

# LIST OF EXPERIMENTS

## **1. To implement Data Definition language**

1.1. Create, alter, drop, truncate

### **1.2. To implement Constraints.**

1.2.1. (a). Primary key, (b).Foreign Key, (c). Check, (d). Unique, (e). Null, (f). Not null , (g) . Default, (h). Enable Constraints, (i). Disable Constraints (j). Drop Constraints

## **2. To implementation on DML, TCL and DRL**

2.1. (a).Insert, (b).Select, (c).Update, (d).Delete, (e).commit, (f).rollback, (g).save point, (i). Like'%', (j).Relational Operator.

## **3. To implement Nested Queries & Join Queries**

3.1.(a). To implementation of Nested Queries

.2.(b). (a) Inner join, (b).Left join, (c).Right join (d).Full join

## **4. To implement Views**

4.1. (a). View, (b).joint view, (c).force view, (d). View with check option

## **5(a). Control Structure**

5.1. To write a PL/SQL block for Addition of Two Numbers

5.2. To write a PL/SQL block for IF Condition

5.3. To write a PL/SQL block for IF and else condition

5.4. To write a PL/SQL block for greatest of three numbers using IF AND ELSEIF

5.5. To write a PL/SQL block for summation of odd numbers using for LOOP

## **5. (b).Procedures**

5.6. To write a PL/SQL Procedure using Positional Parameters

5.7. To write a PL/SQL Procedure using notational parameters

5.8. To write a PL/SQL Procedure for GCD Numbers

5.9. To write a PL/SQL Procedure for cursor implementation

5.10. To write a PL/SQL Procedure for explicit cursors implementation

5.11. To write a PL/SQL Procedure for implicit cursors implementation

## **5. (c). Functions:**

5.13. To write a PL/SQL block to implementation of factorial using function

5.12. To write a PL/SQL function to search an address from the given database

## **DATA DEFINITION LANGUAGE (DDL) COMMANDS IN RDBMS**

To execute and verify the Data Definition Language commands and constraints

### **DDL (DATA DEFINITION LANGUAGE)**

**CREATE**

**ALTER**

**DROP**

**TRUNCATE**

**COMMENT**

**RENAME**

### **PROCEDURE**

STEP 1: Start

STEP 2: Create the table with its essential attributes.

STEP 3: Execute different Commands and extract information from the table.

STEP 4: Stop

### **SQL COMMANDS**

#### **1. COMMAND NAME: CREATE**

COMMAND DESCRIPTION: **CREATE** command is used to create objects in the database.

#### **2. COMMAND NAME: DROP**

COMMAND DESCRIPTION: **DROP** command is used to delete the object from the database.

#### **3. COMMAND NAME: TRUNCATE**

COMMAND DESCRIPTION: **TRUNCATE** command is used to remove all the records from the table

#### **4. COMMAND NAME: ALTER**

COMMAND DESCRIPTION: **ALTER** command is used to alter the structure of database

#### **5. COMMAND NAME: RENAME**

COMMAND DESCRIPTION: **RENAME** command is used to rename the objects.

### **QUERY: 01**

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

**Syntax for creating a table:**

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

**QUERY: 01**

**SQL>CREATE TABLE EMP (EMPNO NUMBER (4),  
ENAME VARCHAR2 (10),  
DESIGNATION VARCHAR2 (10),  
SALARY NUMBER (8,2));**

**Table created.**

**QUERY: 02**

Q2. Write a query to display the column name and datatype of the table employee.

**Syntax for describe the table:**

**SQL: DESC <TABLE NAME>;  
SQL> DESC EMP;**

Name	Null?	Type
EMPNO		NUMBER(4)
ENAME		VARCHAR2(10)
DESIGNATION		VARCHAR2(10)
SALARY		NUMBER(8,2)

**QUERY: 03**

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

**Syntax For Create A from An Existing Table With All Fields**

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

**QUERY: 03**

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

**SQL> DESC EMP1**

<b>Name</b>	<b>Null?</b>	<b>Type</b>
EMPNO		NUMBER(4)
ENAME		VARCHAR2(10)
DESIGNATION		VARCHAR2(10)
SALARY		NUMBER(8,2)

**QUERY: 04**

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

**Syntax For Create A from An Existing Table With Selected Fields**

**SQL> CREATE TABLE <TRAGET TABLE NAME> SELECT EMPNO, ENAME  
FROM <SOURCE TABLE NAME>;**

**QUERY: 04**

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

**SQL> DESC EMP2**

<b>Name</b>	<b>Null?</b>	<b>Type</b>
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 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>;**

**QUERY: 05**

**SQL> CREATE TABLE EMP3 AS SELECT \* FROM EMP WHERE  
1>2;**

**Table created.**

**SQL> DESC EMP3;**

<b>Name</b>	<b>Null?</b>	<b>Type</b>
-------------	--------------	-------------

```
-----  
EMPNO          NUMBER(4)  
ENAME          VARCHAR2(10)  
DESIGNATION    VARCHAR2(10)  
SALARY         NUMBER(8,2);
```

### **ALTER & MODIFICATION ON TABLE**

#### **QUERY: 06**

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

**Syntax for Alter & Modify on a Single Column:**

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

#### **QUERY: 06**

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

**Table altered.**

**SQL> DESC EMP;**

Name	Null?	Type
EMPNO		NUMBER(6)
ENAME		VARCHAR2(10)
DESIGNATION		VARCHAR2(10)
SALARY		NUMBER(8,2)

**QUERY: 07**

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

**Syntax for alter table with multiple column:**

**SQL > ALTER <TABLE NAME> MODIFY <COLUMN NAME1> <DATATYPE> (SIZE), MODIFY <COLUMN NAME2> <DATATYPE> (SIZE) .....**;

**QUERY: 07**

**SQL>ALTER TABLE EMP MODIFY (EMPNO NUMBER (7), ENAME VARCHAR2(12));**  
**Table altered.**

**SQL> DESC EMP;**

Name	Null?	Type
EMPNO		NUMBER(7)
ENAME		VARCHAR2(12)
DESIGNATION		VARCHAR2(10)
SALARY		NUMBER(8,2)

**QUERY: 08**

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

**Syntax for add a new column:**

**SQL> ALTER TABLE <TABLE NAME> ADD (<COLUMN NAME> <DATA TYPE> <SIZE>);**

**QUERY: 08**

**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

##### Syntaxforaddanewcolumn:

SQL> ALTER TABLE <TABLE NAME> ADD (<COLUMN NAME1> <DATA TYPE> <SIZE>,<COLUMN NAME2> <DATA TYPE> <SIZE>,  
.....);

#### QUERY: 09

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

#### QUERY: 10

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

##### Syntaxforaddanewcolumn:

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

**QUERY: 10**

**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

**Syntaxforaddanewcolumn:**

**SQL> ALTER TABLE <TABLE NAME> DROP <COLUMN NAME1>,<COLUMN NAME2>..... ;**

**QUERY: 11**

**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)
		REMOVE

**QUERY: 12**

Q10. Write a query to rename table emp to employee

**Syntaxforaddanewcolumn:**

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



## QUERY: 12

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

Name	Null?	Type
EMPNO		NUMBER(7)
ENAME		VARCHAR2(12)
DESIGNATION		VARCHAR2(10)
SALARY		NUMBER(8,2)

## CONSTRAINTS

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

### 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:

- i) ENABLE
- ii) DISABLE
- iii) DROP

### Column level constraints Using Primary key

Q13. Write a query to create primary constraints with column level

## **Primarykey**

### **SyntaxforColumnlevelconstraintsUsingPrimarykey:**

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

### **QUERY:13**

SQL>CREATE TABLE EMPLOYEE(EMPNO NUMBER(4) **PRIMARY  
KEY,**  
ENAME VARCHAR2(10),  
JOB VARCHAR2(6),  
SAL NUMBER(5),  
DEPTNO NUMBER(7));

### **Column level constraints Using Primary key with naming convention**

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

### **Syntax for Column level constraints Using Primary key:**

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

### **QUERY:14**

SQL>CREATE TABLE EMPLOYEE(EMPNO NUMBER(4)  
**CONSTRAINT EMP\_EMPNO\_PK PRIMARY KEY,**  
ENAME VARCHAR2(10),  
JOB VARCHAR2(6),  
SAL NUMBER(5),  
DEPTNO NUMBER(7));

### **Table Level Primary Key Constraints**

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

#### **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>);

#### **QUERY: 15**

```
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):**

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

#### **Syntax for Column level constraints Using Primary key:**

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

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

#### **QUERY: 16**

```
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 PRIMARY  
KEY (EMPNO);
```

## **Reference/foreign key constraint**

**Column level foreign key constraint:**

Q.17. Write a query to create foreign key constraints with column level

**Parent Table:**

**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 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> .....);
```

**QUERY: 17**

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

```
SQL>CREATE TABLE EMP4  
(EMPNO NUMBER(3),  
DEPTNO NUMBER(2) REFERENCES DEPT(DEPTNO),  
DESIGN VARCHAR2(10));
```

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

**Parent Table:**

**Syntax for Column level constraints Using Primary key:**

Q.18. Write a query to create foreign key constraints with column level

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

#### **Child Table:**

#### **Syntax 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>  
.....);

#### **QUERY:18**

SQL>CREATE TABLE DEPT(DEPTNO NUMBER(2) PRIMARY  
KEY,  
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**

Q.19. Write a query to create foreign key constraints with Table level

#### **Parent Table:**

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

#### **Child Table:**

#### **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> );

#### **QUERY: 19**

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 FOREIGN  
KEY(DEPT NO)REFERENCESDEPT(DEPTNO));
```

### **Table Level Foreign Key Constraints with Alter command**

Q.20. Write a query to create foreign key constraints with Table level with alter command.

#### **Parent Table:**

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

#### **Child Table:**

#### **Syntax for Table level constraints using foreign key:**

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

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

#### **QUERY:20**

```
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 FOREIGN  
KEY(DEPTNO)REFERENCES DEPT(DEPTNO);

## **Checkconstraint**

### **Column Level Check Constraint**

Q.21. Write a query to create Check constraints with column level

#### **SyntaxforclumnlevelconstraintsusingCheck:**

SQL:>CREATE <OBJ.TYPE> <OBJ.NAME> (COLUMN NAME.1 <DATATYPE>  
(SIZE) CONSTRAINT <CONSTRAINTS NAME> <TYPE OF CONSTRAINTS>  
(CONSTRAINTNS CRITERIA) , COLUMN NAME2 <DATATYPE> (SIZE));

### **QUERY:21**

SQL>CREATE TABLE EMP7(EMPNO NUMBER(3),  
ENAME VARCHAR2(20),  
DESIGN VARCHAR2(15),  
SAL NUMBER(5)CONSTRAINT EMP7\_SAL\_CK CHECK(SAL>500 AND  
SAL<10001),  
DEPTNO NUMBER(2));

### **Table Level Check Constraint:**

Q.22. Write a query to create Check constraints with table level

#### **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> (CONSTRAINTNS  
CRITERIA)) ;

### **QUERY:22**

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**

Q.23. Write a query to create Check constraints with table level using alter command.

#### **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> (CONSTRAINTS
CRITERIA)) ;

```

### **QUERY:23**

```

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**

### **Column Level Constraint**

Q.24. Write a query to create unique constraints with column level

#### **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)) ;

```



## QUERY:24

```
SQL>CREATE TABLE EMP10(EMPNO NUMBER(3),
    ENAME VARCHAR2(20),
    DESGIN VARCHAR2(15)CONSTRAINT EMP10_DESIGN_UK UNIQUE,
    SAL NUMBER(5));
```

### Table Level Constraint

Q.25. Write a query to create unique constraints with table level

#### 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)); ;
```

## QUERY:25

```
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

Q.26. Write a query to create unique constraints with table level

#### **Syntax for Table level constraints with Check Using Alter**

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

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

## QUERY:26

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

```
SQL>ALTER TABLE EMP12 ADD CONSTRAINT EMP12_DESIGN_UK  
UNIQUE(DESING);
```

## **NotNull**

### **Column Level Constraint**

Q.27. Write a query to create Not Null constraints with column level

#### **SyntaxforColumnlevelconstraintswithNotNull:**

```
SQL :> CREATE <OBJ.TYPE> <OBJ.NAME> (<COLUMN NAME.1>  
<DATATYPE> (SIZE) CONSTRAINT <NAME OF CONSTRAINTS>  
<CONSTRAINT TYPE>, (COLUMN NAME2 <DATATYPE> (SIZE)) ;
```

### **QUERY: 27**

```
SQL>CREATE TABLE EMP13  
(EMPNO NUMBER(4),  
ENAME VARCHAR2(20) CONSTRAINT EMP13_ENAME_NN NOT NULL,  
DESIGN VARCHAR2(20),  
SAL NUMBER(3));
```

## **Null**

### **Column Level Constraint**

Q.28. Write a query to create Null constraints with column level

#### **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)) ;
```

### **QUERY:28**

```
SQL>CREATE TABLE EMP13  
(EMPNO NUMBER(4),  
ENAME VARCHAR2(20) CONSTRAINT EMP13_ENAME_NN NULL,  
DESIGN VARCHAR2(20),  
SAL NUMBER(3));
```

## **Constraint Disable\Enable**

### **Constraint Disable**

Q.29. Write a query to disable the constraints

#### **Syntax for disabling a single constraint in a table:**

SQL>ALTER TABLE <TABLE-NAME> DISABLE CONSTRAINT <CONSTRAINT-NAME>

### **Constraint Enable**

#### **QUERY:29**

SQL>ALTER TABLE EMP13 DISABLE CONSTRAINT EMP13\_ENAME\_NN  
NULL;

Q.30. Write a query to enable the constraints

#### **Syntax for disabling a single constraint in a table:**

SQL>ALTER TABLE <TABLE-NAME> DISABLE CONSTRAINT <CONSTRAINT-NAME>

#### **QUERY:30**

SQL>ALTER TABLE EMP13 ENABLE CONSTRAINT EMP13\_ENAME\_NN  
NULL;

## **EX: NO: 2**

### **To implementation on DML and DCL Commands in RDBMS**

#### **AIM:**

To execute and verify the DML and TCL Language commands

#### **DML (DATA MANIPULATION LANGUAGE)**

SELECT

INSERT

DELETE

UPDATE

#### **TCL (TRANSACTION CONTROL LANGUAGE)**

COMMIT

ROLL BACK

SAVE POINT

#### **PROCEDURE**

STEP 1: Start

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 6: use commit for permanently save the records.

#### **SQL COMMANDS**

##### **1. COMMAND NAME: INSERT**

COMMAND DESCRIPTION: INSERT command is used to Insert objects in the database.

##### **2. COMMAND NAME: SELECT**

COMMAND DESCRIPTION: **SELECT** command is used to **SELECT** the object from the database.

3. COMMAND NAME: **UPDATE**

COMMAND DESCRIPTION: **UPDATE** command is used to **UPDATE** the records from the table

4. COMMAND NAME: **DELETE**

COMMAND DESCRIPTION: **DELETE** command is used to **DELETE** the Records from the table

5. COMMAND NAME: **COMMIT**

COMMAND DESCRIPTION: **COMMIT** command is used to save the Records.

6. COMMAND NAME: **ROLLBACK**

COMMAND DESCRIPTION: **ROLL BACK** command is used to undo the Records.

6. COMMAND NAME: **SAVE POINT**

COMMAND DESCRIPTION: **SAVE POINT** command is used to undo the Records in a particular transaction.

## **INSERT**

### **QUERY: 01**

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

**Syntax for Insert Records in to a table:**

**SQL :> INSERT INTO <TABLE NAME> VALUES< VAL1, 'VAL2',.....>;**

### **QUERY: 01**

**INSERT A RECORD FROM AN EXISTING TABLE:**

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

1 row created.

## SELECT

### QUERY: 02

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

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

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

### QUERY: 02

**DISPLAY THE EMP TABLE:**

**SQL> SELECT \* FROM EMP;**

EMPNO	ENAME	DESIGNATION	SALARY
101	NAGARAJAN	LECTURER	15000

## INSERT A RECORD USING SUBSTITUTION METHOD

### QUERY: 03

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

**Syntax for Insert Records into the table:**

**SQL :> INSERT INTO <TABLE NAME> VALUES(< '&column name', '&column name 2',.....>);**

### QUERY: 03

**SQL> INSERT INTO EMP**

**VALUES(&EMPNO,'&ENAME','&DESIGNATION','&SALARY');**

Enter value for empno: 102

Enter value for ename: SARAVANAN

Enter value for designatin: LECTURER

Enter value for salary: 15000

```
old 1: INSERT INTO EMP
VALUES(&EMPNO,&ENAME,&DESIGNATIN,&SALARY)
new 1: INSERT INTO EMP VALUES(102,'SARAVANAN','LECTURER','15000')
1 row created.
```

SQL> /

```
Enter value for empno: 103
Enter value for ename: PANNERSELVAM
Enter value for designatin: ASST. PROF
Enter value for salary: 20000
```

```
old 1: INSERT INTO EMP
VALUES(&EMPNO,&ENAME,&DESIGNATIN,&SALARY)
new 1: INSERT INTO EMP VALUES(103,'PANNERSELVAM','ASST.
PROF','20000')
1 row created.
```

SQL> /

```
Enter value for empno: 104
Enter value for ename: CHINNI
Enter value for designatin: HOD, PROF
Enter value for salary: 45000
```

```
old 1: INSERT INTO EMP
VALUES(&EMPNO,&ENAME,&DESIGNATIN,&SALARY)
new 1: INSERT INTO EMP VALUES(104,'CHINNI','HOD, PROF','45000')
1 row created.
```

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

## UPDATE

### QUERY: 04

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

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

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

### QUERY: 04

```
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 for update multiple Records from the table:

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

### QUERY: 05



```
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

### QUERY: 06

Q5. Write a query to delete records from employee.

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

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

### QUERY: 06

```
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

## TCL(TRANSACTION CONTROL LANGUAGE)

### SAVEPOINT:

#### QUERY: 07

Q5. Write a query to implement the save point.

#### Syntax for savepoint:

SQL> SAVEPOINT <SAVE POINT NAME>;

#### QUERY: 07

SQL> SAVEPOINT S1;

Savepoint created.

SQL> SELECT \* FROM EMP;

EMPNO	ENAME	DESIGNATION	SALARY
101	NAGARAJAN	LECTURER	16000
102	SARAVANAN	ASST. PROF	16000
104	CHINNI	HOD, PROF	45000

SQL> INSERT INTO EMP VALUES(105,'PARTHASAR','STUDENT',100);

1 row created.

SQL> SELECT \* FROM EMP;

EMPNO	ENAME	DESIGNATION	SALARY
105	PARTHASAR	STUDENT	100

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

## ROLL BACK

### QUERY: 08

Q5. Write a query to implement the Rollback.

#### Syntaxforsavepoint:

SQL> ROLL BACK <SAVE POINT NAME>;

### QUERY: 08

SQL> ROLL BACK S1;

Rollback complete.

SQL> SELECT \* FROM EMP;

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

## COMMIT

### QUERY: 09

Q5. Write a query to implement the Rollback.

#### Syntaxforcommit:

SQL> COMMIT;

## QUERY: 09

SQL> COMMIT;

Commit complete.

## DCL(DATA CONTROLLANGUAGE)

### CREATING A USER

SQL>CONNECT SYSTEM/MANAGER;

SQL>CREATE USER "USERNAME" IDENTIFIED BY "PASSWORD"

SQL>GRANT DBA TO "USERNAME"

SQL>CONNECT "USERNAME"/"PASSWORD";

EXAMPLE

CREATING A USER

SQL>CONNECT SYSTEM/MANAGER;

SQL>CREATE USER CSE2 IDENTIFIED BY CSECSE;

SQL>GRANT DBA TO CSE2;

SQL>CONNECT CSE2/CSECSE;

SQL>REVOKE DBA FROM CSE2;

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

SQL> select \* from emp;

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800	2000
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	3000
7521	WARD	SALESMAN	7698	22-FEB-81	1250	5000

7566 JONES     MANAGER     7839 02-APR-81     2975     2000

4 rows selected.

SQL> select empno,ename,sal from emp;

EMPNO	ENAME	SAL
7369	SMITH	800
7499	ALLEN	1600
7521	WARD	1250
7566	JONES	2975

SQL>select ename,job,sal,deptno from emp where sal not between 1500 and 5000;

ENAME	JOB	SAL	DEPTNO
SMITH	CLERK	800	20
WARD	SALESMAN	1250	30
MARTIN	SALESMAN	1250	30
ADAMS	CLERK	1100	20
JAMES	CLERK	950	30
MILLER	CLERK	1300	10

6 rows selected.

SQL> select empno,ename,sal from emp where sal in (800,5000);

EMPNO	ENAME	SAL
7369	SMITH	800
7839	KING	5000

SQL> select empno,ename,sal from emp where comm is null;

EMPNO	ENAME	SAL
7369	SMITH	800
7566	JONES	2975
7698	BLAKE	2850
7782	CLARK	2450
7788	SCOTT	3000
7839	KING	5000

7876 ADAMS	1100
7900 JAMES	950
7902 FORD	3000
7934 MILLER	1300

10 rows selected.

SQL> select empno,ename,sal from emp where comm is not null;

EMPNO	ENAME	SAL
7499 ALLEN		1600
7521 WARD		1250
7654 MARTIN		1250
7844 TURNER		1500

SQL> select empno,ename,job,sal from emp where ename like'S%';

EMPNO	ENAME	JOB	SAL
7369 SMITH		CLERK	800
7788 SCOTT		ANALYST	3000

SQL> select empno,ename,job,sal from emp where job not like'S%';

EMPNO	ENAME	JOB	SAL
7369 SMITH		CLERK	800
7566 JONES		MANAGER	2975
7698 BLAKE		MANAGER	2850
7782 CLARK		MANAGER	2450
7788 SCOTT		ANALYST	3000

SQL> select ename,job,sal from emp where sal>2500;

ENAME	JOB	SAL
JONES	MANAGER	2975
BLAKE	MANAGER	2850
SCOTT	ANALYST	3000
KING	PRESIDENT	5000
FORD	ANALYST	3000

SQL> select ename,job,sal from emp where sal<2500;

ENAME	JOB	SAL
-------	-----	-----

SMITH	CLERK	800
ALLEN	SALESMAN	1600
WARD	SALESMAN	1250
MARTIN	SALESMAN	1250
CLARK	MANAGER	2450
TURNER	SALESMAN	1500
ADAMS	CLERK	1100
JAMES	CLERK	950
MILLER	CLERK	1300

9 rows selected.

SQL> select empno,ename,job,sal from emp order by sal;

EMPNO	ENAME	JOB	SAL
7369	SMITH	CLERK	800
7900	JAMES	CLERK	950
7876	ADAMS	CLERK	1100
7521	WARD	SALESMAN	1250
7654	MARTIN	SALESMAN	1250
7934	MILLER	CLERK	1300
7844	TURNER	SALESMAN	1500
7499	ALLEN	SALESMAN	1600
7782	CLARK	MANAGER	2450
7698	BLAKE	MANAGER	2850
7566	JONES	MANAGER	2975
7788	SCOTT	ANALYST	3000
7902	FORD	ANALYST	3000
7839	KING	PRESIDENT	5000

14 rows selected.

SQL> select empno,ename,job,sal from emp order by sal desc;

EMPNO	ENAME	JOB	SAL
7839	KING	PRESIDENT	5000
7788	SCOTT	ANALYST	3000
7902	FORD	ANALYST	3000
7566	JONES	MANAGER	2975
7698	BLAKE	MANAGER	2850
7782	CLARK	MANAGER	2450

7499	ALLEN	SALESMAN	1600
7844	TURNER	SALESMAN	1500
7934	MILLER	CLERK	1300
7521	WARD	SALESMAN	1250
7654	MARTIN	SALESMAN	1250

EMPNO	ENAME	JOB	SAL
-----			
7876	ADAMS	CLERK	1100
7900	JAMES	CLERK	950
7369	SMITH	CLERK	800

14 rows selected.



## **EX: NO: 3**

## **NESTED QUERIES AND JOIN QUERIES**

### **EX: NO: 3 A**

### **Nested Queries**

#### **AIM**

To execute and verify the SQL commands for Nested Queries.

#### **OBJECTIVE:**

Nested Query can have more than one level of nesting in one single query. A SQL nested query is a **SELECT** query that is nested inside a **SELECT**, **UPDATE**, **INSERT**, or **DELETE** SQL query.

#### **PROCEDURE**

STEP 1: Start

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

STEP 3: Insert attribute values into the table.

STEP 4: Create the Nested query from the above created table.

STEP 5: Execute Command and extract information from the tables.

STEP 6: Stop

#### **SQL COMMANDS**

1. COMMAND NAME: **SELECT**

COMMAND DESCRIPTION: **SELECT** command is used to select records from the table.

2. COMMAND NAME: **WHERE**

COMMAND DESCRIPTION: **WHERE** command is used to identify particular elements.

### 3. COMMAND NAME: **HAVING**

COMMAND DESCRIPTION: **HAVING** command is used to identify particular elements.

### 4. COMMAND NAME: **MIN (SAL)**

COMMAND DESCRIPTION: **MIN (SAL)** command is used to find minimum salary.

**Table -1**

#### **SYNTAX FOR CREATING A TABLE:**

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

```
SQL> CREATE TABLE EMP2(EMPNO NUMBER(5),
                        ENAME VARCHAR2(20),
                        JOB VARCHAR2(20),
                        SAL NUMBER(6),
                        MGRNO NUMBER(4),
                        DEPTNO NUMBER(3));
```

#### **SYNTAX FOR INSERT RECORDS IN TO A TABLE:**

**SQL :> INSERT INTO <TABLE NAME> VALUES< VAL1, 'VAL2',.....);**

#### **INSERTION**

```
SQL> INSERT INTO EMP2 VALUES(1001,'MAHESH','PROGRAMMER',15000,1560,200);
1 ROW CREATED.
```

```
SQL> INSERT INTO EMP2 VALUES(1002,'MANOJ','TESTER',12000,1560,200);
1 ROW CREATED.
```

```
SQL> INSERT INTO EMP2 VALUES(1003,'KARTHIK','PROGRAMMER',13000,1400,201);
1 ROW CREATED.
```

```
SQL> INSERT INTO EMP2 VALUES(1004,'NARESH','CLERK',1400,1400,201);
1 ROW CREATED.
```

```
SQL> INSERT INTO EMP2 VALUES(1005,'MANI','TESTER',13000,1400,200);
1 ROW CREATED.
```

```
SQL> INSERT INTO EMP2 VALUES(1006,'VIKI','DESIGNER',12500,1560,201);
1 ROW CREATED.
```

```
SQL> INSERT INTO EMP2 VALUES(1007,'MOHAN','DESIGNER',14000,1560,201);
```

1 ROW CREATED.

SQL> INSERT INTO EMP2 VALUES(1008,'NAVEEN','CREATION',20000,1400,201);

1 ROW CREATED.

SQL> INSERT INTO EMP2 VALUES(1009,'PRASAD','DIR',20000,1560,202);

1 ROW CREATED.

SQL> INSERT INTO EMP2 VALUES(1010,'AGNESH','DIR',15000,1400,200);

1 ROW CREATED.

### **SYNTAXFORSELECTRECORDSFROMTHETABLE:**

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

SQL> SELECT \*FROM EMP2;

EMPNO	ENAME	JOB	SAL	MGRNO	DPTNO
-----	-----	-----	-----	-----	-----
1001	MAHESH	PROGRAMMER	15000	1560	200
1002	MANOJ	TESTER	12000	1560	200
1003	KARTHIK	PROGRAMMER	13000	1400	201
1004	NARESH	CLERK	1400	1400	201
1005	MANI	TESTER	13000	1400	200
1006	VIKI	DESIGNER	12500	1560	201
1007	MOHAN	DESIGNER	14000	1560	201
1008	NAVEEN	CREATION	20000	1400	201
1009	PRASAD	DIR	20000	1560	202
1010	AGNESH	DIR	15000	1400	200

**TABLE- 2**

### **SYNTAXFORCREATINGATABLE:**

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

```
SQL> CREATE TABLE DEPT2(DEPTNO NUMBER(3),
                           DEPTNAME VARCHAR2(10),
                           LOCATION VARCHAR2(15));
```

Table created.

### **SYNTAX FOR INSERT RECORDS INTO A TABLE:**

**SQL :> INSERT INTO <TABLE NAME> VALUES< VAL1, 'VAL2',.....>;**

### **INSERTION**

```
SQL> INSERT INTO DEPT2 VALUES(107,'DEVELOP','ADYAR');
```

1 ROW CREATED.

```
SQL> INSERT INTO DEPT2 VALUES(201,'DEBUG','UK');
```

1 ROW CREATED.

```
SQL> INSERT INTO DEPT2 VALUES(200,'TEST','US');
```

```
SQL> INSERT INTO DEPT2 VALUES(201,'TEST','USSR');
```

1 ROW CREATED.

```
SQL> INSERT INTO DEPT2 VALUES(108,'DEBUG','ADYAR');
```

1 ROW CREATED.

```
SQL> INSERT INTO DEPT2 VALUES(109,'BUILD','POTHERI');
```

1 ROW CREATED.

### **SYNTAX FOR SELECT RECORDS FROM THE TABLE:**

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

```
SQL> SELECT *FROM DEPT2;
```

DEPTNO	DEPTNAME	LOCATION
-----	-----	-----
107	DEVELOP	ADYAR
201	DEBUG	UK
200	TEST	US
201	TEST	USSR
108	DEBUG	ADYAR
109	BUILD	POTHERI

6 rows selected.

**GENERAL SYNTAX FOR NESTED QUERY:**

```
SELECT "COLUMN_NAME1"
FROM "TABLE_NAME1"
WHERE "COLUMN_NAME2" [COMPARISON OPERATOR]
(SELECT "COLUMN_NAME3"
FROM "TABLE_NAME2"
WHERE [CONDITION])
```

**SYNTAX NESTED QUERY STATEMENT:**

```
SQL> SELECT <COLUMN_NAME> FROM FROM <TABLE_1> WHERE
      <COLUMN_NAME> <RELATIONAL _OPERATION> 'VALUE'
      (SELECT (AGGREGATE FUNCTION) FROM <TABLE_1> WHERE <COLUMN
      NAME> = 'VALUE'
      (SELECT <COLUMN_NAME> FROM <TABLE_2> WHERE <COLUMN_NAME>=
      'VALUE'));
```

**NESTED QUERY STATEMENT:**

```
SQL> SELECT ENAME FROM EMP2 WHERE SAL>
```

```
(SELECT MIN(SAL) FROM EMP2 WHERE DPTNO=
```

```
(SELECT DEPTNO FROM DEPT2 WHERE LOCATION='UK'));
```

**Nested Query Output:**

ENAME

-----

MAHESH

MANOJ

KARTHIK

MANI

VIKI

MOHAN

NAVEEN

PRASAD

AGNESH

## **EX:NO:3B - JOINS**

### **AIM**

To execute and verify the SQL commands using Join queries.

### **OBJECTIVE:**

SQL joins are used to query data from two or more tables, based on a relationship between certain columns in these tables.

### **PROCEDURE**

STEP 1: Start

STEP 2: Create the table with its essential attributes.

STEP 3: Insert attribute values into the table

STEP 4: Execute different Commands and extract information from the table.

STEP 5: Stop

### **SQL COMMANDS**

#### **COMMAND NAME: INNER JOIN**

**COMMAND DESCRIPTION:** The INNER JOIN keyword return rows when there is at least one match in both tables.

#### **COMMAND NAME LEFT JOIN**

**COMMAND DESCRIPTION:** The LEFT JOIN keyword returns all rows from the left table (table\_name1), even if there are no matches in the right table (table\_name2).

#### **COMMAND NAME : RIGHT JOIN**

**COMMAND DESCRIPTION:** The RIGHT JOIN keyword Return all rows from the right table (table\_name2), even if there are no matches in the left table (table\_name1).

**COMMAND NAME : FULL JOIN**

**COMMAND DESCRIPTION:** The FULL JOIN keyword return rows when there is a match in one of the tables.



## **LEFT JOIN or LEFT OUTER JOIN**

### **Table:1-ORDERS**

```
SQL> CREATE table orders(O_Id number(5),  
                          Orderno number(5),  
                          P_Id number(3));
```

Table created.

```
SQL> DESC orders;
```

Name	Null?	Type
-----	-----	-----
O_ID		NUMBER(5)
ORDERNO		NUMBER(5)
P_ID		NUMBER(3)

### **INSERTING VALUES INTO ORDERS**

```
SQL> INSERT into orders values(&O_Id,&Orderno,&P_Id);
```

Enter value for o\_id: 1

Enter value for orderno: 77895

Enter value for p\_id: 3

old 1: INSERT into orders values(&O\_Id,&Orderno,&P\_Id)

new 1: INSERT into orders values(1,77895,3)

1 row created.

```
SQL> INSERT into orders values(&O_Id,&Orderno,&P_Id);
```

Enter value for o\_id: 2

Enter value for orderno: 44678

Enter value for p\_id: 3

old 1: INSERT into orders values(&O\_Id,&Orderno,&P\_Id)

new 1: INSERT into orders values(2,44678,3)

1 row created.

SQL> INSERT into orders values(&O\_Id,&Orderno,&P\_Id);

Enter value for o\_id: 3

Enter value for orderno: 22456

Enter value for p\_id: 1

old 1: INSERT into orders values(&O\_Id,&Orderno,&P\_Id)

new 1: INSERT into orders values(3,22456,1)

1 row created.

SQL> INSERT into orders values(&O\_Id,&Orderno,&P\_Id);

Enter value for o\_id: 4

Enter value for orderno: 24562

Enter value for p\_id: 1

old 1: INSERT into orders values(&O\_Id,&Orderno,&P\_Id)

new 1: INSERT into orders values(4,24562,1)

1 row created.

SQL> INSERT into orders values(&O\_Id,&Orderno,&P\_Id);

Enter value for o\_id: 5

Enter value for orderno: 34764

Enter value for p\_id: 15

old 1: INSERT into orders values(&O\_Id,&Orderno,&P\_Id)

new 1: INSERT into orders values(5,34764,15)

1 row created.

### **TABLESECTION:**

SQL> SELECT \* FROM orders;

O_ID	ORDERNO	P_ID
-----	-----	-----
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15

### **TABLE-2:PERSONS**

SQL> CREATE table persons(p\_Id number(5),  
LASTNAME varchar2(10),  
Firstname varchar2(15), Address varchar2(20),  
city varchar2(10));

Table created.

SQL> INSERT into persons values(&p\_Id,&Lastname','&firstname','&Address','&city');

Enter value for p\_id: 1

Enter value for lastname: Hansen

Enter value for firstname: Ola

Enter value for address: Timoteivn 10

Enter value for city: sadnes

old 1: INSERT into persons values(&p\_Id,'&Lastname','&firstname','&Address','&city')

new 1: INSERT into persons values(1,'Hansen','Ola','Timoteivn 10','sadnes')

1 row created.

SQL> INSERT into persons values(&p\_Id,'&Lastname','&firstname','&Address','&city');

Enter value for p\_id: 2

Enter value for lastname: Svendson

Enter value for firstname: Tove

Enter value for address: Borgn 23

Enter value for city: Sandnes

old 1: INSERT into persons values(&p\_Id,'&Lastname','&firstname','&Address','&city')

new 1: INSERT into persons values(2,'Svendson','Tove','Borgn 23','Sandnes')

1 row created.

SQL> INSERT into persons values(&p\_Id,'&Lastname','&firstname','&Address','&city');

Enter value for p\_id: 3

Enter value for lastname: Pettersen

Enter value for firstname: Kari

Enter value for address: Storgt 20

Enter value for city: Stavanger

old 1: INSERT into persons values(&p\_Id,'&Lastname','&firstname','&Address','&city')

new 1: INSERT into persons values(3,'Pettersen','Kari','Storgt 20','Stavanger')

1 row created.

SQL> SELECT \* FROM persons;

P_ID	LASTNAME	FIRSTNAME	ADDRESS	CITY
1	Hansen	Ola	Timoteivn 10	sandnes
2	Svendson	Tove	Borgn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

## **LEFT JOIN SYNTAX**

SQL> SELECT column\_name(s)

FROM table\_name1

LEFT JOIN table\_name2

ON table\_name1.column\_name=table\_name2.column\_name

## **LEFT JOIN EXAMPLE**

SQL> SELECT persons.lastname,persons.firstname,orders.orderno

FROM persons

LEFT JOIN orders

ON persons.p\_Id = orders.p\_Id

ORDER BY persons.lastname;

### **OUTPUT**

LASTNAME	FIRSTNAME	ORDERNO
-----	-----	-----
Hansen	Ola	22456
Hansen	Ola	24562
Pettersen	Kari	77895
Pettersen	Kari	44678
Svendson	Tove	

### **FULL OUTER JOIN**

SQL> SELECT \* FROM persons;

P_ID	LASTNAME	FIRSTNAME	ADDRESS	CITY
-----	-----	-----	-----	-----
1	Hansen	Ola	Timoteivn 10	sandnes
2	Svendson	Tove	Borgn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

SQL> SELECT \* FROM orders;

O_ID	ORDERNO	P_ID
-----	-----	-----
1	77895	3
2	44678	3
3	22456	1

4	24562	1
5	34764	15

### **FULL OUTER JOIN SYNTAX**

```
SQL>SELECT column_name(s)
      FROM table_name1
      FULL JOIN table_name2
      ON table_name1.column_name=table_name2.column_name
```

### **FULL OUTER JOIN EXAMPLE**

```
SQL> SELECT persons.lastname,persons.firstname,orders.orderno
      FROM persons
      FULL OUTER JOIN orders
      ON persons.p_Id = orders.p_Id
      ORDER BY persons.lastname;
```

## **RIGHT OUTER JOIN**

### **RIGHT OUTER JOIN SYNTAX**

```
SQL>SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
      FROM Persons
      RIGHT JOIN Orders
      ON Persons.P_Id=Orders.P_Id
      ORDER BY Persons.LastName
```

### **RIGHT OUTER JOIN EXAMPLE**

```
SQL> SELECT persons.lastname,persons.firstname,orders.orderno
      FROM persons
      RIGHT OUTER JOIN orders
      ON persons.p_Id = orders.p_Id
      ORDER BY persons.lastname;
```

LASTNAME	FIRSTNAME	ORDERNO
-----	-----	-----
Hansen	Ola	24562
Hansen	Ola	22456
Pettersen	Kari	44678
Pettersen	Kari	77895

## **INNERJOIN**

### **INNERJOINSYNTAX**

```
SQL>SELECT column_name(s)
      FROM table_name1
      INNER JOIN table_name2
      ON table_name1.column_name=table_name2.column_name
```

### **INNERJOINEXAMPLE**

```
SQL> SELECT persons.lastname,persons.firstname,orders.orderno
2  FROM persons
3  INNER JOIN orders
4  ON persons.p_Id = orders.p_Id
5  ORDER BY persons.lastname;
```

LASTNAME	FIRSTNAME	ORDERNO
-----	-----	-----
Hansen	Ola	22456
Hansen	Ola	24562
Pettersen	Kari	77895
Pettersen	Kari	44678



S 2258    LASTNAME

FIRSTNAME

ORDERNO

-----	-----	-----
Hansen	Ola	22456
Hansen	Ola	24562
Pettersen	Kari	77895
Pettersen	Kari	44678
Svendson	Tove	34764

6 rows selected.

## **EX: NO: 4**

## **VIEWS**

### **AIM**

To execute and verify the SQL commands for 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.

### **PROCEDURE**

STEP 1: Start

STEP 2: Create the table with its essential attributes.

STEP 3: Insert attribute values into the table.

STEP 4: Create the view from the above created table.

STEP 5: Execute different Commands and extract information from the View.

STEP 6: Stop

### **.SQL COMMANDS**

#### **1. COMMAND NAME: CREATE VIEW**

COMMAND DESCRIPTION: **CREATE VIEW** command is used to define a view.

#### **2. COMMAND NAME: INSERT IN VIEW**

COMMAND DESCRIPTION: **INSERT** command is used to insert a new row into the view.

#### **3. COMMAND NAME: DELETE IN VIEW**

COMMAND DESCRIPTION: **DELETE** command is used to delete a row from the view.

#### **4. COMMAND NAME: UPDATE OF VIEW**

COMMAND DESCRIPTION: **UPDATE** command is used to change a value in a tuple without changing all values in the tuple.

#### **5. COMMAND NAME: DROP OF VIEW**

COMMAND DESCRIPTION: **DROP** command is used to drop the view table

## COMMANDEXECUTION

### **CREATION OF TABLE**

```
SQL> CREATE TABLE EMPLOYEE (  
        EMPLOYEE_NAME VARCHAR2(10),  
        EMPLOYEE_NO NUMBER(8),  
        DEPT_NAME VARCHAR2(10),  
        DEPT_NO NUMBER (5), DATE_OF_JOIN DATE);
```

Table created.

### **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
```

### **SUNTAX FOR CREATION OF VIEW**

```
SQL> CREATE <VIEW> <VIEW NAME> AS SELECT  
        <COLUMN_NAME_1>, <COLUMN_NAME_2> FROM <TABLE NAME>;
```

### **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 INTO VIEW**

-----

**INSERT STATEMENT:****SYNTAX:**

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

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

1 ROW CREATED.

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
SRI	120	CSE	67

SQL> SELECT \* FROM EMPLOYEE;

EMPLOYEE_N	EMPLOYEE_NO	DEPT_NAME	DEPT_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:****DELETE STATEMENT:****SYNTAX:**

SQL> DELETE <VIEW\_NAME> WHERE <COLUMN NAME> ='VALUE';

SQL> DELETE FROM EMPVIEW WHERE EMPLOYEE\_NAME='SRI';

1 ROW DELETED.

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

### **UPDATE STATEMENT:**

#### **SYNTAX:**

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

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

1 ROW UPDATED.

SQL> SELECT \* FROM EMPKAVIVIEW;

EMPLOYEE_N	EMPLOYEE_NO	DEPT_NAME	DEPT_NO
KAVI	124	ECE	89
VIJAY	345	CSE	21
RAJ	98	IT	22
GIRI	100	CSE	67

### **DROP VIEW:**

#### **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 keyword to avoid the error “ora\_0095:name is already used by an existing object”.

**CHANGING THE COLUMN(S) NAME IN 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	SALARY
7369	SMITH	1000
7499	MARK	1050
7565	WILL	1500
7678	JOHN	1800
7578	TOM	1500
7548	TURNER	1500

6 rows selected.

View created.

EMPNO	ENAME	SALARY	MGRNO	DEPTNO
7578	TOM	1500	7298	10
7548	TURNER	1500	7298	10

View created.

```
SQL> SELECT * FROM EMP_TOTSAL;
```

**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.DNAM
E "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	RESEARCH	ZURICH
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;
```

## WITHCHECKOPTIONCLAUSE

### **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;
```

### **JOINVIEW:**

### **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,DEPT B WHERE A.DEPTNO=B.DEPTNO;
```

```
SQL> SELECT * FROM DEPT_EMP_VIEW;  
View created.
```

EMPNO	ENAME	SALARY	DEPTNO
7578	TOM	1500	10
7548	TURNER	1500	10

View created.

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.

## FORCEVIEW

### **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	TURNER	1500	10

6 rows selected.

View 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	TURNER	1500	10

6 rows selected

## **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.

## **EX: NO: 5 A**

## **CONTROLSTRUCTURE**

### **AIM**

To write a PL/SQL block using different control (if, if else, for loop, while loop,...) statements.

### **OBJECTIVE:**

PL/SQL Control Structure provides conditional tests, loops, flow control and branches that let to produce well-structured programs.

### **AdditionofTwoNumbers:**

1. Write a PL/SQL Program for Addition of Two Numbers

### **PROCEDURE**

STEP 1: Start

STEP 2: Initialize the necessary variables.

STEP 3: Develop the set of statements with the essential operational parameters.

STEP 4: Specify the Individual operation to be carried out.

STEP 5: Execute the statements.

STEP 6: Stop.

### **PL/ SQL General Syntax**

SQL> DECLARE

    <VARIABLE DECLARATION>;

    BEGIN

        <EXECUTABLE STATEMENT >;

    END;

## **PL/SQL CODING FOR ADDITION OF TWO NUMBERS**

```
SQL> declare
a number;
b number;
c number;
begin
a:=&a;
b:=&b;
c:=a+b;
dbms_output.put_line('sum of'||a||'and'||b||'is'||c);
end;
/
```

### **INPUT:**

```
Enter value for a: 23
old 6: a:=&a;
new 6: a:=23;
Enter value for b: 12
old 7: b:=&b;
new 7: b:=12;
```

### **OUTPUT:**

```
sum of23and12is35
```

PL/SQL procedure successfully completed.

## **PL/SQL Program for IF Condition:**

2. Write a PL/SQL Program using if condition

### **PROCEDURE**

STEP 1: Start

STEP 2: Initialize the necessary variables.

STEP 3: invoke the if condition.

STEP 4: Execute the statements.

STEP 5: Stop.

### **PL/SQL GENERAL SYNTAX FOR IF CONDITION:**

```
SQL> DECLARE
        <VARIABLE DECLARATION>;
    BEGIN
        IF(CONDITION) THEN
            <EXECUTABLE STATEMENT >;
        END;
```

### **Coding for If Statement:**

```
DECLARE
    b number;
    c number;
BEGIN
    B:=10;
    C:=20;
    if(C>B) THEN
        dbms_output.put_line('C is maximum');
    end if;
end;
/
```

## **OUTPUT:**

C is maximum

PL/SQL procedure successfully completed.

## **PL/SQL GENERAL SYNTAX FOR IF AND ELSE CONDITION:**

```
SQL> DECLARE
        <VARIABLE DECLARATION>;
    BEGIN
        IF (TEST CONDITION) THEN
            <STATEMENTS>;
        ELSE
            <STATEMENTS>;
        ENDIF;
    END;
```

**\*\*\*\*\*Less then or Greater Using IF ELSE \*\*\*\*\***

```
SQL> declare
    n number;
    begin
        dbms_output.put_line('enter a number');
        n:=&number;
        if n<5 then
            dbms_output.put_line('entered number is less than 5');
        else
            dbms_output.put_line('entered number is greater than 5');
```

```
end if;  
end;  
/
```

### **Input**

Enter value for number: 2  
old 5: n:=&number;  
new 5: n:=2;

### **Output:**

entered number is less than 5

PL/SQL procedure successfully completed.

## **PL/ SQL GENERAL SYNTAX FOR NESTED IF:**

```
SQL> DECLARE  
        <VARIABLE DECLARATION>;  
BEGIN  
        IF (TEST CONDITION) THEN  
            <STATEMENTS>;  
        ELSEIF (TEST CONDITION) THEN  
            <STATEMENTS>;  
        ELSE  
            <STATEMENTS>;  
        ENDIF;  
END;
```

**\*\*\*\*\* GREATEST OF THREE NUMBERS USING IF ELSEIF\*\*\*\*\***

```
SQL> declare
  a number;
  b number;
  c number;
  d number;
begin
  a:=&a;
  b:=&b;
  c:=&b;
  if(a>b)and(a>c) then
  dbms_output.put_line('A is maximum');
    elsif(b>a)and(b>c)then
  dbms_output.put_line('B is maximum');
  else
  dbms_output.put_line('C is maximum');
  end if;
end;
/
```

**INPUT:**

```
Enter value for a: 21
old 7: a:=&a;
new 7: a:=21;
Enter value for b: 12
old 8: b:=&b;
new 8: b:=12;
Enter value for b: 45
old 9: c:=&b;
new 9: c:=45;
```

**OUTPUT:**

C is maximum

PL/SQL procedure successfully completed.

## PL/SQL GENERAL SYNTAX FOR LOOPING STATEMENT:

```
SQL> DECLARE
    <VARIABLE DECLARATION>;
BEGIN
    LOOP
    <STATEMENT>;
    END LOOP;
    <EXECUTABLE STATEMENT>;
END;
```

### **\*\*\*\*\*SUMMATION OF ODD NUMBERS USING FOR LOOP\*\*\*\*\***

```
SQL> declare
n number;
sum1 number default 0;
endvalue number;
begin
endvalue:=&endvalue;
n:=1;
for n in 1..endvalue
loop
    if mod(n,2)=1
    then
    sum1:=sum1+n;
    end if;
    end loop;
    dbms_output.put_line('sum ='||sum1);
end;
/
```

### **INPUT:**

```
Enter value for endvalue: 4
old 6: endvalue:=&endvalue;
new 6: endvalue:=4;
```



## OUTPUT:

sum =4

PL/SQL procedure successfully completed.

## PL/SQL GENERAL SYNTAX FOR LOOPING STATEMENT:

```
SQL> DECLARE
        <VARIABLE DECLARATION>;
BEGIN
    WHILE <condition>
    LOOP
        <STATEMENT>;
    END LOOP;
    <EXECUTABLE STATEMENT>;
END;
```

### \*\*\*\*\*SUMMATION OF ODD NUMBERS USING WHILE LOOP\*\*\*\*\*

```
SQL> declare
n number;
sum1 number default 0;
endvalue number;
begin
endvalue:=&endvalue;
n:=1;
while(n<endvalue)
loop
sum1:=sum1+n;
n:=n+2;
end loop;
```

```
dbms_output.put_line('sum of odd no. bt 1 and' ||endvalue||'is'||sum1);  
end;  
/
```

**INPUT:**

Enter value for endvalue: 4  
old 6: endvalue:=&endvalue;  
new 6: endvalue:=4;

**OUTPUT:**

sum of odd no. bt 1 and4is4

PL/SQL procedure successfully completed.

**RESULT:**

Thus the PL/SQL block for different controls are verified and executed.

## **EX: NO:5B**

## **PROCEDURES**

### **AIM**

To write a PL/SQL block to display the student name, marks whose average mark is above 60%.

### **ALGORITHM**

STEP1:Start

STEP2:Create a table with table name stud\_exam

STEP3:Insert the values into the table and Calculate total and average of each student

STEP4: Execute the procedure function the student who get above 60%.

STEP5: Display the total and average of student

STEP6: End

### **EXECUTION**

#### **SETTING SERVEROUTPUT ON:**

```
SQL> SET SERVEROUTPUT ON
```

#### **PROGRAM:**

#### **PROCEDURE USING POSITIONAL PARAMETERS:**

```
SQL> SET SERVEROUTPUT ON
SQL> CREATE OR REPLACE PROCEDURE PROC1 AS
2 BEGIN
3   DBMS_OUTPUT.PUT_LINE('Hello from procedure...');
4 END;
5 /
```

#### **Output:**

Procedure created.

```
SQL> EXECUTE PROC1
Hello from procedure...
```

PL/SQL procedure successfully completed.

## **II)PROGRAM:**

### **PROCEDURE USING NOTATIONAL PARAMETERS:**

```
SQL> CREATE OR REPLACE PROCEDURE PROC2
2      (N1 IN NUMBER,N2 IN NUMBER,TOT OUT NUMBER) IS
3 BEGIN
4     TOT := N1 + N2;
5 END;
6 /
```

Output:

Procedure created.

```
SQL> VARIABLE T NUMBER
```

```
SQL> EXEC PROC2(33,66,:T)
```

PL/SQL procedure successfully completed.

```
SQL> PRINT T
```

```
      T
-----
      99
```

## PROCEDUREFORGCDNUMBERS

### III)PROGRAM:

```
SQL> create or replace procedure pro
is
  a number(3);
  b number(3);
  c number(3);
  d number(3);
begin
  a:=&a;
  b:=&b;
  if(a>b) then
    c:=mod(a,b);
    if(c=0) then
      dbms_output.put_line('GCD is');
      dbms_output.put_line(b);
    else
      dbms_output.put_line('GCD is');
      dbms_output.put_line(c);
    end if; else
    d:=mod(b,a);
    if(d=0) then
      dbms_output.put_line('GCD is');
      dbms_output.put_line(a);
    else
      dbms_output.put_line('GCD is');
      dbms_output.put_line(d);
    end if;
  end if;
end;
/
```

Enter value for a: 8

old 8: a:=&a;

new 8: a:=8;

Enter value for b: 16

old 9: b:=&b;

new 9: b:=16;

Procedure created.

SQL> set serveroutput on;

SQL> execute pro;

GCD is

8

PL/SQL procedure successfully completed.

## PROCEDURE FOR CURSOR IMPLEMENTATION

### IV)PROGRAM:

```
SQL> create table student(regno number(4),name varchar2(20),mark1 number(3), mark2
      number(3), mark3 number(3), mark4 number(3), mark5 number(3));
```

Table created

```
SQL> insert into student values (101,'priya', 78, 88,77,60,89);
```

1 row created.

```
SQL> insert into student values (102,'surya', 99,77,69,81,99);
```

1 row created.

```
SQL> insert into student values (103,'suryapriya', 100,90,97,89,91);
```

1 row created.

```
SQL> select * from student;
```

regno	name	mark1	mark2	mark3	mark4	mark5
101	priya	78	88	77	60	89
102	surya	99	77	69	81	99
103	suryapriya	100	90	97	89	91

```
SQL> declare
ave number(5,2);
tot number(3);
cursor c_mark is select*from student where mark1>=40 and mark2>=40 and
mark3>=40 and mark4>=40 and mark5>=40;
begin
dbms_output.put_line('regno name mark1 mark2 mark3 mark4 mark4 mark5 total
average');
dbms_output.put_line('-----');
for student in c_mark loop
tot:=student.mark1+student.mark2+student.mark3+student.mark4+student.mark5;
ave:=tot/5;
dbms_output.put_line(student.regno||rpad(student.name,15)
||rpad(student.mark1,6)||rpad(student.mark2,6)||rpad(student.mark3,6)
||rpad(student.mark4,6)||rpad(student.mark5,6)||rpad(tot,8)||rpad(ave,5));
end loop;
end;
/
```

### SAMPLE OUTPUT

regno	name	mark1	mark2	mark3	mark4	mark5	total	average
101	priya	78	88	77	60	89	393	79
102	surya	99	77	69	81	99	425	85
103	suryapriya	100	90	97	89	91	467	93

PL/SQL procedure successfully completed.

## EXPLICITCURSORSANDEXPLICITCURSORSIMPLEMENTATION

### CREATING A TABLE EMP IN ORACLE

#### V) PROGRAM

SQL> select \* from EMP;

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM
7369	SMITH	CLERK	7902	17-DEC-80	800	

7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM
7566	JONES	MANAGER	7839	02-APR-81	2975	

7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400
7698	BLAKE	MANAGER	7839	01-MAY-81	2850	

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM
7782	CLARK	MANAGER	7839	09-JUN-81	2450	

7788	SCOTT	ANALYST	7566	09-DEC-82	3000	
------	-------	---------	------	-----------	------	--



7839 KING	PRESIDENT	17-NOV-81	5000
10			

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM
-----						
DEPTNO						
-----						
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0
30						
7876	ADAMS	CLERK	7788	12-JAN-83	1100	
20						
7900	JAMES	CLERK	7698	03-DEC-81	950	
30						

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM
-----						
DEPTNO						
-----						
7902	FORD	ANALYST	7566	03-DEC-81	3000	
20						
7934	MILLER	CLERK	7782	23-JAN-82	1300	
10						

14 rows selected.

### Implicit cursors:

```
SQL> DECLARE
2  ena EMP.ENAME%TYPE;
3  esa EMP.SAL%TYPE;
4  BEGIN
5      SELECT ENAME,SAL INTO ENA,ESA FROM EMP
6          WHERE EMPNO = &EMPNO;
7  DBMS_OUTPUT.PUT_LINE('NAME :' || ENA);
8  DBMS_OUTPUT.PUT_LINE('SALARY :' || ESA);
9
10 EXCEPTION
11  WHEN NO_DATA_FOUND THEN
12  DBMS_OUTPUT.PUT_LINE('Employee no does not exists');
13  END;
```

14 /

Output:

Enter value for empno: 7844

old 6: WHERE EMPNO = &EMPNO;

new 6: WHERE EMPNO = 7844;

PL/SQL procedure successfully completed.

### **Explicit Cursors:**

```
SQL> DECLARE
2   ena EMP.ENAME%TYPE;
3   esa EMP.SAL%TYPE;
4   CURSOR c1 IS SELECT ename,sal FROM EMP;
5   BEGIN
6   OPEN c1;
7   FETCH c1 INTO ena,esa;
8   DBMS_OUTPUT.PUT_LINE(ena || ' salry is $ ' || esa);
9
10  FETCH c1 INTO ena,esa;
11  DBMS_OUTPUT.PUT_LINE(ena || ' salry is $ ' || esa);
12
13  FETCH c1 INTO ena,esa;
14  DBMS_OUTPUT.PUT_LINE(ena || ' salry is $ ' || esa);
15  CLOSE c1;
16  END;
17 /
```

Output:

SMITH salry is \$ 800

ALLEN salry is \$ 1600

WARD salry is \$ 1250

### **RESULT:**

Thus the PL/SQL block to display the student name,marks,average is verified and executed.

## **EX: NO: 5C**

## **FUNCTIONS**

### **AIM**

To write a Functional procedure to search an address from the given database.

### **PROCEDURE**

STEP 1: Start

STEP 2: Create the table with essential attributes.

STEP 3: Initialize the Function to carryout the searching procedure..

STEP 4: Frame the searching procedure for both positive and negative searching.

STEP 5: Execute the Function for both positive and negative result .

STEP 6: Stop

### **EXECUTION**

#### **SETTING SERVEROUTPUT ON:**

SQL> SET SERVEROUTPUT ON

#### **IMPLEMENTATION OF FACTORIAL USING FUNCTION**

##### **I) PROGRAM:**

```
SQL>create function fnfact(n number)
    return number is
    b number;
    begin
    b:=1;
    for i in 1..n
    loop
    b:=b*i;
    end loop;
    return b;
    end;
/
```

```
SQL>Declare
    n number:=&n;
    y number;
```

```

begin y:=fnfact(n);
dbms_output.put_line(y);
end;
/

```

Function created.

```

Enter value for n: 5
old 2: n number:=&n;
new 2: n number:=5;
120

```

PL/SQL procedure successfully completed.

## **II)PROGRAM**

```

SQL> create table phonebook (phone_no number (6) primary key,username
varchar2(30),doorno varchar2(10),
street varchar2(30),place varchar2(30),pincode char(6));

```

Table created.

```

SQL> insert into phonebook values(20312,'vijay','120/5D','bharathi street','NGO
colony','629002');

```

1 row created.

```

SQL> insert into phonebook values(29467,'vasanth','39D4','RK bhavan','sarakkal vilai','629002');

```

1 row created.

```

SQL> select * from phonebook;

```

<b>PHONE_NO</b>	<b>USERNAME</b>	<b>DOORNO</b>	<b>STREET</b>	<b>PLACE</b>	<b>PINCODE</b>
20312	vijay	120/5D	bharathi street	NGO colony	629002
29467	vasanth	39D4	RK bhavan	sarakkal vilai	629002

```

SQL> create or replace function findAddress(phone in number) return varchar2 as
address varchar2(100);

```

```

begin
select username||','||doorno ||','||street ||','||place||','||pincode into address from phonebook
where phone_no=phone;
return address;
exception
when no_data_found then return 'address not found';
end;
/

```

Function created.

```

SQL>declare
2 address varchar2(100);
3 begin
4 address:=findaddress(20312);
5 dbms_output.put_line(address);
6 end;
7 /

```

**OUTPUT 1:**

**Vijay,120/5D,bharathi street,NGO colony,629002**

**PL/SQL procedure successfully completed.**

```

SQL> declare
2 address varchar2(100);
3 begin
4 address:=findaddress(23556);
5 dbms_output.put_line(address);
6 end;
7 /

```

**OUTPUT2:**

**Address not found**

**PL/SQL procedure successfully completed.**

**RESULT:**Thus the Function for searching process has been executed successfully.