**EXP.No. 1. DDL COMMANDS**

**AIM:**

To create a database, table

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**

1

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),**

**DESIGNATIN 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)**

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

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

2

**QUERY: 03**

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

**Table created.**

**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 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**

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

3

**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;**

**Name Null? Type**

**----------------------------------------- -------- ------------------------**

**EMPNO NUMBER(4)**

**ENAME VARCHAR2(10)**

**DESIGNATIN 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)**

**DESIGNATIN VARCHAR2(10)**

**SALARY NUMBER(8,2)**

4

**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)**

**DESIGNATIN 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**

**----------------------------------------- -------- ----------------------------**

5

**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 for add a new column:**

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

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

6

**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**

Q11. Write a query to drop multiple columns from employee **Syntax for add a new column:**

**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)**

7

**RENAME**

**QUERY: 12**

Q12. Write a query to rename table emp to employee

**Syntax for add a new column:**

**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)**

**DESIGNATIN VARCHAR2(10)**

**SALARY NUMBER(8,2)**

**RESULT**

DDL Commands had been successfully executed.

8

**EXP.No. 2. DML COMMANDS**

**AIM:**

To work with DML Commands

**DML COMMANDS**

DML commands are the most frequently used SQL commands and is used to query and manipulate the existing database objects. Some of the commands are Insert, Select, Update, Delete.

**Insert Command:** This is used to add one or more rows to a table. The values are separated by commas and the data types char and date are enclosed in apostrophes. The values must be entered in the same order as they are defined.

**Select Command:** It is used to retrieve information from the table. It is generally referred to as querying the table. We can either display all columns in a table or only specify column from the table.

**Update Command:** It is used to alter the column values in a table. A single column may be updated or more than one column could be updated.

**Delete command:** After inserting row in a table we can also delete them if required. The delete command consists of a from clause followed by an optional where clause.

**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: Select 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 form the table .

9

**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 DESIGNATIN SALARY 

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

101 NAGARAJAN LECTURER 15000

**INSERT A RECORD USING SUBSITUTION 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

10

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

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')

11

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 CHINNIHOD, 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>;

12

**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

13

**EXP. No.3 Basic SELECT statements**

**AIM:**

To Implement Basic Select statements

Q1: Create a pseudo table employee with the same structure as the table emp and insert rows into the table using select clauses.

Ans: SQL> create table employee as select \* from emp;

Table created.

SQL> desc employee;

Name Null? Type

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

EMPNO NUMBER(6)

ENAME NOT NULL VARCHAR2(20)

JOB NOT NULL VARCHAR2(13)

DEPTNO NUMBER(3)

SAL NUMBER(7,2)

Q2: select employee name, job from the emp table

Ans: SQL> select ename, job from emp;

ENAME JOB

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

Mathi AP

Arjun ASP

Gugan ASP

Karthik Prof

Akalya AP

suresh lect

6 rows selected.

Q3: List the records in the emp table orderby salary in ascending order.

Ans: SQL> select \* from emp order by sal;

EMPNO ENAME JOB DEPTNO SAL

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

1 Mathi AP 1 10000

5 Akalya AP 1 10000

14

2 Arjun ASP 2 15000

3 Gugan ASP 1 15000

4 Karthik Prof 2 30000

Q4: List the records in the emp table orderby salary in descending order. Ans: SQL> select \* from emp order by sal desc;

EMPNO ENAME JOB DEPTNO SAL

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

4 Karthik Prof 2 30000

2 Arjun ASP 2 15000

3 Gugan ASP 1 15000

1 Mathi AP 1 10000

5 Akalya AP 1 10000

Q5: Display only those employees whose deptno is 30.

Solution: Use SELECT FROM WHERE syntax.

Ans: SQL> select \* from emp where deptno=1;

EMPNO ENAME JOB DEPTNO SAL

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

1 Mathi AP 1 10000

3 Gugan ASP 1 15000

5 Akalya AP 1 10000

Q6: Display deptno from the table employee avoiding the duplicated values. Solution:

1. Use SELECT FROM syntax.

2.Select should include distinct clause for the deptno.

Ans: SQL> select distinct deptno from emp; DEPTNO

----------

1

2

15

**EXP. No.4 Advanced SELECT statements AIM:**

To Implement Advanced select Statements

COMPARISON OPERATORS:

SQL provides the following comparison operators

|  |  |
| --- | --- |
| Symbol | Meaning |
| **=** | Equal to |
| **<** | Less than |
| **<=** | Less than or equal to |
| **>** | Greater than |
| **>=** | Greater than or equal to |
| **< >** or **! =** | Not equal to |

ARITHMETIC OPERATORS:

SQL provides the following arithmetic operators. We can use arithmetic operators with table attributes in a column list or in a conditional expression.

|  |  |
| --- | --- |
| Arithmetic  operator | Description |
| **+** | Add |
| **-** | Subtract |
| **\*** | Multiply |
| **/** | Divide |
| **^** | Raise to the power of (some application use \*\* instead of ^) |

16

LOGICAL OPERATORS:

SQL allows having multiple conditions in a query through the use of logical operators. The logical operators are: **AND, OR, NOT** .The logical operators are used to connect the Boolean expressions in the where clause.

SPECIAL OPERATORS:

ANSI –standard SQL allows the use of special operators in conjunction with the WHERE clause. These special operators include:

BETWEEN: used to check whether an attribute value is within a range.

IS NULL: used to check whether an attribute value is null.

IN: used to check whether an attribute value matches any value within a value list. EXISTS: used to check whether a sub query returns any rows.

LIKE: used to check whether an attribute value matches a given string pattern. The LIKE special operator is used in conjunction with wildcards to find patterns

within string attributes. Standard SQL allows to use the percent sign (**%**) and underscore ( \_ **)** wildcard characters to make matches when the entire string is not known:

∙ % Means any and all following or preceding characters are eligible. For example „J%‟ includes Johnson, James, and July. „Jo %‟ includes Johnson, Jones. „%n „includes Johnson, Jagan, Kiran.

∙ \_ means any one character may be substituted for the underscore

Frequency distribution can be created quickly and easily using the GROUP BY clause within the SELECT statement. The syntax is:

SELECT *column list*

FROM *table list*

[WHERE *condition list*]

[GROUP BY *column list*]

[HAVING *condition list*]

[ORDER BY *column list* [ASC|DESC]];

17

The GROUP BY clause is generally used when you have attributes columns combined with aggregate functions in the SELECT statement.

The GROUP BY clause is valid only when used in conjunction with one of the SQL aggregate functions such as COUNT, MIN, MAX, AVG and SUM.

EXAMPLE-SQL OPERATORS

CONSIDER THE FOLLOWING EMPLOYEE TABLE AND WRITE THE QUERIES AND CORRESPONDING RESULT FOR EACH QUERY.

**EMPLOYEE**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **EMP\_NUM EMP\_NAME** | **JOB\_TYPE** | **HIREDATE** | **EMP\_SAL** | **EMP\_LOC** |
| 1001 | KRISHNA | MANAGER | 01-JAN-2011 | 50000 | NANDYAL |
| 1002 |  | CLERK | 01-JAN-2011 | 15000 | KURNOOL |
| 1003 | RAM | CLERK | 16-AUG-2011 | 15000 | NANDYAL |
| 1004 | MAHESH | ASSTMANAGER | 03-MAY-2011 | 35000 | HYDERABAD |
| 1005 | VIGNESH | ACCOUNTANT | 01-JAN-2011 | 20000 | KADAPA |
| 1006 | NAGENDRA | MECHANIC | 01-JAN-2011 | 10000 | KADAPA |
| 1007 | KIRAN | CLERK | 08-JUN-2011 | 15000 | HYDERABAD |
| 1008 | LOKESH | ATTENDER | 01-JAN-2011 | 12000 | KURNOOL |
| 1009 | MOHAN | ATTENDER | 16-AUG-2011 | 12000 | KADAPA |
| 1010 | PRAVEEN | ADMINOFFICER | 02-JAN-2011 | 40000 | NANDYAL |

**1. Find the employee whose salary is greater than 25000**

SQL> select \*from employee where emp\_sal>

25000;

EMP\_NUM EMP\_NAME JOB\_TYPE HIREDATE EMP\_SAL EMP\_LOC --------------- ------------------- ------------ --------- ---------- --------------------------------------- 1001 krishna manager 01-JAN-11 50000 nandyal 1004 mahesh asstmanager 03-MAY-11 35000 hyderabad 1010 praveen adminofficer 02-JAN-11 40000 nandyal

18

**2. Find the name of employees whose salary is 50000**

SQL> select emp\_name from employee where emp\_sal=50000;

EMP\_NAME

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

Krishna

**3. Find the employee who are either manager or belongs to nandyal** SQL> select \* from employee where job\_type='manager' or emp\_loc='nandyal';

**4. Find the employee who is manager and belongs to nandyal**

SQL>select \* from employee where job\_type='manager' and emp\_loc='nandyal';

EMP\_NUM EMP\_NAME JOB\_TYPE HIREDATE EMP\_SAL EMP\_LOC

---------------- ------------------ ---------------- ------------------ ------------- -------------- 1001 krishna manager 01-JAN-11 50000 nandyal **5. Find the employee who are either manager or belongs to nandyal** SQL> select \* from employee where job\_type='manager' or emp\_loc='nandyal';

EMP\_NUM EMP\_NAME JOB\_TYPE HIREDATE EMP\_SAL EMP\_LOC ---------------- ------------------ ------------------ ----------------- ------------ ---------------- 1001 krishna manager 01-JAN-11 50000 nandyal 1003 ram clerk 16-AUG-11 15000 nandyal 1010 praveen adminofficer 02-JAN-11 40000 nandyal

**6. Find employee whose salary is not less than 20000.**

SQL> select \* from employee where not(emp\_ sal<20000);

(or)

SQL> select \* from employee where emp\_sal>=20000;

19

EMP\_NUM EMP\_NAME JOB\_TYPE HIREDATE EMP\_SAL EMP\_LOC ---------------- ------------------ ----------------- ----------------- ------------- ------------ 1001 krishna manager 01-JAN-11 50000 nandyal

1004 mahesh asstmanager 03-MAY-11 35000 hyderabad

1005 vignesh accountant 01-JAN-11 20000 kadapa

1010 praveen adminofficer 02-JAN-11 40000 nandyal

**7. Find the name and location of employee who location is nandyal or Kurnool** SQL> select \* from employee where emp\_loc in('nandyal','kurnool');

(or)

SQL> select \* from employee where emp\_loc='nandyal' or emp\_loc='kurnool';

EMP\_NUM EMP\_NAME JOB\_TYPE HIREDATE EMP\_SAL EMP\_LOC --------------- ------------------- ----------------- ---------------- ------------- ------------ 1001 krishna manager 01-JAN-11 50000 nandyal

1002 clerk 01-JAN-11 15000 kurnool

1003 ram clerk 16-AUG-11 15000 nandyal

1008 lokesh attender 01-JAN-11 12000 kurnool

1010 praveen adminofficer 02-JAN-11 40000 nandyal

**8. Find the rows from employee table whose EMP\_NAME column values are null**

SQL> select \* from employee where emp\_name is null;

EMP\_NUM EMP\_NAME JOB\_TYPE HIREDATE EMP\_SAL EMP\_LOC

------------------ ------------------- ---------------- --------------- -------------- --------------- 1002 clerk 01-JAN-11 15000 kurnool

9. **Find the name of employees whose name starts with ‘k’**

SQL> select emp\_name from employee where emp\_name like 'k%';

20

EMP\_NAME

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

krishna

kiran

**10. Find the name of employees whose name starts with ‘k’ .** SQL> select emp\_name from employee where emp\_name like 'k%';

EMP\_NAME

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

krishna

kiran

**11. Find the name of employees in which the second character is ‘a’ .** SQL> select emp\_name from employee where emp\_name like '\_a%';

EMP\_NAME

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

ram

mahesh

nagendra

**12. Find number of employees.**

SQL> select count(\*) from employee;

COUNT(\*)

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

10

**13. Find minimum salary of the employee**

SQL> select min(emp\_sal)from employee;

MIN(EMP\_SAL)

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

10000

**14. Find maximum salary of the employee**

SQL> select max(emp\_sal) from employee;

MAX(EMP\_SAL)

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

50000

15. **Find the average salary of the employees**

SQL> select avg(emp\_sal) from employee;

AVG(EMP\_SAL)

21

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

22400

**16. .Find the number of employees for each job**

SQL> select job\_type,count(emp\_name) from employee group by job\_type;

JOB\_TYPE COUNT(EMP\_NAME)

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

attender 2

asstmanager 1

clerk 2

accountant 1

adminofficer 1

manager 1

mechanic 1

**17. What is the maximum salary for each job**

SQL> select job\_type, max(emp\_sal) from employee group by job\_type;

JOB\_TYPE MAX(EMP\_SAL)

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

attender 12000

asstmanager 35000

clerk 15000

accountant 20000

adminofficer 40000

manager 50000

mechanic 10000

**18. Find the number of each job and name the column that gives number of jobs as jobnum**

SQL> select job\_type, count(job\_type) as jobnum from employee group by job\_type; JOB\_TYPE JOBNUM

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

attender 2

asstmanager 1

clerk 3

accountant 1

adminofficer 1

manager 1

mechanic 1

22

**19. Find the name and salary of the employee with maximum salary SQL> select emp\_name, emp\_sal from employee where emp\_sal=(select max(emp\_sal) from employee);**

EMP\_NAME EMP\_SAL

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

krishna50000

**20. Find the count of employees for each job so that at least two of the employees had salary greater than 10000**

SQL> select job\_type ,count(emp\_num) 2 from employee where emp\_sal>10000 group by job\_type having count(emp\_num)>=2

JOB\_TYPE COUNT(EMP\_NUM)

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

attender 2

clerk 3

**RESULT**

A study on Advanced Select statements had done successfully.

23

**EXP. No.4** Integrity and Constraints

**AIM:**

To Perform Integrity and 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**:

1. ENABLE

2. DISABLE

3. DROP

**Column level constraints Using Primary key**

Q1. Write a query to create primary constraints with column level

**Primary key**

**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)

……………………………);

**QUERY:1**

24

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**

Q2. 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:2**

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**

Q3. 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>);

25

**QUERY: 3**

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):**

Q4. 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: 4**

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.5. Write a query to create foreign key constraints with column level **Parent Table:**

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

26

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: 5**

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.6. 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:**

27

SQL:>CREATE <OBJ.TYPE> <OBJ.NAME> (COLUMN NAME.1 <DATATYPE> (SIZE) , COLUMN NAME2 <DATATYPE> (SIZE) **CONSTRAINT <CONST. NAME>** REFERENCES <TABLE NAME> (COLUMN NAME>

……………………………);

**QUERY:6**

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.7. 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: 7**

28

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.8. 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:8**

SQL>CREATE TABLE DEPT

(DEPTNO NUMBER(2) PRIMARY KEY,

DNAME VARCHAR2(20),

LOCATION VARCHAR2(15));

29

SQL>CREATE TABLE EMP5

(EMPNO NUMBER(3),

DEPTNO NUMBER(2),

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

**Check constraint**

**Column Level Check Constraint**

Q.9 Write a query to create Check constraints with column level

**Syntax for clumn 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));

**QUERY:9**

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.10. Write a query to create Check constraints with table level

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

30

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

**QUERY:10**

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.11. 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> (CONSTRAITNS CRITERIA)) ; 

**QUERY:11**

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);

31

**Unique Constraint**

**Column Level Constraint**

Q.11. 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:11**

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.12. 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:12**

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.13. 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);

32

**QUERY:14**

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);

**Not Null**

**Column Level Constraint**

Q.15. Write a query to create Not Null constraints with column level **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)) ;

**QUERY: 15**

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**

33

Q.16. 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:16**

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.17. 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:17**

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

Q.18. 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>

34

**QUERY:18**

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

**RESULT**

Constraints and Integrity had been implemented successfully.

35

**EXP. No.6** Joining Tables

**SQL JOIN OPERATIONS**

The relational join operation merges rows from two tables and returns the rows with one of the following conditions:

• Have common values in common columns(natural join)

• Meet a given join condition(equality or inequality)

• Have a common value in common columns or have no matching values(outer join) SQL JOIN EXPRESSIN STYLES

|  |  |  |  |
| --- | --- | --- | --- |
| JOIN  CLASSIFICATION | JOIN TYPE | SQL SYNTAX EXAMPLE | DESCRIPTION |
| CROSS | CROSS  JOIN | SELECT \*  FROM T1,T2; (old style)  SELECT \*  FROM T1 CROSS JOIN T2; | Returns the Cartesian product of T1 and T2. |
| INNER | Old-style  join | SELECT \*  FROM T1,T2  WHERE T1.C1=T2.C1; | Returns only the rows that meet the join condition in the where clause (old style).Only the rows with matching values areselected. |
|  | NATURAL JOIN | SELECT \*  FROM T1 NATURAL JOIN T2; | Returns only the rows with matching values in thematching columns. Thematching columns must have the same names and similar data types. |

36

JOIN SELECT \* Returns only the rows with USING FROM T1 JOIN T2 USING( C1) matching values in the columns indicated in the USING clause

|  |  |  |  |
| --- | --- | --- | --- |
|  | JOIN ON | SELECT \*  FROM T1 JOIN T2  ON T1.C1=T2.C1; | Returns only the rows with matching values in the columns indicated in the ON clause |
| OUTER | LEFT JOIN | SELECT \*  FROM T1 LEFT OUTER JOIN T2 values and includes all rowsON T1.C1=T2.C1; | Returns rows with matching from the left table (T1) with unmatched values. |
|  | RIGHT  JOIN | SELECT \*  FROM T1 RIGHT OUTER JOIN T2 ON T1.C1=T2.C1; | Returns rows with matchingvalues and includes all rows from the right table (T2) with unmatched values. |
|  | FULL JOIN | SELECT \*  FROM T1 FULL OUTER JOIN T2 ON T1.C1=T2.C1; | Returns rows with matchingvalues and includes all rows from both tables (T1and T2) with unmatched values. |

Consider the tables CUSTOMER and AGENT; perform different SQL join operations and write corresponding results

37

CUSTOMER

|  |  |  |  |
| --- | --- | --- | --- |
| CUS\_CODE | CUS\_NAME | CUS\_ZIP | AGENT\_CODE |
| 1132445 | W | 145 | 231 |
| 1217782 | A | 145 | 125 |
| 1312243 | Ra | 129 | 167 |
| 1321242 | Ro | 134 | 125 |
| 1542311 | S | 134 | 421 |
| 1657399 | V | 145 | 231 |

|  |  |
| --- | --- |
| AGENT\_CODE | AGENT\_PHONE |
| 125 | 9985707288 |
| 167 | 8985291308 |
| 231 | 9885434311 |
| 333 | 9704128379 |

**CROSS JOIN:**

SQL> select \* (old style)

2 from customer,agent;

CUS\_CODE CUS\_NAME CUS\_ZIP AGENT\_CODE AGENT\_CODE AGENT\_PHONE -------------- ----------------- -------------------------------- -------------------- --------------------- 1132445 W 145 231 125 9985707288 1217782 A 145 125 125 9985707288 1312243 Ra 129 167 125 9985707288 1321242 Ro 134 125 125 9985707288 1542311 S 134 421 125 9985707288 1657399 V 145 231 125 9985707288 1132445 W 145 231 167 8985291308 1217782 A 145 125 167 8985291308 1312243 Ra 129 167 167 8985291308 1321242 Ro 134 125 167 8985291308 1542311 S 134 421 167 8985291308

CUS\_CODE CUS\_NAME CUS\_ZIP AGENT\_CODE AGENT\_CODE AGENT\_PHONE -------------- ----------------- -------------------------------- ------------------- ---------------------

38

1657399 V 145 231 167 8985291308 1132445 W 145 231 231 9885434311 1217782 A 145 125 231 9885434311 1312243 Ra 129 167 231 9885434311 1321242 Ro 134 125 231 9885434311 1542311 S 134 421 231 9885434311 1657399 V 145 231 231 9885434311 1132445 W 145 231 333 9704128379 1217782 A 145 125 333 9704128379 1312243 Ra 129 167 333 9704128379 1321242 Ro 134 125 333 9704128379 CUS\_CODE CUS\_NAME CUS\_ZIP AGENT\_CODE AGENT\_CODE AGENT\_PHONE ------------ ----------------- ------------- ------------------- ------------------- ----------------------

1542311 S 134 421 333 9704128379 1657399 V 145 231 333 9704128379

24 rows selected.

SQL> select \* from customer cross join agent;

This query also results the values same as the above query which is the old style **INNER JOINS**

**Old-style join :**

SQL> select \* from customer, agent where customer.agent\_code=agent.agent\_code;

CUS\_CODE CUS\_NAME CUS\_ZIP AGENT\_CODE AGENT\_CODE AGENT\_PHONE -------------- --------------- ----------- ------------------- ------------------ ------------------- 1132445 w 145 231 231 9885434311 1217782 a 145 125 125 9985707288 1312243 ra 129 167 167 8985291308

39

1321242 ro 134 125 125 9985707288 1657399 v 145 231 231 9885434311

NATURAL JOIN:

SQL> select \*

2 from customer natural join agent;

AGENT\_CODE CUS\_CODE CUS\_NAME CUS\_ZIP AGENT\_PHONE ----------------- --------------- --------------- ------------ -------------------- 231 1132445 W 145 9885434311

125 1217782 A 145 9985707288

167 1312243 Ra 129 8985291308

125 1321242 Ro 134 9985707288

231 1657399 V 145 9885434311

JOIN USING

SQL> select \*

2 from customer join agent using(agent\_code);

AGENT\_CODE CUS\_CODE CUS\_NAME CUS\_ZIP AGENT\_PHONE ----------------- --------------- --------------- ------------ -------------------- 231 1132445 W 145 9885434311

125 1217782 A 145 9985707288

167 1312243 Ra 129 8985291308

125 1321242 Ro 134 9985707288

231 1657399 V 145 9885434311

FULL OUTER JOIN

SQL> select \*

2 from customer full outer join agent on customer.agent\_code=agent.agent\_code;

40

CUS\_CODE CUS\_NAME CUS\_ZIP AGENT\_CODE AGENT\_CODE AGENT\_PHONE ---------------- ------------------- ------------ ------------------ ------------------- ----------------------- 1321242 Ro 134 125 125 9985707288 1217782 A 145 125 125 9985707288 1312243 Ra 129 167 167 8985291308 1657399 V 145 231 231 9885434311 1132445 W 145 231 231 9885434311 1542311 S 134 421

333 9704128379

7 rows selected.

**RESULT**

The SQL Join queries had been implemented successfully.

41

**EXP. No.7** SQL Functions

**CHARACTER/STRING FUNCTION:**

SQL> select upper('welcome') from dual; -----------

WELCOME

SQL> select upper('hai') from dual;

---

HAI

SQL> select lower('HAI') from dual;

LOW

---

hai

SQL> select initcap(„hello world') from dual; INITCAP('Hello

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

Hello World

SQL> select ltrim(' hai') from dual;

LTR

---

hai

SQL> select rtrim('hai ')from dual;

42

RTR

---

hai

SQL> select rtrim(' hai ')from dual;

RTRIM('

------

hai

SQL> select concat('SRM',' university')from dual; ------------------------

SRM university

SQL> select length('SRM‟)from dual;

LENGTH('SRM')

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

12

SQL> select replace('SRM university', 'SRM','Anna')from dual; ----------------

Anna university

SQL> select substr('SRM', 7,6)from dual;

SUBSTR

------

lingam

43

SQL> select rpad('hai',3,'\*')from dual; RPAD('

------

hai\*\*\*

SQL> select lpad('hai',3,'\*')from dual; LPAD('

------

\*\*\*hai

SQL> select replace ('Dany','y','ie') from dual; REPLACE

-------

Danie

SQL> select translate ('cold','ld','ol') from dual; TRANSL

------

cool

**DATE & TIME FUNCTION**

SQL> select sysdate from dual;

SYSDATE

---------

07-APR-10

SQL> select round(sysdate)from dual;

44

ROUND (SYS

---------

07-APR-10

SQL> select add\_months(sysdate,3)from dual; ADD\_MONTH

---------

07-JUL-10

SQL> select last\_day(sysdate)from dual; LAST\_DAY (

---------

30-APR-10

SQL> select sysdate+20 from dual;

SYSDATE+2

---------

27-APR-10

SQL> select next\_day(sysdate,'tuesday')from dual;

NEXT\_DAY(

---------

13-APR-10

45

**NUMERIC FUNCTION**

SQL> select round(15.6789)from dual; ROUND(15.6789)

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

16

SQL> select ceil(23.20)from dual; CEIL(23.20)

-----------

24

SQL> select floor(34.56)from dual; FLOOR(34.56)

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

34

SQL> select trunc(15.56743)from dual; TRUNC(15.56743)

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

15

SQL> select sign(-345)from dual; SIGN(-345)

----------

-1

SQL> select abs(-70)from dual; ABS(-70)

---------

70

46

**MATH FUNCTION:**

SQL> select abs(45) from dual;

ABS(45)

---------

45

SQL> select power(10,12) from dual;

POWER(10,12)

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

1.000E+12

SQL> select mod(11,5) from dual;

MOD(11,5)

---------

1

SQL> select exp(10) from dual;

EXP(10)

---------

22026.466

SQL> select sqrt(225) from dual;

SQRT(225)

---------

15

47

**Exp No: 8 SubQueries**

**AIM**

To execute and verify the SQL commands for SubQueries.

**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

S**QL 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 element 3.COMMAND NAME: **HAVING**

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

3. COMMAND NAME: **MIN (SAL)**

COMMAND DESCRIPTION: **MIN (SAL)** command is used to find minimum salary. **Table -1**

**SYNTAX FOR CREATING A TABLE:**

48

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:**

**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);

49

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.

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**

50

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

Table created.

**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

51

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

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 FRORM <TABLE \_1> WHERE <COLUMN\_NAME> <RELATIONAL \_OPERATION> „VALUE‟

(SELECT (AGGRECATE FUNCTION) FROM <TABLE\_1> WHERE <COLUMN NAME> = „VALUE‟

(SELECT <COLUMN\_NAME> FROM <TABLE\_2> WHERE

<COLUMN\_NAME= „VALUE‟));

**NESTED QUERY STATEMENT:**

52

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

**RESULT**

The SubQueries had been implemented successfully.

53

**EXP NO. 9 VIEWS**

**AIM**

To execute and verify the SQL commands for Views.

**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

**COMMANDS EXECUTION**

**CREATION OF TABLE**

**--------------------------------**

SQL> CREATE TABLE EMPLOYEE (

EMPLOYEE\_NAMEVARCHAR2(10),

EMPLOYEE\_NONUMBER(8),

DEPT\_NAME VARCHAR2(10),

DEPT\_NO NUMBER (5),DATE\_OF\_JOIN DATE);

54

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

55

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

56

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\_NMAE>WHERE <COLUMN NMAE> =‟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:**

57

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 A 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;

58

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 abject”.

**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 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

59

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

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

60

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;

**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;

61

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 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.

**FORCE VIEW**

62

**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.

63

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.

64

**Exp No: 10 Basics of PL/SQL**

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

**Addition of Two Numbers:**

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;

65

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>;

66

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 ELSECONDITION:** SQL> DECLARE

<VARIABLE DECLARATION>;

BEGIN

IF (TEST CONDITION) THEN

<STATEMENTS>;

67

ELSE

<STATEMENTS>;

ENDIF;

END;

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 LOOPING STATEMENT:**

SQL> DECLARE

68

<VARIABLE DECLARATION>;

BEGIN

LOOP

<STATEMENT>;

END LOOP;

<EXECUTAVLE STATEMENT>; END;

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:**

69

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;

<EXECUTAVLE 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;

70

/

**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.

71

**Exp No.11 Design and Develop applications**

**AIM**

To design a form using different tools in Visual Basic.

**PROCEDURE**

STEP 1: Start

STEP 2: Create the form with essential controls in tool box. STEP 3: Write the code for doing the appropriate functions. STEP 4: Save the forms and project.

STEP 5: Execute the form .

STEP 6: Stop

**CODING:**

Private Sub Calendar1\_Click()

Text3.Text = Calendar1.Value

End Sub

Private Sub Combo1\_Change()

Combo1.AddItem "BSC"

Combo1.AddItem "MSC"

Combo1.AddItem "BE"

Combo1.AddItem "ME"

End Sub

Private Sub Command1\_Click()

List1.AddItem Text1.Text

72

List1.AddItem Text2.Text

If Option1.Value = True Then

gender = "male"

End If

If Option2.Value = True Then

gender = "female"

End If

List1.AddItem gender

List1.AddItem Text3.Text

If Check1.Value = 1 And Check2.Value = 1 Then area = "software Engineering & Networks" End If

If Check1.Value = 0 And Check2.Value = 1 Then area = " Networks"

End If

List1.AddItem area

List1.AddItem Text4.Text

End Sub

Private Sub Command2\_Click()

End

End Sub

Private Sub Command3\_Click()

If List1.ListIndex <> 0 Then

List1.RemoveItem (0)

End If

End Sub

73

Private Sub Form\_Load()

Label10.Caption = Date$

MsgBox "Welcome to Registration" End Sub

Private Sub Option1\_Click()

If (Option1.Value = True) Then MsgBox ("You have selected Male") ElseIf (Option2.Value = True) Then MsgBox ("You have selected Female") End If

End Sub

Private Sub Option2\_Click()

If (Option1.Value = True) Then MsgBox ("You have selected Male") ElseIf (Option2.Value = True) Then MsgBox ("You have selected Female") End If

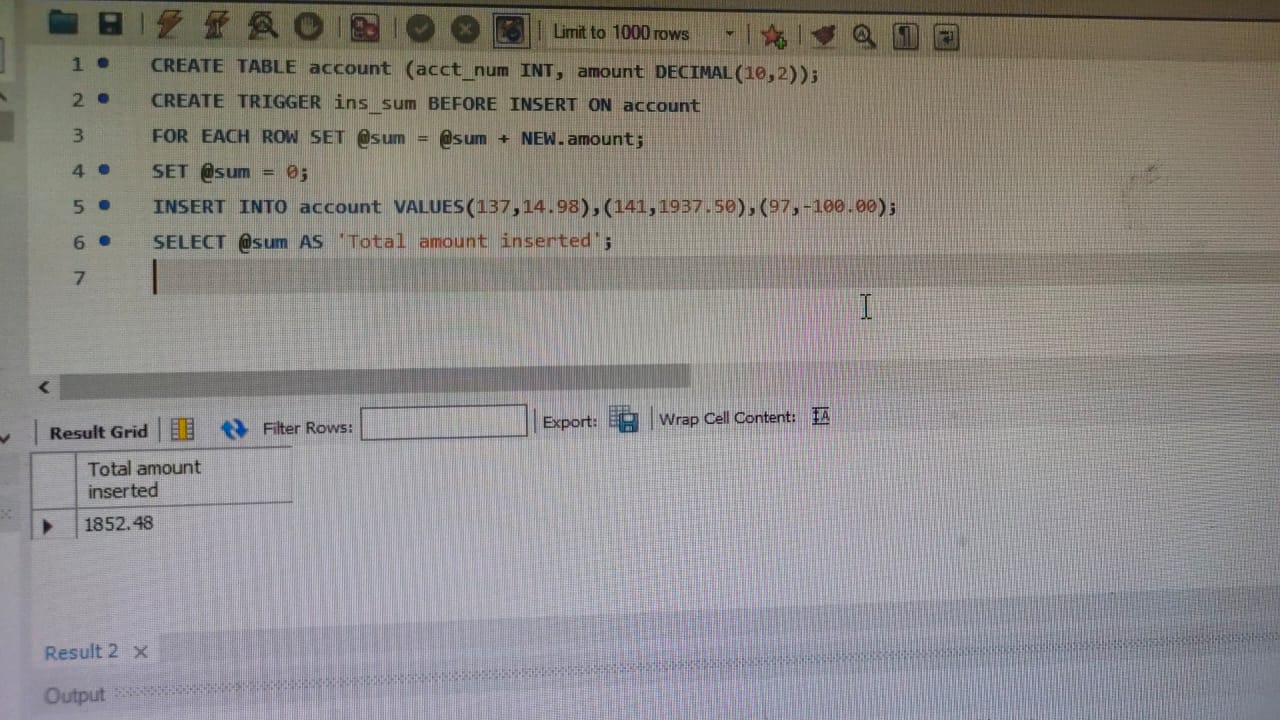
End Sub

**REGISTRATION FORM:**

74

**RESULT**: Thus the program has been loaded and executed successfully.

75



**Procedure:**

create table film(rating int ,name varchar(20),release\_date int);

insert into film values(4,'tomandJerry',90);

insert into film values(5,'harrypotter',21);

insert into film values(2,'jamesBond',85);

insert into film values(3,'jumanji',22);

select \* from film;

DELIMITER //

CREATE PROCEDURE sp\_GetMovies()

BEGIN

select rating,name,release\_date from film;

END //

DELIMITER ;

CALL sp\_GetMovies();

output



**Cursor**

CREATE TABLE GetVatsaCursor(

C\_ID INT PRIMARY KEY AUTO\_INCREMENT,

c\_name VARCHAR(50),

c\_address VARCHAR(200)

);

CREATE TABLE Vbackupdata(

C\_ID INT,

c\_name VARCHAR(50),

c\_address VARCHAR(200)

);

INSERT INTO GetVatsaCursor(c\_name, c\_address) VALUES('Test', '132, Vatsa Colony'),

('Admin', '133, Vatsa Colony'),

('Vatsa', '134, Vatsa Colony'),

('Onkar', '135, Vatsa Colony'),

('Rohit', '136, Vatsa Colony'),

('Simran', '137, Vatsa Colony'),

('Jashmin', '138, Vatsa Colony'),

('Anamika', '139, Vatsa Colony'),

('Radhika', '140, Vatsa Colony');

SELECT \* FROM GetVatsaCursor;

SELECT \* FROM Vbackupdata;

delimiter //

CREATE PROCEDURE firstCurs()

BEGIN

DECLARE d INT DEFAULT 0;

DECLARE c\_id INT;

DECLARE c\_name, c\_address VARCHAR(20);

DECLARE Get\_cur CURSOR FOR SELECT \* FROM GetVatsaCursor;

DECLARE CONTINUE HANDLER FOR SQLSTATE '02000'

SET d = 1;

DECLARE CONTINUE HANDLER FOR SQLSTATE '23000'

SET d = 1;

OPEN Get\_cur;

lbl: LOOP

IF d = 1 THEN

LEAVE lbl;

END IF;

IF NOT d = 1 THEN

FETCH Get\_cur INTO c\_id, c\_name, c\_address;

INSERT INTO Vbackupdata VALUES(c\_id, c\_name, c\_address);

END IF;

END LOOP;

CLOSE Get\_cur;

END;

//

CALL firstCurs();

SELECT \* FROM Vbackupdata;

output:-

