SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be **University** *u/s 3 of UGC Act, 1956*

FACULTY OF SCIENCE AND HUMANITIES

DEPARTMENT OF COMPUTER SCIENCE AND APPLICATIONS

VADAPALANI CAMPUS

LAB RECORD

NAME	:	**MOHAMED FIRNAS M A**
REGISTER NUMBER	:	**RA2232241040034**
CLASS	:	**I Year MCA**
DEPARTMENT	:	**Computer Science and Applications**
SUBJECT CODE & NAME	:	**PCA20C03J – Database Technology Lab**

APRIL-2023

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be **University** *u/s 3 of UGC Act, 1956*

FACULTY OF SCIENCE AND HUMANITIES

DEPARTMENT OF COMPUTER SCIENCE AND APPLICATIONS

VADAPALANI CAMPUS

CERTIFICATE

This is to certify that this is a bonafide record of practical work done by

NAME	**:**	**MOHAMED FIRNAS M A**
REGISTER NUMBER	**:**	**RA2232241040034**
CLASS	**:**	**I MCA**
DEPARTMENT	**:**	**Computer Science and Applications**
SEMESTER, MONTH & YEAR	**:**	**I Semester APR - 2023**
SUBJECT CODE & NAME	**:**	**PCA20C03J – Database Technology Lab**

During the Academic Year 2022 -2024 & Submitted for MCA practical examination held on _____.

LECTURER INCHARGE **H.O.D**

INTERNAL EXAMINER **EXTERNAL EXAMINER**

INDEX

Ex.No :
Date :

DATA DEFINITION LANGUAGE (DDL) STATEMENTS

AIM:

To perform Data Definition Language Commands in RDBMS by using structured query language.

DDL Statements:

1. CREATE TABLE – to create objects in the database.
2. ALTER TABLE –to alter the structure of the database.
3. DROP TABLE – delete objects from the database.
4. TRUNCATE TABLE– removes all records from a table.
5. DESC COMMAND – display the structure of the table.

CREATE TABLE:

- This command is used to create a new table.
- A table is a unit of storage which holds data in the form of rows and columns.
- In a table,
  - ✓ A unique column name should be specified.
  - ✓ Proper data type along with its width should be specified.
  - ✓ 'Not null' condition can be included when needed, by default it is 'Null'.

Syntax: create table <table name> (column_name 1 datatype(size), column_name 2 data(size),…….column_name n datatype(size));

Example:

SQL> create table student(regno number(6), name varchar2(15),age number(3), gender varchar2(1), phone number(10));

Output: Table created.

ALTER TABLE:

- This command is used to add a new column, modify the existing column definition, include or drop integrity constraints.

ADD column

Syntax: alter table <table name> add (column_name datatype(size));

Example:

SQL> alter table student add(address varchar2(20));

Output: Table altered.

MODIFY column:

Syntax: alter table <table name> modify (column_name datatype (size));

Example:

SQL> alter table student modify(name varchar2(30));

Output: Table altered.

DESC COMMAND:

- This command will display the structure of the table.

Syntax: Desc <table name>;

Example:

SQL> desc student;

Output:

Name	Null?	Type
REGNO		NUMBER(6)
NAME		VARCHAR2(30)
AGE		NUMBER(3)
GENDER		VARCHAR2(1)
PHONE		NUMBER(10)
ADDRESS		VARCHAR2(20)

TRUNCATE TABLE:

- The truncate command deletes all rows from the table. Only the structure of the table remains.

Syntax: Truncate table <table name>;

Example:

SQL> truncate table student;

Output:

Table truncated.

DROP TABLE:

- The drop table command is used to delete the table permanently from the database.
- It is used to delete the structure of the table.

Syntax: Drop table <table name>;

Example:

SQL> drop table student;

Output:

Table dropped.

Result:

Thus the above DDL commands are performed and the required output has been obtained and verified.

Ex.No:`
Date :

INTEGRITY CONSTRAINTS

AIM:

To perform Data Definition Language Commands with constraints in RDBMS by using structured query language.

Integrity Constraints:

An integrity constraint is a mechanism used by Oracle to prevent invalid data entry into the table, by enforcing rule for the column in a table.

1. **Domain Integrity Constraints** – Maintains value according to the specifications.

✓ Not Null Constraint – When a not null constraint is enforced on a column or a set of columns in a table, it will not allow null values.

Example: create table student (regno number (11), name varchar2 (25) constraint nm Not Null, addr varchar2 (15), dept varchar2 (3));

Output:

Table created.

✓ Check Constraint – These are rules governed by logical expressions or Boolean expressions.

Example:

Enforcing a check constraint using column level syntax:

SQL> create table student (regno number (11) constraint checkit check (regno >= 4000 and regno <= 5000), name varchar2 (25), addr address, dept varchar2 (3));

Enforcing a check constraint using table level syntax:

SQL> create table student (regno number (11), name varchar2 (25), addr address, dept varchar2 (3), constraint checkit check (regno >= 42207621001 and regno <= 42207621060));

Output:

Table created.

- ✓ If the table is already created without specifying the check constraint, then use the alter table command to add a check constraint.

Example: alter table student add constraint checkit check (regno >= 42207621001 and regno <= 42207621060);

Output:

Table altered.

2. **Entity Integrity Constraints** – To identify each row in a table uniquely, this constraint is used.

- Unique Constraint – Usage of the unique key constraint is to prevent the duplication of values within the rows of a specified column or a set of columns in a table. Columns defined with this constraint can also allow Null values.

Example:

Enforcing a unique constraint using column level syntax:

SQL> create table rrr (regno number (11) constraint uregno unique, name varchar2(25), addr address, dept varchar2 (3));

Enforcing a unique constraint using table level syntax:

SQL> create table student (regno number (11), name varchar2 (25), addr address, dept varchar2 (3), constraint uregno unique (regno));

Output:

Table created.

- If the table is already created without specifying the unique constraint, then use the alter table command to add a unique constraint.

Example: alter table student add constraint uregno unique (regno);

Output:

Table altered.

- Primary Key Constraint – This constraint avoids duplication of rows and does not allow Null values, when enforced in a column or set of columns. A table can have only one primary key, and it is used to identify a row.

Example:

Enforcing a primary key constraint using column level syntax:

SQL> create table student (regno number (11) constraint pk primary key, name varchar2(25), addr address, dept varchar2 (3));

Enforcing a primary key constraint using table level syntax:

SQL> create table student (regno number (11), name varchar2 (25), addr address, dept varchar2 (3), constraint pk primary key (regno));

Output:

Table created.

- If the table is already created without specifying the primary key constraint, then use the alter table command to add a primary key constraint.

Example: alter table student add constraint pk primary key (regno);

Output:

Table altered.

3. **Referential Integrity Constraints** – To establish a 'parent–child' relationship between two tables having a common column, make use of referential integrity constraints.
   - To implement this constraint, define the column in the parent table as a primary key and the same column in the child table as a foreign key referring to the corresponding parent key.
   - On delete cascade clause: If all the rows under the referenced key column in a parent table are deleted, then all rows in the child table with dependent foreign key column will also be deleted automatically.

   Example:

   Parent Table:

   SQL> create table department (dcode varchar2 (3) constraint pkdc primary key, dname varchar2 (30), HODName varchar2 (30));

   Output:

   Table created.

   Child Table

   Enforcing a foreign key constraint using column level syntax:

SQL> create table student (regno number (11) constraint pk primary key, name varchar2(25), addr address, dept varchar2 (3) constraint fk_dept references department (dcode) on delete cascade);

Enforcing a foreign key constraint using table level syntax:

SQL> create table student (regno number (11) constraint pk primary key, name varchar2(25), addr address, dept varchar2 (3), constraint fk_dept foreign key (dept) references department (dcode) on delete cascade);

Output:

Table created.

- If the child table is already created without specifying the foreign key constraint, then use the alter table command to add a foreign key constraint.

SQL> alter table student add constraint fk_dept foreign key (dept) references department (dcode) on delete cascade;

Output:

Table altered.

Result:

Thus the above DDL commands using integrity constraints are performed and the required output has been obtained and verified.

Ex.No:`
Date :

DATA MANIPULATION LANGUAGE (DML)

AIM:

To perform Data Manipulation Language Commands in RDBMS by using structured query language.

1. INSERT COMMAND

- Used to add one or more rows to a table.
- While using this command 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 in the table.

Syntax:

- To insert all column values for each row
  SQL> Insert into <table_name> values (a list of data values);
- To insert specific column values for each row
  SQL> Insert into <table_name> (col_names) values (list of values);

Example:

SQL> Insert Into Account Values (&Account_No, '&Name', &Balance);

Output:

1 row created.

2. SELECT COMMAND

- Used to perform a query. The query is a request for information.

Syntax: Select column_name … from table_name … where conditions [order by column_name …];

- To select only specific columns, specify the column names instead of * in the select command.

Example:

SQL> select * from account;

Output:

ACCOUNT_NO	NAME	BALANCE
1021	Sarika	5000
101	Sarika	6000
102	Amar	500

3. UPDATE COMMAND

- Used to alter the column values in a table.
- Specific rows could be updated based on a specific condition.

Syntax: Update table_name set field = value, … where condition;

- The 'where' clause and the 'set' clause can also include queries.

Example:

```
SQL> update account set balance=1000 where name='Amar';
```

Output:

```
1 row updated.
```

4. DELETE COMMAND

- Used to delete one or more rows to a table.

Syntax: Delete from <table_name> where conditions;

Example:

```
SQL> delete from account where name='Amar';
```

Output:

```
1 row deleted.
```

Result:

Thus the above DML commands are performed and the required output has been obtained and verified.

Ex.No:
Date :

TRANSACTION CONTROL LANGUAGE (TCL) COMMANDS

AIM:

To perform Data Manipulation Language Commands in RDBMS by using structured query language.

TCL Commands:

- A transaction is a logical unit of work.
- All changes made to the database between commit and / or rollback operations can be referred to as a transaction.
- A transaction begins with an executable SQL statement and ends explicitly with either rollback or commit statements and implicitly, i.e., automatically, when a DDL statement is used.

1. COMMIT

- Used to end a transaction and transaction changes are made permanent to the database.
- Erases all savepoints in the transaction thus releasing the transaction locks.

Syntax: Commit work; (or) Commit;

Example:

SQL> commit;

Output:

Commit complete.

2. ROLLBACK

- Used to undo the work done in the current transaction.

- Rollback the entire transaction so that all changes made by SQL statements are undone or rollback a transaction to a savepoint so that the SQL statements after the savepoint are rolled back.

Syntax: Rollback to savepoint savepoint_id;

Example:

SQL> rollback;

Output:

Rollback complete.

3. SAVEPOINT

- Savepoints are like markers to divide a very lengthy transaction to smaller ones.
- Used to identify a point in a transaction to which it can be rolled back later.
- Used in conjunction with rollback, to rollback portions of the current transaction.

Syntax: Savepoint savepoint_id;

Example:

SQL> savepoint ins;

Output:

Savepoint created.

Result:

Thus the above TCL commands are performed and the required output has been obtained and verified.

Ex.No:
Date :

DATA CONTROL LANGUAGE (DCL) COMMANDS

AIM:

To perform Data Manipulation Language Commands in RDBMS by using structured query language.

- **GRANT PRIVILEGE COMMAND –** Used to grant database object's privileges [i.e. right to access another user's objects (table, view, …)] to other users.

  Syntax: Grant privileges on <object-name> to <username>;

  Example:

  SQL> grant create table, create view, create sequence to scott;

- **REVOKE PRIVILEGE COMMAND –** Used to withdraw the privilege which has been granted to a user.

  Syntax: Revoke privileges on <object-name> from <username>;

  Example:

  SQL> revoke select, insert on dept from scott;

Result:

Thus the above DCL commands are performed and the required output has been obtained and verified.

Ex.No:
Date :

SQL OPERATORS

Aim:

To perform SQL operators in RDBMS by using structured query language.

Arithmetic Operators

+ Addition
- Subtraction
* Multiplication
/ Division

Comparison Operators

= Equal to
<> Not Equal to
< Less than
> Greater than
<= Less than or equal to
>= Greater than or equal to
IN (List)Match any of list of values
LIKE Match a character pattern (% any no. of characters, -
IS NULLIs a null value
BETWEEN…AND…Between two values

Logical Operators

ANDReturns TRUE if both component conditions are TRUE
ORReturns TRUE if either component condition is TRUE
NOTReturns TRUE if the following condition is FALSE

Concatenation Operator (||)

Concatenates the Columns of any data type.
A Resultant column will be a Single column.

Order by Clause

Sort rows specified by the order ASC / DESC

SELECTcolumn1, column2, … …
FROM table
ORDER BY sort-column DESC;
Sorts table by sort-column in descending order
Omitting the keyword DESC will sort the table in ascending order

Examples:

1. SQL> update mgr set sal=sal+sal*10/100;

   1 row updated.

2. SQL> delete from mgr where sal<2750;

   1 rows deleted.

3. SQL> select ename || job from empl;

   ENAME||JOB
   ---
   smithclerk
   jonesmanager

4. SQL> select ename from empl where empno **in** (7369,7839,7934,7788);

   ENAME

   Smith

5. SQL> select ename from empl where job<>'manager';

   ENAME

   Smith

6. SQL> select ename from empl where hiredate between '30-JUN-81' **and** '31-DEC-81';

   no rows selected

7. SQL> select empno,ename from empl where comm=0 **or** comm is NULL;

   EMPNO ENAME

   7369 smith
   7566 jones

8. SQL> select ename from empl where ename **like** 's%' or ename **like** '%s';

```
ENAME
------------------
smith
jones
```

9. SQL> select ename from empl where ename **like** '_i%';

```
ENAME
---------------
Miller
```

10. SQL> select * from empl order by hiredate;

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL
7369	smith	clerk	7902	17-DEC-80	800
7566	jones	manager	7839	02-APR-81	2975

11. SQL> select ename,deptno,sal from empl order by deptno,sal desc;

ENAME	DEPTNO	SAL
jones	20	2975
smith	20	800

Result:

Thus the above SQL operators are performed and the required output has been obtained and verified.

Ex.No:
Date :

SQL FUNCTIONS

AIM:

To perform SQL functions in RDBMS by using structured query language.

SINGLE ROW FUNCTIONS

1. DATE FUNCTIONS

I. ADD_MONTHS –Returns a data after adding a specified date with the specified number of months.

Syntax: add_months (d, n)

Example:

SQL> select sysdate,add_months(sysdate,4) from dual;

Output

```
SYSDATE     ADD_MONTH
----------- -----------
20-APR-12   20-AUG-12
```

II. LAST_DAY – Returns the date corresponding to the last day of the month.

Syntax: last_day (d)

Example:

SQL> select sysdate,last_day(sysdate) from dual;

Output:

```
SYSDATE    LAST_DAY(
---------- ----------
20-APR-12  30-APR-12
```

III. MONTHS_BETWEEN – Returns the number of months between two dates.
Syntax: months_between (d1, d2)

Example:

SQL> select sysdate,saledt,months_between(saledt,sysdate) as diff from sale where saledt>sysdate;

Output:

```
SYSDATE      SALEDT       DIFF
-----------  -----------  ----------
 20-APR-12   14-NOV-12    6.78925403
```

IV. NEXT_DAY

Syntax: next_day (d, day)

Example: SQL> select sysdate,next_day (sysdate,'saturday') from dual;

Output:

```
SYSDATE     NEXT_DAY
----------- -----------
20-APR-12    21-APR-12
```

V. ROUND – Returns the date which is rounded to the unit specified by the format model.

Syntax: round (d, [fmt])

Example:

SQL> select sysdate,round(sysdate,'month') from dual;

Output:

```
SYSDATE     ROUND(SYS
----------- ----------
20-APR-12    01-MAY-12
```

VI. TRUNCATE – Returns the date with the time portion of the day truncated to the unit specified by format model.

Syntax: trunc (d, [fmt])

Example:

```
SQL> select sysdate,trunc(saledt,'month') from sale;
```

Output:

```
SYSDATE   TRUNC(SAL
--------- ---------
20-APR-12 01-NOV-12
```

VII. GREATEST – Returns the latest date present in the argument.

Syntax: greatest (d1, d2, …)

Example:

```
SQL> select greatest(sysdate,saledt) as big,saledt,sysdate from sale;
```

Output:

```
BIG       SALEDT    SYSDATE
--------- --------- ---------
14-NOV-12 14-NOV-12 20-APR-12
```

VIII. Arithmetic Operations on Date Values:

1) date + number (adds number of days to date)
2) date – number (subtracts number of days from date)
3) date – date (subtracts dates, producing number of days)

2. CHARACTER FUNCTIONS

FUNCTION	INPUT	OUTPUT
Initcap (char)	SQL> select initcap('pradip') from dual;	INITCA ------- Pradip
Lower (char)	SQL> select lower('TUSHAR') from dual;	LOWER(------- Tushar
Upper (char)	SQL> select upper('tushar') from dual;	UPPER(------- TUSHAR
trim (char, set)	SQL> select trim('p' from 'pradip') from dual;	TRIM ---- Radi
Instr(col,m)	SQL> select instr('pradip','d') from dual;	INSTR('PRADIP','D') ---------- 4
Length(col)	SQL> select length('pradip') from dual;	LENGTH('PRADIP') ---------- 6
Lpad(col,N,'string')	SQL> select lpad(45,6,'#') from dual;	LPAD(4 ------- ####45
Rpad(col,n)	SQL> select rpad(45,6,'#') from dual;	RPAD(4 ------- 45####

Substr (char, m, n)	SQL> select substr('demodulate',3,5) from dual;	SUBST ----- Modul
Concat(col 1,col 2,… col n)	SQL>select concat('pradip',upper('CHENNAI')) from dual;	CONCAT('PRADI ---------- pradipCHENNAI

3. NUMERIC FUNCTIONS

FUNCTION	INPUT	OUTPUT
Abs	SQL> select abs(-5) from dual;	ABS(-5) ---------- 5
Ceil (n)	SQL> select ceil(3.5) from dual;	CEIL(3.5) ---------- 4
Cos (n)	SQL> select cos(45) from dual;	COS(45) ---------- .525321989
Cosh (n)	SQL> select cosh(45) from dual;	COSH(45) ---------- 1.7467E+19
Floor (n)	SQL> select floor(3.5) from dual;	FLOOR(3.5) ---------- 3
Power (m, n)	SQL> select power(2,3) from dual;	POWER(2,3) ---------- 8
Mod (m, n)	SQL> select mod(9,2) from dual;	MOD(9,2) ---------- 1

Round (m, n)	SQL> select round(18.56789,2) from dual;	ROUND(18.56789,2) ---------- 18.57
Trunc (m, n)	SQL> select trunc(18.234665,2) from dual;	TRUNC(18.234665,2) ---------- 18.23
Sqrt (n)	SQL> select sqrt(25) from dual;	SQRT(25) ---------- 5

LN – Returns the natural logarithmic value from the system table dual.

Example:

SQL> select ln(2) from dual;

Output:

LN(2)

.693147181

4. CONVERSION FUNCTIONS

I. TO_CHAR

Syntax: to_char (d [, fmt])

- Converts date to a value of varchar2 data type in a form specified by date format fmt. If fmt is neglected then it converts date to varchar2 in the default date format.

Example:

SQL> select to_char(sysdate,'DD-MM-YY')word from dual;

Output:

WORD

20-04-12

II. TO_DATE

Syntax: to_date (char [, fmt])

- Converts char or varchar data type to date data type. Format model, fmt specifies the form of character.

Example:

```
SQL>select '31-DEC-12' last_date,to_date('31-DEC-12')-saledt diff from dual,sale;
```

Output:

```
LAST_DATE       DIFF
----------------------  ------------
   31-DEC-12        47
```

Result:

Thus the above SQL functions are performed and the required output has been obtained and verified.

Ex.No:
Date :

SQL GROUP FUNCTIONS

AIM:

To perform SQL group functions in RDBMS by using structured query language.

SQL GROUP FUNCTIONS:

Group functions Operate on sets of rows to give one result per group

FUNCTION	EXAMPLE	RESULT
Avg	SQL> select avg(sal) from std;	AVG(SAL) ---------- 1750
Max	SQL> select max(sal) from std;	MAX(SAL) ---------- 2500
Min	SQL> select min(sal) from std;	MIN(SAL) --------- 1000
Sum	SQL> select sum(sal) from std;	SUM(SAL) ---------- 7000
Count	SQL> select count(sal) from std;	COUNT(SAL) ----------- 4

Syntax:

Select column,groupfunction(column)
From table
[Where condition]
[Group by expression]
[Having group condition]
[Order by column name]

Example:

1. SQL> select job,count(*) from emp
   group by job
   order by count(*) desc;

JOB	COUNT(*)
CLERK	4
SALESMAN	4
MANAGER	3
ANALYST	2
PRESIDENT	1

2. SQL> select job,sum(sal),min(sal),max(sal),avg(sal)
   from emp
   group by job,deptno
   having deptno=20 and avg(sal)>2000;

JOB	SUM(SAL)	MIN(SAL)	MAX(SAL)	AVG(SAL)
ANALYST	6000	3000	3000	3000
MANAGER	2975	2975	2975	2975

Result:

Thus the above SQL group functions are performed and the required output has been obtained and verified.

Ex.No:
Date :

SET OPERATIONS AND JOINS

AIM:

To perform SQL operations and joins functions in RDBMS by using structured query language.

Set Operations

Combines the results of two queries into a single one.

Operator	Function
Union	Returns all distinct rows selected by either query
Union all	Returns all distinct rows selected by either query including duplicates
Intersect	Returns only rows that are common to both the queries
Minus	Returns all rows selected only by the first query but not by the second.

Examples:

1. SQL> select distinct(cus_name) from depositor union select distinct(cus_name) from borrower;

   CUS_NAME

   Jones
   paul
   smith

2. SQL> select cus_name from depositor union all select cus_name from borrower;

CUS_NAME

Jones
smith
paul
smith
paul

3. SQL> select cus_name from depositor intersect select cus_name from borrower;

CUS_NAME
---
paul
smith

4. SQL> select cus_name from depositor minus select cus_name from borrower;

CUS_NAME

Jones

Joins:

- They are used to combine the data spread across the tables. A JOIN Basically involves more than one Table to interact with. Where clause specifies the JOIN Condition.

- Ambiguous Column names are identified by the Table name.

- If join condition is omitted, then a Cartesian product is formed. That is all rows in the first table are joined to all rows in the second table

Types of Joins

Inner Join (Simple Join) : Retrieves rows from 2 tables having common columns.
1. Equi Join : A join condition with =.
2. Non Equi Join :A join condition other than =

Self Join : Joining table to itself.

Outer Join : Returns all the rows returned by simple join as well as those rows from one table that do not match any row from The symbol (+) represents outer joins.

JOIN:

```
SQL> create table univers(id number(10),name varchar(25));
Table created.

SQL> create table instit(id number(10),name varchar(25),universityid number(10));
Table created.

SQL> insert into univers values(&id,'&name');
Enter value for id: 1
Enter value for name: srm university
old  1: insert into univers values(&id,'&name')
new  1: insert into univers values(1,'srm university')
1 row created.

SQL> /
Enter value for id: 2
Enter value for name: anna university
old  1: insert into univers values(&id,'&name')
new  1: insert into univers values(2,'anna university')
1 row created.

SQL> /
Enter value for id: 3
Enter value for name: madras university
old  1: insert into univers values(&id,'&name')
new  1: insert into univers values(3,'madras university')
1 row created.

SQL> insert into instit values(&id,'&name',&universityid);
Enter value for id: 1
Enter value for name: valliamal
Enter value for universityid: 1
old 1: insert into instit values(&id,'&name',&universityid)
new 1: insert into instit values(1,'valliamal',1)
1 row created.

SQL> /
Enter value for id: 2
Enter value for name: stella mary
Enter value for universityid: 1
old 1: insert into instit values(&id,'&name',&universityid)
new 1: insert into instit values(2,'stella mary',1)
1 row created.
```

```
SQL> /
Enter value for id: 3
Enter value for name: kings eng
Enter value for universityid: 2
old 1: insert into instit values(&id,'&name',&universityid)
new 1: insert into instit values(3,'kings eng',2)
1 row created.
SQL> /

Enter value for id: 4
Enter value for name: menakshi eng
Enter value for universityid: 2
old 1: insert into instit values(&id,'&name',&universityid)
new 1: insert into instit values(4,'menakshi eng',2)
1 row created.
SQL> /

Enter value for id: 5
Enter value for name: acs eng
Enter value for universityid: 4
old 1: insert into instit values(&id,'&name',&universityid)
new 1: insert into instit values(5,'acs eng',4)
1 row created.

SQL> select * from univers;

        ID NAME
---------- ------------------------------
         1 srm university
         2 anna university
         3 madras university

SQL> select * from instit;

        ID NAME                           UNIVERSITYID
---------- ------------------------------ ------------
         1 valliamal                               1
         2 stella mary                             1
         3 kings eng                               2
         4 menakshi eng                            2
         5 acs eng                                 4
```

SQL> select * from univers **join** instit on univers.id= instit.universityid;

ID	NAME	ID	NAME	UNIVERSITYID
1	srm university	1	valliamal	1
1	srm university	2	stella mary	1
2	anna university	3	kings eng	2
2	anna university	4	menakshi eng	2

SQL> select * from univers **inner join** instit on univers.id= instit.universityid;

ID	NAME	ID	NAME	UNIVERSITYID
1	srm university	1	valliamal	1
1	srm university	2	stella mary	1
2	anna university	3	kings eng	2
2	anna university	4	menakshi eng	2

SQL> select * from univers **full outer join** instit on univers.id= instit.universityid;

ID	NAME	ID	NAME	UNIVERSITYID
1	srm university	1	valliamal	1
1	srm university	2	stella mary	1
2	anna university	3	kings eng	2
2	anna university	4	menakshi eng	2
3	madras university	<----------Null values------->		
<----------Null values------->		5	acs eng	4

6 rows selected.

SQL> select * from univers **left outer join** instit on univers.id= instit.universityid;

ID	NAME	ID	NAME	UNIVERSITYID
1	srm university	1	valliamal	1
1	srm university	2	stella mary	1
2	anna university	3	kings eng	2
2	anna university	4	menakshi eng	2
3	madras university	<----------Null value ---------->		

SQL> select * from univers **right outer join** instit on univers.id= instit.universityid;

ID	NAME	ID	NAME	UNIVERSITYID
1	srm university	2	stella mary	1
1	srm university	1	valliamal	1
2	anna university	4	menakshi eng	2
2	anna university	3	kings eng	2
<-------Null value ----- >		5	acs eng	4

Result:

Thus the above SQL operations and join functions are performed and the required output has been obtained and verified.

Ex.No:
Date :

SUB QUERIES

AIM:

To perform sub queries in RDBMS by using structured query language.

SUB QUERIES

- Nesting of Queries one within the other is termed as SubQueries.

Syntax:

Select select-list from table
Where expr oerator (select select-list from tablename);

- The subquery (inner query) executes once before the main query.The result of the subquery is used by the main query (outer query).

Example:

```
SQL> SELECT ENAME FROM EMP WHERE JOB=(SELECT JOB FROM EMP
WHERE     EMPNO=7499) and SAL>(SELECT SAL FROM EMP WHERE
EMPNO=7521);
```

Output:

```
ENAME
----------
ALLEN
```

Result:

Thus the above sub queries are performed and the required output has been obtained and verified.

Ex.No:
Date :

OTHER DATABASE OBJECTS

AIM:

To perform views, index and sequence in RDBMS by using structured query language.

I. VIEWS

- An Imaginary table contains no data and the tables upon which a view is based are called base tables.

- Logically represents subsets of data from one or more tables

Advantages of view

- To restrict database access
- To make complex queries easy
- To allow data independence
- To present different views of the same data

SYNTAX:

```
CREATE [OR REPLACE] VIEW view [col1 alias, col2 alias,...]
AS subquery
[ WITH CHECK OPTION [ CONSTRAINT constraint ] ]
[ WITH READ ONLY ]
```

Example:

1. SQL> create view empview as select empno,name,age from emp1;

Output:

View created.

2. SQL> create view emview as select empno,name,age from emp1 where empno='120' with check option;

Output:

View created.

II.SEQUENCE

- Automatically generates unique numbers
- Is a sharable object
- Is typically used to create a primary key value
- Replaces application code
- Speeds up the efficiency of accessing sequence values when cached in memory

SYNTAX:

```
CREATE SEQUENCE sequence
[INCREMENT BY n]
[START WITH n]
[{MAXVALUE n | NOMAXVALUE}]
[{MINVALUE n | NOMINVALUE}]
[{CYCLE | NOCYCLE}]
[{CACHE n | NOCACHE}];
```

Increment by n - interval between sequence numbers. The default is 1.

Start with n - first sequence number

Minvalue - minimum value of the sequence.

Maxvalue - maximum value of the sequence.

Cycle - continues to generate values after reaching either its max or min value. The default is 'nocycle'

Cache - Oracle pre allocates sequence numbers and keep the in memory for faster access. The default is 'nocache'.

Pseudo columns

- NEXTVAL returns the next available sequence value.
- CURRVAL obtains the current sequence value.
- ROWID uniquely identify the rows in your table.
- LEVEL – a special column you can reference only in a hierarchical query

Example:

SQL> create sequence empno start with 5 increment by 1 minvalue 5 maxvalue 15 cache 5 cycle;

Output:

Sequence created.

III.INDEX

- Used by the Oracle Server to speed up the retrieval of rows by using a pointer
- Reduces disk I/O by using rapid path access method to locate the data quickly
- Independent of the table it indexes
- Automatically used and maintained by the Oracle Server

SYNTAX:

CREATE INDEX index ON table (column1, column2...);

Example:

SQL> create index iemp on emp1(name);

Output:

Index created.

UNIQUE INDEX

Example:

SQL> create unique index iem on emp1(dob);

Output:

Index created.

Result:

Thus the above views, sequence, index are performed and the required output has been obtained and verified.

Ex.No:
Date :

CURSORS

AIM:

To write a PL/SQL program to create a work area in memory.

PURPOSE:

A cursor is a temporary work area created in the system memory when a SQL statement is executed. A cursor contains information on a select statement and the rows of data accessed by it. This temporary work area is used to store the data retrieved from the database, and manipulate this data. A cursor can hold more than one row, but can process only one row at a time. The set of rows the cursor holds is called the active set.

PROGRAM
IMPLICIT CURSOR

```
SQL> create table employee(empid number(10),empname varchar2(15),deptno number(10),
sal number(6,2));
Table created.
SQL> desc employee;

 Name                      Null?    Type
 ----------------------------------------------------------------------------
 EMPID                              NUMBER(10)
 EMPNAME                            VARCHAR2(15)
 DEPTNO                             NUMBER(10)
 SAL                                NUMBER(6,2)

SQL> insert into employee values(&empid,'&empname',&deptno,&sal);
Enter value for empid: 1001
Enter value for empname: AAA
Enter value for deptno: 101
Enter value for sal: 9800
old 1: insert into employee values(&empid,'&empname',&deptno,&sal)
new 1: insert into employee values(1001,'AAA',101,9800)
1 row created.
SQL> /
Enter value for empid: 1002
Enter value for empname: BBB
Enter value for deptno: 102
Enter value for sal: 9860
old  1: insert into employee values(&empid,'&empname',&deptno,&sal)
```

```
new  1: insert into employee values(1002,'BBB',102,9860)
1 row created.
SQL> select * from employee;
 EMPID    EMPNAME          DEPTNO       SAL
------ ---------- --------- ---------
     1001 AAA                 101         9800
     1002 BBB                 102         9860
SQL> alter table employee modify sal number(9,2);
Table altered.
SQL> select * from employee;
 EMPID    EMPNAME    DEPTNO       SAL
------ ---------- --------- ---------
     1001    AAA              101         9800
     1002    BBB              102         9860
SQL> Declare
  2          num number(5);
  3    Begin
  4          update employee set sal = sal + sal*0.15 where deptno=101;
  5          if SQL%NOTFOUND then
  6          dbms_output.put_line('none of the salaries were updated');
  7          elsif SQL%FOUND then
  8          num := SQL%ROWCOUNT;
  9          dbms_output.put_line('salaries for ' || num || 'employees are updated');
 10         end if;
 11    End;
 12 /
```

Salaries for 1 Employees are updated

PL/SQL procedure successfully completed.

OUTPUT:

SQL> select * from employee;

EMPID	EMPNAME	DEPTNO	SAL
1001	AAA	101	11270
1002	BBB	102	9860

Result:

Thus the above PL/SQL program is executed and the output is verified.

Ex.No:
Date :

TRIGGERS

AIM:

To write a PL/SQL program to raise the trigger for a particular event in RDBMS using SQL.

PURPOSE:

A trigger is a PL/SQL block structure which is fired when DML statements like Insert, Delete and Update is executed on a database table. A trigger is triggered automatically when an associated DML statement is executed.

SYNTAX

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
BEGIN
 -- SQL Statements
END;
```

PROGRAM

```
SQL> set serverout on;
SQL> create table cust(id number(10),name varchar2(15),age number(3),address var
char2(15),salary number(9,2));
Table created.
SQL> insert into cust values(&id,'&name',&age,'&address',&salary);
Enter value for id: 1
Enter value for name: Ramesh
Enter value for age: 23
Enter value for address: chennai
Enter value for salary: 20000
old 1: insert into cust values(&id,'&name',&age,'&address',&salary)
new 1: insert into cust values(1,'Ramesh',23,'chennai',20000)
1 row created.
```

```
SQL> /
Enter value for id: 2
Enter value for name: Suresh
Enter value for age: 22
Enter value for address: kanpur
Enter value for salary: 22000
old  1: insert into cust values(&id,'&name',&age,'&address',&salary)
new  1: insert into cust values(2,'Suresh',22,'kanpur',22000)
1 row created.
SQL> /
Enter value for id: 3
Enter value for name: Mahesh
Enter value for age: 24
Enter value for address: bengalur
Enter value for salary: 24000
old  1: insert into cust values(&id,'&name',&age,'&address',&salary)
new  1: insert into cust values(3,'Mahesh',24,'bengalur',24000)
1 row created.
SQL> select * from cust;
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	chennai	20000
2	Suresh	22	kanpur	22000
3	Mahesh	24	bengalur	24000

PROGRAM

```
SQL> CREATE OR REPLACE TRIGGER display_salary_changes
  2  BEFORE DELETE OR INSERT OR UPDATE ON cust
  3  FOR EACH ROW
  4  WHEN (NEW.ID > 0)
  5  DECLARE
  6     sal_diff number;
  7 BEGIN
  8  sal_diff := :NEW.salary - :OLD.salary;
  9   dbms_output.put_line('Old salary: ' || :OLD.salary);
 10  dbms_output.put_line('New salary: ' || :NEW.salary);
 11   dbms_output.put_line('Salary difference: ' || sal_diff);
12 END;
 13 /

Trigger created.
```

PROGRAM

```
SQL> DECLARE
  2     total_rows number(2);
  3  BEGIN
  4    UPDATE cust
  5    SET salary = salary + 5000;
  6     IF sql%notfound THEN
  7    dbms_output.put_line('no customers updated');
  8   ELSIF sql%found THEN
  9    total_rows := sql%rowcount;
 10     dbms_output.put_line( total_rows || ' customers updated ');
 11     END IF;
 12  END;
 13
 14 /
```

OUTPUT:

```
Old salary: 20000
New salary: 25000
Salary difference: 5000
Old salary: 22000
New salary: 27000
Salary difference: 5000
Old salary: 24000
New salary: 29000
Salary difference: 5000
3 customers updated
PL/SQL procedure successfully completed.
```

Result:

Thus the above PL/SQL program is executed and the output is verified.