure of an already existing table to add one more columns and also modify the existing columns.

Alter command is used to:

* 1. Add a new column.
  2. Modify the existing column definition.
  3. To include or drop integrity constraint.

#### QUERY: 06

**Q6:** Write a Query to Alter the column EMPNO NUMBER (4) TO EMPNO NUMBER (6).

**Syntax:** The syntax for alter & modify on a single column.

SQL > ALTER <TABLE NAME> MODIFY <COLUMN NAME><DATATYPE>(SIZE);

#### Command:

SQL>ALTER TABLE **EMP** MODIFY **EMPNO** NUMBER (6);

Table altered.

#### Command:

SQL> DESC EMP;

Name Null? Type

EMPNO NUMBER(6)

ENAME VARCHAR2(10) DESIGNATIN VARCHAR2(10) SALARY NUMBER(8,2) **QUERY: 07**

Q7. Write a Query to Alter the table employee with multiple columns (EMPNO,ENAME.)

**Syntax:** To alter table with multiple column.

SQL > ALTER <TABLE NAME> MODIFY <COLUMN NAME1><DATATYPE>(SIZE),

MODIFY <COLUMN NAME2><DATATYPE>(SIZE) ;

**Command:**

SQL>ALTER TABLE EMP MODIFY (EMPNO NUMBER (7),

ENAMEVARCHAR2(12)); Table altered.

**Command:**

SQL> DESC EMP;

Name Null? Type

EMPNO NUMBER(7)

ENAME VARCHAR2(12) DESIGNATIN VARCHAR2(10)

SALARY NUMBER(8,2);

**QUERY: 08**

Q8. Write a query to add a new column in to employee

**Syntax:** To add a new column.

SQL> ALTER TABLE <TABLE NAME> ADD (<COLUMN NAME><DATATYPE><SIZE>);

**Command:**

SQL> ALTER TABLE EMP ADD QUALIFICATION VARCHAR2(6);

Table altered.

SQL> DESC EMP;

Name Null? Type

EMPNO NUMBER(7)

ENAME VARCHAR2(12) DESIGNATIN VARCHAR2(10) SALARY NUMBER(8,2) QUALIFICATION VARCHAR2(6)

**QUERY: 09**

**Q9:** Write a query to add multiple columns in to employee **Syntax:** Syntax for add a new column.

SQL> ALTER TABLE <TABLE NAME> ADD (<COLUMN NAME1><DATATYPE><SIZE>,

(<COLUMN NAME2><DATA TYPE><SIZE>…);

**Command:**

SQL>ALTER TABLE EMP ADD (DOB DATE, DOJ DATE);

Table altered.

SQL> DESC EMP;

Name Null? Type

EMPNO NUMBER(7)

ENAME VARCHAR2(12) DESIGNATIN VARCHAR2(10) SALARY NUMBER(8,2) QUALIFICATION VARCHAR2(6) DOB DATE

DOJ DATE

**REMOVE / DROP**

It will delete the table structure provided the table should be empty.

**QUERY: 10**

Q10. Write a query to drop a column from an existing table employee

**Syntax:** syntax for add a new column.

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

**Command:**

SQL> ALTER TABLE EMP DROP COLUMN DOJ;

Table altered.

SQL> DESC EMP;

Name Null? Type

EMPNO NUMBER(7)

ENAME VARCHAR2(12) DESIGNATIN VARCHAR2(10) SALARY NUMBER(8,2) QUALIFICATION VARCHAR2(6) DOB DATE

**QUERY: 11**

Q10. Write a query to drop multiple columns from employee

**Syntax:**The Syntax for add a new column.

SQL> ALTER TABLE <TABLE NAME> DROP <COLUMNNAME1>,<COLUMN NAME2>, ;

**Command:**

SQL> ALTER TABLE EMP DROP (DOB, QUALIFICATION);

Table altered.

SQL> DESC EMP;

Name Null? Type

EMPNO NUMBER(7)

ENAME VARCHAR2(12) DESIGNATIN VARCHAR2(10) SALARY NUMBER(8,2)

**RENAME**

**QUERY: 12**

Q10. Write a query to rename table emp to employee

**Syntax:**The Syntax for add a new column.

SQL> ALTER TABLE RENAME <OLD NAME> TO <NEW NAME>

**Command:**

SQL> ALTER TABLE RENAME EMP TO EMPLOYEE; SQL> DESC EMPLOYEE;

Name Null? Type

EMPNO NUMBER(7)

ENAME VARCHAR2(12) DESIGNATIN VARCHAR2(10) SALARY NUMBER(8,2)

**TRUNCATE TABLE**

If there is no further use of records stored in a table and the structure has to be retained then the records alone can be deleted.

**Syntax:**

TRUNCATE TABLE <TABLE NAME>;

**Example:**

Truncate table EMP;

**DROP:**

To remove a table along with its structure and data.

**Syntax:**The Syntax for add a new column.

SQL> Drop table<table name>;

**Command:**

SQL> drop table employee;

**RESULT:**

Thus the SQL commands for DDL commands in RDBMS has been verified and executed successfully.

**DATA MANIPULATION LANGUAGE (DML) COMMANDS IN RDBMS**

#### Ex: No: 1.2

**: :**

**AIM:**

To execute and verify the DML commands are the most frequently used SQL commands and is used to query and manipulate the existing database objects.

**DML (DATA MANIPULATION LANGUAGE)**

SELECT INSERT DELETE UPDATE

**ALGORITHM:**

**STEP 1:** Start the DBMS.

**STEP 2:** Create the table with its essential attributes.

**STEP 3:** Insert the record into table

**STEP 4:** Update the existing records into the table

**STEP 5:** Delete the records in to the table

**STEP 6:** use save point if any changes occur in any portion of the record to undo its original state.

**STEP 7:** use rollback for completely undo the records

**STEP 8:** use commit for permanently save the records

#### INSERT

The SQL INSERT INTO Statement is used to add new rows of data to a table in the database.

#### Insert a record from an existing table:

**QUERY: 01**

Q1. Write a query to insert the records in to employee.

**Syntax:** syntax for insert records in to a table

SQL :> INSERT INTO <TABLE NAME> VALUES< VAL1, ‘VAL2’,…..);

#### Command:

SQL>INSERT INTO EMP VALUES (101,'NAGARAJAN','LECTURER',15000);

1 row created.

#### Insert A Record Using Substitution Method:

**QUERY: 03**

Q3. Write a query to insert the records in to employee using substitution method.

**Syntax:** syntax for insert records into the table.

SQL :> INSERT INTO <TABLE NAME> VALUES< ‘&column name’, ‘&column nam); e 2’, …..

#### Command:

SQL> INSERT INTO EMP VALUES(&EMPNO,'&ENAME','&DESIGNATIN','&SALARY'); Enter value for empno: 102

Enter value for ename: SARAVANAN Enter value for designatin: LECTURER Enter value for salary: 15000

1 row created.

**old 1:** INSERT INTO EMP VALUES(&EMPNO,'&ENAME','&DESIGNATIN','&SALARY')

**new 1:** INSERT INTO EMP VALUES(102,'SARAVANAN','LECTURER','15000')

SQL> /

Enter value for empno: 103

Enter value for ename: PANNERSELVAM Enter value for designatin: ASST. PROF Enter value for salary: 20000

1 row created.

**old 1:** INSERT INTO EMP VALUES(&EMPNO,'&ENAME','&DESIGNATIN','&SALARY')

**new 1:** INSERT INTO EMP VALUES(103,'PANNERSELVAM','ASST.PROF','20000')

SQL> /

Enter value for empno: 104 Enter value for ename: CHINNI

Enter value for designatin: HOD, PROF Enter value for salary: 45000

1 row created.

**old 1:** INSERT INTO EMP VALUES(&EMPNO,'&ENAME','&DESIGNATIN','&SALARY')

**new 1:** INSERT INTO EMP VALUES(104,'CHINNI','HOD, PROF','45000')

SQL> SELECT \* FROM EMP;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| EMPNO |  | ENAME | DESIGNATIN |  | SALARY |
| 101 |  | NAGARAJAN | LECTURER |  | 15000 |
| 102 |  | SARAVANAN | LECTURER |  | 15000 |
| 103 |  | PANNERSELVAM | ASST. PROF |  | 20000 |
| 104 |  | CHINNI | HOD, PROF |  | 45000 |

#### SELECT

**SELECT** Statement is used to fetch the data from a database table which returns data in the form of result table. These result tables are called result-sets.

#### Display the EMP table:

**QUERY: 02**

Q3. Write a query to display the records from employee.

**Syntax:** Syntax for select Records from the table.

#### SQL> SELECT \* FROM <TABLE NAME>;

**Command:**

SQL> SELECT \* FROM EMP;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| EMPNO |  | ENAME | DESIGNATIN | SALARY |
| 101 |  | NAGARAJAN | LECTURER | 15000 |

#### UPDATE

The SQL **UPDATE** Query is used to modify the existing records in a table. You can use **WHERE**

clause with **UPDATE** query to update selected rows, otherwise all the rows would be affected.

#### QUERY: 04

Q1. Write a query to update the records from employee.

**Syntax:** syntax for update records from the table.

SQL> UPDATE <<TABLE NAME> SET <COLUMNANE>=<VALUE> WHERE <COLUMN NAME=<VALUE>;

#### Command:

SQL> UPDATE EMP SET SALARY=16000 WHERE EMPNO=101;

1 row updated.

SQL> SELECT \* FROM EMP;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| EMPNO |  | ENAME | DESIGNATIN |  | SALARY |
| 101 |  | NAGARAJAN | LECTURER |  | 16000 |
| 102 |  | SARAVANAN | LECTURER |  | 15000 |
| 103 |  | PANNERSELVAM | ASST. PROF |  | 20000 |
| 104 |  | CHINNI | HOD,PROF |  | 45000 |

#### Update Multiple Columns:

**QUERY: 05**

Q5. Write a query to update multiple records from employee.

**Syntax:** syntax for update multiple records from the table.

SQL> UPDATE <<TABLE NAME> SET <COLUMNANE>=<VALUE> WHERE <COLUMN NAME=<VALUE>;

#### Command:

SQL>UPDATE EMP SET SALARY = 16000, DESIGNATIN='ASST. PROF' WHERE EMPNO=102;

1 row updated.

SQL> SELECT \* FROM EMP;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| EMPNO | | ENAME | DESIGNATIN | SALARY | |
| 101 | NAGARAJAN | | LECTURER |  | 16000 |
| 102 | SARAVANAN | | ASST. PROF |  | 16000 |
| 103 | PANNERSELVAM | | ASST. PROF |  | 20000 |
| 104 | CHINNI | | HOD, PROF |  | 45000 |

#### DELETE

The **SQL DELETE** Query is used to delete the existing records from a table. You can use

**WHERE** clause with **DELETE** query to delete selected rows, otherwise all the records would be deleted.

#### QUERY: 06

Q5. Write a query to delete records from employee.

#### Syntax: Syntax for delete Records from the table:

SQL> DELETE <TABLE NAME> WHERE <COLUMN NAME>=<VALUE>;

#### Command:

SQL> DELETE EMP WHERE EMPNO=103;

1 row deleted.

SQL> SELECT \* FROM EMP;

EMPNO ENAME DESIGNATIN SALARY

|  |  |  |  |
| --- | --- | --- | --- |
| 101 | NAGARAJAN | LECTURER | 16000 |
| 102 | SARAVANAN | ASST. PROF | 16000 |
| 104 | CHINNI | HOD, PROF | 45000 |

**RESULT:**

Thus the SQL commands for DML has been verified and executed successfully.

#### Performing Insertion, Deletion, Modifying, Altering, Updating and Viewing records based on conditions.

**Ex: No: 02**

**: :**

**AIM:**

To performing insertion, deletion, modifying, altering, updating and viewing records based on conditions.

**ALGORITHM:**

**STEP 1:** Start the DBMS

**STEP 2:** Connect to the database (DB)

**STEP 3:** Create the table with its essential attributes.

**STEP 4:** Insert the record into table based on some condition using WHERE CLAUSE

**STEP 5:** Update the existing records into the table based on some condition

**STEP 6:** Delete the records in to the table based on some condition

**STEP 7:** Use commit for permanently save the records

**STEP 8:** Stop the program

#### DRL-DATA RETRIEVAL IMPLEMENTING ON SELECT COMMANDS

|  |  |  |
| --- | --- | --- |
| **Command:** |  | |
| SQL> CREATE TABLE EMP( |
|  | EMPNO | NUMBER (4), |
|  | ENAME | VARCHAR2 (10), |
|  | JOB | VARCHAR2(20), |
|  | MGR | NUMBER(4), |
|  | HIREDATE | DATE, |
|  | SAL | NUMBER(8,2), |
|  | DEPTNO | NUMBER(3) |
| ); |  |  |

Table created.

SQL> INSERT INTO EMP VALUES(7369,'SMITH','CLERK',5001,'17-DEC-

80','8000',200); 1 row created.

SQL> INSERT INTO EMP VALUES(7499,'ALLEN','SALESMAN',5002,'20-FEB-

80','3000',300); 1 row created.

SQL> INSERT INTO EMP VALUES(7521,'WARD','SALESMAN',5003,'22-FEB-

80','5000',500); 1 row created.

SQL> INSERT INTO EMP VALUES(7566,'JONES','MANAGER',5002,'02-APR-85','75000',200);

1 row created.

SQL> INSERT INTO EMP VALUES(7566,'RAJA','OWNER',5000,'30-APR-75',NULL,100);

1 row created.

SQL> INSERT INTO EMP VALUES(7566,'KUMAR','COE',5002,'12-JAN-87','55000',300);

1 row created.

SQL> INSERT INTO EMP VALUES(7499,'RAM KUMAR','SR.SALESMAN',5003,'22-JAN-87','12000.55',200);

1 row created.

SQL> INSERT INTO EMP VALUES(7521,'SAM KUMAR','SR.SALESMAN',5003,'22-JAN-

75','22000',300); 1 row created.

**THE SELECT STATEMENT SYNTAX WITH ADDITIONAL CLAUSES:**

Select [ Distinct / Unique ] ( \*columnname [ As alias}, ….]

From tablename

[ where condition ]

## [ Group BY group \_by\_expression ] [Having group\_condition ]

[ORDER BY {*col(s)|expr|numeric\_pos*} [ASC|DESC] [NULLS FIRST|LAST]];

SQL> SELECT \* FROM EMP;

EMPNO ENAME JOB MGR HIREDATE SAL DEPTNO

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 7369 | SMITH | CLERK | 5001 | 17-DEC-80 | 8000 | 200 |
| 7499 | ALLEN | SALESMAN | 5002 | 20-FEB-80 | 3000 | 300 |
| 7521 | WARD | SALESMAN | 5003 | 22-FEB-80 | 5000 | 500 |
| 7566 | JONES | MANAGER | 5002 | 02-APR-85 | 75000 | 200 |
| 7566 | RAJA | OWNER | 5000 | 30-APR-75 |  | 100 |
| 7566 | KUMAR | COE | 5002 | 12-JAN-87 | 55000 | 300 |
| 7499 | RAM KUMAR | SR.SALESMAN | 5003 | 22-JAN-87 | 12000.55 | 200 |
| 7521 | SAM KUMAR | SR.SALESMAN | 5003 | 22-JAN-75 | 22000 | 300 |

8 rows selected.

**BY USING SELECTED COLUNMS**

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| EMPNO |  | ENAME |  | JOB | SAL |
| 7369 |  | SMITH |  | CLERK | 8000 |
| 7499 |  | ALLEN |  | SALESMAN | 3000 |
| 7521 |  | WARD |  | SALESMAN | 5000 |
| 7566 |  | JONES |  | MANAGER | 75000 |
| 7566 |  | RAJA |  | OWNER |  |
| 7566 |  | KUMAR |  | COE | 55000 |
| 7499 | | RAM KUMAR SR.SALESMAN | | | 12000.55 |
| 7521 | | SAM KUMARSR.SALESMAN | | | 22000 |
| 8 rows selected. | |  | | |  |

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE SAL=5000;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| EMPNO | ENAME |  | JOB |  | SAL |
| 7521 | WARD |  | SALESMAN |  | 5000 |

**BY USING BETWEEN / NOT / IN / NULL / LIKE**

### BETWEEN Syntax:

SELECT *column\_name(s)*

FROM *table\_name*

WHERE *column\_name* BETWEEN *value1* AND *value2;*

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE SAL **BETWEEN** 10000 AND 30000;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| EMPNO |  | ENAME |  | JOB | SAL |
| 7499 |  | RAM KUMAR |  | SR.SALESMAN | 12000.55 |
| 7521 |  | SAM KUMAR |  | SR.SALESMAN | 22000 |

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE SAL **NOT BETWEEN** 10000 **AND** 30000;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| EMPNO | ENAME |  | JOB | SAL |
| 7369 | SMITH |  | CLERK | 8000 |
| 7499 | ALLEN |  | SALESMAN | 3000 |
| 7521 | WARD |  | SALESMAN | 5000 |
| 7566 | JONES |  | MANAGER | 75000 |
| 7566 | KUMAR |  | COE | 55000 |
| **IN Syntax** |  |  |  |  |

SELECT column\_name(s) FROM table\_name

WHERE column\_name IN (value1,value2,...);

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE SAL **IN** (1000.5,75000);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| EMPNO |  | ENAME | JOB | SAL |
| 7566 |  | JONES | MANAGER | 75000 |

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE SAL **NOT IN** (1000.5,75000);

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| EMPNO | ENAME | | JOB | SAL | |
| 7369 |  | SMITH | CLERK |  | 8000 |
| 7499 |  | ALLEN | SALESMAN |  | 3000 |
| 7521 |  | WARD | SALESMAN |  | 5000 |
| 7566 |  | KUMAR | COE |  | 55000 |
| 7499 | | RAM KUMAR SR.SALESMAN | | 12000.55 | |
| 7521  6 rows selected. | | SAM KUMARSR.SALESMAN | | 22000 | |

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE SAL **IS NULL**;

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| EMPNO |  | ENAME |  | JOB |  | SAL |
| 7566 |  | RAJA |  | OWNER |  |  |

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE SAL **IS NOT NULL**;

|  |  |  |  |
| --- | --- | --- | --- |
| EMPNO | ENAME | JOB | SAL |
| 7369 | SMITH | CLERK | 8000 |
| 7499 | ALLEN | SALESMAN | 3000 |
| 7521 | WARD | SALESMAN | 5000 |
| 7566 | JONES | MANAGER | 75000 |
| 7566 | KUMAR | COE | 55000 |
| 7499 | RAM KUMAR | SR.SALESMAN | 12000.55 |
| 7521 | SAM KUMAR | SR.SALESMAN | 22000 |

7 rows selected.

#### LIKE Syntax:

SELECT *column\_name(s)*

FROM *table\_name*

WHERE *column\_name* LIKE *pattern*;

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE SAL **LIKE** 55000;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| EMPNO |  | ENAME | JOB |  | SAL |
| 7566 |  | KUMAR | COE |  | 55000 |

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE ENAME **LIKE 'S%';**

EMPNO ENAME JOB SAL

7369 SMITH CLERK 8000

7521 SAM KUMARSR.SALESMAN 22000

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE ENAME **LIKE '%R';**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| EMPNO | ENAME |  | JOB |  | SAL |
| 7566 | KUMAR |  | COE |  | 55000 |
| 7499 | RAM KUMAR SR.SALESMAN | | | | 12000.55 |
| 7521 | SAM KUMARSR.SALESMAN | | | | 22000 |

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE ENAME **LIKE '%U%';**

EMPNO ENAME JOB SAL

-

7566 KUMAR COE 55000

7499 RAM KUMAR SR.SALESMAN 12000.55

7521 SAM KUMAR SR.SALESMAN 22000

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE ENAME **LIKE '%A%';**

EMPNO ENAME JOB SAL

|  |  |  |  |
| --- | --- | --- | --- |
| 7499 | ALLEN | SALESMAN | 3000 |
| 7521 | WARD | SALESMAN | 5000 |
| 7566 | RAJA | OWNER |  |
| 7566 | KUMAR | COE | 55000 |
| 7499 | RAM KUMAR | SR.SALESMAN | 12000.55 |
| 7521 | SAM KUMAR | SR.SALESMAN | 22000 |

6 rows selected.

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE ENAME **LIKE '%LL%';**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EMPNO |  | | ENAME |  | JOB | | | SAL |
| 7499 |  | | ALLEN |  | SALESMAN | | | 3000 |
| SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE ENAME | | | | | | | | |
| EMPNO | |  | ENAME | JOB | |  | SAL | |
| 7499 | |  | ALLEN | SALESMAN | |  | 3000 | |
| 7566 | |  | JONES | MANAGER | |  | 75000 | |

**LIKE '%E%';**

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE ENAME **LIKE '%U%A%';**

EMPNO ENAME JOB SAL

-

7566 KUMAR COE 55000

7499 RAM KUMAR SR.SALESMAN 12000.55

7521 SAM KUMAR SR.SALESMAN 22000

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE ENAME **LIKE 'R ';** //**3\_**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| EMPNO |  | ENAME |  | JOB |  | SAL |
| 7566 |  | RAJA |  | OWNER |  |  |

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE ENAME **LIKE 'R\_J\_';**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| EMPNO |  | ENAME | JOB |  | SAL |
| 7566 |  | RAJAOWNER |  |  |  |

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE ENAME **LIKE '\_M%';**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| EMPNO |  | ENAME |  | JOB |  | SAL |
| 7369 |  | SMITH |  | CLERK |  | 8000 |

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE ENAME **LIKE '\_M';**

no rows selected

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE ENAME **LIKE ' R'; // 4\_**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| EMPNO |  | ENAME | JOB |  | SAL |
| 7566 |  | KUMAR | COE |  | 55000 |

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE ENAME **LIKE 'K R'; // 3\_**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| EMPNO |  | ENAME |  | JOB |  | SAL |
| 7566 |  | KUMAR |  | COE |  | 55000 |

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE ENAME NOT **LIKE 'R\_J\_';**

|  |  |  |  |
| --- | --- | --- | --- |
| EMPNO | ENAME | JOB | SAL |
| 7369 | SMITH | CLERK | 8000 |
| 7499 | ALLEN | SALESMAN | 3000 |
| 7521 | WARD | SALESMAN | 5000 |
| 7566 | JONES | MANAGER | 75000 |
| 7566 | KUMAR | COE | 55000 |
| 7499 | RAM KUMAR | SR.SALESMAN | 12000.55 |
| 7521 | SAM KUMAR | SR.SALESMAN | 22000 |
| 7 rows selected. |  |  |  |

**RELATIONAL OPERATOR**

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE SAL=55000;

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| EMPNO |  | ENAME | JOB | SAL | |  | |
| 7566 |  | KUMAR | COE | 55000 | |
| SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE EMPNO ENAME JOB | | | | | | | SAL!=55 SAL |
| 7369 | | SMITH | | CLERK |  | | 8000 |
| 7499 | | ALLEN | | SALESMAN |  | | 3000 |
| 7521 | | WARD | | SALESMAN |  | | 5000 |
| 7566 | | JONES | | MANAGER |  | | 75000 |
| 7499 | | RAM KUMAR | | SR.SALESMAN |  | | 12000.55 |
| 7521  6 rows selected. | | SAM KUMAR | | SR.SALESMAN |  | | 22000 |

000;

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE SAL<>55000; EMPNO ENAMEJOB SAL

|  |  |  |  |
| --- | --- | --- | --- |
| 7369 SMITH | CLERK |  | 8000 |
| 7499 ALLEN | SALESMAN |  | 3000 |
| 7521 WARD | SALESMAN |  | 5000 |
| 7566 JONES | MANAGER |  | 75000 |

7499 RAM KUMARSR.SALESMAN12000.55

7521 SAM KUMARSR.SALESMAN 22000

6 rows selected.

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE SAL>55000;

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| EMPNO |  | ENAME |  | JOB |  | SAL |
| 7566 |  | JONES |  | MANAGER |  | 75000 |

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE SAL<55000;

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| EMPNO |  | ENAME |  | JOB |  | SAL |  |
| 7369 |  | SMITH |  | CLERK |  | 8000 |
| 7499 |  | ALLEN |  | SALESMAN |  | 3000 |
| 7521 |  | WARD |  | SALESMAN |  | 5000 |
| 7499 | RAM KUMAR SR.SALESMAN | | | | 12000.55 | | |
| 7521 | SAM KUMARSR.SALESMAN | | | | 22000 | | |

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE SAL<=55000;

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| EMPNO |  | ENAME |  | JOB |  | SAL |  |
| 7369 |  | SMITH |  | CLERK |  | 8000 |
| 7499 |  | ALLEN |  | SALESMAN |  | 3000 |
| 7521 |  | WARD |  | SALESMAN |  | 5000 |
| 7566 |  | KUMAR |  | COE |  | 55000 |
| 7499 | | RAM KUMAR SR.SALESMAN | | | 12000.55 | | |
| 7521  6 rows selected. | | SAM KUMARSR.SALESMAN | | | 22000 | | |

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE SAL>=55000;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| EMPNO |  | ENAME | JOB |  | SAL |
| 7566 |  | JONES | MANAGER |  | 75000 |
| 7566 |  | KUMAR | COE |  | 55000 |

**AND / OR**

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE JOB='SR.SALESMAN' **AND** SAL=22000; EMPNO ENAME JOB SAL

7521 SAM KUMARSR.SALESMAN 22000

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE JOB='SR.SALESMAN' **OR** SAL=22000;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| EMPNO |  | ENAME | JOB |  | SAL |
| 7499 |  | RAM KUMAR | SR.SALESMAN |  | 12000.55 |
| 7521 |  | SAM KUMAR | SR.SALESMAN |  | 22000 |

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

WHERE SAL=5000 **AND** (JOB='SR.SALESMAN' **OR** JOB='SALESMAN'); EMPNO ENAME JOB SAL

7521 WARD SALESMAN 5000

**Syntax:**

**ORDER BY**

SELECT *column\_name*,*column\_name*

FROM *table\_name*

ORDER BY *column\_name*,*column\_name* ASC|DESC;

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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 7499 |  | ALLEN |  | SALESMAN |  | 3000 |
| 7566 |  | JONES |  | MANAGER |  | 75000 |
| 7566 |  | KUMAR |  | COE |  | 55000 |
| 7566 |  | RAJA |  | OWNER |  |  |
| 7499 | | RAM KUMAR SR.SALESMAN | | | 12000.55 | |
| 7521 | | SAM KUMARSR.SALESMAN | | | 22000 | |
| 7369 | | SMITH CLERK | | | 8000 | |
| 7521 | | WARD SALESMAN | | | 5000 | |
| 8 rows selected. | |  | | |  | |

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP ORDER BY ENAME **DESC;**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| EMPNO | ENAME | JOB |  | SAL |
| 7521 | WARD | SALESMAN |  | 5000 |
| 7369 | SMITH | CLERK |  | 8000 |
| 7521 | SAM KUMAR | SR.SALESMAN |  | 22000 |
| 7499 | RAM KUMAR | SR.SALESMAN |  | 12000.55 |
| 7566 | RAJA | OWNER |  |  |
| 7566 | KUMAR | COE |  | 55000 |
| 7566 | JONES | MANAGER |  | 75000 |
| 7499 | ALLEN | SALESMAN |  | 3000 |
| 8 rows selected. |  |  |  |  |

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP ORDER BY ENAME **ASC;**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| EMPNO |  | ENAME |  | JOB |  | SAL |
| 7499 |  | ALLEN |  | SALESMAN |  | 3000 |
| 7566 |  | JONES |  | MANAGER |  | 75000 |
| 7566 |  | KUMAR |  | COE |  | 55000 |
| 7566 |  | RAJA |  | OWNER |  |  |
| 7499 | | RAM KUMAR SR.SALESMAN | | | 12000.55 | |
| 7521 | | SAM KUMARSR.SALESMAN | | | 22000 | |
| 7369 | | SMITH CLERK | | | 8000 | |
| 7521  8 rows selected. | | WARD SALESMAN | | | 5000 | |
|  | |  | | | **TOP** | |

// **TOP** clause is not in oracle instead of that **ROWNUM**

**Syntax**

SELECT *column\_name(s)*

FROM *table\_name*

WHERE ROWNUM <= *number*;

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE **ROWNUM <4;**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EMPNO | |  | ENAME |  | JOB | |  | SAL |
| 7369 | |  | SMITH |  | CLERK | |  | 8000 |
| 7499 | |  | ALLEN |  | SALESMAN | |  | 3000 |
| 7521 | |  | WARD |  | SALESMAN | |  | 5000 |
| SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE | | | | | | | | |
| EMPNO |  | | ENAME |  | JOB | SAL | |  |
| 7499 |  | | ALLEN |  | SALESMAN |  | 3000 |  |
| 7369 |  | | SMITH |  | CLERK |  | 8000 |  |
| 7521 |  | | WARD |  | SALESMAN |  | 5000 |  |

**ROWNUM <4 ORDER BY ENAME;**

### Syntax:

**Ex:**

### DISTINCT

SELECT DISTINCT column\_name,column\_name FROM table\_name;

SQL> SELECT **DISTINCT** JOB FROM EMP; JOB

CLERK SALESMAN SR.SALESMAN MANAGER COE

OWNER

6 rows selected.

**USING ALTER**

This can be used to add or remove columns and to modify the precision of the datatype.

1. **ADDING COLUMN Syntax:**

alter table <*table\_name*> add <*col datatype*>;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ex:**  SQL> DESC EMP; |  | | | |
| Name |  | Null? |  | Type |
| EMPNO |  |  |  | NUMBER(4) |
| ENAME |  |  |  | VARCHAR2(10) |
| JOB |  |  |  | VARCHAR2(20) |
| MGR |  |  |  | NUMBER(4) |
| HIREDATE |  |  |  | DATE |
| SAL |  |  |  | NUMBER(8,2) |
| DEPTNO |  |  |  | NUMBER(3) |

SQL> alter table EMP add TAX number; Table altered.

SQL> DESC EMP;

Name Null? Type

EMPNO NUMBER(4)

ENAME VARCHAR2(10)

JOB VARCHAR2(20)

MGR NUMBER(4)

HIREDATE DATE

SAL NUMBER(8,2)

DEPTNO NUMBER(3)

TAX NUMBER

1. **REMOVING COLUMN Syntax:**

alter table <*table\_name*> drop <*col datatype*>;

**Ex:**

SQL> alter table EMP drop column TAX; Table altered.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SQL> DESC EMP; |  | | | |
| Name |  | Null? |  | Type |
| EMPNO |  |  |  | NUMBER(4) |
| ENAME |  |  |  | VARCHAR2(10) |
| JOB |  |  |  | VARCHAR2(20) |
| MGR |  |  |  | NUMBER(4) |
| HIREDATE |  |  |  | DATE |
| SAL |  |  |  | NUMBER(8,2) |
| DEPTNO |  |  |  | NUMBER(3) |

1. **INCREASING OR DECREASING PRECISION OF A COLUMN Syntax:**

alter table <*table\_name*> modify <*col datatype*>;

**Ex:**

SQL> alter table EMP modify DEPTNO number(5); Table altered.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SQL> DESC EMP; |  | | | |
| Name |  | Null? |  | Type |
| EMPNO |  |  |  | NUMBER(4) |
| ENAME |  |  |  | VARCHAR2(10) |
| JOB |  |  |  | VARCHAR2(20) |
| MGR |  |  |  | NUMBER(4) |
| HIREDATE |  |  |  | DATE |
| SAL |  |  |  | NUMBER(8,2) |
| DEPTNO |  |  |  | NUMBER(5) |

\* To decrease precision the column should be empty.

1. **MAKING COLUMN UNUSED Syntax:**

alter table <*table\_name*> set unused column <*col*>;

**Ex:**

SQL> alter table EMP set unused column DEPTNO; Table altered.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SQL> DESC EMP; |  | | | |
| Name |  | Null? |  | Type |
| EMPNO |  |  |  | NUMBER(4) |
| ENAME |  |  |  | VARCHAR2(10) |
| JOB |  |  |  | VARCHAR2(20) |
| MGR |  |  |  | NUMBER(4) |
| HIREDATE |  |  |  | DATE |
| SAL |  |  |  | NUMBER(8,2) |

SQL> SELECT \* FROM EMP;

EMPNO ENAME JOB MGR HIREDATE SAL

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 7369 | SMITH | CLERK | 5001 17-DEC-80 | 8000 |
| 7499 | ALLEN | SALESMAN | 5002 20-FEB-80 | 3000 |
| 7521 | WARD | SALESMAN | 5003 22-FEB-80 | 5000 |
| 9 rows selected. |  |  |  |  |

Even though the column is unused still it will occupy memory.

1. **DROPPING UNUSED COLUMNS Syntax:**

alter table <*table\_name*> drop unused columns;

**Ex:**

SQL> alter table EMP drop unused columns; Table altered.

\* You can not drop individual unused columns of a table.

1. **RENAMING COLUMN Syntax:**

alter table <*table\_name*> rename column <*old\_col\_name*> to <*new\_col\_name*>;

**Ex:**

SQL> alter table EMP rename column SAL to SALARY; Table altered.

SQL> DESC EMP;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name |  | Null? |  | Type |
| EMPNO |  |  |  | NUMBER(4) |
| ENAME |  |  |  | VARCHAR2(10) |
| JOB |  |  |  | VARCHAR2(20) |
| MGR |  |  |  | NUMBER(4) |
| HIREDATE SALARY |  |  |  | DATE NUMBER(8,2) |
| **Method 1** |  |  |  | **INSERT** |

**GENERAL INSERT COMMAND:**

SQL> INSERT INTO EMP(EMPNO,ENAME,JOB,MGR,HIREDATE,SALARY) VALUES(1111,'RAMU','SALESMAN',5063,'12-JAN-87','5643.55');

1 row created.

**Method 2**

**WITHOUT SPECIFY THE COLUMNS DETAILS**

SQL> INSERT INTO Emp VALUES(1111,'RAMU','SALESMAN',5063,'12-JAN-

87','5643.55'); 1 row created.

**Method 3**

**INSERTING DATA INTO SPECIFIED COLUMNS**

SQL> INSERT INTO EMP(EMPNO,ENAME,JOB) VALUES(1111,'RAMU','SALESMAN'); 1 row created.

**Method 4**

**BY CHANGE THE ORDER OF COLUMNS**

SQL> INSERT INTO EMP(salary,EMPNO,ENAME,JOB) VALUES(35000,1111,'RAMU','SALESMAN'); 1 row created.

SQL> select \* from emp;

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| EMPNO | ENAME | JOB | MGR |  | HIREDATE | SALARY |
| 7369 | SMITH | CLERK | 5001 |  | 17-DEC-80 | 8000 |
| 7499 | ALLEN | SALESMAN | 5002 |  | 20-FEB-80 | 3000 |
| 7521 | WARD | SALESMAN | 5003 |  | 22-FEB-80 | 5000 |
| 7566 | JONES | MANAGER | 5002 |  | 02-APR-85 | 75000 |
| 7566 | RAJA | OWNER | 5000 |  | 30-APR-75 |  |
| 7566 | KUMAR | COE | 5002 |  | 12-JAN-87 | 55000 |
| 7499 | RAM KUMAR | SR.SALESMAN | 5003 |  | 22-JAN-87 | 12000.55 |
| 7521 | SAM KUMAR | SR.SALESMAN | 5003 |  | 22-JAN-75 | 22000 |
| 7521 | SAM KUMAR | SR.SALESMAN | 5003 |  | 22-JAN-75 | 22000 |
| 1111 | RAMU | SALESMAN | 5063 |  | 12-JAN-87 | 5643.55 |
| 1111 | RAMU | SALESMAN | 5063 |  | 12-JAN-87 | 5643.55 |
| 1111 | RAMU | SALESMAN |  |  |  |  |
| 1111  13 rows selected. | RAMU | SALESMAN |  |  |  | 35000 |
| **Method 5** |  |  |  |  |  |  |

**INSERT WITH SELECT**

Using this we can insert existing table data to another table in a single trip. But the table structure should be same.

**Syntax:**

Insert into <*table1*> select \* from <*table2*>;

|  |  |  |
| --- | --- | --- |
| **Ex:**  SQL> DESC EMP |  | |
| Name | Null? | Type |
| EMPNO |  | NUMBER(4) |
| ENAME |  | VARCHAR2(10) |
| JOB |  | VARCHAR2(20) |
| MGR |  | NUMBER(4) |
| HIREDATE  SALARY |  | DATE  NUMBER(8,2) |

SQL> create table EMPLOYEE(EMP\_NO,EMP\_NAME,EMP\_JOB,HR,HIREDATE,SALARY) as

select \* from EMP where 1 = 2;

Table created.

SQL> DESC EMPLOYEE

Name Null? Type

EMP\_NO NUMBER(4)

EMP\_NAME VARCHAR2(10)

EMP\_JOB VARCHAR2(20)

HR NUMBER(4)

HIREDATE DATE

SALARY NUMBER(8,2)

SQL> SELECT \* FROM

EMPLOYEE; no rows selected

SQL> insert into EMPLOYEE select \* from EMP; 13 rows created.

SQL> SELECT \* FROM EMPLOYEE;

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| EMP\_NO | EMP\_NAME |  | EMP\_JOB |  | HR | HIREDATE | SALARY |
| 7369 | SMITH |  | CLERK |  | 5001 | 17-DEC-80 | 8000 |
| 7499 | ALLEN |  | SALESMAN |  | 5002 | 20-FEB-80 | 3000 |
| 7521 | WARD |  | SALESMAN |  | 5003 | 22-FEB-80 | 5000 |
| 7566 | JONES |  | MANAGER |  | 5002 | 02-APR-85 | 75000 |
| 7566 | RAJA |  | OWNER |  | 5000 | 30-APR-75 |  |
| 7566 | KUMAR |  | COE |  | 5002 | 12-JAN-87 | 55000 |
| 7499 | RAM KUMAR SR.SALESMAN | | | 5003 | | 22-JAN-87 | 12000.55 |
| 7521 | SAM KUMARSR.SALESMAN | | | 5003 | | 22-JAN-75 | 22000 |
| 7521 | SAM KUMARSR.SALESMAN | | | 5003 | | 22-JAN-75 | 22000 |
| 1111 | RAMU SALESMAN | | | 5063 | | 12-JAN-87 | 5643.55 |
| 1111 | RAMU SALESMAN | | | 5063 | | 12-JAN-87 | 5643.55 |
| 1111 | RAMU SALESMAN | | |  | |  |  |
| 1111 | RAMU SALESMAN | | |  | |  | 35000 |

13 rows selected.

**Method 6**

**MULTIBLE INSERTS**

We have table called DEPT with the following columns and data SQL> select \* from DEPT;

|  |  |  |
| --- | --- | --- |
| DEPTNO | DNAME | LOC |
| ----------- | ---------- | ------- |
| 10 | accounting | new york |
| 20 | research | dallas |
| 30 | sales | Chicago |
| 40 | operations | boston |

**CREATE STUDENT TABLE**

SQL> Create table student(no number(2),name varchar(2),marks number(3));

1. **MULTI INSERT WITH ALL FIELDS**

SQL> Insert all

Into student values(1,’a’,100) Into student values(2,’b’,200) Into student values(3,’c’,300) Select \*from dept where deptno=10;

3 rows created.

SQL> Select \* from student;

|  |  |  |
| --- | --- | --- |
| NO | NAME | MARKS |
| ------- | -------- | ---------- |
| 1 | a | 100 |
| 2 | b | 200 |
| 3 | c | 300 |

1. **MULTI INSERT WITH SPECIFIED FIELDS**

SQL> insert all

Into student (no,name) values(4,’d’)

Into student(name,marks) values(’e’,400) Into student values(3,’c’,300)

Select \*from dept where deptno=10; 3 rows created.

SQL> Select \* from student; NO NAME MARKS

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | a |  | 100 |
| 2 | b |  | 200 |
| 3 | c |  | 300 |
| 4 | d |  |  |
|  | e |  | 400 |
| 3  6 rows selected. | c |  | 300 |

1. **MULTI INSERT WITH DUPLICATE ROWS**

SQL> insert all

Into student values(1,’a’,100) Into student values(2,’b’,200) Into student values(3,’c’,300)

Select \*from dept where deptno > 10; 9 rows created.

-- This inserts 9 rows because in the select statement retrieves 3 records (3 inserts for each row retrieved) SQL> Select \* from student;

NO NAME MARKS

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 |  | a |  | 100 |
| 2 |  | b |  | 200 |
| 3 |  | c |  | 300 |
| 4 |  | d |  |  |
|  |  | e |  | 400 |
| 3 |  | c |  | 300 |
| 1 |  | a |  | 100 |
| 1 |  | a |  | 100 |
| 1 |  | a |  | 100 |
| 2 |  | b |  | 200 |
| 2 |  | b |  | 200 |
| 2 |  | b |  | 200 |
| 3 |  | c |  | 300 |
| 3 |  | c |  | 300 |
| 3 |  | c |  | 300 |

15 rows selected.

1. **MULTI INSERT WITH CONDITIONS BASED**

SQL> create table mytbl1(name varchar2(20),no number(10));

**Table created.**

SQL> insert into mytbl1 values('ram',111);

**1 row created.**

SQL> insert into mytbl1 values('sam',222);

**1 row created.**

SQL> insert into mytbl1 values('tam',333);

**1 row created.**

SQL> select \* from mytbl1;

**NAME NO**

|  |  |
| --- | --- |
| **ram** | **111** |
| **sam** | **222** |
| **tam** | **333** |

SQL> create table yourtbl1(name varchar2(20),no number(10));

**Table created.**

SQL> create table yourtbl2(name varchar2(20),no number(10));

**Table created.**

SQL> create table yourtbl3(name varchar2(20),no number(10));

**Table created**.

SQL> select \* from yourtbl1; **no rows selected**

SQL> select \* from yourtbl2; **no rows selected**

SQL> select \* from yourtbl3; **no rows selected** SQL> insert all

when no > 111 then

into yourtbl1 values('ramu',1111) when name = 'sam' then

into yourtbl2 values('samu',2222) when name = 'tam' then

into yourtbl3 values('tamu',3333) select

\* from mytbl1 where no > 111;

**4 rows created.**

SQL> select \* from mytbl1;

**NAME NO**

|  |  |
| --- | --- |
| **ram** | **111** |
| **sam** | **222** |
| **tam** | **333** |

SQL> select \* from yourtbl1;

**NAME NO**

**ramu 1111**

**ramu 1111**

SQL> select \* from yourtbl2;

**NAME NO**

**samu 2222**

SQL> select \* from yourtbl3;

**NAME NO**

**tamu 3333**

-- This inserts 4 rows because the first condition satisfied 3 times, second condition satisfied once and the last none.

1. **MULTI INSERT WITH CONDITIONS BASED AND ELSE**

SQL> create table mytbl1(name varchar2(20),no number(10)); Table created.

SQL> insert into mytbl1 values('ram',111); 1 row created.

SQL> insert into mytbl1 values('sam',222); 1 row created.

SQL> insert into mytbl1 values('tam',333); 1 row created.

SQL> select \* from mytbl1; NAME NO

|  |  |
| --- | --- |
| ram | 111 |
| sam | 222 |
| tam | 333 |

SQL> create table yourtbl1(name varchar2(20),no number(10)); Table created.

SQL> create table yourtbl2(name varchar2(20),no number(10)); Table created.

SQL> create table yourtbl3(name varchar2(20),no number(10)); Table created.

SQL> create table yourtbl4(name varchar2(20),no number(10)); Table created.

SQL> insert all

when no > 111 then

into yourtbl1 values('ramu',1111) when name = 'sam' then

into yourtbl2 values('samu',2222) when name = 'tam' then

into yourtbl3 values('tamu',3333) else

into yourtbl4 values('chotta',4444) select \* from mytbl1 where no > 111;

**4 rows created.**

SQL> select \* from yourtbl1; NAME NO

ramu 1111

ramu 1111

SQL> select \* from yourtbl2; NAME NO

samu 2222

SQL> select \* from yourtbl3; NAME NO

tamu 3333

1. **MULTI INSERT WITH CONDITIONS BASED AND FIRST**

SQL> create table mytbl1(name varchar2(20),no number(10)); Table created.

SQL> insert into mytbl1 values('ram',111); 1 row created.

SQL> insert into mytbl1 values('sam',222); 1 row created.

SQL> insert into mytbl1 values('tam',333); 1 row created.

SQL> create table yourtbl1(name varchar2(20),no number(10)); Table created.

SQL> create table yourtbl2(name varchar2(20),no number(10)); Table created.

SQL> create table yourtbl3(name varchar2(20),no number(10)); Table created.

SQL> create table yourtbl4(name varchar2(20),no number(10)); Table created.

SQL> select \* from mytbl1; NAME NO

|  |  |
| --- | --- |
| ram | 111 |
| sam | 222 |
| tam | 333 |

SQL> insert first

when no=111 then

into yourtbl1 values('ramu',1111) when name = 'sam' then

into yourtbl2 values('samu',2222) when name = 'tam' then

into yourtbl3 values('tamu',3333)

select \* from mytbl1 where name='ram'; 1 row created.

SQL> select \* from yourtbl1; NAME NO

ramu 1111

-- This inserts 1 record because the first clause avoid to check the remaining conditions once the condition is satisfied.

1. **MULTI INSERT WITH CONDITIONS BASED, FIRST AND ELSE**

SQL> create table mytbl1(name varchar2(20),no number(10)); Table created.

SQL> insert into mytbl1 values('ram',111); 1 row created.

SQL> insert into mytbl1 values('sam',222); 1 row created.

SQL> insert into mytbl1 values('tam',333); 1 row created.

SQL> create table yourtbl1(name varchar2(20),no number(10)); Table created.

SQL> create table yourtbl2(name varchar2(20),no number(10)); Table created.

SQL> create table yourtbl3(name varchar2(20),no number(10)); Table created.

SQL> create table yourtbl4(name varchar2(20),no number(10)); Table created.

SQL> select \* from mytbl1;

|  |  |
| --- | --- |
| NAME | NO |
| ram | 111 |
| sam | 222 |
| tam | 333 |
| SQL> insert first |  |

when no=111 then

into yourtbl1 values('ramu',1111) when name = 'bam' then

into yourtbl2 values('samu',2222) when name = 'tam' then

into yourtbl3 values('tamu',3333) else

into yourtbl4 values('kamu',4444) select

\* from mytbl1 where name='ram'; 1 row created.

SQL> select \* from yourtbl1; NAME NO

ramu 1111

SQL> select \* from yourtbl2; no rows selected

SQL> select \* from yourtbl3; no rows selected

SQL> select \* from yourtbl4; no rows selected

1. **MULTI INSERT WITH MULTIBLE TABLES**

SQL> create table mytbl1(name varchar2(20),no number(10)); Table created.

SQL> insert into mytbl1 values('ram',111); 1 row created.

SQL> insert into mytbl1 values('sam',222); 1 row created.

SQL> insert into mytbl1 values('tam',333); 1 row created.

SQL> select \* from mytbl1;

NAME NO

|  |  |
| --- | --- |
| ram | 111 |
| sam | 222 |
| tam | 333 |

SQL> create table yourtbl1(name varchar2(20),no number(10)); Table created.

SQL> create table yourtbl2(name varchar2(20),no number(10)); Table created.

SQL> create table yourtbl3(name varchar2(20),no number(10)); Table created.

SQL> create table yourtbl4(name varchar2(20),no number(10)); Table created.

SQL> insert all

into yourtbl1 values('ramu',11111) into yourtbl2 values('samu',22222) into yourtbl3 values('tamu',33333) into yourtbl4 values('kamu',44444)

select \* from mytbl1 where name='ram'; 4 rows created.

SQL> select \* from yourtbl1; NAME NO

ramu 11111

SQL> select \* from yourtbl2; NAME NO

samu 22222

SQL> select \* from yourtbl3; NAME NO

tamu 33333

SQL> select \* from yourtbl4; NAME NO

kamu 44444

\*\* You can use multi tables with specified fields, with duplicate rows, with conditions, with first and else clauses.

**GROUP BY**

Using group by, we can create groups of related information. Columns used in select must be used with group by; otherwise it was not a group by expression.

**Ex:**

SQL> select \* from emp;

EMPNO ENAME JOB MGR HIREDATE SAL DEPTNO

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7369 | |  | SMITH |  | CLERK |  | 500117-DEC-80 | 8000 | 200 |
| 7499 | |  | ALLEN |  | SALESMAN |  | 500220-FEB-80 | 3000 | 300 |
| 7521 | WARD SALESMAN | | | | | 5003 22-FEB-80 5000 | | | 500 |
| 7499 | RAM KUMARSR.SALESMAN | | | | | 5003 22-JAN-87 12000.55200 | | |  |
| 7566 | JONES MANAGER | | | | | 5002 02-APR-85 75000 | | | 200 |
| 7521 | SAM KUMARSR.SALESMAN | | | | | 5003 22-JAN-75 22000 300 | | |  |

6 rows selected.

SQL> select job from EMP group by job; JOB

CLERK SALESMAN SR.SALESMAN MANAGER

SQL> select job,SUM(SAL) from EMP group by job;

|  |  |  |
| --- | --- | --- |
| JOB |  | SUM(SAL) |
| CLERK SALESMAN SR.SALESMAN  MANAGER | 8000  75000 | 8000  34000.55 |

**HAVING**

This will work as where clause which can be used only with group by because of absence of where clause in group by.

SQL> select deptno,job,sum(sal) Total\_Salary\_Of\_Each\_Dept from emp group by deptno,job having sum(sal) > 3000;

DEPTNO JOB TOTAL\_SALARY\_OF\_EACH\_DEPT

|  |  |  |  |
| --- | --- | --- | --- |
| 200 |  | MANAGER | 75000 |
| 200 |  | SR.SALESMAN | 12000.55 |
| 200 |  | CLERK | 8000 |
| 500 |  | SALESMAN | 5000 |
| 300 |  | SR.SALESMAN | 22000 |

SQL> select deptno,job,sum(sal) Total\_Salary\_of\_Each\_Dept from emp

group by deptno,job having sum(sal) > 3000 order by job;

DEPTNO JOB TOTAL\_SALARY\_OF\_EACH\_DEPT

|  |  |  |  |
| --- | --- | --- | --- |
| 200 |  | CLERK | 8000 |
| 200 |  | MANAGER | 75000 |
| 500 |  | SALESMAN | 5000 |
| 200 |  | SR.SALESMAN | 12000.55 |
| 300 |  | SR.SALESMAN | 22000 |

#### USING DELETE

SQL> select \* from EMP;

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EMPNO |  | ENAME |  | JOB | MGR |  | HIREDATE |  | SAL |  | DEPTNO |
| 1001 |  | RAM |  | CLERK | 5001 |  | 17-DEC-84 |  | 8000 |  | 301 |
| 1002 |  | SAM |  | MANAGER | 5001 |  | 11-JAN-81 |  | 85000 |  | 301 |
| 1003 |  | SAMU |  | SALESMAN | 5003 |  | 09-FEB-82 |  | 8000 |  | 302 |
| 1004 |  | RAMU |  | SR.SALESMAN | 5002 |  | 22-JUN-85 |  | 45000 |  | 303 |

SQL> DELETE EMP WHERE ENAME='SAM';

1 row deleted.

SQL> select \* from EMP;

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EMPNO |  | ENAME |  | JOB | MGR |  | HIREDATE |  | SAL DEPTNO |
| 1001 |  | RAM |  | CLERK | 5001 |  | 17-DEC-84 |  | 8000 301 |
| 1003 |  | SAMU |  | SALESMAN | 5003 |  | 09-FEB-82 |  | 8000 302 |
| 1004 |  | RAMU |  | SR.SALESMAN | 5002 |  | 22-JUN-85 |  | 45000 303 |

SQL> DELETE EMP WHERE ENAME LIKE 'R ';

1 row deleted.

SQL> select \* from EMP;

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EMPNO |  | ENAME |  | JOB | MGR |  | HIREDATE |  | SAL |  | DEPTNO |
| 1003 |  | SAMU |  | SALESMAN | 5003 |  | 09-FEB-82 |  | 8000 |  | 302 |
| 1004 |  | RAMU |  | SR.SALESMAN | 5002 |  | 22-JUN-85 |  | 45000 |  | 303 |

SQL> DELETE FROM EMP WHERE

ENAME='SAMU'; 1 row deleted.

#### TO DELETE ALL RECORDS

SQL> DELETE FROM

EMP; 1 row deleted.

#### DELETE DUPLICATE ROWS

SQL> SELECT \* FROM myTBL;

|  |  |
| --- | --- |
| NAME | MARK |
| RAM | 101 |
| RAM | 101 |
| SAM | 102 |
| SAM | 102 |
| RAMU |  |
| RAMU |  |
| SAMU | 103 |
| SAMU | 103 |
| SAMU | 103 |
| TAM |  |
| RAJA | 555 |
| KAJA | 123 |

12 rows selected.

SQL> delete from myTBL t1

where t1.rowid > (select min(t2.rowID) from myTBL t2 where t1.name = t2.name and t1.mark = t2.mark); 4 rows deleted.

SQL> SELECT \* FROM myTBL;

|  |  |
| --- | --- |
| NAME | MARK |
| --------- | ---------- |
| RAM | 101 |
| SAM | 102 |
| RAMU |  |
| SAMU | 103 |
| TAM |  |
| RAJA | 555 |
| KAJA  8 rows selected. | 123 |

#### Using UPDATE

SQL> select \* from EMP;

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EMPNO |  | ENAME |  | JOB | MGR |  | HIREDATE |  | SAL |  | DEPTNO |
| 1001 |  | RAM |  | CLERK | 5001 |  | 17-DEC-84 |  | 8000 |  | 301 |
| 1002 |  | SAM |  | MANAGER | 5001 |  | 11-JAN-81 |  | 85000 |  | 301 |
| 1003 |  | SAMU |  | SALESMAN | 5003 |  | 09-FEB-82 |  | 8000 |  | 302 |

SQL> UPDATE EMP SET SAL = 55555,JOB = 'SR.MANAGER' WHERE ENAME LIKE

'R '; 1 row updated.

SQL> select \* from EMP;

EMPNO ENAME JOB MGR HIREDATE SAL DEPTNO

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1001 |  | RAM |  | SR.MANAGER | 5001 |  | 17-DEC-84 | 55555 |  | 301 |
| 1002 |  | SAM |  | MANAGER | 5001 |  | 11-JAN-81 | 85000 |  | 301 |
| 1003 |  | SAMU |  | SALESMAN | 5003 |  | 09-FEB-82 | 8000 |  | 302 |

SQL> UPDATE EMP SET SAL = 55555,JOB = 'SR.MANAGER';

3 rows updated.

SQL> select \* from EMP;

EMPNO ENAME JOB MGR HIREDATE SAL DEPTNO

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1001 |  | RAM |  | SR.MANAGER | 5001 |  | 17-DEC-84 |  | 55555 |  | 301 |
| 1002 |  | SAM |  | SR.MANAGER | 5001 |  | 11-JAN-81 |  | 55555 |  | 301 |
| 1003 |  | SAMU |  | SR.MANAGER | 5003 |  | 09-FEB-82 |  | 55555 |  | 302 |

**RESULT:**

Thus the SQL commands for Performing Insertion, Deletion, Modifying, Altering, Updating and

Viewing records based on conditions has been verified and executed successfully.

#### Creation of Views, Synonyms, Sequence, Indexes, Save point.

**Ex: No: 03 (3.1) VIEWS**

**: :**

**AIM:**

To create the view, execute and verify the various operations on views.

**OBJECTIVE:**

Views Helps to encapsulate complex query and make it reusable.

Provides user security on each view - it depends on your data policy security.

Using view to convert units - if you have a financial data in US currency, you can create view to convert them into Euro for viewing in Euro currency.

A view is nothing more than a SQL statement that is stored in the database with an associated name. A view is actually a composition of a table in the form of a predefined SQL query.

A view can contain all rows of a table or select rows from a table. A view can be created from one or many tables which depends on the written SQL query to create a view.

Views, which are kind of virtual tables, allow users to do the following: Structure data in a way that users or classes of users find natural or intuitive.

Restrict access to the data such that a user can see and (sometimes) modify exactly what they need and no more.

**ALGORITHM:**

STEP 1: Start the DMBS.

STEP 2: Connect to the existing database(DB) STEP 3: Create the table with its essential attributes. STEP 4: Insert record values into the table.

STEP 5: Create the view from the above created table. STEP 6: Display the data presented on the VIEW. STEP 7: Insert the records into the VIEW,

STEP 8: Check the database object table and view the inserted values presented STEP 9: Execute different Commands and extract information from the View. STEP 10: Stop the DBMS.

**COMMANDS EXECUTION**

#### CREATION OF TABLE:

Database views are created using the **CREATE VIEW** statement. Views can be created from a single table, multiple tables, or another view. To create a view, a user must have the appropriate system privilege according to the specific implementation.

SQL> CREATE TABLE **EMPLOYEE** (

);

Table created.

EMPLOYEE\_NAME VARCHAR2(10), EMPLOYEE\_NO NUMBER(8), DEPT\_NAME VARCHAR2(10),

DEPT\_NO NUMBER (5),

DATE\_OF\_JOIN DATE

**TABLE DESCRIPTION:**

SQL> DESC **EMPLOYEE**;

NAME NULL? TYPE

EMPLOYEE\_NAME VARCHAR2(10)

EMPLOYEE\_NO NUMBER(8)

DEPT\_NAME VARCHAR2(10)

DEPT\_NO NUMBER(5)

DATE\_OF\_JOIN DATE

**CREATE VIEW**

**SUNTAX FOR CREATION OF VIEW**

**CREATE [OR REPLACE] [FORCE ] VIEW viewname [(column-name, column-name)] AS Query [with check option];**

**CREATION OF VIEW**

SQL> CREATE VIEW **EMPVIEW** AS SELECT EMPLOYEE\_NAME, EMPLOYEE\_NO,

DEPT\_NAME, DEPT\_NO,

DATE\_OF\_JOIN FROM **EMPLOYEE**;

View Created. **DESCRIPTION OF VIEW** SQL> DESC EMPVIEW;

NAME NULL? TYPE

EMPLOYEE\_NAME VARCHAR2(10)

EMPLOYEE\_NO NUMBER(8)

DEPT\_NAME VARCHAR2(10)

DEPT\_NO NUMBER(5)

**DISPLAY VIEW:**

SQL> SELECT \* FROM EMPVIEW;

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| EMPLOYEE\_N |  | EMPLOYEE\_NO |  | DEPT\_NAME |  | DEPT\_NO |
| RAVI |  | 124 |  | ECE |  | 89 |
| VIJAY |  | 345 |  | CSE |  | 21 |
| RAJ |  | 98 |  | IT |  | 22 |
| GIRI |  | 100 |  | CSE |  | 67 |

**INSERTION OF VALUES INTO VIEW**

Rows of data can be inserted into a view. The same rules that apply to the UPDATE command also apply to the INSERT command. Here, we can not insert rows in CUSTOMERS\_VIEW because we have not included all the NOT NULL columns in this view, otherwise you can insert rows in a view in similar way as you insert them in a table.

**INSERT STATEMENT SYNTAX:**

SQL> INSERT INTO <VIEW\_NAME> (COLUMN NAME1, …) VALUES(VALUE1,….);

**COMMAND:**

SQL> INSERT INTO EMPVIEW VALUES ('SRI', 120,'CSE', 67,'16-NOV-1981'); 1 ROW CREATED.

SQL> SELECT \* FROM EMPVIEW;

EMPLOYEE\_N EMPLOYEE\_NO DEPT\_NAMEDEPT\_NO

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| RAVI |  | 124 |  | ECE |  | 89 |
| VIJAY |  | 345 |  | CSE |  | 21 |
| RAJ |  | 98 |  | IT |  | 22 |
| GIRI |  | 100 |  | CSE |  | 67 |
| SRI |  | 120 |  | CSE |  | 67 |

SQL> SELECT \* FROM EMPLOYEE;

EMPLOYEE\_N EMPLOYEE\_NO DEPT\_NAMEDEPT\_NO DATE\_OF\_J

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| RAVI |  | 124 |  | ECE | 89 |  | 15-JUN-05 |
| VIJAY |  | 345 |  | CSE | 21 |  | 21-JUN-06 |
| RAJ |  | 98 |  | IT | 22 |  | 30-SEP-06 |
| GIRI |  | 100 |  | CSE | 67 |  | 14-NOV-81 |
| SRI |  | 120 |  | CSE | 67 |  | 16-NOV-81 |

**DELETION OF VIEW:**

Rows of data can be deleted from a view. The same rules that apply to the UPDATE and INSERT commands apply to the DELETE command.

**DELETE STATEMENT SYNTAX:**

SQL> DELETE <VIEW\_NMAE>WHERE <COLUMN NMAE> =’VALUE’;

**Command:**

SQL> DELETE FROM EMPVIEW WHERE

EMPLOYEE\_NAME='SRI'; 1 row deleted.

SQL> SELECT \* FROM EMPVIEW;

EMPLOYEE\_N EMPLOYEE\_NO DEPT\_NAMEDEPT\_NO

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| RAVI |  | 124 |  | ECE |  | 89 |
| VIJAY |  | 345 |  | CSE |  | 21 |
| RAJ |  | 98 |  | IT |  | 22 |
| GIRI |  | 100 |  | CSE |  | 67 |

**UPDATE STATEMENT:**

A view can be updated under certain conditions:

The SELECT clause may not contain the keyword DISTINCT. The SELECT clause may not contain summary functions.

The SELECT clause may not contain set functions. The SELECT clause may not contain set operators.

The SELECT clause may not contain an ORDER BY clause. The FROM clause may not contain multiple tables.

The WHERE clause may not contain subqueries.

The query may not contain GROUP BY or HAVING. Calculated columns may not be updated.

All NOT NULL columns from the base table must be included in the view in order for the INSERT query to function.

**SYNTAX:**

SQL>UPDATE <VIEW\_NAME> SET< COLUMN NAME> = <COLUMN NAME> +<VIEW> WHERE <COLUMN NAME>=VALUE;

**Command:**

SQL> UPDATE EMPKAVIVIEW SET EMPLOYEE\_NAME='KAVI' WHERE

EMPLOYEE\_NAME='RAVI'; 1 row updated.

SQL> SELECT \* FROM EMPKAVIVIEW;

EMPLOYEE\_N EMPLOYEE\_NO DEPT\_NAMEDEPT\_NO

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| KAVI |  | 124 |  | ECE |  | 89 |
| VIJAY |  | 345 |  | CSE |  | 21 |
| RAJ |  | 98 |  | IT |  | 22 |
| GIRI |  | 100 |  | CSE |  | 67 |

**DROP A VIEW:**

Obviously, where you have a view, you need a way to drop the view if it is no longer needed.

**SYNTAX:**

SQL> DROP VIEW <VIEW\_NAME>

**EXAMPLE**

SQL>DROP VIEW

EMPVIEW; view droped

**CREATE A VIEW WITH SELECTED FIELDS:**

**SYNTAX:**

SQL>CREATE [OR REPLACE] VIEW <VIEW NAME>AS SELECT <COLUMN NAME1>…..FROM

<TABLE ANME>;

**EXAMPLE-2:**

SQL> CREATE OR REPLACE VIEW EMPL\_VIEW1 AS SELECT EMPNO, ENAME,SALARY FROM EMPL; SQL> SELECT \* FROM EMPL\_VIEW1;

**EXAMPLE-3:**

SQL> CREATE OR REPLACE VIEW EMPL\_VIEW2 AS SELECT \* FROM EMPL WHERE DEPTNO=10; SQL> SELECT \* FROM EMPL\_VIEW2;

**Note:**

Replace is the keyboard to avoid the error “ora\_0095:name is already used by an existing ab

**CHANGING THE COLUMN(S) NAME M THE VIEW DURING AS SELECT**

**STATEMENT: TYPE-1:**

**SQL>** CREATE OR REPLACE VIEW EMP\_TOTSAL(EID,NAME,SAL) AS SELECT EMPNO, ENAME,SALARY

FROM EMPL;

View created.

|  |  |  |
| --- | --- | --- |
| EMPNO | ENAME S | ALARY |
| 7369 | SMITH | 1000 |
| 7499 | MARK 1050 |  |
| 7565 | WILL | 1500 |
| 7678 | JOHN | 1800 |
| 7578 | TOM | 1500 |
| 7548 | TURNER | 1500 |
| 6 rows selected. |  |  |
| View created. |  |  |

SQL> SELECT \* FROM EMP\_TOTSAL;

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| EMPNO | ENAME |  | SALARY |  | MGRNO |  | DEPTNO |
| 7578 | TOM |  | 1500 |  | 7298 |  | 10 |
| 7548  View created. | TURNER |  | 1500 |  | 7298 |  | 10 |
| **TYPE-2:** |  |  |  |  |  |  |  |

SQL> CREATE OR REPLACE VIEW EMP\_TOTSAL AS SELECT EMPNO "EID", ENAME "NAME", SALARY "SAL" FROM EMPL;

SQL> SELECT \* FROM EMP\_TOTSAL;

**EXAMPLE FOR JOIN VIEW:**

**TYPE-3:**

SQL> CREATE OR REPLACE VIEW DEPT\_EMP AS SELECT A.EMPNO "EID", A.ENAME "EMPNAME",

A.DEPTNO "DNO", B.DNAME "D\_NAME", B.LOC "D\_LOC"

FROM EMPL A,DEPMT B WHERE A.DEPTNO=B.DEPTNO;

SQL> SELECT \* FROM DEPT\_EMP;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| EID | NAME SAL | |  | |
| 7369 | SMITH | | 1000 | |
| 7499 | MARK 1050 | |  | |
| 7565 | WILL | | 1500 | |
| 7678 | JOHN | | 1800 | |
| 7578 | TOM | | 1500 | |
| 7548 | TURNER | | 1500 | |
| 6 rows selected. View created. | |  | |  |
| EID | | NAME SAL | |  |
| 7369 | | SMITH | | 1000 |
| 7499 | | MARK 1050 | |  |
| 7565 | | WILL | | 1500 |
| 7678 | | JOHN | | 1800 |
| 7578 | | TOM | | 1500 |
| 7548 | | TURNER | | 1500 |
| 6 rows selected. | |  | |  |
| View created. | |  | |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| EID |  | EMPNAME | DNO | D\_NAME | D\_LOC |
| 7578 |  | TOM | 10 | ACCOUNT | NEW YORK |
| 7548 |  | TURNER | 10 | ACCOUNT | NEW YORK |
| 7369 |  | SMITH | 20 | SALES | CHICAGO |
| 7678 |  | JOHN | 20 | SALES | CHICAGO |
| 7499 | MARK 30 | | RESEARCHZURICH | |  |
| 7565 | WILL | | 30 RESEARCH | | ZURICH |

**VIEW READ ONLY AND CHECK OPTION:**

**READ ONLY CLAUSE:**

You can create a view with read only option which enable other to only query .no DML operation can be performed to this type of a view.

**EXAMPLE-4:**

SQL>CREATE OR REPLACE VIEW EMP\_NO\_DML AS SELECT \* FROM EMPL WITH READ ONLY;

**WITH CHECK OPTION CLAUSE:**

**EXAMPLE-4:**

SQL> CREATE OR REPLACE VIEW EMP\_CK\_OPTION AS SELECT EMPNO, ENAME, SALARY, DEPTNO

FROM EMPL WHERE DEPTNO=10 WITH CHECK OPTION; SQL> SELECT \* FROM EMP\_CK\_OPTION;

**JOIN VIEW:**

**EXAMPLE-5:**

SQL> CREATE OR REPLACE VIEW DEPT\_EMP\_VIEW AS SELECT A.EMPNO,

A.ENAME, A.DEPTNO, B.DNAME, B.LOC

FROM EMPL A, DEPMT B WHERE A.DEPTNO=B.DEPTNO;

SQL> SELECT \* FROM DEPT\_EMP\_VIEW;

View created.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| EMPNO | ENAME |  | SALARY |  | DEPTNO |
| 7578 | TOM |  | 1500 |  | 10 |
| 7548  View created. | TURNER |  | 1500 |  | 10 |

EMPNO ENAME DEPTNO DNAME LOC

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 7578 | TOM |  | 10 |  | ACCOUNT | NEW YORK |
| 7548 | TURNER |  | 10 |  | ACCOUNT | NEW YORK |
| 7369 | SMITH |  | 20 |  | SALES | CHICAGO |
| 7678 | JOHN |  | 20 |  | SALES | CHICAGO |
| 7499 | MARK |  | 30 |  | RESEARCH | ZURICH |
| 7565 | WILL |  | 30 |  | RESEARCH | ZURICH |
| 6 rows selected. |  |  |  |  |  |  |

**FORCE VIEW:**

**EXAMPLE-6:**

SQL> CREATE OR REPLACE FORCE VIEW MYVIEW AS SELECT \* FROM XYZ; SQL> SELECT \* FROM MYVIEW;

SQL> CREATE TABLE XYZ AS SELECT EMPNO,ENAME,SALARY,DEPTNO FROM EMPL; SQL> SELECT \* FROM XYZ;

SQL> CREATE OR REPLACE FORCE VIEW MYVIEW AS SELECT \* FROM XYZ; SQL> SELECT \* FROM MYVIEW;

Warning: View created with compilation errors. SELECT \* FROM MYVIEW

\*

ERROR at line 1:

ORA-04063: view "4039.MYVIEW" has errors Table created.

EMPNO ENAME SALARY DEPTNO

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 7369 | SMITH | 1000 | 20 | |
| 7499 | MARK 1050 | 30 |  | |
| 7565 | WILL | 1500 | 30 | |
| 7678 | JOHN | 1800 | 20 | |
| 7578 | TOM | 1500 | 10 | |
| 7548  6 rows selected. View created. | TURNER | 1500 | 10 | |
| EMPNO | ENAME | SALARY | | DEPTNO |
| 7369 | SMITH | 1000 | | 20 |
| 7499 | MARK 1050 | 30 | |  |
| 7565 | WILL | 1500 | | 30 |
| 7678 | JOHN | 1800 | | 20 |
| 7578 | TOM | 1500 | | 10 |
| 7548  6 rows selected | TURNER | 1500 | | 10 |

**COMPILING A VIEW:**

**SYNTAX:**

ALTER VIEW <VIEW\_NAME> COMPILE;

**EXAMPLE:**

SQL> ALTER VIEW MYVIEW COMPILE;

**RESULT**:

Thus the SQL commands for View has been verified and executed successfully.

#### Synonyms

**Ex: No: 03 (3.2)**

**: :**

**AIM:**

To create the Synonyms and verify the various operations on Synonyms

**OBJECTIVE:**

A **synonym** is an alias for any table, view, materialized view, sequence, procedure, function, package, type, Java class schema object, user-defined object type, or another synonym. Because a synonym is simply an alias, it requires no storage other than its definition in the data dictionary.

Synonyms are often used for security and convenience. For example, they can do the following:

Mask the name and owner of an object

Provide location transparency for remote objects of a distributed database Simplify SQL statements for database users

Enable restricted access similar to specialized views when exercising fine-grained access control

You can create both public and private synonyms. A **public** synonym is owned by the special user group named PUBLIC and every user in a database can access it. A **private** synonym is in the schema of a specific user who has control over its availability to others.

**ALGORITHM:**

STEP 1: Start the DMBS.

STEP 2: Connect to the existing database(DB) STEP 3: Create the table with its essential attributes. STEP 4: Insert record values into the table.

STEP 5: Create the synonyms from the above created table or any data object. STEP 6: Display the data presented on the synonyms.

STEP 7: Insert the records into the synonyms,

STEP 8: Check the database object table and view the inserted values presented STEP 9: Stop the DBMS.

**Example:**

**Syntax:**

**SQL>CREATE SYNONYM** synonymName **FOR** object;

OR

**SQL>**CREATE SYNONYM **tt** for **test1;**

Dependent Oject - **tt** (SYNONYM NAME ) Referenced Object - **test1** (TABLE NAME) **USAGE:**

Using emp you can perform DML operation as you have create sysnonm for table object If employees table is dropped then status of emp will be invalid.

Local Dependencies are automatically managed by oracle server.

#### COMMANDS:

**CREATE THE TABLE**

SQL> CREATE TABLE student\_table(Reg\_No number(5),NAME varchar2(5),MARK number(3)); Table created.

#### INSERT THE VALUES INTO THE TABLE

SQL> insert into student\_table values(10001,'ram',85); 1 row created.

SQL> insert into student\_table values(10002,'sam',75); 1 row created.

SQL> insert into student\_table values(10003,'samu',95); 1 row created.

SQL> select \* from STUDENT\_TABLE;

|  |  |  |
| --- | --- | --- |
| REG\_NO | NAME | MARK |
| ------------ | --------- | ---------- |
| 10001 | ram | 85 |
| 10002 | sam | 75 |
| 10003 | samu | 95 |

#### CREATE THE SYNONYM FROM TABLE

SQL> CREATE SYNONYM STUDENT\_SYNONYM FOR STUDENT\_TABLE;

Synonym created.

#### DISPLAY THE VALUES OF TABLE BY USING THE SYSNONYM

SQL> select \* from STUDENT\_SYNONYM;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| REG\_NO |  | NAME |  | MARK |
| 10001 |  | ram |  | 85 |
| 10002 |  | sam |  | 75 |
| 10003 |  | samu |  | 95 |

#### INSERT THE VALUES TO THE SYNONYM

SQL> insert into student\_SYNONYM values(10006,'RAJA',80); 1 row created.

#### DISPLAY THE VALUES IN BOTH TABLE AND SYNONYM

SQL> select \* from STUDENT\_TABLE;

|  |  |  |
| --- | --- | --- |
| REG\_NO | NAME | MARK |
|  |  | - |
| 10001 | ram | 85 |
| 10002 | sam | 75 |
| 10003 | samu | 95 |
| 10006 | RAJA | 80 |

SQL> select \* from STUDENT\_SYNONYM;

|  |  |  |
| --- | --- | --- |
| REG\_NO | NAME | MARK |
| ----------- | --------- | --------- |
| 10001 | ram | 85 |
| 10002 | sam | 75 |
| 10003 | samu | 95 |
| 10006 | RAJA | 80 |

#### YOU CAN UPDATE THE TABLE BY USING SYNONYM

SQL> UPDATE STUDENT\_SYNONYM SET MARK=100 WHERE REG\_NO=10006;

1. row updated.

SQL> select \* from STUDENT\_SYNONYM;

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| REG\_NO |  | NAME |  | | MARK | |
| 10001 |  | ram |  | | 85 | |
| 10002 |  | sam |  | | 75 | |
| 10003 |  | samu |  | | 95 | |
| 10006 |  | RAJA |  | | 100 | |
| SQL> select \* from STUDENT\_TABLE; | | | | | | |
| REG\_NO |  | NAME | |  | MARK |  |
| 10001 |  | ram | |  | 85 |  |
| 10002 |  | sam | |  | 75 |  |
| 10003 |  | samu | |  | 95 |  |
| 10006 |  | RAJA | |  | 100 |  |

#### TO DROP SYSNONYM

SQL> DROP SYNONYM STUDENT\_SYNONYM;

Synonym dropped.

#### BUT WE CAN USE THE TABLE

SQL> select \* from STUDENT\_TABLE;

|  |  |  |  |
| --- | --- | --- | --- |
| REG\_NO | NAME |  | MARK |
| 10001 | ram |  | 85 |
| 10002 | sam |  | 75 |
| 10003 | samu |  | 95 |
| 10006 | RAJA |  | 100 |
| **RESULT:** |  |  |  |

Thus the SQL commands for creation and various operation on Synonyms has been verified and

executed successfully.

#### Sequence

**Ex: No: 03 (3.3)**

**: :**

#### AIM:

To create the Sequence and verify the various operations on Sequence to get the incremented number.

**OBJECTIVE:**

The sequence generator provides a sequential series of numbers. The sequence generator is especially useful in multiuser environments for generating unique sequential numbers without the overhead of disk I/O or transaction locking

Sequence numbers are integers of up to 38 digits defined in the database. A sequence definition indicates general information, such as the following:

The name of the sequence

Whether the sequence ascends or descends The interval between numbers

Whether Oracle Database should cache sets of generated sequence numbers in memory

**ALGORITHM:**

Step 1: Start the DMBS.

Step 2: Connect to the existing database (DB)

Step 3: Create the sequence with its essential optional parameter.

Step 4: Display the data presented on the sequence by using pseudo column. Step 5: Alter the sequence with different optional parameter.

Step 6: Drop the sequence Step 7: Stop the DBMS.

**Creating a Sequence**

You create a sequence using the CREATE SEQUENCE statement, which has the following.

**SYNTAX:**

**SQL>CREATE SEQUENCE** sequence\_name

[**START WITH** start\_num] [**INCREMENT BY** increment\_num]

[ { **MAXVALUE** maximum\_num | **NOMAXVALUE** } ] [

{ **MINVALUE** minimum\_num | **NOMINVALUE** } ]

[ { **CYCLE** | **NOCYCLE** } ]

[ { **CACHE** cache\_num | **NOCACHE** } ] [

{ **ORDER** | **NOORDER** } ];

Where

***sequence\_name*** is the name of the sequence.

***start\_num*** is the integer to start the sequence. The default start number is 1.

***increment\_num*** is the integer to increment the sequence by. The default increment number is 1. The absolute value of *increment\_num* must be less than the difference between *maximum\_num* and *minimum\_num*.

***maximum\_num*** is the maximum integer of the sequence; *maximum\_num* must be greater than or equal to *start\_num*, and *maximum\_num* must be greater than *minimum\_num*.

**NOMAXVALUE** specifies the maximum is 1027 for an ascending sequence or –1 for a descending sequence. NOMAXVALUE is the default.

***minimum\_num*** is the minimum integer of the sequence; *minimum\_num* must be less than or equal to *start\_num*, and *minimum\_num* must be less than *maximum\_num*.

**NOMINVALUE** specifies the minimum is 1 for an ascending sequence or –1026 for a descending sequence. NOMINVALUE is the default.

**CYCLE** means the sequence generates integers even after reaching its maximum or minimum value. When an ascending sequence reaches its maximum value, the next value generated is the minimum. When a descending sequence reaches its minimum value, the next value generated is the maximum.

**NOCYCLE** means the sequence cannot generate any more integers after reaching its maximum or minimum value. NOCYCLE is the default.

***cache\_num*** is the number of integers to keep in memory. The default number of integers to cache is

20. The minimum number of integers that may be cached is 2. The maximum integers that may be cached is determined by the formula CEIL(*maximum\_num* - *minimum\_num*)/ABS(*increment\_num*).

**NOCACHE** means no caching. This stops the database from pre-allocating values for the sequence, which prevents numeric gaps in the sequence but reduces performance. Gaps occur because cached values are lost when the database is shut down. If you omit CACHE and NOCACHE, the database caches 20 sequence numbers by default.

**ORDER** guarantees the integers are generated in the order of the request. You typically use ORDER when using Real Application Clusters, which are set up and managed by database administrators.

**NOORDER** doesn’tguaranteethe integersare generatedin the order of the request. NOORDER is the default.

#### Example: 1 Command:

SQL> CREATE SEQUENCE seq1

INCREMENT BY 1

START with 1

MAXVALUE 5

MINVALUE 0;

#### Sequence created.

**TO DISPLAY THE VALUES OF SEQUENCES**

After creating sequence use nextval as nextval is used to generate sequence values SQL> select seq1.nextval from dual;

NEXTVAL

1

SQL> select seq1.nextval from dual;

NEXTVAL

2

SQL> select seq1.nextval from dual;

NEXTVAL

3

SQL> select seq1.currval from dual;

CURRVAL

3

The following is the list of available pseudo columns in Oracle.

**Pseudo Column Meaning**

CURRVAL - Returns the current value of a sequence. NEXTVAL - Returns the next value of a sequence.

NULL - Return a null value.

ROWID - Returns the ROWID of a row. See ROWID section below.

ROWNUM - Returns the number indicating in which order Oracle selects rows. First row selected will be ROWNUM of 1 and second row ROWNUM of 2 and so on.

SYSDATE - Returns current date and time.

USER - Returns the name of the current user.

UID - Returns the unique number assigned to the current user.

**TO ALTER THE SEQUENCES**

alter SEQUENCE seq1 maxvalue 25

INCREMENT BY

1. cycle

cache 2

drop SEQUENCE seq1;

#### EXAMPLE: 2

CREATE SEQUENCE seq2 INCREMENT BY

1 start with 1

maxvalue 5

minvalue 0

cycle CACHE 4;

#### EXAMPLE: 3

CREATE SEQUENCE seq3 INCREMENT BY -

1 start with 2

maxvalue 5

minvalue 0;

#### EXAMPLE: 4

CREATE SEQUENCE seq3 INCREMENT BY -

1 start with 2

maxvalue 5

minvalue 0

cycle cache 4;

#### EXAMPLE: 5

CREATE SEQUENCE seq1 INCREMENT BY

1 start with 1

maxvalue 10

minvalue 0;

#### EXAMPLE: 6

create table test1(a number primary key);

**TO INSERT THE VALUES FROM SEQUENCES TO TABLE:**

insert into test1 values(seq1.nextval)

**TO DROP SEQUENCES**

drop sequence sequenceNAme

**RESULT:**

Thus the SQL commands for creation and various operations on Sequence has been verified and executed successfully.

#### Indexes

**Ex: No: 03 (3.4)**

**: :**

**AIM:**

To create the Index for the table data and verify the various operations on Index.

**ALGORITHM:**

STEP 1: Start the DMBS.

STEP 2: Connect to the existing database (DB) STEP 3: Create the table with its essential attributes. STEP 4: Insert record values into the table.

STEP 5: Create the Index from the above created table or any data object. STEP 6: Display the data presented on the Index.

STEP 7: Stop the DBMS.

**Index**

The indexes are special objects which built on top of tables. The indexes can do operation like

SELECT , DELETE and UPDATE statement faster to manipulate a large amount of data. An INDEX can also be called a table and it has a data structure. An INDEX is created on columns of a table. One table may contain one or more INDEX tables

**The SQL INDEX does the following:**

INDEXES can locate information within a database very fast.

An INDEX makes a catalog of rows of a database table as row can be pointed within a fraction of time with a minimum effort.

A table INDEX is a database structure which arranges the values of one or more columns in a specific order.

The performance of an INDEX can not be recognized much when dealing with relatively small tables.

INDEX can work properly and quickly for the columns that have many different values.

It takes a long time to find an information for one or combination of columns from a table when there are thousands of records in the table. In that case if indexes are created on that columns, which are accessed frequently, the information can be retrieved quickly.

The INDEX first sorts the data and then it assigns an identification for each row.

The INDEX table having only two columns, one is a rowid and another is indexed-column (ordered).

When data is retrieved from a database table based on the indexed-column, the index pointer

searches the rowid and quickly locates that position.in the actual table and display the rows sought for.

**How it differ from view**

An INDEX is also a table. So it has a data structure.

INDEXES are pointers that represents the physical address of a data. An INDEX is created on columns of a table.

An INDEX makes a catalog based on one or more columns of a table. One table may contain one or more INDEX tables.

An INDEX can be created on a single column or combination of columns of a database table.

#### Types of Indexes:

**Command SAMPLE TABLE:**

1. Simple Index
2. Composite Index

SQL> SELECT \* FROM STUDENT\_TBL;

SL\_NO REG\_NONAME SEX DOB TOTAL\_PERCENTAGE MOBILE\_NO ADDRESS BLOOD

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 |  | 10001 |  | RAM | M | 11-DEC-85 75 |  | 9756435789 |  | PLOT.No:30/6 ABC | B+ |
| 2 |  | 10002 |  | RAJA | M | 16-JAN-7487.5 |  | 9456435458 |  | ABC Nager | O+ |
| 3 |  | 10003 |  | NIRMALA | F | 22-FEB-87 95.5 |  | 9461135411 |  | SAKTHI Nager | A+ |
| 4 |  | 10004 |  | Anitha | F | 05-MAR-8764.3 |  | 9461135555 |  | ANNA Nager | AB+ |

#### Simple Index:

When index is created on one column it is called as simple index.

#### Syntax:

CREATE INDEX <INDEX\_NAME> ON <TABLE\_NAME> (COL\_NAME);

#### Ex:

SQL> create index myIndex on student\_tbl(name); Index created.

#### \*notes

Index should be created on columns which we regularly use in the where clause.

When a index is created a separate structure is created with first column is ROWID and second column as indexed column.

The Rows in the index will be arranged in the ascending order of indexed column.

#### Composite Index:

when Index is created multiple columns it is called composite index.

#### Ex:

SQL> create index myCompIndex on student\_tbl(DOB,ADDRESS); Index created.

The above index **myCompIndex** is used only when both the columns are used in the where clause.

#### Disadvantages of index:

Index will consume memory.

The performance of DML command will be decreased.

Index can also be categorized two types:

* 1. Unique index
  2. Non-unique index

#### Unique Index:

If the indexed column contains unique value it is called unique index.

A unique index is automatically created. When we create a table with primary key constraint or unique constraint.

#### Cmd

SQL> create unique index myIndex on student\_tbl(name);

#### Non-unique index:

If an index column contains duplicated values they are called as non-unique index.

#### See to index tables:

SQL> Select index\_name from user\_indexes; INDEX\_NAME

MYCOMPINDEX

MYINDEX SYS\_C0011164

Query to see list of all the indexes.

SQL> Select INDEX\_NAME,TABLE\_NAME from user\_indexes; INDEX\_NAME TABLE\_NAME

SYS\_C0011164 TBL\_PKEY MYCOMPINDEX STUDENT\_TBL MYINDEX STUDENT\_TBL

Query to see list of all the indexes along with column name.

SQL> Select index\_name, table\_name, column\_name from user\_ind\_columns;

|  |  |  |
| --- | --- | --- |
| INDEX\_NAME | TABLE\_NAME | COLUMN\_NAME |
| -------------------- | ------------------- | ------------------------ |
| MYCOMPINDEX | STUDENT\_TBL | ADDRESS |
| MYCOMPINDEX | STUDENT\_TBL | DOB |
| MYINDEX  **.** | STUDENT\_TBL | NAME |

SQL> Desc user\_indexes; SQL> Desc user\_ind\_columns; **Function based index:**

When index is created by using functions it is called as function based index.

#### Ex:

SQL> CREATE INDEX myFuncIndex ON STUDENT\_TBL(lower(name)); Index created.

#### To drop on index:

**Ex:**

SQL> drop index index\_of\_student\_tbl;

**RESULT:**

Thus the SQL commands for creation and various operations on Indexes has been verified and executed successfully.

#### SAVE POINT

**Ex: No: 03 (3.5)**

**: :**

#### AIM:

To create the SAVE POINT for the transaction and verify the various operations of TCL commands.

**OBJECTIVE:**

The **SAVEPOINT** statement names and marks the current point in the processing of a transaction. With the **ROLLBACK TO** statement, savepoints undo parts of a transaction instead of the whole transaction.

An implicit savepoint is marked before executing an **INSERT, UPDATE, or DELETE** statement. If the statement fails, a rollback to the implicit savepoint is done. Normally, just the failed SQL statement is rolled back, not the whole transaction; if the statement raises an unhandled exception, the host environment

**ALGORITHM:**

STEP 1: Start the DMBS.

STEP 2: Connect to the existing database (DB) STEP 3: Create the table with its essential attributes.

STEP 4: Insert record values into the table or perform any kind of DML operation.

STEP 5: Create the **SAVE POINT**s for some set of statement on the transaction of database object. STEP 6: Use the **COMMIT** command to save the effect of the previous command operation except

DDL command

STEP 7: Use the **ROLLBACK TO SP\_LABLE** / **ROLLBACK** command for restore the database status up to the save point

STEP 8: Check the status of the database. STEP 9: Stop the DBMS.

**Syntax**: SAVEPOINT<SAVEPOINT\_NAME>;

**Ex**:

SQL> create table ORDER\_PROCESSING(

Order\_ID number(3), Product\_ID varchar2(10), Quantity number(3,2), Price number(4,2)

);

#### Table created.

SQL> insert into ORDER\_PROCESSING values(101,'RICE-22','6.5','30.50');

#### 1 row created.

SQL> insert into ORDER\_PROCESSING values(102,'OIL','2.0','90.50');

#### 1 row created.

SQL> SELECT \* FROM ORDER\_PROCESSING; ORDER\_IDPRODUCT\_ID QUANTITY PRICE

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 101 |  | RICE-22 |  | 6.5 |  | 30.5 |
| 102 |  | OIL |  | 2 |  | 90.5 |

SQL> **COMMIT;**

#### Commit complete.

SQL> insert into ORDER\_PROCESSING values(103,'BAGS','2','95');

#### 1 row created.

SQL> insert into ORDER\_PROCESSING values(104,'WATER BOTS','2','20');

#### 1 row created.

SQL> **SAVEPOINT A;**

#### Savepoint created.

SQL> insert into ORDER\_PROCESSING values(105,'EGG','8','40.50');

#### 1 row created.

SQL> insert into ORDER\_PROCESSING values(106,'SHAMPOO','1','75.50');

#### 1 row created.

SQL> **SAVEPOINT B;**

#### Savepoint created.

SQL> insert into ORDER\_PROCESSING values(107,'BAR SOAP','1','45.50');

#### 1 row created.

SQL> insert into ORDER\_PROCESSING values(108,'TONER','1','75.50');

#### 1 row created.

SQL> **SAVEPOINT C;**

#### Savepoint created.

SQL> insert into ORDER\_PROCESSING values(109,'SUGAR','2.0','60.50');

#### 1 row created.

SQL> SELECT \* FROM ORDER\_PROCESSING;

ORDER\_ID PRODUCT\_ID QUANTITY PRICE

|  |  |  |
| --- | --- | --- |
| 101 | RICE-22 6.5 | 30.5 |
| 102 | OIL 2 | 90.5 |
| 103 | BAGS 2 | 95 |
| 104 | WATER BOTS2 | 20 |
| 105 | EGG 8 | 40.5 |
| 106 | SHAMPOO 1 | 75.5 |
| 107 | BAR SOAP 1 | 45.5 |
| 108 | TONER 1 | 75.5 |
| 109  9 rows selected. | SUGAR 2 | 60.5 |

SQL> ROLLBACK TO B;

#### Rollback complete.

SQL> SELECT \* FROM ORDER\_PROCESSING;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ORDER\_ID | PRODUCT\_ID |  | QUANTITY | PRICE |
| 101 | RICE-22 |  | 6.5 | 30.5 |
| 102 | OIL |  | 2 | 90.5 |
| 103 | BAGS |  | 2 | 95 |
| 104 | WATER BOTS |  | 2 | 20 |
| 105 | EGG |  | 8 | 40.5 |
| 106  6 rows selected. | SHAMPOO |  | 1 | 75.5 |

SQL> ROLLBACK TO A;

#### Rollback complete.

SQL> SELECT \* FROM ORDER\_PROCESSING;

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ORDER\_ID |  | PRODUCT\_ID |  | QUANTITY |  | PRICE |
| 101 |  | RICE-22 |  | 6.5 |  | 30.5 |
| 102 |  | OIL |  | 2 |  | 90.5 |
| 103 |  | BAGS |  | 2 |  | 95 |
| 104 |  | WATER BOTS |  | 2 |  | 20 |

SQL> ROLLBACK;

#### Rollback complete.

SQL> SELECT \* FROM ORDER\_PROCESSING;

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ORDER\_ID |  | PRODUCT\_ID |  | QUANTITY |  |  | PRICE |
| 101 |  | RICE-22 |  | 6.5 |  | 30.5 |  |
| 102 | OIL 2 90.5 | | | | | | |

SQL> ROLLBACK;

#### Rollback complete.

SQL> SELECT \* FROM ORDER\_PROCESSING; ORDER\_IDPRODUCT\_ID QUANTITY PRICE

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 101 | RICE-22 |  | 6.5 |  | 30.5 |
| 102 | OIL | 2 |  | 90.5 |  |

**RESULT:**

Thus the SQL commands for creation and various operations on transaction (TCL COMMAND) save point has been verified and executed successfully

#### Creating an Employee database to set various constraints in RDBMS

**Ex: No: 04**

**: :**

**AIM:**

At the end of this exercise students are able

To differentiate between self referential constraints and foreign key constraint. To refer a field of a given table or another table by using foreign key.

To apply check constraint & default constraint in an effective manner.

**ALGORITHM:**

STEP 1: Start the DMBS.

STEP 2: Connect to the existing database (DB) STEP 3: Create the table with its essential constraint.

STEP 4: Insert record values into the table and then check the constraint. STEP 5: disable the constraints and insert the values into the table.

STEP 6: if you want to re-enable the constraint then enable you can do. STEP 7: Stop the DBMS.

#### CONSTRAINTS

Constraints are part of the table definition that limits and restriction on the value entered into its columns.

#### INTEGRITY CONSTRAINT

An integrity constraint is a mechanism used by oracle to prevent invalid data entry into the table. It has enforcing the rules for the columns in a table.

The types of the integrity constraints are:

1. Domain Integrity
2. Entity Integrity
3. Referential Integrity

#### TYPES OF CONSTRAINTS:

1. Primary key
2. Foreign key/references
3. Check
4. Unique
5. Not null
6. Null
7. Default

#### CONSTRAINTS CAN BE CREATED IN THREE WAYS:

1. Column level constraints
2. Table level constraints
3. Using DDL statements-alter table command

#### OPERATION ON CONSTRAINT:

* 1. ENABLE
  2. DISABLE
  3. DROP

**PRIMARY KEY CONSTRAINTS**

A primary key avoids duplication of rows and does not allow null values. It can be defined on one or more columns in a table and is used to uniquely identify each row in a table. These values should never be changed and should never be null. A table should have only one primary key. If a primary key constraint is assigned to more than one column or combination of column is said to be composite primary key, which can contain 16 columns.

**Column level constraints using primary key:**

**QUERY: 13**

Q13. Write a query to create primary constraints with column level **Syntax:** Column level constraints using primary key.

SQL> CREATE<OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1 <DATATYPE> (SIZE)<TYPE OF

CONSTRAINTS>, COLUMN NAME.1 <DATATYPE> (SIZE) );

**Command:**

SQL> CREATE TABLE TBL\_PKEY(

**Table created.**

RegNo NUMBER(5) PRIMARY KEY, Name VARCHAR2(20), ANY\_SUB\_MARK NUMBER(3)

);

SQL> insert into result values(10001,'raju',75);

**1 row created.**

SQL> insert into result values(10002,'KAMAL;',100);

**1 row created.**

SQL> insert into result values(0,'RAVI;',75);

**1 row created.**

SQL> insert into result values(NULL,'KAVI',65);

**1 row created.**

SQL> insert into TBL\_PKEY values(10001,'raju',75);

**1 row created.**

SQL> insert into TBL\_PKEY values(10002,'raj',85);

**1 row created.**

SQL> insert into TBL\_PKEY values(0,'Kaj',22);

**1 row created.**

SQL> insert into TBL\_PKEY values(NULL,'Kaj',22); insert into TBL\_PKEY values(NULL,'Kaj',22)

\* ERROR at line 1:

**ORA-01400: cannot insert NULL into ("SENTHIL"."TBL\_PKEY"."REGNO")**

SQL> insert into TBL\_PKEY values(10002,'RAJAN',95); insert into TBL\_PKEY values(10002,'RAJAN',95)

\*

ERROR at line 1:

**ORA-00001: unique constraint (SENTHIL.SYS\_C0011650) violated**

SQL> insert into TBL\_PKEY values(10003,'RAJA',85);

**1 row created.**

SQL> select \* FROM TBL\_PKEY;

REGNO NAME ANY\_SUB\_MARK

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 10001 |  | raju |  | 75 |
| 10002 |  | raj |  | 85 |
| 0 |  | Kaj |  | 22 |
| 10003 |  | RAJA |  | 85 |

**Column level constraints using primary key with naming convention:**

**QUERY: 14**

Q14. Write a query to create primary constraints with column level with naming convention

**Syntax:** syntax for column level constraints using primary key.

SQL >CREATE <OBJ.TYPE><OBJ.NAME> (

COL NAME.1 <DATATYPE> (SIZE)CONSTRAINTS <NAME OF CONSTRAINTS><TYPE OF CONSTRAINTS>, COL NAME.2 <DATATYPE> (SIZE)… );

**Command:**

SQL>CREATE TABLE EMPLOYEE (

EMPNO NUMBER (4) **CONSTRAINT EMP\_EMPNO\_PK PRIMARY KEY,**

ENAMEVARCHAR2 (10),JOB VARCHAR2 (6),SAL NUMBER (5),DEPTNO NUMBER (7));

**Table level primary key constraints:**

**QUERY: 15**

Q15. Write a query to create primary constraints with table level with naming convention

**Syntax:** The syntax for table level constraints using primary key

SQL: >CREATE <OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1 <DATATYPE> (SIZE), COLUMN NAME.1 <DATATYPE> (SIZE),

CONSTRAINTS <NAME OF THE CONSTRAINTS><TYPE OF THE CONSTRAINTS>);

**Command:**

SQL>CREATE TABLE EMPLOYEE (EMPNO NUMBER(6),ENAME VARCHAR2(20),JOB VARCHAR2(6), SAL NUMBER(7), DEPTNO NUMBER(5),

**CONSTRAINT EMP\_EMPNO\_PK PRIMARY KEY(EMPNO));**

**Table level constraint with alter command (primary key):**

**QUERY: 16**

Q16. Write a query to create primary constraints with alter command

**Syntax:** The syntax for column level constraints using primary key.

SQL:>CREATE <OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1 <DATATYPE>(SIZE),

COLUMN NAME.1 <DATATYPE> (SIZE));

**[OR]**

SQL> ALTER TABLE <TABLE NAME> ADD CONSTRAINTS <NAME OF THECONSTRAINTS> <TYPE OF THE CONSTRAINTS><COLUMN NAME>);

**Command:**

SQL>CREATE TABLE EMPLOYEE(EMPNO NUMBER(5),ENAME VARCHAR2(6),JOB VARCHAR2(6), SAL NUMBER(6),DEPTNO NUMBER(6));

SQL>ALTER TABLE EMP3 ADD CONSTRAINT **EMP3\_EMPNO\_PK PRIMARYKEY (EMPNO);**

**REFERENCE /FOREIGN KEY CONSTRAINT**

It enforces relationship between tables. To establish parent-child relationship between 2 tables having a common column definition, we make use of this constraint. To implement this, we should define the column in the parent table as primary key and same column in the child table as foreign key referring to the corresponding parent entry.

**Foreign key**

A column or combination of column included in the definition of referential integrity, which would refer to a referenced key.

**Referenced key**

It is a unique or primary key upon which is defined on a column belonging to the parent table.

**Column level foreign key constraint**

**QUERY: 17**

* 1. Write a query to create foreign key constraints with column level

**Parent Table:**

**Syntax:** Syntax for Column level constraints Using Primary key SQL:>CREATE <OBJ.TYPE><OBJ.NAME> (

COLUMN NAME.1 <DATATYPE>(SIZE)<TYPE OF CONSTRAINTS> ,

COLUMN NAME.1 <DATATYPE> (SIZE) );

**Command:**

SQL> CREATE TABLE **DEPT**( **DEPTNO** NUMBER(3) **PRIMARY KEY**,

**DNAME** VARCHAR2(20),**LOCATION** VARCHAR2(15));

Table created.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| SQL> desc DEPT; | | | | | |
| Name |  | Null? |  | Type | |
| DEPTNO |  | NOT NULL |  | NUMBER(3) | |
| DNAME |  |  |  | VARCHAR2(20) | |
| LOCATION |  |  |  | VARCHAR2(15) | |
| SQL> select \* from DEPT; | | | | | |
| DEPTNO | | DNAME |  | LOCATION |  |
| 101 | | kamal |  | chennai |  |
| 102 | | rajini |  | madurai |  |
| 103 | | Ajith |  | kovai |  |
| **Child Table:** | |  |  |  |  |

**Syntax:** The syntax for column level constraints using foreign key.

SQL:>CREATE <OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1 <DATATYPE>(SIZE),

COLUMN NAME2 <DATATYPE> (SIZE) REFERENCES <TABLE NAME>(COLUMN NAME> ….);

**Command:**

SQL> CREATE TABLE **EMPL**(**EMPNO** NUMBER(4), **DEPTNO** NUMBER(3) **REFERENCES** DEPT(**DEPTNO**), **DESIGN** VARCHAR2(10));

Table created.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SQL> desc EMPL; Name |  | Null? |  | Type |
| EMPNO  DEPTNO DESIGN |  |  |  | NUMBER(4)  NUMBER(3) VARCHAR2(10) |

SQL> insert into EMPL values(5001,101,'RAJA'); 1 row created.

SQL> insert into EMPL values(5003,103,'KAJA'); 1 row created.

SQL> insert into EMPL values(5006,104,'RAMYA'); insert into EMPL values(5006,104,'RAMYA')

\*

ERROR at line 1:

ORA-02291: integrity constraint (SYSTEM.SYS\_C0011294) violated - parent key not found

SQL> select \* from EMPL;

|  |  |  |
| --- | --- | --- |
| EMPNO | DEPTNO | DESIGN |
| ---------- | ------------ | ------------- |
| 5001 | 101 | RAJA |
| 5003 | 103 | KAJA |

**Column level foreign key constraint with naming conversions**

**QUERY: 18**

* 1. Write a query to create foreign key constraints with column level

**Parent Table:**

**Syntax:** The syntax for column level constraints using primary key.

SQL :> CREATE<OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1 <DATATYPE>(SIZE)<TYPE OF CONSTRAINTS>,COLUMN NAME.1 <DATATYPE> (SIZE)…);

**Child Table:**

**Syntax: s**yntax for column level constraints using foreign key.

SQL :> CREATE<OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1 <DATATYPE>(SIZE) ,

COLUMN NAME2 <DATATYPE> (SIZE) **CONSTRAINT <CONST.NAME>**REFERENCES <TABLE NAME>

(COLUMN NAME>…);

**Command:**

SQL>CREATE TABLE DEPT (DEPTNO NUMBER (2) **PRIMARYKEY**, DNAME VARCHAR2 (20), LOCATION VARCHAR2 (15)); SQL>CREATE TABLE EMP4A (EMPNO NUMBER (3),

DEPTNO NUMBER (2) **CONSTRAINT EMP4A\_DEPTNO\_FK REFERENCES DEPT (DEPTNO),**

DESIGN VARCHAR2 (10));

**Table level foreign key constraints:**

**QUERY: 19**

* 1. Write a query to create foreign key constraints with Table level.

**Parent Table:**

**Syntax:**

SQL :> CREATE<OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1 <DATATYPE>(SIZE) <TYPE OF CONSTRAINTS>, COLUMN NAME.1 <DATATYPE> (SIZE)…);

**Child Table:**

**Syntax:** The syntax for table level constraints using foreign key.

SQL :> CREATE<OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1

<DATATYPE>(SIZE), COLUMN NAME2 <DATATYPE> (SIZE),

**CONSTRAINT <CONST.NAME>**REFERENCES <TABLE NAME> (COLUMN NAME>);

**Command:**

SQL>CREATE TABLE DEPT(DEPTNO NUMBER(2) **PRIMARY KEY**, DNAME VARCHAR2(20),LOCATION VARCHAR2(15));

SQL>CREATE TABLE EMP5(EMPNO NUMBER(3),DEPTNO NUMBER(2),

DESIGN VARCHAR2(10) **CONSTRAINT ENP2\_DEPTNO\_FK FOREIGNKEY(DEPT NO) REFERENCES DEPT(DEPTNO));**

**Table level foreign key constraints with alter command:**

**QUERY:20**

* 1. Write a query to create foreign key constraints with Table level with altercommand.

**Parent Table:**

**Syntax:**

SQL :>CREATE<OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1 <DATATYPE>(SIZE)<TYPE OF

CONSTRAINTS> , COLUMN NAME.1 <DATATYPE> (SIZE) );

**Child Table:**

**Syntax:** The syntax for table level constraints using foreign key.

SQL:>CREATE <OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1 <DATATYPE>(SIZE),

COLUMN NAME2 <DATATYPE> (SIZE));

**Syntax:**

SQL> ALTER TABLE <TABLE NAME> ADD CONSTRAINT <CONST. NAME>REFERENCES <TABLE NAME> (COLUMN NAME>);

**Command:**

SQL>CREATE TABLE DEPT(DEPTNO NUMBER(2) PRIMARY KEY, DNAME VARCHAR2(20), LOCATION VARCHAR2 (15));

SQL>CREATE TABLE EMP5 (EMPNO NUMBER(3), DEPTNO NUMBER (2), DESIGN VARCHAR2 (10));

SQL>ALTER TABLE EMP6 ADD CONSTRAINT EMP6\_DEPTNO\_FK FOREIGNKEY(DEPTNO)REFERENCES DEPT(DEPTNO);

**CHECK CONSTRAINT**

Check constraint can be defined to allow only a particular range of values .when the manipulation violates this constraint, the record will be rejected. Check condition cannot contain sub queries.

**Column level checks constraint:**

**QUERY: 21**

* 1. Write a query to create Check constraints with column level

**Syntax:** syntax for column level constraints using check.

SQL:>CREATE <OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1 <DATATYPE>(SIZE) CONSTRAINT <CONSTRAINTS NAME><TYPE OF CONSTRAINTS>(CONSTRAITNS CRITERIA)

, COLUMN NAME2 <DATATYPE> (SIZE));

**Command:**

SQL>CREATE TABLE EMP7(EMPNO NUMBER(3),ENAME VARCHAR2(20),DESIGN VARCHAR2(15), SAL NUMBER(5)**CONSTRAINT** EMP7\_SAL\_CK CHECK(SAL>500 ANDSAL<10001),

DEPTNO NUMBER(2));

**Table Level Check Constraint:**

**QUERY: 22**

* 1. Write a query to create Check constraints with table level

**Syntax:** Syntax for Table level constraints using Check.

SQL:>CREATE <OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1

<DATATYPE>(SIZE), (COLUMN NAME2 <DATATYPE> (SIZE),

CONSTRAINT<CONSTRAINTS NAME><TYPE OF CONSTRAINTS> (CONSTRAITNSCRITERIA));

**Command:**

SQL>CREATE TABLE EMP8(EMPNO NUMBER(3),ENAME VARCHAR2(20),DESIGN VARCHAR2(15), SAL NUMBER(5),DEPTNO NUMBER(2),

**CONSTRAINTS** EMP8\_SAL\_CK CHECK(SAL>500 **AND**SAL<10001));

**Check Constraint with Alter Command:**

**QUERY:23**

* 1. Write a query to create Check constraints with table level using alter command.

**Syntax:** Syntax for Table level constraints using Check.

SQL:>CREATE <OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1

<DATATYPE>(SIZE), (COLUMN NAME2 <DATATYPE> (SIZE),

CONSTRAINT<CONSTRAINTS NAME><TYPE OF CONSTRAINTS> (CONSTRAITNSCRITERIA)) ;

**Command:**

SQL>CREATE TABLE EMP9(EMPNO NUMBER,ENAME VARCHAR2(20),DESIGN VARCHAR2(15), SAL NUMBER(5));

SQL>ALTER TABLE EMP9 ADD **CONSTRAINTS** EMP9\_SAL\_CK**CHECK**(SAL>500 AND SAL<10001);

**UNIQUE CONSTRAINT**

It is used to ensure that information in the column for each record is unique, as with telephone or drivers license numbers. It prevents the duplication of value with rows of a specified column in a set of column. A column defined with the constraint can allow null value.

If unique key constraint is defined in more than one column i.e., combination of column cannot be specified.

Maximum combination of columns that a composite unique key can contain is 16.

**Column Level Constraint:**

**QUERY:24**

* 1. Write a query to create unique constraints with column level

**Syntax:** syntax for column level constraints with unique.

SQL :> CREATE <OBJ.TYPE><OBJ.NAME> (<COLUMN NAME.1> <DATATYPE> (SIZE) CONSTRAINT

<NAME OF CONSTRAINTS><CONSTRAINT TYPE>, (COLUMN NAME2 <DATATYPE> (SIZE));

**Command:**

SQL>CREATE TABLE EMP10(EMPNO NUMBER(3),ENAME VARCHAR2(20), DESGIN VARCHAR2 (15)**CONSTRAINT** EMP10\_DESIGN\_UK UNIQUE,

SAL NUMBER (5));

**Table Level Constraint:**

**QUERY: 25**

* 1. Write a query to create unique constraints with table level **Syntax:** syntax for table level constraints with unique.

SQL :> CREATE <OBJ.TYPE><OBJ.NAME> (<COLUMN NAME.1><DATATYPE> (SIZE), (COLUMN NAME2 <DATATYPE> (SIZE),

CONSTRAINT<NAME OF CONSTRAINTS><CONSTRAINT TYPE>(COLUMN NAME);) ;

**Command:**

SQL>CREATE TABLE EMP11(EMPNO NUMBER(3),ENAME VARCHAR2(20),DESIGN VARCHAR2(15), SAL NUMBER(5),

CONSTRAINT EMP11\_DESIGN\_UK UNIGUE(DESIGN));

**Table Level Constraint Alter Command:**

**QUERY:26**

* 1. Write a query to create unique constraints with table level

**Syntax:** syntax for table level constraints with check using alter.

SQL :> CREATE <OBJ.TYPE><OBJ.NAME> (<COLUMN NAME.1><DATATYPE> (SIZE),

(<COLUMN NAME.2><DATATYPE> (SIZE)) ;

SQL> ALTER TABLE ADD <CONSTRAINTS><CONSTRAINTS NAME><CONSTRAINTS TYPE> (COLUMN NAME);

**Command:**

SQL>CREATE TABLE EMP12(EMPNO NUMBER(3),ENAME VARCHAR2(20),DESIGN VARCHAR2(15), SAL NUMBER(5));

SQL>ALTER TABLE EMP12 ADD CONSTRAINT EMP12\_DESIGN\_UKUNIQUE(DESING);

**NOT NULL CONSTRAINTS**

While creating tables, by default the rows can have null value .the enforcement of not null constraint in a table ensure that the table contains values.

**Column Level Constraint:**

**QUERY: 27**

* 1. Write a query to create Not Null constraints with column level

**Syntax:** syntax for column level constraints with not null

SQL :> CREATE <OBJ.TYPE><OBJ.NAME>(<COLUMN NAME.1><DATATYPE> (SIZE) CONSTRAINT

<NAME OF CONSTRAINTS> <CONSTRAINT TYPE>, (COLUMN NAME2 <DATATYPE> (SIZE)) ;

**Command:**

SQL>CREATE TABLE EMP13(EMPNO NUMBER(4),

ENAME VARCHAR2(20) CONSTRAINT EMP13\_ENAME\_NN NOT NULL, DESIGN VARCHAR2(20),SAL NUMBER(3));

**NULL CONSTRAINTS**

Setting null value is appropriate when the actual value is unknown, or when a value would not be meaningful. A null value is not equivalent to a value of zero.

A null value will always evaluate to null in any expression.

When a column name is defined as not null, that column becomes a mandatory i.e., the user has to enter data into it.

Not null Integrity constraint cannot be defined using the alter table command when the table contain rows.

**Column Level Constraint:**

**QUERY:28**

* 1. Write a query to create Null constraints with column level

**Syntax:** syntax for column level constraints with null

SQL :> CREATE <OBJ.TYPE><OBJ.NAME> (

<COLUMN NAME.1><DATATYPE> (SIZE) **CONSTRAINT** <NAME OF CONSTRAINTS>

<CONSTRAINT TYPE>,(COLUMN NAME2 <DATATYPE> (SIZE)) ;

**Command:**

SQL>CREATE TABLE EMP13(EMPNO NUMBER(4),

ENAME VARCHAR2(20) CONSTRAINT EMP13\_ENAME\_NN NULL, DESIGN VARCHAR2(20),SAL NUMBER(3));

**DEFAULT CONSTRAINTS**

Default constraints assign the default values if the values is not passed at the time of inserting the values to the table

**QUERY:28**

* 1. Write a query to create default constraints with column level

**Syntax:** syntax for column level constraints with default

SQL :> CREATE <OBJ.TYPE><OBJ.NAME> (

<COLUMN NAME.1><DATATYPE> (SIZE) ,

<COLUMN NAME.2 <DATATYPE> (SIZE) **Default** <**default value**>) ;

**Command:**

SQL> CREATE TABLE DF(

);

Table created.

REGNO NUMBER(5), NAME VARCHAR2(20),

MARKS NUMBER(3) **DEFAULT 55**

SQL> INSERT INTO DF VALUES(1001,'ARJUN',NULL);

1 row created.

SQL> INSERT INTO DF(REGNO) VALUES(**1005**);

1 row created.

SQL> INSERT INTO DF VALUES(1001,'RAJ',78);

1 row created.

SQL> SELECT \* FROM DF;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| REGNO |  | NAME |  | MARKS |
| 1001 |  | ARJUN |  |  |
| 1005 |  |  |  | 55 |
| 1001 |  | RAJ |  | 78 |

#### CREATING RELATIONSHIP BETWEEN THE DATABASES IN RDBMS

**Ex: No: 05 (5.3) To implementation the Join Operations**

**: :**

**AIM:**

To execute and verify the SQL commands for various join operation.

**ALGORITHM:**

**STEP 1:** Start the program.

**STEP 2:** Create two different tables with its essential attributes.

**STEP 3:** Insert attribute values into the table. **STEP 4:** Create the table object for easy reference. **STEP 5:** Join two tables by using JOIN operator. **STEP 6:** Display the result of the result table.

**STEP 7:** Stop the program.

#### JOINS:

Joins are used to retrieve the data from multiple tables.

#### Types of Joins:

1. EQUI\_JOIN
2. NON EQUI\_JOIN
3. SELF JOIN
4. OUTER JOIN
   1. Right outer join
   2. Left outer join
   3. Full outer join

1. **EQUI\_JOIN**:

When tables are joined basing on a common column it is called EQUI\_JOIN.

**Ex**:

select empno, ename, dname from emp, dept where emp.deptno = dept.deptno;

|  |  |  |
| --- | --- | --- |
| **EMPNO** | **ENAME** | **DNAME** |
| 7369 | SMITH | RESEARCH |
| 7499 | ALLEN | SALES |
| 7521 | WARD | SALES |
| **Note**: |  |  |

We need to mention join conditions in the where clause.

In EQUI\_JOINS we along use to equal to operator in join condition.

**Ex**:

**SQL>Selete empno, ename, sal, job, dname, loc from emp, dept**

**where emp.deptno = dept.deptno;**

**SQL>**Selete empno, ename, sal, deptno, dname, loc from emp, dept

where emp.deptno = dept.deptno;// error

**SQL>**Selete empno, ename, sal, emp.deptno, dname, loc from emp, dept

where emp.deptno = dept.deptno; //valid

#### Note:

we need to mention table name dot column(emp.deptno) name for the common column to resolve the any table.

The common column can be retrieved from any of the table. We can filter the data from the result of join.

**Ex**:

**SQL>**Select empno, ename, sal, emp.deptno, dname, loc from emp, dept

where emp.deptno = dept.deptno AND sal > 2000;

To improve the performance of the join we need mention table name dot column name for all the columns.

**Ex**:

**SQL>**Select emp.empno, emp.ename, emp.sal,emp.deptno, dept.dname, dept.loc from emp,dept

where emp.deptno = dept.deptno AND sal > 2000;

#### Table alias:

**Ex**:

Table alias is an alternate name given to a table.

By using a table alias length of the table reduces and at the same time performance is maintains. Table alias are create in same clause can be used in select clause as well as where clause.

Table alias is temporary once the query is executed the table alias are losed.

**SQL>**Select E.Empno, E.Ename, E.sal, E.deptno, D.Dname,

D.loc from emp E, Dept D where E.deptno = D.deptno;

#### Join the multiple tables(3 tables):

Select \* from Areas;

#### City State

Newyork AP

Dallas Mh

**Ex**:

**SQL>**Select E.empno, E.ename, E.sal,D.dname,A.state from emp E, dept D, Areas A

where E.deptno = D.deptno AND D.loc = A.city;

**Note**: To join ‘n’ tables we ne-e1dconnditions.

#### NON EQUI JOIN:

When we do not use NON EQUI JOIN to operator in the join condition is NON EQUI JOIN.

**Ex**:

**SQL>**Select \* from SALGRADE;

|  |  |  |
| --- | --- | --- |
| GRADE | LOSAL | HISAL |
| 1 | 700 | 1200 |
| 2 | 1201 | 1400 |
| 3 | 1401 | 2000 |
| 4 | 2001 | 3000 |
| 5 | 3001 | 9999 |

**SQL>**Select e.empno, e.ename, e.sal, s.grade from emp e, salgrade s

where e.sal BETWEEN s.losal AND hisal;

|  |  |  |
| --- | --- | --- |
| EMPNO | ENAME | GRADE |
| 7369 | SMITH | 1 |
| 7876 | ADAMS | 1 |
| 7900 | JAMES | 2 |

**SQL>**Select e.empno, e.ename, s.grade from emp e, salgrade s

where e.sal BETWEEN s.losal AND s.hisal AND s.grade = 4;

#### SELF JOIN:

When a table is joining to it self it is called self join. In self joins we need to create two table aliases for the same table.

**SQL>**Select empno, ename, job, mgr, from emp;

**SQL>**Select e.empno, e.ename, e.job, m.ename from emp e, emp m

where e.mgr = m.empno;

|  |  |  |  |
| --- | --- | --- | --- |
| **Empno** | **Ename** | **Job** | **Ename** |
| 7902 | FORD | ANALYST | JONES |
| 7869 | SCOTT | CLERK | JONES |
| 7900 | JAMES | SALESMAN | BLAKE |

#### CARTESIAN PRODUCT:

When tables are joined without any join condition it is called Cartesian product. In the result we get all possible combination.

**SQL>**Select e.empno, e.ename, e.sal, e.deptno, d.dname, d.loc

from emp e, dept d; //14\*4=56 rows are selected

#### ANSI JOINS:

They are the three types.

#### INNER JOINS:

It is same as Equi join.

**Ex**:

**SQL>**Select e.empno, e.ename, e.sal, e.deptno, d.dname, d.loc from emp e INNER JOIN dept d ON(e.deptno = d.deptno); 2.**NATURAL JOIN**:

It is same as Equi join.

**Ex**:

**SQL>**Select empno, ename, sal, deptno, dname,loc from NATURAL JOIN dept;

#### CROSS PRODUCT/CROSS JOIN:

It is same as Cartesian product.

**Ex**:

**SQL>**Select e.empno, e.ename, e.sal, e.deptno, d.dname, d.loc

from emp e CROSS JOIN dept d; //14\*4 = 56 rows are displayed.

#### DEFAULT:

**Ex**:

**SQL>**Create table stu1(sno number(3), Sname varchar2(10),

Marks number(3) default 100, Doj Date DEFAULT sysdate);

**SQL>**Insert into stu1(sno, sname) values(101,’malli’); **SQL>**Insert into stu1 values(102,’ARUN’-,J4A0N,’-1019’); **SQL>**Insert into stu1 values (103,’KIRAN’,NU-LFLE,B’1-120’);

|  |  |  |  |
| --- | --- | --- | --- |
| SNO | SNAME | MARKS | DOJ |
| 101 | malli | 100 | 26-JUN-12 |
| 102 | ARUN | 40 | 11-JAN-09 |
| 103 | KIRAN |  | 12-FEB-10 |

#### SUPER KEY:

Combination of columns which can be used unique key identify every row is called as super key. Table object

Column Attributes Row Tuple/Record **OUTER JOINS**:

It is extension of EQUI JOINS.

In outer joins we get match as well as non matching rows. (+) This called as outer join operator.

#### RIGHT OUTER JOIN:

**SQL Syntax**:

**SQL>**Select e.empno, e.ename, e.sal, e.deptno, d.dname, d.loc from emp e, dept d

where e.deptno(+) = d.deptno; //14 + 1 = 15 rows

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **empno** | **ename** | **sal** | **deptno** | **dname** | **loc** |
| 7900 | james | 950 | 30 | sales | chicago |
| 8963 | adams | 1400 | 20 | clerk | newyork |
| 6798 | adams | 2000 | 10 | sales | india |

#### ANSI SYNTAX OF RIGHT OUTER JOIN:

ANSI SYSTAX:

**SQL>**Select e.empno, e.ename, e.sal, e.deptno, d.dname, d.loc

from emp e RIGHT OUTER JOIN dept d ON(e.deptno = d.deptno);

#### LEFT OUTER JOIN:

**SQL Syntax**:

**SQL>**Select e.empno, e.ename, e.sal, e.deptno, d.dname, d.loc from emp e, dept d

where e.deptno = d.deptno(+); //14+3 = 17 row displayed

#### ANSI SYNTAX OF LEFT OUTER JOIN: ANSI SYNTAX:

**SQL>**Select e.empno, e.ename, e.sal, e.deptno, d.dname, d.loc from emp e LEFT OUTER JOIN dept d ON(e.deptno = d.deptno);

#### FULL OUTER JOIN: ANSI SYNTAX:

**SQL>**Select e.empno, e.ename, e.sal, e.deptno, d.dname, d.loc from emp e FULL OUTER JOIN dept d ON(e.deptno = d.deptno);

//14 + 2 + 3 = 19 rows are displayed.

**RESULT:**

Thus the SQL commands **to** implementation the join operations has been verified and executed successfully.