EX.NO: 1(A)

DATE:

DATA DEFINITION LANGUAGE

AIM:

To execute the following commands in data definition language.

COMMANDS

1(a). CREATION OF TABLE

DESCRIPTION: Create command is used to create tables in database.

SYNTAX: Create table <tablename>(col1 datatype,col2 datatype,col3 datatype);

EXAMPLE: SQL> create table employee(ename varchar2(15),eno number(3), designation varchar2(15),salary number(6));

OUTPUT : **Table created.**

1(b). CREATION OF TABLE WITH PRIMARY KEY

DESCRIPTION: A primary key is a column in a table whose values uniquely identifies the rows in the table. Primary keys must contain uniquely values. A primary key column cannot contain NULL values. Each table should have a primary key and each table can have only one primary key.

SYNTAX: create table table_name(col1 datatype primary key,col2 datatype, col3 datatype);

EXAMPLE: SQL> create table emp1(eno number(5) primary key, ename varchar2(15),designation varchar2(15), salary number(6));

OUTPUT: **Table created**.

2(a). ALTERING THE TABLE

DESCRIPTION: Alter command is used to alter the structure of database like modifying the size of the column, adding a new column, remaining column, deleting a column

SYNTAX: alter table<tablename> modify(colname datatype);

EXAMPLE: SQL> alter table employee modify(ename varchar2(20))

OUTPUT: **Table altered.**

SYNTAX: alter table<tablename>add(colname datatype);

EXAMPLE: SQL> alter table employee add(place varchar2(20));

OUTPUT: **Table altered.**

SYNTAX: Alter table<tablename>rename column<old colname>to<new colname>;

EXAMPLE: SQL> alter table employee rename column place to exp;

OUTPUT: **Table altered.**

2(b).CREATING PRIMARY KEY USING ALTER COMMAND

SYNTAX: alter table employee add primary key(eno)

EXAMPLE: SQL> alter table employee add primary key(eno);

OUTPUT: **Table altered.**

2(c).DROPPING PRIMARY KEY USING ALTER COMMAND

SYNTAX: alter table employee drop primary key;

EXAMPLE: SQL>alter table employee drop primary key;

OUTPUT: **Table dropped.**

3. VIEW THE DESCRIPTION OF THE TABLE

DESCRIPTION: Desc command is used to view the description of the table.

SYNTAX: desc<table_name>;

EXAMPLE: SQL> desc emp1;

OUTPUT:

```
 Name                              Null?            Type
-----------------------------    ------------     ---------------------------
ENAME                           NOTNULL          VARCHAR2(15)
ENO                                              NUMBER(3)
DESIGNATION                                      VARCHAR2(15)
SALARY                                           NUMBER(6)
```

4. TRUNCATING THE TABLE

DESCRIPTION: Deletes only the records of the table, schema remains safe.

SYNTAX: truncate table<tablename>;

EXAMPLE: SQL> truncate table employee;

OUTPUT: **Table truncated.**

5. DELETING THE TABLE

DESCRIPTION: Deletes the record as well as the schema entirely.

SYNTAX: Drop table <table_name>

EXAMPLE: SQL> drop table employee;

OUTPUT: **Table dropped.**

6.RENAMING THE TABLE

DESCRIPTION: Used to rename the table

SYNTAX: rename<old tablename>to <new tablename>;

EXAMPLE: SQL> rename employee to emp1;

OUTPUT: **Table renamed.**

7. COMMENTING THE TABLE

DESCRIPTION: used to add the comments to the table

SYNTAX: comment on table<table_name> is ‘comment details’;

EXAMPLE: SQL> comment on table emp1 is 'EMPLOYEE DETAILS';

OUTPUT: **Comment created.**

<u>RESULT:</u>

Thus the above commands were successfully executed and the output was verified.

Ex.No :1(B) Creating an Employee database to set various constraints.

DATE:

AIM

To study the various constraints available in the SQL query language.

1.PRIMARY KEY:

DESCRIPTION: A primary key is one which uniquely identifies a row of a table. This key does not allow null values and also does not allow duplicate values.

SYNTAX: Create table table_name (col_name datatype primary key);

EXAMPLE:

```
SQL> insert into employee values('rahul',5,'chennai',15000,'3yrs');
insert into employee values('rahul',5,'chennai',15000,'3yrs')
*
ERROR at line 1:
ORA-00001: unique constraint (RAJI.SYS_C003995) violated
```

2.UNIQUE

DESCRIPTION: The UNIQUE constraint uniquely identifies each record in a database table. The UNIQUE and primary key constraints both provide a guarantee for uniqueness for a column or a set of columns.

SYNTAX: Create table table_name(col_name datatype unique);

EXAMPLE:

```
SQL> create table t1(a number(4) primary key,b number(4) unique,c number(4));

Table created.

SQL> insert into t1 values(1,2,3);

1 row created.

SQL> insert into t1 values(2,2,3);
insert into t1 values(2,2,3)
ERROR at line 1:
ORA-00001: unique constraint (RAJI.SYS_C003997) violated

SQL> insert into t1(a,c) values(3,3);

1 row created.
```

3.NOT NULL:

DESCRIPTION: The NOT NULL constraints enforces a column to not accept NULL values.The not null constraints enforces a field to always contain a value. This means that you cannot insert a new record, or update a record without adding a value to this field.

SYNTAX: Create table table_name(col_name datatype NOT NULL);

EXAMPLE:

```
SQL> create table p1(eno number(4),name varchar2(12) not null);

Table created.

SQL> insert into p1 values(1,'latha');

1 row created.

SQL> insert into p1(name) values('kavi');

1 row created.

SQL> insert into p1(eno) values(1);
insert into p1(eno) values(1)
*
ERROR at line 1:
ORA-01400: cannot insert NULL into ("RAJI"."P1"."NAME")
```

4.CHECK:

DESCRIPTION: The check constraint is used to limit the value range that can be placed in a column. If you want to define a check constraint on a single column it allows only certain values for this column. If you define a check constraint on a table it can limit the values in certain columns based on values in other columns in a row.

SYNTAX: CREATE table table_name (….column_name type CHECK(predicate),…..);

EXAMPLE:

```
SQL> create table voter(vid number(4),age number check(age>18));

Table created.


SQL> insert into voter values(1001,20);

1 row created.

SQL> insert into voter values(1001,15);
```

```
insert into voter values(1001,15)
ERROR at line 1:
ORA-02290: check constraint (RAJI.SYS_C003999) violated
```

5.FOREIGN KEY:

DESCRIPTION: Foreign key is reference of another table primary key. It will not allows null values.

SYNTAX: Create table table_name(col_name datatype references table_name(col_name));
NOTE: References table column should be a primary key

EXAMPLE:

```
SQL>create table stu(rno number(3) references employee(eno),sname
varchar2(20));

Table created.

SQL> insert into stu values(1,'ram');

1 row created.

SQL> insert into stu values(1,'priya');

ORA-02291:integrity constrains(RAJI.SYS_C003000) violated – parent key
not found..
```

6.DEFAULT:

DESCRIPTION: The DEFAULT constraint is used to insert a default value into a column. The default value will be added to all new records, if no other value is specified.

SYNTAX: Create table table_name(col_name datatype default ‘value’);

EXAMPLE:

```
SQL> create table a1(no number,name varchar2(12) default 'narmadha');

Table created.

SQL> insert into a1 values(1,'priya');

1 row created.

SQL> insert into a1(no) values(2);

1 row created.

SQL> select * from a1;
```

```
      NO          NAME
- - - - - - - - - - - - - - - - - - - -
       1            priya
       2     narmadha
```

DOMAIN INTEGRITY CONSTRAINTS

NOT NULL CONSTRAINT

```
SQL> create table empl (ename varchar2(30) not null, eid varchar2(20) not null);

Table created.

SQL> insert into empl values ('abcde',11);

1 row created.

SQL> insert into empl values ('fghij',12);

1 row created.

SQL> insert into empl values ('klmno',null);

insert into empl values ('klmno',null)

*

ERROR at line 1:

ORA-01400: cannot insert NULL into ("ITA"."EMPL"."EID")

SQL> select * from empl;

ENAME                    EID

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

abcde                    11

fghij                    12
```

CHECK AS A COLUMN CONSTRAINT

```
SQL> create table depts ( dname varchar2(30) not null, did number(20) not null check (did<10000));

Table created.

SQL> insert into depts values ('sales ',9876);

1 row created.
```

```
SQL> insert into depts values ('marketing',5432);

1 row created.

SQL> insert into depts values ('accounts',789645);

insert into depts values ('accounts',789645)

*

ERROR at line 1:

ORA-02290: check constraint (ITA.SYS_C003179) violated

SQL> select * from depts;

DNAME                    DID

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

sales                  9876

marketing              5432
```

CHECK AS A TABLE CONSTRAINT

```
SQL> create table airports (aname varchar2(30) not null , aid number(20) not null, acity varchar2(30) check(
acity in ('chennai','hyderabad','bangalore')));

Table created.

SQL> insert into airports values( 'abcde', 100,'chennai');

1 row created.

SQL> insert into airports values( 'fghij', 101,'hyderabad');

1 row created.

SQL> insert into airports values( 'klmno', 102,'bangalore');

1 row created.

SQL> insert into airports values( 'pqrst', 103,'mumbai');

insert into airports values( 'pqrst', 103,'mumbai')

*

ERROR at line 1:
```

```
ORA-02290: check constraint (ITA.SYS_C003187) violated

SQL> select * from airports;

ANAME                          AID ACITY

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

abcde                          100   chennai

fghij                          101   hyderabad

klmno                          102   bangalore
```

ENTITY INTEGRITY CONSTRAINTS

UNIQUE AS A COLUMN CONSTRAINT

```
SQL> create table book (bname varchar2(30) not null, bid number(20) not null unique);

Table created.

SQL> insert into book values ('fairy tales',1000);

1 row created.

SQL> insert into book values ('bedtime stories',1001);

1 row created.

SQL> insert into book values ('comics',1001);

insert into book values ('comics',1001)

*

ERROR at line 1:

ORA-00001: unique constraint (ITA.SYS_C003130) violated

SQL> select * from book;

BNAME                          BID

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

fairy tales                    1000

bedtime stories                1001
```

UNIQUE AS A TABLE CONSTRAINT

```
SQL> create table orders( oname varchar2(30) not null , oid number(20) not null , unique(oname,oid));

Table created.

SQL> insert into orders values ('chair', 2005);

1 row created.

SQL> insert into orders values ('table',2006);

1 row created.

SQL> insert into orders values ('chair',2007);

1 row created.

SQL> insert into orders values ('chair', 2005);

insert into orders values ('chair', 2005)

*

ERROR at line 1:

ORA-00001: unique constraint (ITA.SYS_C003152) violated

SQL> select * from orders;

ONAME                          OID

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

chair                     2005

table                     2006

chair                     2007
```

PRIMARY KEY AS A COLUMN CONSTRAINT

```
SQL> create table custo ( cname varchar2(30) not null , cid number(20) not null primary key);

Table created.

SQL> insert into custo values ( 'jones', 506);

1 row created.

SQL> insert into custo values ('hayden',508);
```

```
1 row created.

SQL> insert into custo values ('ricky',506);

insert into custo values ('ricky',506)

*

ERROR at line 1:

ORA-00001: unique constraint (ITA.SYS_C003165) violated

SQL> select * from custo;

CNAME                          CID

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

jones                      506

hayden                     508
```

PRIMARY KEY AS A TABLE CONSTRAINT

```
SQL> create table branches( bname varchar2(30) not null , bid number(20) not null , primary key(bnam
e,bid));

Table created.

SQL> insert into branches values ('anna nagar', 1005);

1 row created.

SQL> insert into branches values ('adyar',1006);

1 row created.

SQL> insert into branches values ('anna nagar',1007);

1 row created.

SQL> insert into branches values ('anna nagar', 1005);

insert into branches values ('anna nagar', 1005)

*

ERROR at line 1:

ORA-00001: unique constraint (ITA.SYS_C003173) violated
```

```
SQL> select * from branches;

BNAME                          BID

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

anna nagar                    1005

adyar                         1006

anna nagar                    1007
```

REFERENTIAL INTEGRITY CONSTRAINTS

TO CREATE ‘DEPTS’ TABLE

```
SQL> create table depts(city varchar2(20), dno number(5) primary key);

Table created.

SQL> insert into depts values('chennai', 11);

1 row created.

SQL> insert into depts values('hyderabad', 22);

1 row created.
```

TO CREATE ‘SEMP’ TABLE

```
SQL> create table semp(ename varchar2(20), dno number(5) references depts(dno));

Table created.

SQL> insert into semp values('x', 11);

1 row created.

SQL> insert into semp values('y', 22);

1 row created.

SQL> select * from semp;

ENAME                 DNO

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

x                  11

y                  22
```

ALTER TABLE

```
SQL> alter table semp add(eddress varchar2(20));

Table altered.

SQL> update semp set eddress='10 gandhi road' where dno=11;

1 row updated.

SQL> update semp set eddress='12 m.g. road' where dno=22;

1 row updated.

SQL> select * from semp;

ENAME                  DNO    EDDRESS

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

x                     11          10 gandhi road

y                     22          12 m.g. road


SQL> select city, ename from depts, s2emp where depts.dno = s2emp.dno;

CITY               ENAME

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

chennai            x

hyderabad          y
```

RESULT

The various constraints were implemented and the tables were created using the respective constraints.

EX.NO: 2A
DATE:

DATA MANIPULATION LANGUAGE

AIM:

To manipulate the data using data manipulation language.

COMMANDS

1. INSERTING THE ROWS

DESCRIPTION: Used to insert the values to the table

SYNTAX: insert into <tablename>values(‘col1 values’,’col2 values’);

EXAMPLE: SQL>insert into employee values('priya',1,'chennai',10000);

OUTPUT: **1 row created.**

EXAMPLE:

```
SQL> insert into employee values('&ename',&eno,'&designation',&salary);
Enter value for ename: latha
Enter value for eno: 2
Enter value for designation: trichy
Enter value for salary: 25000
```

OUTPUT:

```
old 1: insert into employee values('&ename',&eno,'&designation',&salary)
new 1: insert into employee values('latha',2,'trichy',25000)
1 row created.
```

```
SQL> /
Enter value for ename: raji
Enter value for eno: 3
Enter value for designation: madurai
Enter value for salary: 20000
```

OUTPUT:

```
old 1: insert into employee values('&ename',&eno,'&designation',&salary)
new 1: insert into employee values('raji',3,'madurai',20000)
1 row created.
```

```
SQL> /
Enter value for ename: veni
Enter value for eno: 4
Enter value for designation: chennai
Enter value for salary: 15000
```

OUTPUT:

```
old 1: insert into employee values('&ename',&eno,'&designation',&salary)
new 1: insert into employee values('veni',4,'chennai',15000)
1 row created.
```

```
SQL> /
```

```
                    Enter value for ename: divya
                    Enter value for eno: 5
                    Enter value for designation: trichy
                    Enter value for salary: 25000
```

OUTPUT:

```
old 1: insert into employee values('&ename',&eno,'&designation',&salary)
new 1: insert into employee values('divya',5,'trichy',25000)
1 row created.
```

2. SELECTING THE ROWS OF THE TABLE

DESCRIPTION: Used to display the table and table values

SYNTAX: Select * from <tablename>;

EXAMPLE: SQL> select * from employee;

OUTPUT:

```
ENAME                  ENO      DESIGNATION          SALARY
--------------------   ------------   --------------------------   ----------------
priya                     1          chennai            10000
latha                     2           trichy            25000
raji                      3          madurai            20000
veni                      4          chennai            15000
divya                     5           trichy            25000
```

EXAMPLE: SQL> select ename from employee;

OUTPUT:

```
ENAME
---------------
priya
latha
raji
veni
divya
```

EXAMPLE: SQL> select ename from employee where eno=2;

OUTPUT:

```
ENAME
---------------
 latha
```

3.UPDATING THE TABLE

DESCRIPTION: Used to update the values of the table

SYNTAX: update<tablename>set colname=’value’where colname=’values’;

EXAMPLE: SQL> update employee set ename='kavi' where eno=1;

OUTPUT: **1 row updated.**

EXAMPLE: SQL> select ename,eno from employee;

OUTPUT:

```
ENAME          ENO
```

```
--------------- -----------------
kavi                  1
latha                 2
raji                  3
veni                  4
divya                 5
```

4.DELETING THE TABLE

DESCRIPTION: Used to delete the specified rows of the table.

SYNTAX: delete from<tablename>where colname=value;

EXAMPLE: SQL> delete from employee where ename='veni';

OUTPUT: **1 row deleted.**

EXAMPLE: SQL> select * from employee;

OUTPUT:

```
ENAME                 ENO          DESIGNATION           SALARY
--------------------- ---------- ---------------------------------  -------------------
kavi                  1              chennai                 10000
latha                 2              trichy                  25000
raji                  3              madurai                 20000
divya                 5              trichy                  25000
```

RESULT:

Thus the database is manipulated using data manipulation language and the output was verified.

EX.NO: 2B

DATE:

VARIOUS CLAUSES OF SELECT

AIM:

To study about the various clauses of select command to retrieve the data from the database table

Employee:

ENAME	ENO	DESIGNATION	SALARY
kavi	1	chennai	10000
latha	2	trichy	25000
raji	3	madurai	20000
divya	5	trichy	25000

COMMANDS

1. USING DISTINCT

DESCRIPTION: The distinct keyword can be used to return only Distinct (different).

SYNTAX: Select DISTINCT column_name from table_name;

EXAMPLE: SQL> select distinct designation from employee;

OUTPUT:

```
DESIGNATION
----------
chennai
trichy
madurai
```

2.WHERE CLAUSE

DESCRIPTION: The WHERE clause is used to execute only the records that fulfill a specified criterion.

SYNTAX: select column_name from table_name Where column_name operator value.

EXAMPLE: SQL> select ename from employee where salary between 15000 and 25000;

OUTPUT:

```
ENAME
-----------
latha
raji
divya
```

SQL> select ename from employee where salary!=25000;

OUTPUT:

```
ENAME
-----------
 kavi
 raji
```

SQL> select ename from employee where designation='chennai';

```
OUTPUT: ENAME
        --------
        kavi
```

SQL> select ename from employee where designation='trichy' and ename='divya';

```
OUTPUT:  ENAME
         -----------
         divya
```

SQL> select * from employee where eno<3;

```
OUTPUT:  ENAME    ENO       DESIGNATION       SALARY       EXP
         ----------  -----------   ---------------------    ---------------   ----------
         kavi         1           chennai              10000        2yrs
         latha        2           trichy               25000         3yrs
```

SQL> select * from employee where ename like 'l%';

```
OUTPUT:  ENAME    ENO       DESIGNATION       SALARY     EXP
         ---------- -------------- --------------------- ---------------- ----------
         latha        2          trichy              25000      3yrs
```

SQL> select * from employee where ename like '%a';

```
OUTPUT:  ENAME    ENO       DESIGNATION       SALARY     EXP
        ----------    ------------    ---------------------     ---------------- ----------
        latha         2          trichy                 25000       3yrs
        divya         5          trichy                 25000       5yrs
```

SQL> select * from employee where ename like '%th%';

```
OUTPUT:  ENAME    ENO       DESIGNATION       SALARY     EXP
         ----------   ------------   ---------------------    ---------------- ----------
         latha        2            trichy               25000       3yrs
```

3.ORDER BY

DESCRIPTION: The ORDER BY keyword is used to sort the result_set by a specified column. The ORDER BY keyword sort the records in ascending order by default. If you want to sort the records in a descending order, you can use the DESC keyword.

SYNTAX: select column_name from table_name ORDER BY column_name(s);

EXAMPLE: SQL> select * from employee order by ename;

```
OUTPUT:   ENAME              ENO          DESIGNATION      SALARY    EXP
          -------------------      ----------      -----------------------------  ----------  -----------
           kavi                   1             chennai                 10000      2yrs
           latha                  2             trichy                  25000      3yrs
           raji                   3             madurai                 20000      5yrs
             divya                  5             trichy                25000      5yrs
```

```
SQL> select * from employee order by ename desc;
OUTPUT:  ENAME              ENO          DESIGNATION      SALARY    EXP
         -------------     ----------   ----------------------  ------------  -------
         raji                3            madurai              20000     5yrs
         latha               2            trichy               25000     3yrs
         kavi                1            chennai              10000     2yrs
         divya               5            trichy               25000     5yrs
```

```
SQL> select * from employee where ename like '%a' order by eno;
OUTPUT:  ENAME     ENO      DESIGNATION       SALARY   EXP
          ------------  ---------   --------------------     ------------- ----------
          latha      2        trichy             25000     3yrs
          divya      5        trichy             25000     5yrs
```

4.ALIAS

DESCRIPTION: Giving someone another name to column or table in the output.

SYNTAX:

For tables,   select column_name(s) from table_name as alias_name

For columns,   select column_name as alias_name from table_name;

```
EXAMPLE:       SQL> select eno as employeeid from employee;
OUTPUT:        EMPLOYEEID
               - - - - - - - - - - -
                     1
                     2
                     3
                     5
SQL> select e.ename from employee e;
OUTPUT:        ENAME
               -----------
               kavi
               latha
               raji
               divya
```

```
SQL> select e.eno,e.ename,d.department,d.manager from employee e,depart d where
     e.eno=d.eno;
OUTPUT:         ENO       ENAME    DEPARTMENT   MANAGER
               ---------- --------------    -------------------- --------------
                    1      kavi          it                   ramya
                    2      latha         csc                  vinay
```

RESULT:

Thus the various clauses of select command to retrieve the data from a database table was studied and the output was verified.

EX.NO: 2C
DATE:

FUNCTIONS IN SQL

AIM

To study about the various functions of oracle SQL.

GROUP FUNCTIONS

1. COUNT

DESCRIPTION : Returns the total number of rows in a table.
SYNTAX : select count(col_name) from table_name
EXAMPLE: **SQL>** select count(salary) from employee;
OUTPUT:

```
COUNT(SALARY)
-------------
          4
```

2.MIN

DESCRIPTION : Returns the Minimum value of the specified column in a table.
SYNTAX : select min(col_name) from table_name
EXAMPLE: **SQL>** select min(salary) from employee;
OUTPUT:

```
MIN(SALARY)
 -----------
   10000
```

3.MAX

DESCRIPTION : Returns the Maximum value of the specified column in a table.
SYNTAX : select max(col_name) from table_name
EXAMPLE: **SQL>** select max(salary) from employee;
OUTPUT:

```
MAX(SALARY)
 ------------
   25000
```

4.SUM

DESCRIPTION : Returns the total sumof the specified column in a table.
SYNTAX : select sum(col_name) from table_name
EXAMPLE: **SQL>** select sum(salary) from employee;
OUTPUT:

```
SUM(SALARY)
------------
  80000
```

5.AVG:

DESCRIPTION : Returns the average of the specified column in a table.
SYNTAX : select avg(col_name) from table_name
EXAMPLE: **SQL>** select avg(salary) from employee;
OUTPUT:

```
 AVG(SALARY)
-----------
  20000
```

6.GROUPBY

DESCRIPTION : The GROUP BY statement is used in conjunction with the aggregate functions to group the result-set by one or more columns.

SYNTAX :
```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value GROUP BY column_name
```

EXAMPLE: **SQL>** select designation,max(salary) from employee group by designation;

OUTPUT:
```
DESIGNATION MAX(SALARY)
 -------------------------
 chennai         10000
 trichy           25000
 madurai         20000
```

SQL> select designation,sum(salary) from employee group by designation;

OUTPUT:
```
DESIGNATION  SUM(SALARY)
 -------------  -------------
 chennai          10000
 trichy            50000
 madurai           20000
```

SQL> select designation,sum(salary) from employee where designation='trichy'group by designation;

OUTPUT:
```
DESIGNATION  SUM(SALARY)
 ------------    -----------
     trichy          50000
```

SQL> select * from employee where salary=(select max(salary) from employee);

OUTPUT:
```
ENAME    ENO   DESIGNATION SALARY  EXP
--------       ------------------------------------
latha        2       trichy           2500      3yrs
divya        5       trichy           25000     6yrs
```

NUMERIC FUNCTIONS:

1.ROUND

DESCRIPTION: Rounds a column, an expression, or a value to n decimal places

SYNTAX: SELECT ROUND(column name/expression/value,n)FROM DUAL;

EXAMPLE: **SQL>** select round(56.889124,2) from dual;

OUTPUT:
```
ROUND(56.889124,2)
------------------
     56.89
```

2.MOD

DESCRIPTION: Returns the remainder of x divided by y

SYNTAX: SELECT MOD(X, Y) FROM dual;

EXAMPLE: **SQL>** select mod(15,4) modulated_value from dual;

OUTPUT: **MODULATED_VALUE**

```
------------------
       3
```

3.TRUNC

DESCRIPTION: Truncates a column, an expression, or a value to n decimal places

SYNTAX: SELECT TRUNC(X, Y) FROM dual;

EXAMPLE: **SQL>** select trunc(23.56921,2) truncated_value from dual;

OUTPUT: **TRUNCATED_VALUE**

```
-----------------
    23.56
```

4.ABS

DESCRIPTION: Returns the absolute value of the number

SYNTAX: SELECT ABS(expression/column name/value) FROM dual;

EXAMPLE: **SQL>** select abs(-5.6) absolute_value from dual;

OUTPUT: **ABSOLUTE_VALUE**

```
----------------
    5.6
```

5.CEIL

DESCRIPTION: Returns the next integer that is higher the number entered

SYNTAX: SELECT CEIL(expression/column name/value) FROM dual;

EXAMPLE: **SQL>** select ceil(34.55) from dual;

OUTPUT: **CEIL(34.55)**

```
----------
  35
```

SQL> select ceil(-34.25) from dual;

OUTPUT: **CEIL(-34.25)**

```
----------
  -34
```

6.FLOOR

DESCRIPTION: Returns the next integer that is lower than the number entered

SYNTAX: SELECT FLOOR(expression/column name/value) FROM dual;

EXAMPLE: **SQL>** select floor(34.33) flooring from dual;

OUTPUT: **FLOORING**

```
---------
  34
```

SQL> select floor(-34.33) flooring from dual;

OUTPUT: **FLOORING**

```
---------
    -35
```

7.POWER

DESCRIPTION: Raises the value of m to n

SYNTAX: SELECT POWER(X, Y) FROM dual;

EXAMPLE: **SQL>** select power(3,2) power from dual;

OUTPUT: **POWER**

```
-------
   9
```

8.SQRT
DESCRIPTION: returns the square root of the number entered
SYNTAX: SELECT SQRT(expression/column name/value) FROM dual;
EXAMPLE: **SQL>** select sqrt(256) sq_root from dual;
OUTPUT: **SQ_ROOT**

```
---------
    16
```

CHARACTER FUNCTIONS
1.CONCAT
DESCRIPTION: Concatenates the specified columns or expression
SYNTAX: SELECT CONCAT(arg1,arg2) from Dual;
where arg1 and arg2 can be column names or expressions
EXAMPLE: **SQL>** select concat('hello','world') concat from dual;
OUTPUT: **CONCAT**

```
---------
helloworld
```

2.INITCAP
DESCRIPTION: Capitalizes the initial letters
SYNTAX: SELECT INITCAP(col_name(or)expr) FROM DUAL;
EXAMPLE: **SQL>** select initcap('chennai') initial_capital from dual;
OUTPUT: **INITIAL**

```
-------
Chennai
```

3.LOWER
DESCRIPTION: Converts to lower case
SYNTAX: SELECT LOWER(expression/column name/string) FROM dual;
EXAMPLE: **SQL>** select lower('HELLOWORLD') lower from dual;
OUTPUT: **LOWER**

```
---------
helloworld
```

4.UPPER
DESCRIPTION: Converts to Upper case
SYNTAX: SELECT UPPER(expression/column name/string) FROM dual;
EXAMPLE: **SQL>** select upper('tajmahal') from dual;
OUTPUT: **UPPER('T**

```
--------
TAJMAHAL
```

5.SUBSTR:
DESCRIPTION: Extracts the specified number of characters from the position given

SYNTAX: SELECT SUBSTR(string,<position>,[<no, of characters to extract>) FROM dual;

EXAMPLE: **SQL>** select substr('HELLO_WORLD',5,4) from dual;

OUTPUT:

```
SUBS
-----
O_WO
```

6.LTRIM:

DESCRIPTION: Clips of the specified portion from the left side

SYNTAX: SELECT LTRIM(string1,substring) FROM dual;

EXAMPLE: **SQL>** select ltrim('HELLO_WORLD','HE')from dual;

OUTPUT:

```
LTRIM('HE
   ------
LLO_WORLD
```

7.RTRIM:

DESCRIPTION: Clips of the specified portion from the right side

SYNTAX: SELECT RTRIM(string,substring) FROM dual;

EXAMPLE: **SQL>** select rtrim('HELLO_WORLD','RLD') from dual;

OUTPUT:

```
RTRIM('H
 ---------
HELLO_WO
```

8.INSTR:

DESCRIPTION: Returns the position of the sub-string in the given string

SYNTAX: SELECT INSTR(string,substring) FROM dual;

EXAMPLE: SQL> select instr('HELLO','E') from dual;

OUTPUT:

```
INSTR('HELLO','E')
 ---------------
        2
```

MISCELLANEOUS FUNCTIONS:

1.NVL

DESCRIPTION: Returns either of the two expressions which is not null

SYNTAX: SELECT NVL(column name, value to be substituted) FROM tablename;

EXAMPLE: **SQL>** select eno,nvl(salary,2) from employee;

OUTPUT:

```
ENO   NVL(SALARY,2)
----  --------------
  1    10000
  2    25000
  3    20000
  5    25000
```

2.VSIZE:

DESCRIPTION: Returns the size of the specified string

SYNTAX: SELECT VSIZE(expression/column name/value) FROM dual;

EXAMPLE: **SQL>** select designation,vsize(designation) from employee;

OUTPUT:

```
DESIGNATION VSIZE(DESIGNATION)
```

```
------------------------      -------------------------------------
    chennai                  7
    trichy                   6
    madurai                  7
    trichy                   6
```

DATE FUNCTIONS:

1. ADD MONTHS:

DESCRIPTION: Adds the specified number of months to the date given

SYNTAX: SELECT ADD_MONTHS(‘DATE’, NUMBER) FROM DUAL;

EXAMPLE: **SQL>** select add_months('31-july-2010',2) from dual;

OUTPUT:

```
ADD_MONTH
- - - - - - - - - - - -
    30-SEP-10
```

2.LAST DAY:

DESCRIPTION: Returns the last day of the specified month

SYNTAX: SELECT LAST_DAY(‘DATE’) FROM DUAL;

EXAMPLE: **SQL**> select last_day('10-feb-2010') from dual;

OUTPUT:

```
LAST_DAY(
 - - - - - - - - -
 28-FEB-10
```

3.MONTHS BETWEEN:

DESCRIPTION: Returns the number of months between the dates

SYNTAX: SELECT MONTHS_BETWEEN(‘DATE1,DATE2) FROM DUAL;

EXAMPLE: SQL> select months_between('05-jan-2010','05-may-2010') from dual;

OUTPUT:

```
MONTHS_BETWEEN('05-JAN-2010','05-MAY-2010')
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        -4
```

4.SYSTEM DATE:

DESCRIPTION: Returns the current date

EXAMPLE: **SQL>** select sysdate from dual;

OUTPUT:

```
SYSDATE
 - - - - - - - -
26-JAN-11
```

5.NEXT DAY:

DESCRIPTION: Returns the next date that would fall on the same day

SYNTAX: SELECT NEXT_DAY(‘DATE’, ‘DAY’) FROM DUAL;

EXAMPLE: **SQL**> select next_day('05-jan-2010','wed') from dual;

OUTPUT:

```
NEXT_DAY(
- - - - - - - - - -
06-JAN-10
```

RESULT:

Thus the various functions of oracle SQL were studied and the output was verified.

Ex. No: 3(A)

Ex. No: 3(A)

DATE: **CREATION OF VIEWS, SYNONYMS, SEQUENCE, INDEXES.**

AIM:
To create views, synonyms, sequences, and indexes using DDL and DML statements

DESCRIPTION:

VIEWS

A database view is a *logical* or *virtual table* based on a query. It is useful to think of a *view* as a stored query. Views are queried just like tables.
A DBA or view owner can drop a view with the DROP VIEW command.

TYPES OF VIEWS

- Updatable views – Allow data manipulation
- Read only views – Do not allow data manipulation

To Create The Table 'Fviews'

```
SQL> create table fviews( name varchar2(20),no number(5), sal number(5), dno number(5));
Table created.
SQL> insert into fviews values('xxx',1,19000,11);
1 row created.
SQL> insert into fviews values('aaa',2,19000,12);
1 row created.
SQL> insert into fviews values('yyy',3,40000,13);
1 row created.

SQL> select * from fviews;
OUTPUT:
NAME              NO      SAL      DNO
------------ ----- ---------- ------
xxx                1    19000       11
aaa               2    19000       12
yyy                3    40000       13
```

TO CREATE THE TABLE 'DVIEWS'

```
SQL> create table dviews( dno number(5), dname varchar2(20));
Table created.
SQL> insert into dviews values(11,'x');
1 row created.
SQL> insert into dviews values(12,'y');
1 row created.

SQL> select * from dviews;

  OUTPUT:    DNO  DNAME
             ----- ---------
                11   x
```

```
                  12    y
```

CREATING THE VIEW 'SVIEW' ON 'FVIEWS' TABLE

```
SQL> create view sview as select name,no,sal,dno from fviews where dno=11;
View created.
SQL> select * from sview;

OUTPUT:    NAME        NO      SAL      DNO
          ---------- ------- -------- --------
            xxx         1     19000      11
```

Updates made on the view are reflected only on the table when
the structure of the table and the view are not similar -- proof

```
SQL> insert into sview values ('zzz',4,20000,14);
1 row created.
SQL> select * from sview;

OUTPUT:    NAME        NO      SAL      DNO
          ---------- ------- -------- --------
            xxx         1     19000      11

SQL> select * from fviews;

OUTPUT:    NAME        NO      SAL      DNO
          ---------- ------- -------- --------
            xxx         1     19000      11
            aaa         2      19000      12
            yyy         3     40000      13
            zzz          4     20000      14
```

Updates made on the view are reflected on both the view and
the table when the structure of the table and the view are similar – proof

CREATING A VIEW 'IVIEW' FOR THE TABLE 'FVIEWS'

```
SQL> create view iview as select * from fviews;
View created.
SQL> select * from iview;
OUTPUT:    NAME        NO      SAL      DNO
          ---------- ------- -------- --------
            xxx         1     19000      11
            aaa         2      19000      12
            yyy         3     40000      13
            zzz          4     20000      14
```

PERFORMING UPDATE OPERATION

```
SQL> insert into iview values ('bbb',5,30000,15);
1 row created.

SQL> select * from iview;
OUTPUT:    NAME          NO      SAL       DNO
          --------------------  ----------  -----------  -------------
             xxx                1    19000       11
             bbb                5    30000       15

SQL> select * from fviews;
OUTPUT:    NAME          NO      SAL       DNO
          --------------------  ----------  -----------  -------------
             xxx            1        19000       11
             aaa             2        19000       12
             yyy            3        40000       13
             zzz             4        20000       14
             bbb                5    30000       15
```

CREATE A NEW VIEW ‘SSVIEW’ AND DROP THE VIEW

```
SQL> create view ssview( cusname,id) as select name, no from fviews where dno=12;
View created.
SQL> select * from ssview;

OUTPUT:  CUSNAME    ID
         ------------------
              aaa          2

SQL> drop view ssview;
View dropped.
```

TO CREATE A VIEW ‘COMBO’ USING BOTH THE TABLES ‘FVIEWS’ AND ‘DVIEWS’

```
SQL> select * from combo;
NAME          NO      SAL       DNO    DNAME
----------------      ----------  -------------   -------------  ----------------
xxx                 1     19000       11        x
aaa                 2     19000       12        y
```

TO PERFORM MANIPULATIONS ON THIS VIEW

```
SQL> insert into combo values('ccc',12,1000,13,'x');
insert into combo values('ccc',12,1000,13,'x')
*
```

ERROR at line 1:
ORA-01779: cannot modify a column which maps to a non key-preserved table

This shows that when a view is created from two different tables no manipulations can be performed using that view and the above error is displayed.

SYNONYMS

Description:

- A *synonym* is an *alias*, that is, a form of shorthand used to simplify the task of referencing a database object.
- There are two categories of synonyms, *public* and *private*.

Syntax: CREATE SYNONYM privilege

Ex:

```
SQL> select * from class;
NAME            ID
-----------------------------------
anu              1
brindha          2
chinthiya        3
divya            4
ezhil            5
fairoz           7
hema             9
7 rows selected.

Create synonym:
SQL> create synonym c1 for class;
Synonym created.

SQL> insert into c1 values('kalai',20);
1 row created.

SQL> select * from class;
NAME            ID
-----------------------------------
anu              1
brindha          2
chinthiya        3
```

```
divya              4
ezhil              5
fairoz             7
hema               9
kalai             20
```

8 rows selected.

SQL> select * from c1;

```
OUTPUT:  NAME            ID
          -------      -------
         anu              1
         brindha          2
         chinthiya        3
         divya            4
         ezhil            5
         fairoz           7
         hema             9
         kalai           20
```

8 rows selected.

SQL> insert into class values('Manu',21);

1 row created.

SQL> select * from c1;

```
OUTPUT: NAME            ID
        --------      ------
          anu              1
         brindha           2
         chinthiya         3
         divya             4
         ezhil             5
         fairoz            7
         hema              9
         kalai            20
         Manu             21
```

9 rows selected.

Drop Synonym:

```
SQL> drop synonym c1;
Synonym dropped.

SQL> select * from c1;
select * from c1
              *
ERROR at line 1:
ORA-00942: table or view does not exist
```

SEQUENCES

Description

- Oracle provides the capability to generate sequences of unique numbers, and they are called **sequences.**
- Just like tables, views, indexes, and synonyms, a sequence is a type of database object.
- Sequences are used to generate unique, sequential integer values that are used as primary key values in database tables.
- The sequence of numbers can be generated in either ascending or descending order.

CREATION OF TABLE:

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

Insert values into table:

```
SQL> insert into class values('&name',&id);
      Enter value for name: anu
      Enter value for id: 1
OUTPUT: old 1: insert into class values('&name',&id)
             new 1: insert into class values('anu',1)
             1 row created.

SQL> /
Enter value for name: brindha
Enter value for id: 02
OUTPUT: old 1: insert into class values('&name',&id)
             new  1: insert into class values('brindha',02)
             1 row created.

SQL> /
```

```
Enter value for name: chinthiya
Enter value for id: 03
OUTPUT: old 1: insert into class values('&name',&id)
                 new 1: insert into class values('chinthiya',03)
                 1 row created.

SQL> select * from class;

OUTPUT:   NAME             ID
       ---------------------------------
         anu                  1
           brindha              2
           chinthiya            3
```

Create Sequence:

```
SQL> create sequence s_1
  2  start with 4
  3  increment by 1
  4  maxvalue 100
  5  cycle;
Sequence created.
```

```
SQL> insert into class values('divya',s_1.nextval);
1 row created.

SQL> select * from class;

OUTPUT:   NAME            ID
          - - - - - - -    - - - - - - -
          anu               1
          brindha            2
          chinthiya          3
          divya             4
```

Alter Sequence:

```
SQL> alter sequence s_1
2 increment by 2;
Sequence altered.

SQL> insert into class values('fairoz',s_1.nextval);
1 row created.
```

```
SQL> select * from class;
```

OUTPUT:

```
NAME            ID
-------     -------
anu              1
brindha          2
chinthiya        3
divya            4
ezhil            5
fairoz           7
```

Drop Sequence:

```
SQL> drop sequence s_1;
Sequence dropped.
```

INDEXES

Description

- An index can be created in a table to find data more quickly and efficiently.
- The users cannot see the indexes; they are just used to speed up searches/queries.
- Updating a table with indexes takes more time than updating a table without; because the indexes also need an update. So we should only create indexes on columns (and tables) that will be frequently searched against.

Syntax:

Create Index:

CREATE INDEX index_name ON table_name (column_name)

```
SQL> create table splr(sname varchar(10),sid number(10),scity varchar(10));
Table created.

SQL> insert into splr values('hcl',01,'chennai');
1 row created.

SQL> insert into splr values('dell',04,'madurai');
1 row created.

SQL> insert into splr values('HP',02,'kovai');
1 row created.

SQL> insert into splr values('Lenovo',03,'trichy');
1 row created.
```

```
SQL> select * from splr;
OUTPUT:   SNAME      ID   SCITY
          ---------------------
           hcl         1 chennai
          dell        4  madurai
           HP         2 kovai
          Lenovo      3  trichy
SQL> create index sp1 on splr(sid);
Index created.
SQL> create index sp2 on splr(sid,scity);
Index created.
```

Drop Index:

```
    SQL> drop index sp1;
    Index dropped.

    SQL> drop index sp2;
    Index dropped.
```

RESULT: Thus the commands for creating views,sequences,synonyms,index of oracle SQL were studied and the output was verified

Ex. No: 3(B)

DATE

TRANSACTION CONTROL LANGUAGE

AIM:

To study about the Transaction Control Language.

COMMANDS:

1.COMMIT:

DESCRIPTIION : The COMMIT statement makes permanent any changes made to the database during the current transaction.

SYNTAX :commit

EXAMPLE :Commit

2.SAVEPOINT

DESCRIPTION : SAVEPOINT establishes a new savepoint within the current transaction. A savepoint is a special mark inside a transaction that allows all commands that are executed after it was established to be rolled back, restoring the transaction state to what it was at the time of the savepoint.

SYNTAX : savepoint identifier;

EXAMPLE : savepoint empsave;

3.ROLLBACK

DESCRIPTION: The ROLLBACK statement in SQL cancels the proposed changes in a pending database transaction. The transaction can be rolled back completely by specifying the transaction name in the ROLLBACK statement.

SYNTAX :rollback to identifier

EXAMPLE :rollback to empsave

Example:

```
SQL> select * from borrow;
LOANN CNAME          BNAME              AMOUNT
- - -·- - - - - - - - - - - - - - - - - - - - - - - - - - -
l111  narmadha       tambaram           100000
l112  kavitha        chrompet           120000
l113  sri          guidy              10000
l114  harsha         saidapet           12000

SQL> update borrow set bname='vadapalani' where cname='narmadha';
1 row updated.
```

```
SQL> commit;
Commit complete.
SQL> savepoint dd;
Savepoint created.
SQL> update borrow set bname='mumbai' where cname='harsha';
1 row updated.
SQL> update borrow set cname='priya' where bname='mumbai';
1 row updated.
SQL> select * from borrow;
LOANN CNAME        BNAME          AMOUNT
------------------------------------
l111  narmadha     vadapalani     100000
l112  kavitha      chrompet       120000
l113  sri          guidy          10000
l114  priya        mumbai         12000
SQL> rollback to dd;
Rollback complete.
SQL> select * from borrow;
LOANN CNAME        BNAME          AMOUNT
------------------------------------
l111  narmadha     vadapalani     100000
l112  kavitha      chrompet       120000
l113  sri          guidy          10000
l114  harsha       saidapet       12000
```

RESULT: Thus the TCL command of oracle SQL were studied and the output was verified

EX. NO: 4

DATE: **CREATING RELATIONSHIP BETWEEN THE DATABASES.**

AIM

To create databases and implement the relationship between databases using join operation.

DESCRIPTION: JOIN OPERATIONS

INNER JOIN/ NATURAL JOIN: It is a binary operation that allows us to combine certain selections and a Cartesian product into one operation.

OUTER JOIN: It is an extension of join operation to deal with missing information.

- **Left Outer Join:** It takes tuples in the left relation that did not match with any tuple in the right relation, pads the tuples with null values for all other attributes from the right relation and adds them to the result of the natural join.
- **Right Outer Join:** It takes tuples in the right relation that did not match with any tuple in the left relation, pads the tuples with null values for all other attributes from the left relation and adds them to the result of the natural join.
- **Full Outer Join:** It combines tuples from both the left and the right relation and pads the tuples with null values for the missing attributes and hem to the result of the natural join.

CREATING TABLES
FOR DOING JOIN AND NESTED QUERY OPERATIONS

Creating Dept TABLE:

```
SQL> create table dept(dno number(10),dname varchar(10),loc varchar(10));
Table created.

SQL> insert into dept values(10,'inventory','hyd');
1 row created.

SQL> insert into dept values(20,'finance','bglr');
1 row created
.
SQL> insert into dept values(30,'HR','mumbai');
1 row created.

SQL> select * from dept;
OUTPUT:    DNO     DNAME       LOC
         ---------  ------------  ----------
            10      inventory    hyd
            20      finance      bglr
            30      HR           mumbai
```

Creating Emp2 TABLE:

```
SQL> create table emp2(eno number(10),ename varchar(10),job varchar(10),M
        number(10),dno number(10));
Table created.

SQL> insert into emp2 values(111,'saketh','analyst',444,10);
1 row created.

SQL> insert into emp2 values(222,'sandeep','clerk',333,20);
1 row created.

SQL> insert into emp2 values(333,'jagan','manager',111,10);
1 row created.

SQL> insert into emp2 values(444,'madhu','engineer',222,40);
1 row created.

SQL> select * from emp2;
OUTPUT:        ENO  ENAME    JOB        MGR    DNO
             ---------- ---------- ---------- ---------- ----------
                111  saketh    analyst     444      10
                222  sandeep   clerk       333      20
                333  jagan     manager     111      10
                444  madhu     engineer    222      40
```

EQUIJOIN:

A join which contains an equal to ‘=’ operator in this joins condition

```
SQL> select eno,ename,job,dname,loc from emp2 e,dept d where e.dno=d.dno;

OUTPUT:        ENO    ENAME      JOB       DNAME      LOC
               ---------- ------------- ----------- ---------------------------
               111     saketh      analyst   inventory   hyd
               222     sandeep     clerk     finance     bglr
               333     jagan       manager  inventory   hyd
```

Using Clause:

```
SQL> select eno,ename,job,dname,loc from emp2 e join dept d using(dno);

OUTPUT:        ENO    ENAME      JOB       DNAME      LOC
               ---------- ------------- ----------- ---------------------------
               111     saketh      analyst   inventory   hyd
               222     sandeep     clerk     finance     bglr
               333     jagan       manager  inventory   hyd
```

On Clause:

```
SQL> select eno,ename,job,dname,loc from emp2 e join dept d on(e.dno=d.dno);
OUTPUT:      ENO    ENAME      JOB       DNAME      LOC
             -----  --------   -------   -----------------
             111    saketh     analyst   inventory  hyd
             222    sandeep    clerk     finance    bglr
             333    jagan      manager   inventory  hyd
```

NON-EQUIJOIN:

A join which contains an operator other than equal to ‘=’ in the join condition.

```
SQL> select eno,ename,job,dname,loc from emp2 e,dept d where e.dno>d.dno;
OUTPUT:      ENO    ENAME      JOB       DNAME      LOC
                    --------   -------   ------------------
             222    sandeep    clerk     inventory  hyd
             444    madhu      engineer  inventory  hyd
             444    madhu      engineer  finance    bglr
             444    madhu      engineer  HR         Mumbai
```

SELF JOIN:

Joining the table itself is called self join.

```
SQL> select e1.eno,e2.ename,e1.job,e2.dno from emp2 e1,emp2 e2 where e1.eno=e2.eno;
OUTPUT:      ENO    ENAME      JOB       DNO
             -----  --------   -------   -----
             444    saketh     engineer   10
             333    sandeep    manager    20
             111    jagan      analyst    10
             222    madhu      clerk      40
```

NATURAL JOIN:

It compares all the common columns.

```
SQL> select eno, ename, job, dname, loc from emp2 natural join dept;

OUTPUT:      ENO    ENAME      JOB       DNAME      LOC
          ------------------   -------   -----------------
             111    saketh     analyst   inventory  hyd
             222    sandeep    clerk     finance    bglr
             333    jagan      manager   inventory  hyd
```

CROSS JOIN:

This will give the cross product.

```
SQL> select eno,ename,job,dname,loc from emp2 cross join dept;
```

```
OUTPUT:        ENO   ENAME      JOB       DNAME      LOC
             ------------------------------------------------- ----------------------------------
                111      saketh      analyst     inventory    hyd
                222      sandeep    clerk       inventory     hyd
                333      jagan       manager   inventory     hyd
                444      madhu      engineer    inventory     hyd
                111      saketh      analyst      finance         bglr
                222      sandeep    clerk         finance        bglr
                333      jagan       manager    finance        bglr
                444      madhu      engineer    finance        bglr
                111      saketh      analyst      HR             mumbai
                222      sandeep    clerk         HR             mumbai
                333      jagan       manager     HR             mumbai
                444      madhu      engineer     HR               mumbai
12 rows selected.
```

OUTER JOIN:

It gives the non matching records along with matching records.

Left Outer Join:

This will display the all matching records and the records which are in left hand side table those that are in right hand side table.

SQL> select eno,ename,job,dname,loc from emp2 e left outer join dept d on(e.dno= d.dno);

(OR)

SQL> select eno,ename,job,dname,loc from emp2 e,dept d where e.dno=d.dno(+);

```
OUTPUT:        ENO   ENAME      JOB      DNAME      LOC
               ------  ---------   ---------  ---------------------
               111     saketh       analyst    inventory    hyd
               222     sandeep     clerk       finance       bglr
               444     madhu       engineer
```

Right Outer Join:

This will display the all matching records and the records which are in right hand side table those that are not in left hand side table.

SQL> select eno,ename,job,dname,loc from emp2 e right outer join dept d on (e.dno =d.dno);

(OR)

SQL> select eno,ename,job,dname,loc from emp2 e,dept d where e.dno(+)=d.dno;

```
OUTPUT:        ENO   ENAME      JOB      DNAME      LOC
               ------  ---------   --------  ---------------------
               111     saketh       analyst    inventory    hyd
               222     sandeep     clerk       finance       bglr
```

```
222      sandeep      clerk       finance        bglr
333       jagan       manager     HR             mumbai
                                  HR             mumbai
```

Full Outer Join:

This will display the all matching records and the non matching records from both tables.

SQL> select eno,ename,job,dname,loc from emp2 e full outer join dept d on(e.dno= d.dno);

OUTPUT:

```
     ENO   ENAME     JOB       DNAME     LOC
---------------- -----------------   ---------------------------------------------------------
     333     jagan         manager   inventory     hyd
     111     saketh        analyst   inventory    hyd
     222     sandeep      clerk       finance       bglr
    444     madhu        engineer  HR                Mumbai
```

RESULT

Thus the relationship between databases has been implemented using join operation.

Ex.No.5 **SQL QUERIES**

DATE:

AIM

To create tables for **‘Employee Database’** and to form and execute simple queries in SQL .

PROBLEM DEFINITION

Create tables for the Employee Database in a company as per the constraints given below

Table: DEPARTMENTS

NAME	**NULL?**	**TYPE**
DEPT_ID	NOT NULL	NUMBER(3)
DEPT_NAME	NOT NULL	VARCHAR2(20)
SALARY		NUMBER(5)
LOCATION_ID	NOT NULL	NUMBER(3)

Table: EMPLOYEES

NAME	**NULL?**	**TYPE**
EMP_ID	NOT NULL	NUMBER(3)
FIRST_NAME	NOT NULL	VARCHAR2(10)
LAST_NAME		VARCHAR2(10)
GENDER	NOT NULL	VARCHAR2(1)
PHONE_NO		VARCHAR2(10)
HIRE_DATE	NOT NULL	DATE
CITY		VARCHAR2(10)
DEPT_ID	NOT NULL	NUMBER(3)
JOB_ID	NOT NULL	NUMBER(3)
DESIGNATION		VARCHAR2(15)

Table: JOBS

NAME	**NULL?**	**TYPE**
JOB_ID	NOT NULL	NUMBER(3)
JOB_TITLE		VARCHAR2(15)
START_DATE	NOT NULL	DATE
END_DATE	NOT NULL	DATE

Table: LOCATION

NAME	**NULL?**	**TYPE**
LOCATION_ID	NOT NULL	NUMBER(3)
LOCATION_NAME	NOT NULL	VARCHAR2(10)
CITY		VARCHAR2(10)

TABLE CREATION

EMPLOYEES TABLE

```
SQL>CREATE TABLE EMPLOYEES(EMP_ID NUMBER(3) PRIMARY KEY,
                           FIRST_NAME VARCHAR2(10) NOT NULL,
                           LAST_NAME VARCHAR2(10),
                           GENDER VARCHAR2(1) NOT NULL,
                           PHONE_NO NUMBER(10),
                           CITY VARCHAR2(10),
                           DEPT_ID NUMBER (3) NOT NULL,
                           JOB_ID NUMBER(3) NOT NULL,
                           DESIGNATION VARCHAR2(15));
```

Table created.

DEPARTMENTS TABLE

```
SQL> CREATE TABLE DEPARTMENTS (DEPT_ID NUMBER(3) PRIMARY KEY,
                           DEPT_NAME VARCHAR2(20) NOT NULL,
                           SALARY NUMBER(5),
                       LOCATION_ID NUMBER(3) NOT NULL);
```

Table created.

JOBS TABLE

```
SQL> CREATE TABLE JOBS(JOB_ID NUMBER(3) PRIMARY KEY,
                           JOB_NAME VARCHAR2(15),
                           START_DATE DATE NOT NULL,
                           END_DATE DATE NOT NULL);
```

Table created.

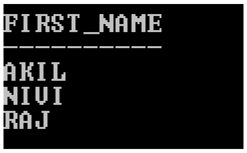
LOCATION TABLE

```
SQL> CREATE TABLE LOCATIONS(LOCATION_ID NUMBER(3) PRIMARY KEY,
                           LOCATION_NAME VARCHAR2(10) NOT NULL,
                           CITY VARCHAR2(10));
```

Table created.

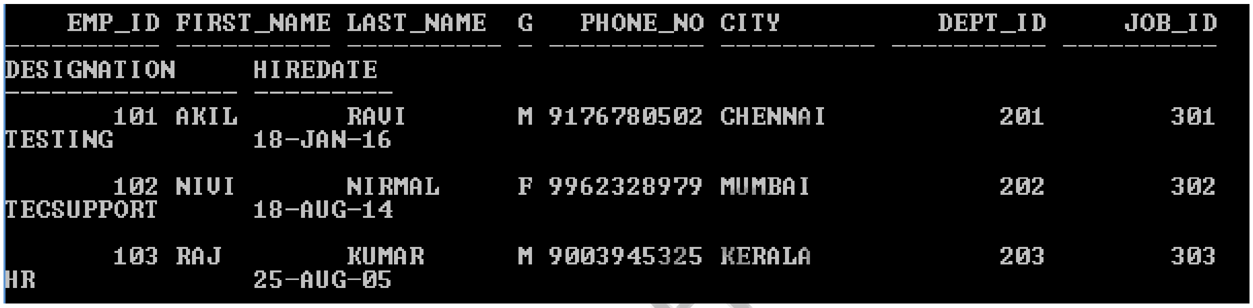
QUERIES

1. List the First names of all the employees.
   SQL> SELECT FIRST_NAME FROM EMPLOYEES;



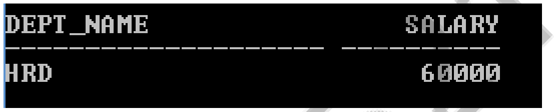
```
FIRST_NAME
----------
AKIL
NIVI
RAJ
```

2. Print the entire Employee table.
   SQL> SELECT * FROM EMPLOYEES;



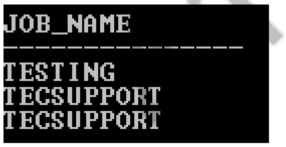
```
    EMP_ID FIRST_NAME LAST_NAME  G   PHONE_NO CITY           DEPT_ID     JOB_ID
---------- ---------- ---------- - ---------- ---------- ---------- ----------
DESIGNATION     HIREDATE
--------------- ---------
       101 AKIL       RAVI       M 9176780502 CHENNAI           201        301
TESTING         18-JAN-16

       102 NIVI       NIRMAL     F 9962328979 MUMBAI            202        302
TECSUPPORT      18-AUG-14

       103 RAJ        KUMAR      M 9003945325 KERALA            203        303
HR              25-AUG-05
```

3. Retrieve the Dept.name and the maximum salary in each department.
   SQL> SELECT DEPT_NAME, SALARY FROM DEPARTMENTS WHERE SALARY=(SELECT MAX(SALARY)FROM DEPARTMENTS);



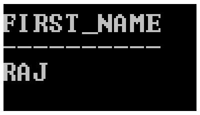
```
DEPT_NAME                SALARY
-------------------- ----------
HRD                       60000
```

4. List the various jobs positions available in the Company
   SQL> SELECT JOB_NAME FROM JOBS;



```
JOB_NAME
---------------
TESTING
TECSUPPORT
TECSUPPORT
```

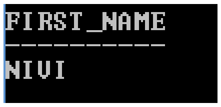
5. Find the names of all the clients having 'a' as the second letter in their names.
   SQL> SELECT FIRST_NAME FROM EMPLOYEES
   WHERE FIRST_NAME LIKE '_A%';



```
FIRST_NAME
----------
RAJ
```

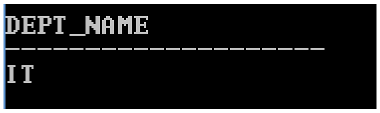
6. Find the first names of all the clients whose names start with 'N'.
   SQL> SELECT FIRST_NAME FROM EMPLOYEES

WHERE FIRST_NAME LIKE 'N%';



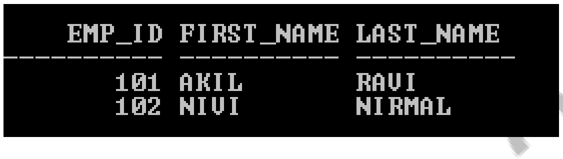
```
FIRST_NAME
----------
NIVI
```

7. Find the first names of all the departments that end with ‘T’.
   SQL> SELECT DEPT_NAME FROM DEPARTMENTS WHERE DEPT_NAME LIKE'%T';



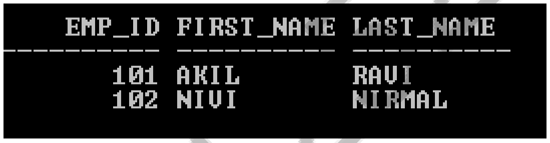
```
DEPT_NAME
--------------------
IT
```

8. List the emp_id, names and phone numbers of employees who stay either in ‘MUMBAI’ or ‘CHENNAI’
   SQL> SELECT EMP_ID, FIRST_NAME, LAST_NAME FROM EMPLOYEES WHERE CITY='MUMBAI' OR CITY='CHENNAI';



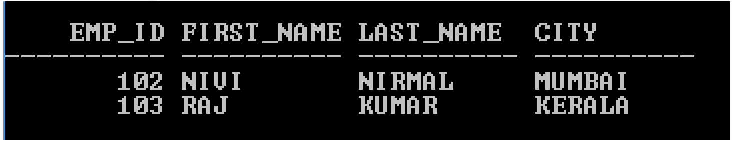
```
    EMP_ID FIRST_NAME LAST_NAME
---------- ---------- ----------
       101 AKIL       RAVI
       102 NIVI       NIRMAL
```

9. Find all the employees whose city names ends with the letter ‘a’ second from the last.
   SQL> SELECT EMP_ID, FIRST_NAME, LAST_NAME FROM EMPLOYEES WHERE CITY LIKE '%A_';



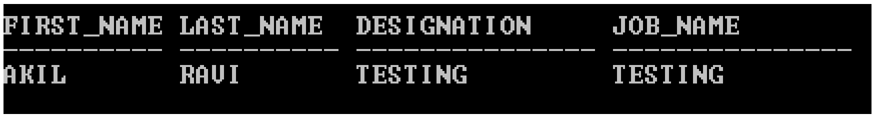
```
    EMP_ID FIRST_NAME LAST_NAME
---------- ---------- ----------
       101 AKIL       RAVI
       102 NIVI       NIRMAL
```

10. Find names and city of the employees who joined in the month of December.
    SQL> SELECT EMP_ID,FIRST_NAME,LAST_NAME,CITY FROM EMPLOYEES WHERE TO_CHAR(HIRE_DATE,'MON')=TO_CHAR('AUG');



```
    EMP_ID FIRST_NAME LAST_NAME  CITY
---------- ---------- ---------- ----------
       102 NIVI       NIRMAL     MUMBAI
       103 RAJ        KUMAR      KERALA
```

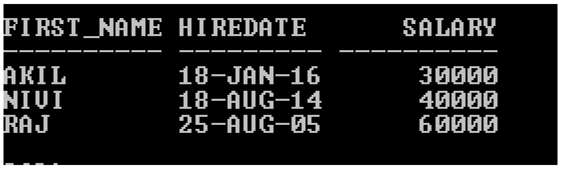
11. Display Employee information as well as Job information for employees with ID 104 and 102
    SQL>SELECT EMPLOYEES.FIRST_NAME,EMPLOYEES.LAST_NAME, EMPLOYEES.DESIGNATION, JOBS.JOB_NAME FROM EMPLOYEES, JOBS WHERE EMPLOYEES.JOB_ID=JOBS.JOB_ID AND (EMPLOYEES.EMP_ID=104 OR EMPLOYEES.EMP_ID=101);



```
FIRST_NAME LAST_NAME  DESIGNATION     JOB_NAME
---------- ---------- --------------- ---------------
AKIL       RAVI       TESTING         TESTING
```

12. Find the first names and hire date of employees with salary greater than 10000.

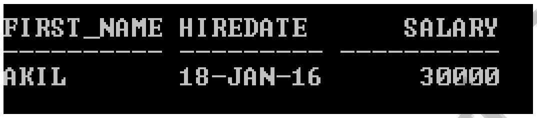
SQL>SELECT EMPLOYEES.FIRST_NAME, EMPLOYEES.HIRE_DATE, DEPARTMENTS.SALARY FROM EMPLOYEES,DEPARTMENTS WHERE EMPLOYEES.DEPT_ID=DEPARTMENTS.DEPT_ID AND DEPARTMENTS.SALARY>10000;



```
FIRST_NAME HIREDATE      SALARY
---------- --------- ----------
AKIL       18-JAN-16      30000
NIVI       18-AUG-14      40000
RAJ        25-AUG-05      60000
```

13. Find the names and hire date of employees with salary greater than 10000 and less than 40000.

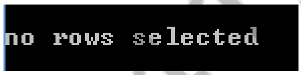
SQL> SELECT EMPLOYEES.FIRST_NAME, EMPLOYEES.HIREDATE, DEPARTMENTS.SALARY FROM EMPLOYEES, DEPARTMENTS WHERE EMPLOYEES.DEPT_ID=DEPARTMENTS.DEPT_ID AND DEPARTMENTS.SALARY > 10000 AND SALARY<40000;



```
FIRST_NAME HIREDATE      SALARY
---------- --------- ----------
AKIL       18-JAN-16      30000
```

14. List the names of the employees whose salary is less than 20000 and list out the original salary multiplied by 1.5 under the new heading 'Hiked_Salary'.

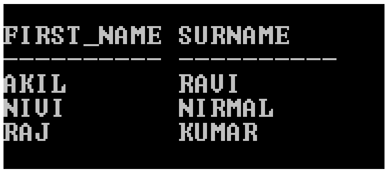
SQL>SELECT EMPLOYEES.FIRST_NAME, DEPARTMENTS.SALARY, DEPARTMENTS.SALARY*1.5 HIKED_SALARY FROM EMPLOYEES, DEPARTMENTS WHERE EMPLOYEES.DEPT_ID=DEPARTMENTS.DEPT_ID AND DEPARTMENTS.SALARY<20000;



```
no rows selected
```

15. Select first name, last name from employees table and rename the column last name as 'Surname'.

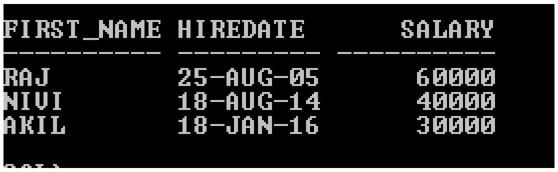
SQL> SELECT FIRST_NAME, LAST_NAME SURNAME FROM EMPLOYEES;



```
FIRST_NAME SURNAME
---------- ----------
AKIL       RAVI
NIVI       NIRMAL
RAJ        KUMAR
```

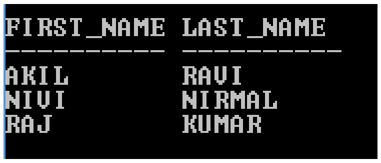
16. List all the name, hire date, emp_id and salary in descending order of salary.

SQL> SELECT EMPLOYEES.FIRST_NAME, EMPLOYEES.HIREDATE, DEPARTMENTS.SALARY FROM EMPLOYEES,DEPARTMENTS WHERE EMPLOYEES.DEPT_ID=DEPARTMENTS.DEPT_ID ORDER BY SALARY DESC;



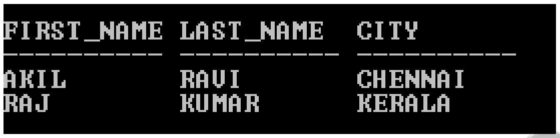
```
FIRST_NAME HIREDATE      SALARY
---------- --------- ----------
RAJ        25-AUG-05      60000
NIVI       18-AUG-14      40000
AKIL       18-JAN-16      30000
```

17. Select the names, city and phone number of the employees who do not live in London.
    SQL> SELECT FIRST_NAME, LAST_NAME FROM EMPLOYEES WHERE CITY <> 'LONDON';



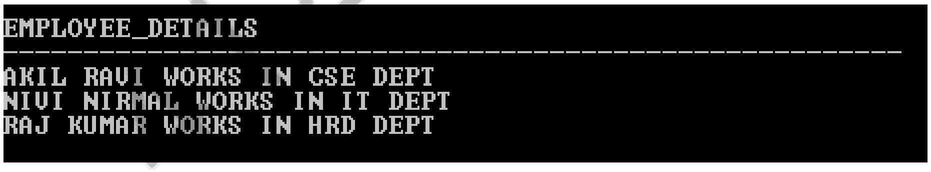
```
FIRST_NAME LAST_NAME
---------- ----------
AKIL       RAVI
NIVI       NIRMAL
RAJ        KUMAR
```

18. Select the names and phone numbers of employees whose city does not begin with the letter ‘M’.
    SQL> SELECT FIRST_NAME, LAST_NAME, CITY FROM EMPLOYEES WHERE CITY NOT LIKE 'M%';



```
FIRST_NAME LAST_NAME  CITY
---------- ---------- ----------
AKIL       RAVI       CHENNAI
RAJ        KUMAR      KERALA
```

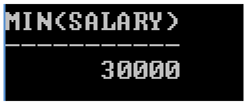
19. Print the names of all the employees in the following format
    EMPLOYEE_DETAILS
    -
    EMP_NAME   WORKS IN   DEPT_NAME
    SQL>SELECT CONCAT(CONCAT(FIRST_NAME,' '), LAST_NAME) ||' WORKS IN ' || CONCAT(DEPT_NAME,' DEPT') EMPLOYEE_DETAILS FROM EMPLOYEES, DEPARTMENTS WHERE EMPLOYEES.DEPT_ID=DEPARTMENTS.DEPT_ID;



```
EMPLOYEE_DETAILS
-----------------------------------------------------------
AKIL RAVI WORKS IN CSE DEPT
NIVI NIRMAL WORKS IN IT DEPT
RAJ KUMAR WORKS IN HRD DEPT
```

Using functions

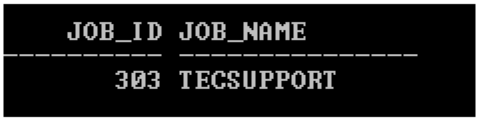
20. Find out the minimum of the salary.
    SQL> SELECT MIN(SALARY) FROM DEPARTMENTS;



```
MIN(SALARY)
-----------
      30000
```

21. List all the jobs that were started in the month of August.
    SQL> SELECT JOB_ID, JOB_NAME FROM JOBS WHERE

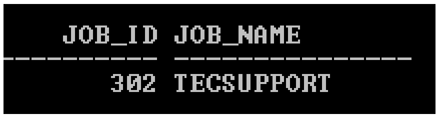
TO_CHAR(START_DATE,'MON')='AUG';



```
    JOB_ID JOB_NAME
---------- ---------------
       303 TECSUPPORT
```

22. List all the jobs that were finished in the month of December.
    SQL> SELECT JOB_ID, JOB_NAME FROM JOBS WHERE

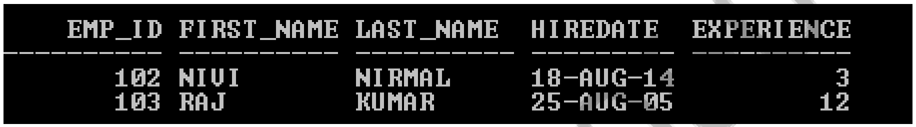
TO_CHAR(END_DATE,'MON')='DEC';



```
    JOB_ID JOB_NAME
---------- ---------------
       302 TECSUPPORT
```

23. Select the names,emp_id,hire_date of the employees who have an experience of two or more years and display their experience.
    SQL> SELECT EMP_ID, FIRST_NAME, LAST_NAME, HIREDATE,
    ROUND ((MONTHS_BETWEEN(SYSDATE,HIREDATE))/12) EXPERIENCE

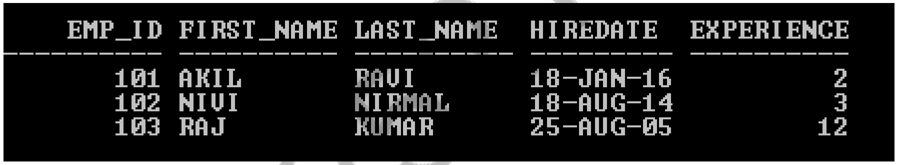
FROM EMPLOYEES WHERE MONTHS_BETWEEN(SYSDATE,HIREDATE) >= 24;



```
    EMP_ID FIRST_NAME LAST_NAME  HIREDATE  EXPERIENCE
---------- ---------- ---------- --------- ----------
       102 NIVI       NIRMAL     18-AUG-14          3
       103 RAJ        KUMAR      25-AUG-05         12
```

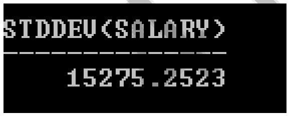
24. Select the names of the employees who have worked for not more than three months.
    SQL> SELECT EMP_ID, FIRST_NAME, LAST_NAME, HIREDATE,

ROUND((MONTHS_BETWEEN(SYSDATE,HIREDATE))/12) EXPERIENCE FROM EMPLOYEES WHERE MONTHS_BETWEEN(SYSDATE,HIREDATE)<>3;



```
    EMP_ID FIRST_NAME LAST_NAME  HIREDATE  EXPERIENCE
---------- ---------- ---------- --------- ----------
       101 AKIL       RAVI       18-JAN-16          2
       102 NIVI       NIRMAL     18-AUG-14          3
       103 RAJ        KUMAR      25-AUG-05         12
```

25. Calculate the standard deviation of the salaries of the employees.
    SQL> SELECT STDDEV(SALARY) FROM DEPARTMENTS;

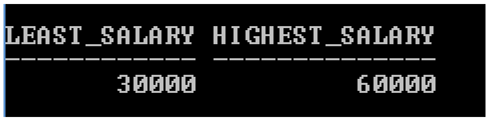


```
STDDEV(SALARY)
--------------
    15275.2523
```

26. Determine the minimum and maximum salary and rename them as Lowest_Salary and Highest_Salary.
    SQL> SELECT MIN(SALARY) LEAST_SALARY, MAX(SALARY)

HIGHEST_SALARY

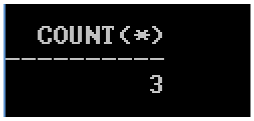
FROM DEPARTMENTS;



```
LEAST_SALARY HIGHEST_SALARY
------------ --------------
       30000          60000
```

27. Calculate the number of Employees whose salary is greater than 10000.

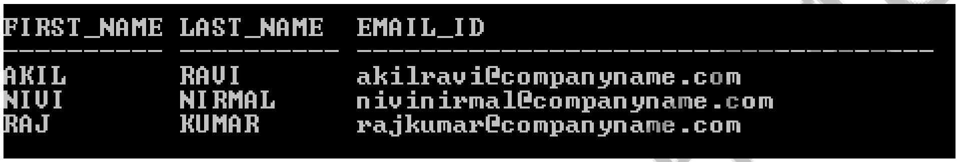
SQL> SELECT COUNT(*) FROM EMPLOYEES,DEPARTMENTS WHERE EMPLOYEES.DEPT_ID=DEPARTMENTS.DEPT_ID AND SALARY>10000;



```
  COUNT(*)
----------
         3
```

28. List the names, Phone number, Email Id of employees who joined in the month of February and print the sum of all their salaries.
Give each employee an E-mail account. The username should be the employee’s fullname@companyname.com. Username should be in lower case.

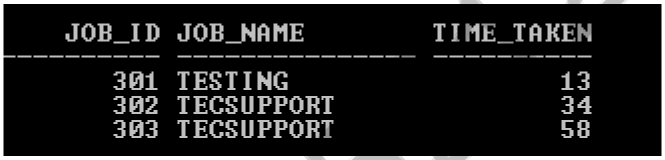
SQL> SELECT FIRST_NAME, LAST_NAME, LOWER (CONCAT (CONCAT (FIRST_NAME, LAST_NAME), '@COMPANYNAME.COM')) EMAIL_ID FROM EMPLOYEES;



```
FIRST_NAME LAST_NAME  EMAIL_ID
---------- ---------- ----------------------------------------
AKIL       RAVI       akilravi@companyname.com
NIVI       NIRMAL     nivinirmal@companyname.com
RAJ        KUMAR      rajkumar@companyname.com
```

29. Find the time taken for each job

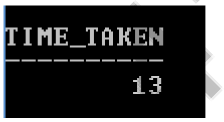
SQL>SELECT JOB_ID, JOB_NAME, ROUND (MONTHS_BETWEEN (END_DATE, START_DATE)) TIME_TAKEN FROM JOBS;



```
    JOB_ID JOB_NAME        TIME_TAKEN
---------- --------------- ----------
       301 TESTING                 13
       302 TECSUPPORT              34
       303 TECSUPPORT              58
```

30. Find the Job which took least time

SQL> SELECT MIN(ROUND(MONTHS_BETWEEN(END_DATE,START_DATE))) AS TIME_TAKEN FROM JOBS;

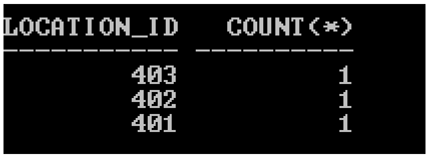


```
TIME_TAKEN
----------
        13
```

HAVING GROUP BY

31. List the number of departments in each location;

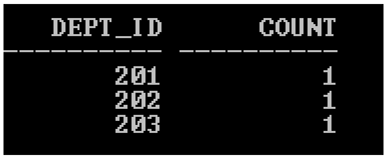
SQL> SELECT LOCATIONS.LOCATION_ID, COUNT(*) FROM DEPARTMENTS,
LOCATIONS WHERE DEPARTMENTS.LOCATION_ID=LOCATIONS.LOCATION_ID GROUP BY(LOCATIONS.LOCATION_ID);



```
LOCATION_ID   COUNT(*)
----------- ----------
        403          1
        402          1
        401          1
```

32. Find the number of employees in each dept.

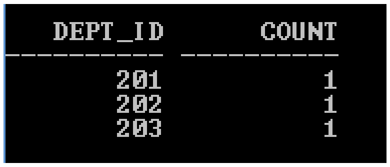
SQL> SELECT DEPT_ID, COUNT(*) AS COUNT FROM EMPLOYEES GROUP BY DEPT_ID;



```
   DEPT_ID      COUNT
---------- ----------
       201          1
       202          1
       203          1
```

33. Find the number of employees working on each job.

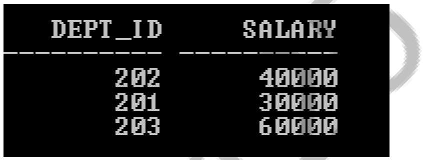
SQL> SELECT DEPT_ID, COUNT(*) AS COUNT FROM EMPLOYEES GROUP BY DEPT_ID;



```
   DEPT_ID      COUNT
---------- ----------
       201          1
       202          1
       203          1
```

34. Find the salary disbursed in each department.

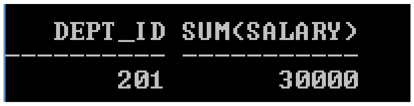
SQL> SELECT DEPT_ID, SALARY FROM DEPARTMENTS WHERE DEPT_ID IN(SELECT DEPT_ID FROM DEPARTMENTS GROUP BY DEPT_ID)*;*



```
   DEPT_ID     SALARY
---------- ----------
       202      40000
       201      30000
       203      60000
```

35. Find the salary disbursed in departments 201 and 204

SQL>SELECT DEPT_ID,SUM(SALARY) FROM DEPARTMENTS GROUP BY(DEPT_ID) HAVING DEPT_ID=201 OR DEPT_ID=204;



```
   DEPT_ID SUM(SALARY)
---------- -----------
       201       30000
```

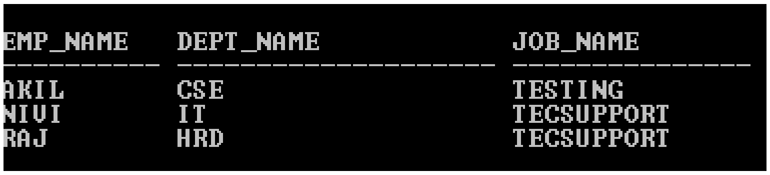
JOINS AND CORRELATIONS

36. Print the information of employee name, dept name and job name for all the employees in the following format.

EMP_NAME   DEPT_NAME   JOB_NAME   SALARY

SQL> SELECT FIRST_NAME EMP_NAME, DEPT_NAME,JOB_NAME FROM EMPLOYEES,JOBS,DEPARTMENTS WHERE

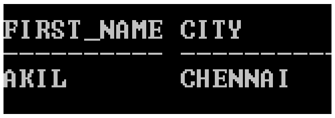
EMPLOYEES.DEPT_ID=DEPARTMENTS.DEPT_ID AND EMPLOYEES.JOB_ID=JOBS.JOB_ID;



```
EMP_NAME   DEPT_NAME            JOB_NAME
---------- -------------------- ---------------
AKIL       CSE                  TESTING
NIVI       IT                   TECSUPPORT
RAJ        HRD                  TECSUPPORT
```

37. Select the names of the employees who live in the city CHENNAI and who work in CSE department

SQL>SELECT FIRST_NAME, CITY FROM EMPLOYEES, DEPARTMENTS WHERE CITY='CHENNAI' AND DEPT_NAME='CSE';



```
FIRST_NAME CITY
---------- ----------
AKIL       CHENNAI
```

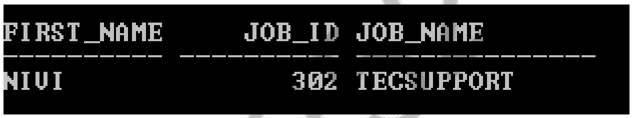
NESTED QUERIES

38. Find out the Job id and job name done by "THOMAS"

SQL> SELECT JOB_ID,JOB_NAME FROM JOBS WHERE JOB_ID IN(SELECT JOB_ID FROM EMPLOYEES WHERE FIRST_NAME='NIVI');

(OR)

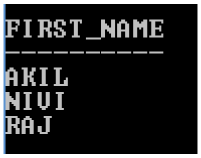
SQL> SELECT FIRST_NAME,JOBS.JOB_ID,JOB_NAME FROM JOBS,EMPLOYEES
WHERE JOBS.JOB_ID = EMPLOYEES.JOB_ID AND FIRST_NAME='NIVI';



```
FIRST_NAME     JOB_ID JOB_NAME
---------- ---------- ---------------
NIVI              302 TECSUPPORT
```

39. Select the names of employees who have worked on a job for more than 12 months

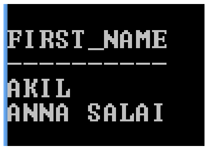
SQL>SELECT FIRST_NAME FROM EMPLOYEES WHERE EMPLOYEES.JOB_ID
IN(SELECT JOB_ID FROM JOBS WHERE ROUND
(MONTHS_BETWEEN(END_DATE,START_DATE))>12);



```
FIRST_NAME
----------
AKIL
NIVI
RAJ
```

40. Select the names of employees or names of locations whose city is Chennai.

SQL> SELECT FIRST_NAME FROM EMPLOYEES WHERE EMPLOYEES.CITY = 'CHENNAI' UNION SELECT LOCATION_NAME AS LOC FROM LOCATIONS WHERE CITY='CHENNAI';



```
FIRST_NAME
----------
AKIL
ANNA SALAI
```

RESULT

The Employee database was created successfully as per the constraints given and simple queries were studied and executed.

Ex. No: 6 Study of PL/SQL block

DATE:

AIM

To study about PL/SQL block in database management systems

DESCRIPTION

PL/SQL PROGRAMMING

Procedural Language/Structured Query Language (PL/SQL) is an extension of SQL.

Basic Syntax of PL/SQL

```
DECLARE
/* Variables can be declared here */
BEGIN
/* Executable statements can be written here */
EXCEPTION
/* Error handlers can be written here. */
END;
```

STEPS TO WRITE & EXECUTE PL/SQL

- As we want output of PL/SQL Program on screen, before Starting writing anything type (Only Once per session)

```
SQL> SET SERVEROUTPUT ON
```

- To write program, use Notepad through Oracle using ED command.

```
SQL> ED ProName
```

Type the program Save & Exit.

- To Run the program

```
SQL> @ProName
```

(i) TO DISPLAY HELLO MESSAGE

```
SQL> set serveroutput on;

SQL> declare
  2  a varchar2(20);
  3  begin
  4  a:='Hello';
  5  dbms_output.put_line(a);
  6  end;
  7 /
```

```
Hello

PL/SQL procedure successfully completed.
```

(ii).Insert the record into Sailors table by reading the values from the Keyboard.

```
SQL> create table sailors(sid number(10),sname varchar(10),rating number(10),age
 number(10));

Table created.

SQL> set serveroutput on
SQL> declare
  2  sid number(10):=&sid;
  3  sname varchar(10):='&sname';
  4  rating number(10):=&rating;
  5  age number(10):=&age;
  6  begin
  7  insert into sailors values(sid,sname,rating,age);
  8  end;
  9 /
Enter value for sid: 02
old 2: sid number(10):=&sid;
new 2: sid number(10):=02;
Enter value for sname: lavanya
old 3: sname varchar(10):='&sname';
new  3: sname varchar(10):='lavanya';
Enter value for rating: 01
old 4: rating number(10):=&rating;
new 4: rating number(10):=01;
Enter value for age: 25
old 5: age number(10):=&age;
new 5: age number(10):=25;

PL/SQL procedure successfully completed.


SQL> select * from sailors;

       SID SNAME          RATING        AGE
- - - - - - - - - - - - - - - - - - - - - - - - - - - -
         2 lavanya            1          25
         3 vani              2          25
```

RESULT

Thus the PL/SQL block has been studied and implemented.

Ex. No: 7 Control Blocks in PL/SQL

DATE:

AIM

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

PROGRAMS

(i) ADDITION OF 2 NUMBERS

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

INPUT:

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

OUTPUT:

```
sum of23and12is35
PL/SQL procedure successfully completed.
```

(ii) GREATEST OF THREE NUMBERS USING IF ELSE

```
SQL> declare
 a number;
b number;
c number;
d number;
```

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

INPUT:

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

OUTPUT:

```
C is maximum
PL/SQL procedure successfully completed.
```

(iii) SUMMATION OF ODD NUMBERS USING FOR LOOP

```
SQL> declare
n number;
sum1 number default 0;
endvalue  number;
begin
endvalue:=&endvalue;
n:=1;
```

```
for n in 1..endvalue
loop
  if mod(n,2)=1
then
sum1:=sum1+n;
end if;
 end  loop;
dbms_output.put_line('sum ='||sum1);
end;
  /
```

INPUT:

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

OUTPUT:

```
  sum =4
PL/SQL procedure successfully completed.
```

(iv) SUMMATION OF ODD NUMBERS USING WHILE LOOP

```
SQL> declare
n number;
sum1 number default 0;
endvalue  number;
begin
endvalue:=&endvalue;
n:=1;
while(n<endvalue)
loop
sum1:=sum1+n;
n:=n+2;
end loop;
dbms_output.put_line('sum of odd no. bt 1 and' ||endvalue||'is'||sum1);
end;
```

```
/
```

INPUT:

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

OUTPUT:

```
sum of odd no. bt 1 and4is4
PL/SQL procedure successfully completed.
```

(v) To Update The Database From Pl/Sql

To Debit The Amount From The Account Table

To Create Saccount Table

```
SQL> create table saccount ( accno number(5), name varchar2(20), bal number(10));

Table created.

SQL> insert into saccount values ( 1,'mala',20000);

1 row created.

SQL> insert into saccount values (2,'kala',30000);

1 row created.

SQL> select * from saccount;

    ACCNO NAME                    BAL
------ --------------------- -------

        1 mala                  20000

        2 kala                  30000

SQL> set serveroutput on;

SQL> declare

  2  a_bal number(7);

  3  a_no varchar2(20);
  4  debit number(7):=2000;
  5  minamt number(7):=500;
  6  begin
  7  a_no:=&a_no;
  8  select bal into a_bal from saccount where accno= a_no;
  9  a_bal:= a_bal-debit;
 10  if (a_bal > minamt) then
 11  update saccount  set bal=bal-debit where accno=a_no;
 12  end if;
 13  end;
 14
 15 /
Enter value for a_no: 1
old 7: a_no:=&a_no;
new  7: a_no:=1;
PL/SQL procedure successfully completed.
SQL> select * from saccount;
  ACCNO NAME                   BAL
------ -------------------- ------
       1 mala                18000
       2 kala                30000
```

(vi).To Update The Fairs Of The Routes Table

To Create Table Sroutes

```
SQL> create table sroutes ( rno number(5), origin varchar2(20), destination varchar2(20), fare numbe
r(10), distance number(10));
Table created.
```

```
SQL> insert into sroutes values ( 2, 'chennai', 'dindugal', 400,230);

1 row created.

SQL> insert into sroutes values ( 3, 'chennai', 'madurai', 250,300);

1 row created.

SQL> insert into sroutes values ( 6, 'thanjavur', 'palani', 350,370);

1 row created.

SQL> select * from sroutes;

   RNO ORIGIN          DESTINATION          FARE DISTANCE

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

     2 chennai         dindugal              400      230

     3 chennai         madurai               250      300

     6 thanjavur       palani               350      370

SQL> set serveroutput on;

SQL> declare

  2  route sroutes.rno % type;

  3  fares sroutes.fare % type;

  4  dist sroutes.distance % type;

  5  begin

  6  route:=&route;

  7  select fare, distance into fares , dist from sroutes where rno=route;

  8  if (dist < 250) then

  9  update sroutes set fare=300 where rno=route;

 10  else if dist between 250 and 370 then

 11  update sroutes set fare=400 where rno=route;

 12  else if (dist > 400) then

 13  dbms_output.put_line('Sorry');
 14  end if;
 15  end if;
 16  end if;
 17  end;
 18 /
Enter value for route: 3
old 6: route:=&route;
new  6: route:=3;
PL/SQL procedure successfully completed.
SQL> select * from sroutes;
      RNO ORIGIN              DESTINATION              FARE DISTANCE
------ ------------- ------------- ---- ------
        2 chennai            dindugal                 400       230
        3 chennai            madurai                  400       300
        6 thanjavur           palani                 350       370
```

RESULT:

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

Ex. No: 8 **EXCEPTION HANDLING IN PL/SQL**

Date:

AIM:

To write PL/SQL blocks that handles all types of exceptions

EXCEPTION

Description:

An error condition during a program execution is called an exception in PL/SQL. PL/SQL supports programmers to catch such conditions using **EXCEPTION** block in the program and an appropriate action is taken against the error condition

STRUCTURE OF EXCEPTION HANDLING

Syntax for Exception Handling:

The General Syntax for exception handling is as follows. Here you can list down as many as exceptions you want to handle. The default exception will be handled using *WHEN others THEN*

```
DECLARE
   <declarations section>
BEGIN
   <executable command(s)>
EXCEPTION
   <exception handling goes here >
   WHEN exception1 THEN
       exception1-handling-statements
   WHEN exception2 THEN
      exception2-handling-statements
   WHEN exception3 THEN
      exception3-handling-statements
   ........
   WHEN others THEN
      exception3-handling-statements
END;
```

RAISING EXCEPTIONS

DESCRIPTION:

The database server raises exceptions automatically whenever there is any internal database error, also exceptions can be raised explicitly by the programmer by using the command **RAISE**.

SYNTAX FOR RAISING EXCEPTIONS:

```
DECLARE
  exception_name EXCEPTION;
BEGIN
  IF condition THEN
    RAISE exception_name;
  END IF;
EXCEPTION
  WHEN exception_name THEN
  statement;
END;
```

You can use above syntax in raising Oracle standard exception or any user-defined exception. Next section will give you an example on raising user-defined exception, similar way you can raise Oracle standard exceptions as well.

TYPES OF EXCEPTIONS:

- **User defined exceptions**
- **Predefined exceptions**

User-defined Exceptions

DESCRIPTION:

PL/SQL allows you to define your own exceptions according to the need of your program. A user-defined exception must be declared and then raised explicitly, using either a RAISE statement or the procedure DBMS_STANDARD.RAISE_APPLICATION_ERROR.

The syntax for declaring an exception is:

```
DECLARE
        my-exception EXCEPTION;
```

Pre-defined Exceptions

DESCRIPTION:

PL/SQL provides many pre-defined exceptions, which are executed when any database rule is violated by a program. For example, the predefined exception NO_DATA_FOUND is raised when a SELECT INTO statement returns no rows. The following table lists few of the important pre-defined exceptions:

SOME COMMON PREDEFINED EXCEPTIONS:

Example

Exception	**Oracle Error**	**SQLCODE**	**Description**
ACCESS_INTO_NULL	06530	-6530	It is raised when a null object is automatically assigned a value.
CASE_NOT_FOUND	06592	-6592	It is raised when none of the choices in the WHEN clauses of a CASE statement is selected, and there is no ELSE clause.
COLLECTION_IS_NULL	06531	-6531	It is raised when a program attempts to apply collection methods other than EXISTS to an uninitialized nested table or array, or the program attempts to assign values to the elements of an uninitialized nested table or array.
DUP_VAL_ON_INDEX	00001	-1	It is raised when duplicate values are attempted to be stored in a column with unique index.
INVALID_CURSOR	01001	-1001	It is raised when attempts are made to make a cursor operation that is not allowed, such as closing an unopened cursor.
INVALID_NUMBER	01722	-1722	It is raised when the conversion of a character string into a number fails because the string does not represent a valid number.
LOGIN_DENIED	01017	-1017	It is raised when s program attempts to log on to the database with an invalid username or password.
NO_DATA_FOUND	01403	+100	It is raised when a SELECT INTO statement returns no rows.
NOT_LOGGED_ON	01012	-1012	It is raised when a database call is issued without being connected to the database.
PROGRAM_ERROR	06501	-6501	It is raised when PL/SQL has an internal problem.
ROWTYPE_MISMATCH	06504	-6504	It is raised when a cursor fetches value in a variable having incompatible data type.
SELF_IS_NULL	30625	-30625	It is raised when a member method is invoked, but the instance of the object type was not initialized.

STORAGE_ERROR	06500	-6500	It is raised when PL/SQL ran out of memory or memory was corrupted.
TOO_MANY_ROWS	01422	-1422	It is raised when s SELECT INTO statement returns more than one row.
VALUE_ERROR	06502	-6502	It is raised when an arithmetic, conversion, truncation, or size-constraint error occurs.
ZERO_DIVIDE	01476	1476	It is raised when an attempt is made to divide a number by zero

PROGRAM ILLUSTRATING EXCEPTION HANDLING

```
DECLARE
  c_id customers.id%type := 8;
  c_name customers.name%type;
  c_addr customers.address%type;
BEGIN
  SELECT name, address INTO c_name, c_addr
  FROM customers
  WHERE id = c_id;

  DBMS_OUTPUT.PUT_LINE ('Name: '|| c_name);
  DBMS_OUTPUT.PUT_LINE ('Address: ' || c_addr);
EXCEPTION
  WHEN no_data_found THEN
    dbms_output.put_line('No such customer!');
  WHEN others THEN
    dbms_output.put_line('Error!');
END;
/
```

OUTPUT:

```
NO such customer!
PL/SQL procedure successfully completed.
```

PROGRAM ILLUSTRATING PREDEFINED EXCEPTION

```
CREATE OR REPLACE PROCEDURE add_new_supplier
  (supplier_id_in IN NUMBER, supplier_name_in IN VARCHAR2)
IS

BEGIN
  INSERT INTO suppliers (supplier_id, supplier_name )
  VALUES ( supplier_id_in, supplier_name_in );

EXCEPTION
```

```
  WHEN DUP_VAL_ON_INDEX THEN
        raise_application_error (-20001,'You have tried to insert a duplicate supplier_id.');

  WHEN OTHERS THEN
     raise_application_error (-20002,'An error has occurred inserting a supplier.');

END;
```

OUTPUT:

```
exec add_new_supplier(10,’ram’);
begin add_new_supplier(10,’ram’);end;

    *
error at line1:
ORA-20001:You have tried to insert a duplicate supplier_id
ORA-06512:at”s3itc18.ADD_NEW_SUPPLIER”,line 11
ORA_06512:at line 1
```

PROGRAM ILLUSTRATING USER DEFINED EXCEPTION

```
DECLARE

  c_id customers.id%type := &cc_id;
  c_name customers.name%type;
  c_addr customers.address%type;

  -- user defined exception
  ex_invalid_id  EXCEPTION;
BEGIN
  IF c_id <= 0 THEN
     RAISE ex_invalid_id;
  ELSE
     SELECT name, address INTO c_name, c_addr
     FROM customers
     WHERE id = c_id;

     DBMS_OUTPUT.PUT_LINE ('Name: '|| c_name);
     DBMS_OUTPUT.PUT_LINE ('Address: ' || c_addr);
  END IF;
EXCEPTION
  WHEN ex_invalid_id THEN
     dbms_output.put_line('ID must be greater than zero!');
  WHEN no_data_found THEN
     dbms_output.put_line('No such customer!');
  WHEN others THEN
     dbms_output.put_line('Error!');
```

```
END;
/
```

OUTPUT:

```
Enter value for cc_id: -6 (let's enter a value -6)
old  2: c_id customers.id%type := &cc_id;
new 2: c_id customers.id%type := -6;
ID must be greater than zero!

PL/SQL procedure successfully completed.
```

RESULT:

Thus PL/SQL blocks for handling all types of exception has been written and implemented successfully.

Ex. No: 9 Creation of Procedures.

DATE:

AIM

To write PL/SQL programs that executes the concept of procedures.

DEFINITION

A procedure or function is a logically grouped set of SQL and PL/SQL statements that perform a specific task. They are essentially sub-programs. Procedures and functions are made up of,

- Declarative part
- Executable part
- Optional exception handling part

These procedures and functions do not show the errors.

KEYWORDS AND THEIR PURPOSES

REPLACE: It recreates the procedure if it already exists.

PROCEDURE: It is the name of the procedure to be created.

ARGUMENT: It is the name of the argument to the procedure. Paranthesis can be omitted if no arguments are present.

IN: Specifies that a value for the argument must be specified when calling the procedure ie. used to pass values to a sub-program. This is the default parameter.

OUT: Specifies that the procedure passes a value for this argument back to it's calling environment after execution ie. used to return values to a caller of the sub-program.

INOUT: Specifies that a value for the argument must be specified when calling the procedure and that procedure passes a value for this argument back to it's calling environment after execution.

RETURN: It is the datatype of the function's return value because every function must return a value, this clause is required.

PROCEDURES – SYNTAX

```
create or replace procedure <procedure name> (argument {in,out,inout} datatype ) {is,as}
variable declaration;
constant declaration;
begin
PL/SQL subprogram body;
```

exception

exception PL/SQL block; end;

CREATING THE TABLE ‘ITITEMS’ AND DISPLAYING THE CONTENTS

```
SQL> create table ititems(itemid number(3), actualprice number(5), ordid number(4), prodid number(4));

Table created.
SQL> insert into ititems values(101, 2000, 500, 201);
1 row created.
SQL> insert into ititems values(102, 3000, 1600, 202);
1 row created.
SQL> insert into ititems values(103, 4000, 600, 202);
1 row created.
SQL> select * from ititems;
ITEMID  ACTUALPRICE     ORDID    PRODID
------      -------          -----      ------
     101        2000                500       201
     102        3000              1600       202
     103        4000                600       202
```

PROGRAM FOR GENERAL PROCEDURE – SELECTED RECORD’S PRICE IS INCREMENTED BY 500 , EXECUTING THE PROCEDURE CREATED AND DISPLAYING THE UPDATED TABLE

```
SQL> create procedure itsum(identity number, total number) is price number;
null_price exception;
begin
select actualprice into price from ititems where itemid=identity;
if price is null then
raise null_price;
else
update ititems set actualprice=actualprice+total where itemid=identity;
end if;
exception
when null_price then
dbms_output.put_line('price is null');
end;
/
```

```
Procedure created.
SQL> exec itsum(101, 500);

PL/SQL procedure successfully completed.

SQL> select * from ititems;

 ITEMID   ACTUALPRICE      ORDID     PRODID

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

      101           2500                  500        201

      102           3000                1600        202

      103          4000                   600        202
```

PROCEDURE FOR ‘IN’ PARAMETER – CREATION, EXECUTION

```
SQL> set serveroutput on;

SQL> create procedure yyy (a IN number) is price number;
  2  begin
  3  select actualprice into price from ititems where itemid=a;
  4  dbms_output.put_line('Actual price is ' || price);
  5  if price is null then
  6  dbms_output.put_line('price is null');
  7  end if;
  8  end;
  9 /
Procedure created.
SQL> exec yyy(103);
Actual price is 4000
PL/SQL procedure successfully completed.
```

PROCEDURE FOR ‘OUT’ PARAMETER – CREATION, EXECUTION

```
SQL> set serveroutput on;
SQL> create procedure zzz (a in number, b out number) is identity number;
  2  begin
  3  select ordid into identity from ititems where itemid=a;
  4  if identity<1000 then
  5 b:=100;
  6  end if;
  7  end;
  8 /
Procedure created.
SQL> declare
  2  a number;
  3  b number;
  4  begin
  5 zzz(101,b);
  6  dbms_output.put_line('The value of b is '|| b);
  7  end;
  8 /
The value of b is 100
PL/SQL procedure successfully completed.
```

PROCEDURE FOR ‘INOUT’ PARAMETER – CREATION, EXECUTION

```
SQL> create procedure itit ( a in out number) is
  2 begin
  3 a:=a+1;
  4  end;
  5 /
  Procedure created.
SQL> declare
  2  a number:=7;
  3  begin
  4  itit(a);
  5  dbms_output.put_line(‘The updated value is ‘||a);
  6  end;
  7 /
The updated value is 8
PL/SQL procedure successfully completed.
```

RESULT

The PL/SQL programs were executed and their respective outputs were verified.

Ex. No: 10 Creation of database triggers and functions

DATE:

AIM

To study and implement the concepts of triggers and functions.

DEFINITION

- A trigger is a statement that is executed automatically by the system as a side effect of a modification to the database. The parts of a trigger are,
- Trigger statement: Specifies the DML statements and fires the trigger body. It also specifies the table to which the trigger is associated.
- Trigger body or trigger action: It is a PL/SQL block that is executed when the triggering statement is used.
- Trigger restriction: Restrictions on the trigger can be achieved

The different uses of triggers are as follows,

- To generate data automatically
- To enforce complex integrity constraints
- To customize complex securing authorizations
- To maintain the replicate table
- To audit data modifications

TYPES OF TRIGGERS

The various types of triggers are as follows,

- Before: It fires the trigger before executing the trigger statement.
- After: It fires the trigger after executing the trigger statement.
- For each row: It specifies that the trigger fires once per row.
- For each statement: This is the default trigger that is invoked. It specifies that the trigger fires once per statement.

VARIABLES USED IN TRIGGERS

- :new
- :old

These two variables retain the new and old values of the column updated in the database. The values in these variables can be used in the database triggers for data manipulation

TRIGGERS - SYNTAX

```
 create or replace trigger triggername [before/after] {DML statements}
 on [tablename] [for each row/statement]
 begin
 -----------------
 -----------------
 -----------------
exception
end;
```

USER DEFINED ERROR MESSAGE

The package “raise_application_error” is used to issue the user defined error messages

Syntax: raise_application_error(error number,‘error message‘);

The error number can lie between -20000 and -20999.

The error message should be a character string.

TO CREATE A SIMPLE TRIGGER THAT DOES NOT ALLOW INSERT UPDATE AND DELETE OPERATIONS ON THE TABLE

```
SQL> create trigger ittrigg before insert or update or delete on itempls for each row
  2  begin
  3  raise_application_error(-20010,'You cannot do manipulation');
  4  end;
  5
  6 /
Trigger created.
SQL> insert into itempls values('aaa',14,34000);
insert into itempls values('aaa',14,34000)
            *
ERROR at line 1:
ORA-20010: You cannot do manipulation
ORA-06512: at "STUDENT.ITTRIGG", line 2
ORA-04088: error during execution of trigger 'STUDENT.ITTRIGG'
SQL> delete from itempls where ename='xxx';
delete from itempls where ename='xxx'
            *
ERROR at line 1:
ORA-20010: You cannot do manipulation
ORA-06512: at "STUDENT.ITTRIGG", line 2
ORA-04088: error during execution of trigger 'STUDENT.ITTRIGG'
SQL> update itempls set eid=15 where ename='yyy';
update itempls set eid=15 where ename='yyy'
       *
ERROR at line 1:
ORA-20010: You cannot do manipulation
ORA-06512: at "STUDENT.ITTRIGG", line 2
ORA-04088: error during execution of trigger 'STUDENT.ITTRIGG'
```

TO DROP THE CREATED TRIGGER

```
SQL> drop trigger ittrigg;
Trigger dropped.
```

TO CREATE A TRIGGER THAT RAISES AN USER DEFINED ERROR MESSAGE AND DOES NOT ALLOW UPDATION AND INSERTION

```
SQL> create trigger ittriggs before insert or update of salary on itempls for each row
  2  declare
  3  triggsal itempls.salary%type;
  4  begin
  5  select salary into triggsal from itempls where eid=12;
  6  if(:new.salary>triggsal or :new.salary<triggsal) then
  7  raise_application_error(-20100,'Salary has not been changed');
  8  end if;
  9  end;
 10  /
Trigger created.
SQL> insert into itempls values ('bbb',16,45000);
insert into itempls values ('bbb',16,45000)
            *
ERROR at line 1:
ORA-04098: trigger 'STUDENT.ITTRIGGS' is invalid and failed re-validation
SQL> update itempls set eid=18 where ename='zzz';
update itempls set eid=18 where ename='zzz'
       *
ERROR at line 1:
ORA-04298: trigger 'STUDENT.ITTRIGGS' is invalid and failed re-validation
```

PL/SQL-FUNCTIONS – SYNTAX

```
 create or replace function <function name> (argument in datatype,……) return datatype {is,as}
 variable declaration;
 constant declaration;
 begin
 PL/SQL subprogram body;
 exception
 exception PL/SQL block;
 end;
```

CREATE THE TABLE ‘ITTRAIN’ TO BE USED FOR FUNCTIONS

```
SQL>create table ittrain ( tno number(10), tfare number(10));
Table created.
SQL>insert into ittrain values (1001, 550);
1 row created.
SQL>insert into ittrain values (1002, 600);
1 row created.
SQL>select * from ittrain;
    TNO      TFARE
  ------    --------
   1001        550
   1002        600
```

TO CREATE THE TABLE ‘ITEMPLS’

```
SQL> create table itempls (ename varchar2(10), eid number(5), salary number(10));
Table created.
SQL> insert into itempls values('xxx',11,10000);
1 row created.
SQL> insert into itempls values('yyy',12,10500);
1 row created.
SQL> insert into itempls values('zzz',13,15500);
1 row created.
SQL> select * from itempls;
ENAME          EID    SALARY
------------------ ------
xxx             11     10000
yyy             12     10500
zzz            13    15500
```

PROGRAM FOR FUNCTION AND IT’S EXECUTION

```
SQL> create function aaa (trainnumber number) return number is
  2  trainfunction ittrain.tfare % type;
  3  begin
  4  select tfare into trainfunction from ittrain where tno=trainnumber;
  5  return(trainfunction);
  6  end;
  7 /
Function created.
SQL> set serveroutput on;
SQL> declare
  2  total number;
  3  begin
  4  total:=aaa (1001);
  5  dbms_output.put_line('Train fare is Rs. '||total);
  6  end;
  7 /
Train fare is Rs.550
PL/SQL procedure successfully completed.
```

FACTORIAL OF A NUMBER USING FUNCTION — PROGRAM AND EXECUTION

```
  SQL> create function itfact (a number) return number is
  2   fact number:=1;
  3   b number;
  4   begin
  5  b:=a;
  6   while b>0
  7   loop
  8   fact:=fact*b;
  9  b:=b-1;
 10   end loop;
 11  return(fact);
 12  end;
 13 /
Function created.
 SQL> set serveroutput on;
SQL> declare
  2  a number:=7;
  3  f number(10);
  4  begin
  5  f:=itfact(a);
  6  dbms_output.put_line('The factorial of the given number is'||f);
  7 end;
  8 /
The factorial of the given number is 5040
PL/SQL procedure successfully completed.
```

RESULT

The triggers and functions were created, executed and their respective outputs were verified.

EX.NO.11 (a) **ARITHMETIC OPERATIONS USING VB FORM**
DATE:

AIM:

To write a program to perform Arithmetic operations using visual basic forms.

PROCEDURE

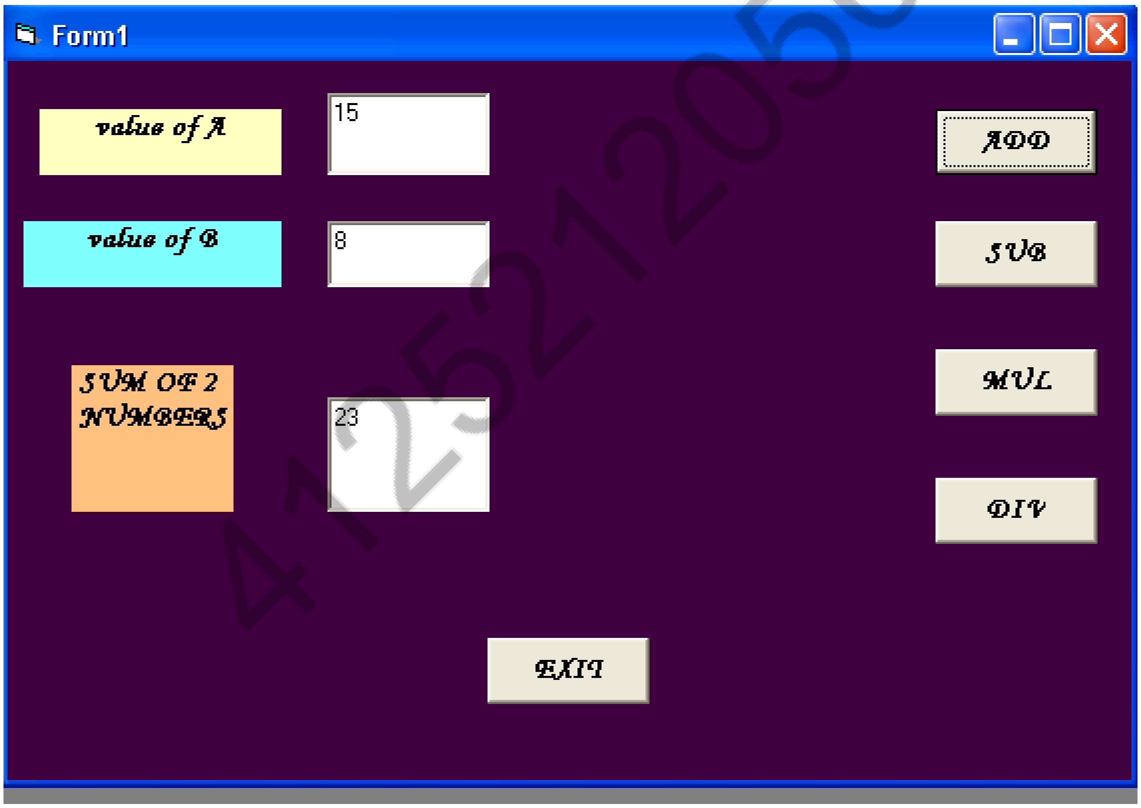
1.Start.
2.Open Microsoft visual studio.
3.Open a new file,drop text box,command box and label after saving the file.
4.Enter the codings for command box to perform addition.
5.Similarly,enter the codings for command box to perform subtraction,multiplication, division.
6.Run them,Display the result.
7.Stop.

VB CODINGS:

```
Dim t As Integer
Dim tt As Integer
Dim res As Integer

Private Sub Command1_Click()
Label3.Caption = "SUM OF 2 NUMBERS"
t = Text1.Text
tt = Text2.Text
res = t + tt
Text3.Text = res
End Sub

Private Sub Command2_Click()
Dim diff As Integer
Label3.Caption = "DIFFERENCE OF TWO NUMBERS"
t = Text1.Text
tt = Text2.Text
diff = t - tt
Text3.Text = diff
End Sub

Private Sub Command3_Click()
Dim prod As Integer
Label3.Caption = "PRODUCT OF 2 NUMBERS"
t = Text1.Text
tt = Text2.Text
prod = t * tt
Text3.Text = prod
End Sub
```

```
Private Sub Command4_Click()
Dim di As Integer
Label3.Caption = "QUOTIENT "
t = Text1.Text
tt = Text2.Text
di = t / tt
Text3.Text = di
End Sub

Private Sub Command5_Click()
End
End Sub
```

OUTPUT:



15

8

23

RESULT:

Thus a program to perform arithmetic operations using visual basic is executed.

Ex.No.11(b) **TEXT EDITOR USING MENU CONTROLS**

Date :

AIM:

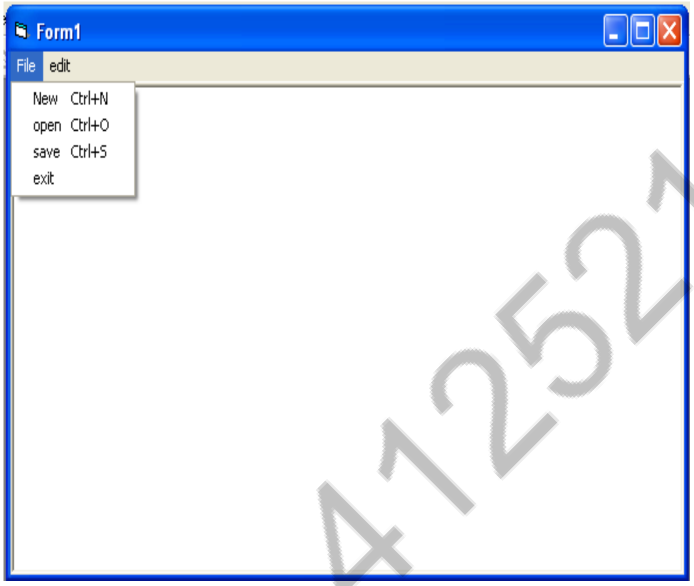
To design a text editor using menu controls in Visual Basic.

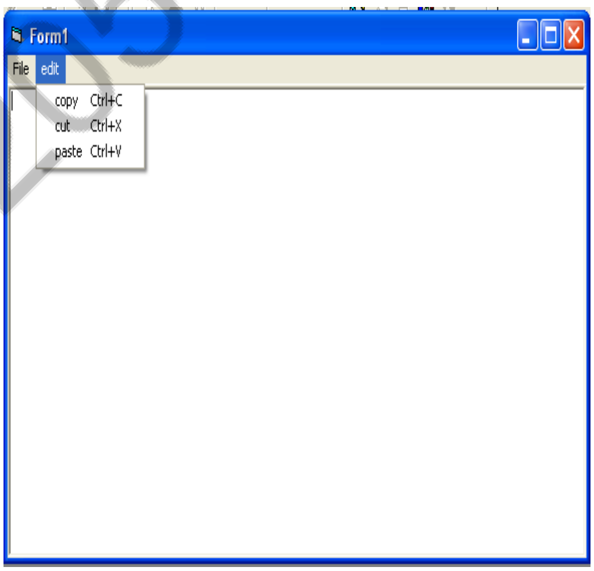
PROCEDURE:

STEP 1: Start the Visual Basic 6.0
STEP 2: Create the form with essential controls and insert the menu using menu editor.
STEP 3: Write the code for doing the appropriate functions for text editing.
STEP 4: Save the forms and project.
STEP 5: Execute the form.
STEP 6: Stop

EXECUTION:

FORM DESIGN



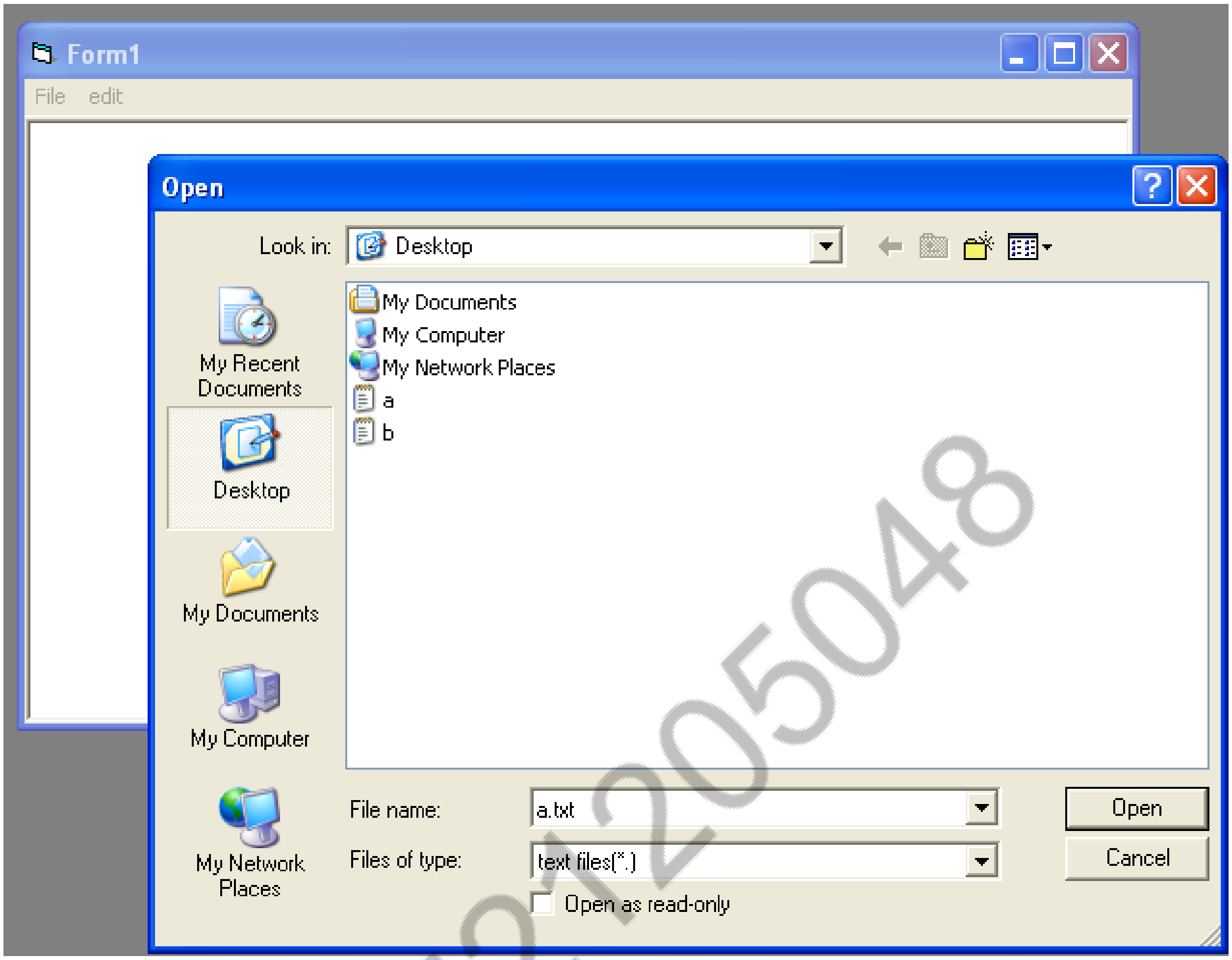


CODING

```
Dim a

Private Sub Form_Load()
RichTextBox1.Height = Me.ScaleHeight
RichTextBox1.Width = Me.ScaleWidth
End Sub

Private Sub New_Click()
RichTextBox1.Visible = True
RichTextBox1.Text = Clear
```

```
RichTextBox1.SetFocus
End Sub

Private Sub open_Click()
Dim filename As String
CommonDialog1.Filter = "text files(*.)|*.txt"
CommonDialog1.ShowOpen
RichTextBox1.filename = CommonDialog1.filename
End Sub

Private Sub save_Click()
Dim filename As String
CommonDialog1.DefaultExt = ".txt"
CommonDialog1.Filter = "textfiles(*.txt)|*.txt"
CommonDialog1.ShowSave
filename = CommonDialog1.filename
RichTextBox1.SaveFile (filename)
End Sub

Private Sub exit_Click()
End
End Sub

Private Sub cut_Click()
a = RichTextBox1.SelText
RichTextBox1.SelText = Clear
End Sub

Private Sub paste_Click()
RichTextBox1.SelText = a
End Sub

Private Sub  copy_Click()
a = RichTextBox1.SelText
End Sub
```

OUTPUT:



Look in: Desktop

File name: a.txt

Files of type: text files(*.)

RESULT: Thus the program for Text editor with menu controls has been developed and executed successfully.

EX NO : 12

DATE :

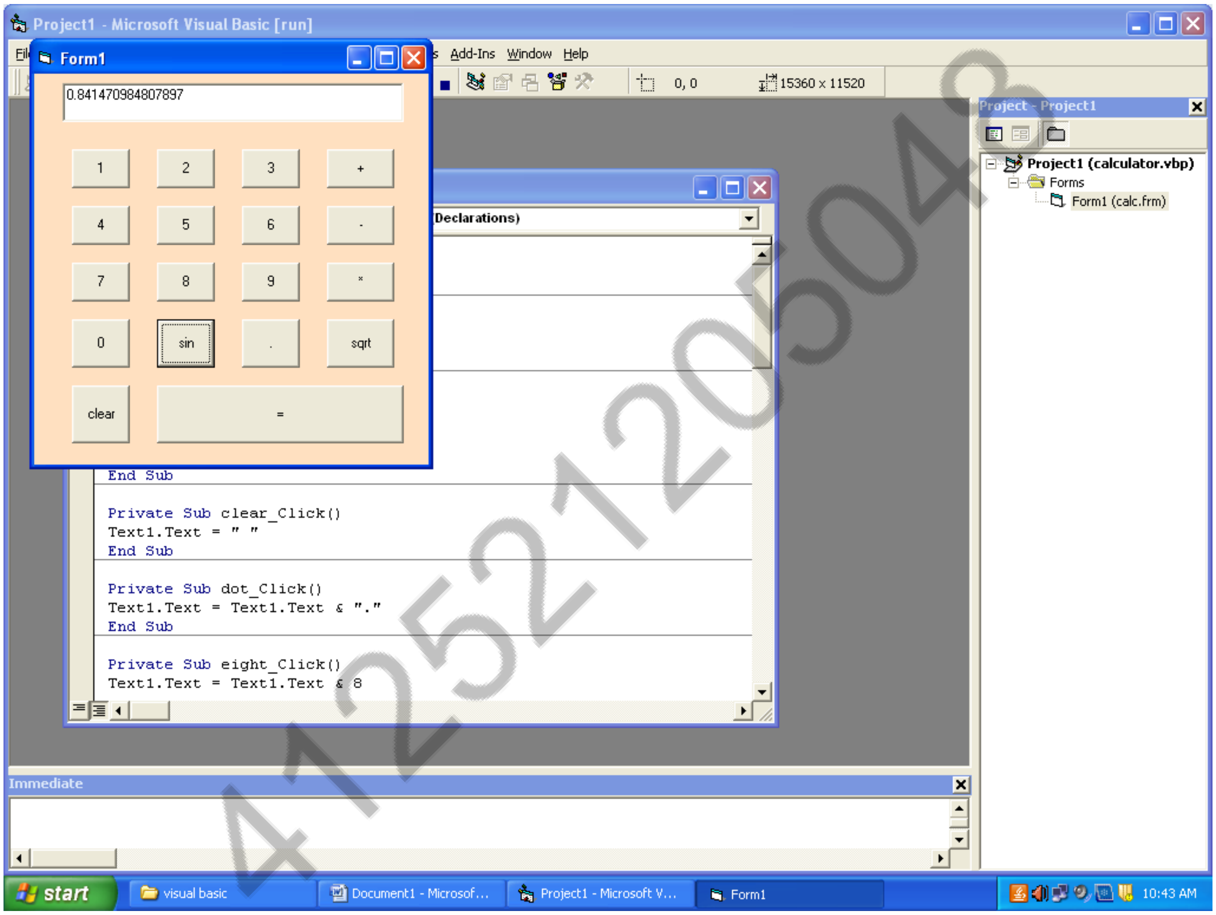
SIMPLE CALCULATOR

```
Dim a As Integer
Dim b As Integer
Dim op As String

Private Sub Command16_Click()

End Sub

Private Sub add_Click()
a = Text1.Text
Text1.Text = " "
op = "+"
End Sub

Private Sub clear_Click()
Text1.Text = " "
End Sub

Private Sub dot_Click()
Text1.Text = Text1.Text & "."
End Sub

Private Sub eight_Click()
Text1.Text = Text1.Text & 8
End Sub

Private Sub equal_Click()
b = Text1.Text
If op = "+" Then
Text1.Text = a + b
ElseIf op = "-" Then
Text1.Text = a - b
ElseIf op = "*" Then
Text1.Text = a * b
ElseIf op = "/" Then
Text1.Text = a / b
End If

End Sub

Private Sub five_Click()
Text1.Text = Text1.Text & 5
```

```
End Sub

Private Sub four_Click()
Text1.Text = Text1.Text & 4
End Sub

Private Sub mul_Click()
a = Text1.Text
Text1.Text = " "
op = "*"
End Sub

Private Sub nine_Click()
Text1.Text = Text1.Text & 9
End Sub

Private Sub one_Click()
Text1.Text = Text1.Text & 1
End Sub

Private Sub seven_Click()
Text1.Text = Text1.Text & 7
End Sub

Private Sub sine_Click()
Text1.Text = Math.Sin(Text1.Text)
End Sub

Private Sub six_Click()
Text1.Text = Text1.Text & 6
End Sub

Private Sub  sqrt_Click()
Text1.Text = Math.Sqr(Text1.Text)
End Sub

Private Sub sub_Click()
a = Text1.Text
Text1.Text = " "
op = "-"
End Sub

Private Sub three_Click()
Text1.Text = Text1.Text & 3
End Sub
```

```
Private Sub two_Click()
Text1.Text = Text1.Text & 2
End Sub

Private Sub zero_Click()
Text1.Text = Text1.Text & 0
End Sub
```

OUTPUT:



```
0.841470984807897
```

```
End Sub

Private Sub clear_Click()
Text1.Text = " "
End Sub

Private Sub dot_Click()
Text1.Text = Text1.Text & "."
End Sub

Private Sub eight_Click()
Text1.Text = Text1.Text & 8
```

RESULT: Thus the program for Calculator has been developed and executed successfully.

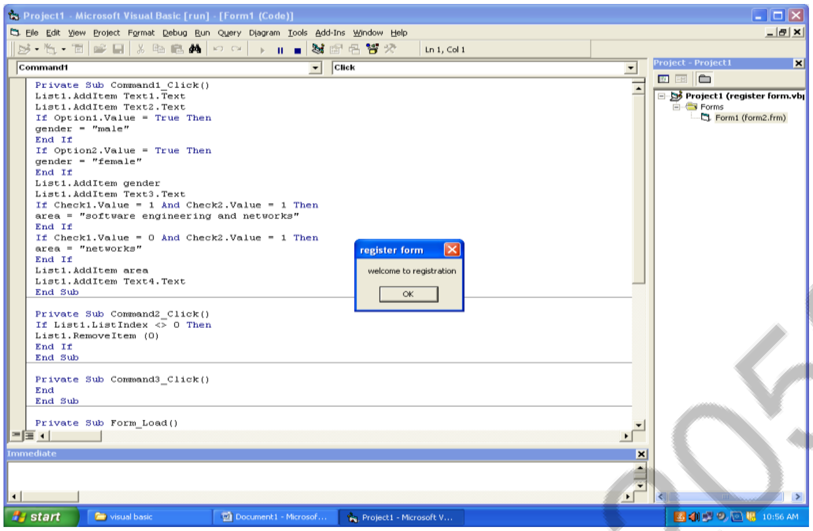
EX NO : 13

DATE : **REGISTRATION FORM**

```
Private Sub Command1_Click()
List1.AddItem Text1.Text
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 and 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()
If List1.ListIndex <> 0 Then
List1.RemoveItem (0)
End If
End Sub

Private Sub Command3_Click()
End
End Sub

Private Sub Form_Load()
Label10.Caption = Date$
MsgBox "welcome to registration"
End Sub

Private Sub Label10_Click()
Label10.Caption =  Date
End Sub

Private Sub Label9_Click()
Label9.Caption = Time
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
```

OUTPUT:



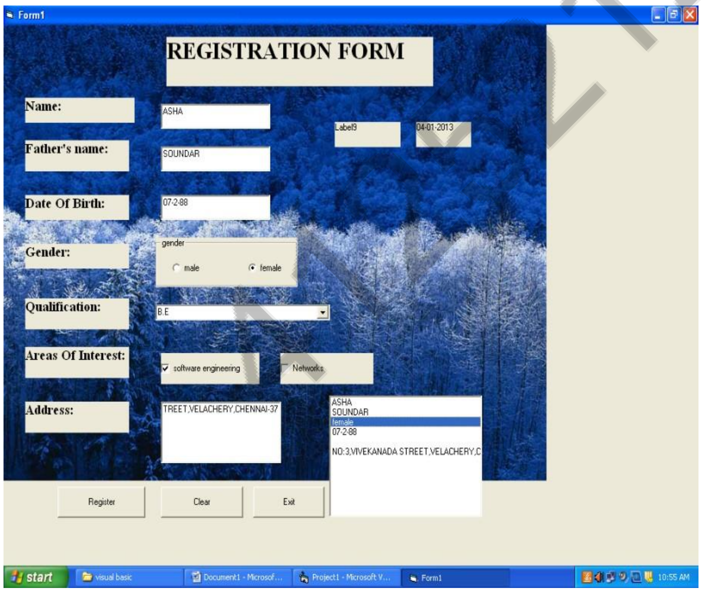
Project1 - Microsoft Visual Basic [run] - [Form1 (Code)]
File Edit View Project Format Debug Run Query Diagram Tools Add-Ins Window Help
Ln 1, Col 1
Command1
Click
Project - Project1
Project1 (register form.vbp
Forms
Form1 (form2.frm)
Immediate
start
visual basic
Document1 - Microsof...
Project1 - Microsoft V...
10:56 AM

```
Private Sub Command1_Click()
List1.AddItem Text1.Text
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 and 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()
If List1.ListIndex <> 0 Then
List1.RemoveItem (0)
End If
End Sub

Private Sub Command3_Click()
End
End Sub

Private Sub Form_Load()
```

register form

welcome to registration

OK



REGISTRATION FORM

Name: ASHA

Father's name: SOUNDAR

Date Of Birth: 07-2-88

Label9 04-01-2013

Gender: gender ○ male ● female

Qualification: B.E

Areas Of Interest: ☑ software engineering □ Networks

Address: TREET,VELACHERY,CHENNAI-37

ASHA
SOUNDAR
female
07-2-88

NO:3,VIVEKANADA STREET,VELACHERY,C

Register | Clear | Exit

RESULT: Thus the program for Calculator has been developed and executed successfully.

EX NO : 14

DATE : **MINI PROJECT**

EMPLOYEE INFORMATION SYSTEM

REPORT GENERATION

AIM

To create a software package for updating and retrieving the employee details using Oracle as the Back end and VB as the front end

PROBLEM DEFINITION

1. The details of a employee can be entered and stored in a employee database
2. The details of all the employees can be viewed as a consolidated report.
3. The information of individual employee can also retrieved.
4. The unwanted details can also be deleted.
5. STEPS FOR ACCESSING AN EMPLOYEE DATABASE

PROCEDURE:

1.Create a form with the following information:

Label- 3

empid, ename, salary.

Textbox-3

One corresponding to each label.

Command buttons-5

New, save, delete, generate report, exit.

2.Change the names of labels:

- Rename label1 as empid, label2 as ename, label3 as salary.

Similarly rename command buttons:

- Rename command1 as new, command2 as save, command3 as delete, command4 as generate report, command5 as exit.

Empty the textboxes.

3.Double click on the command button new and insert the code:

```
If Not Adodc1.Recordset.EOF Then

        Adodc1.Recordset.MoveLast

        Adodc1.Recordset.AddNew

Else
```

```
    Adodc1.Recordset.AddNew
End If
```

4.Double click on the command save and insert the code:

```
Adodc1.Recordset.Fields(0) = Val(Text1.Text)
Adodc1.Recordset.Fields(1) = Text2.Text
Adodc1.Recordset.Fields(2) = Val(Text3.Text)
Adodc1.Recordset.Update
MsgBox "Record saved Successfully"
```

5.Double click on the command button report and insert the following code:

```
DataReport1.Show
```

6.Double click on the command button exit and insert the code:

```
End
```

7.Similarly double click on the command button delete and insert the corresponding code.

8.Click project from menu bar

- ○ Click components->Microsoft ADO data control
- ○ Click apply and ok.
- ○ ADO data control icon will be added to the toolbox on the left side of the window. Drag and drop the icon in the form.

9.Right click on the icon in the form and select properties->general(tab)->build.

- Select Microsoft OLE Database provider for oracle and click next button.
- Enter the server name as “orcl11” and type username and password(e.g.username:s2ita.. password:s2ita..)
- Click on the test connection button to check the connection.(A prompt will be displayed if the connection is successful. Click ok)
- Click the authentication(tab). Enter the username and password as before and click apply ,ok.
- Click on the record source tab and select ADCMD table
- Select any one table(e.g. emp)from the list of tables and click apply, ok.
- Go to form design. To connect the text boxes to the database, click on the first textbox.
    - ○ In the property dialog box(which appears in the right corner of the window)select data source as adodc1 and data field as eno. Similarly do for other two textboxes(select the respective data fields).

10. Save the form. Build using the following icon:

►

- Click new to add a new field to the table.
- Click save to save the newly added informations.
- Click delete to delete a field from the table.
- Click generate report to generate a report.
- Click exit to exit the form.

To generate a report,

11. Click project->components
    - In the designers(tab), click data environment check box, data report check box.
    - Click apply, close.

12. Click project->add data environment
    - Right click on “connection”->properties->Microsoft OLE database provided for oracle.
    - Click next. Enter username and password as before and click ok.
    - Right click on “connection” and select add command. command1 will be added.
    - In the properties dialog box, select data object as table and a suitable object name(e.g.emp)

13. Click project->add data report.
    - In the properties dialog box, select data source as dataenvironment1 and data member as command1.
    - Click command1 so that all the fields in the table will be displayed.
    - Drag and drop the required fields to the report screen and click file->save to save the report.

TABLE DESIGN

Employee table

```
Name                          Null?    Type
------------------------------------------------
 ENO                          NOT NULL NUMBER(3)
 ENAME                                 VARCHAR2(12)
 DESIGNATION                           VARCHAR2(15)
 PLACE                                 VARCHAR2(20)
 SALARY                                NUMBER
```

TABLE CREATION

create table employee (eno number(3),ename varchar2(12),desigantion varchar2(15),place varchar2(20),salary number);

VB CODE

MAIN FORM

```
Private Sub Command1_Click()
Form2.Show
End Sub

Private Sub Command2_Click()
Form3.Show
End Sub

Private Sub Command3_Click()
End
End Sub
```

ADD FORM

```
Public acno As Integer

Private Sub cmdadd_Click()
If Adodc1.Recordset.EOF Then
Adodc1.Recordset.MoveLast
Adodc1.Recordset.AddNew
Else
Adodc1.Recordset.AddNew
End If

Adodc1.Recordset.Fields.Item("ename").Value = Text1.Text
Adodc1.Recordset.Fields.Item("eno").Value = Val(Text2.Text)
Adodc1.Recordset.Fields.Item("designation").Value = Text3.Text
Adodc1.Recordset.Fields.Item("salary").Value = Text4.Text
Adodc1.Recordset.Fields.Item("place").Value = Text5.Text
Adodc1.Recordset.Update
MsgBox "DONE"
Text1.Text = ""
Text2.Text = ""
Text3.Text = ""
Text4.Text = ""
Text5.Text = ""
End Sub

Private Sub cmdback_Click()
Unload Me
Form1.Show
End Sub
Dim msg As String
Private Sub cmddel_Click()
msg = MsgBox("Delete surely?", vbYesNo, "Deletion Confiramion")
If msg = vbYes Then
Adodc1.Recordset.Delete
MsgBox "Record Deleted"
```

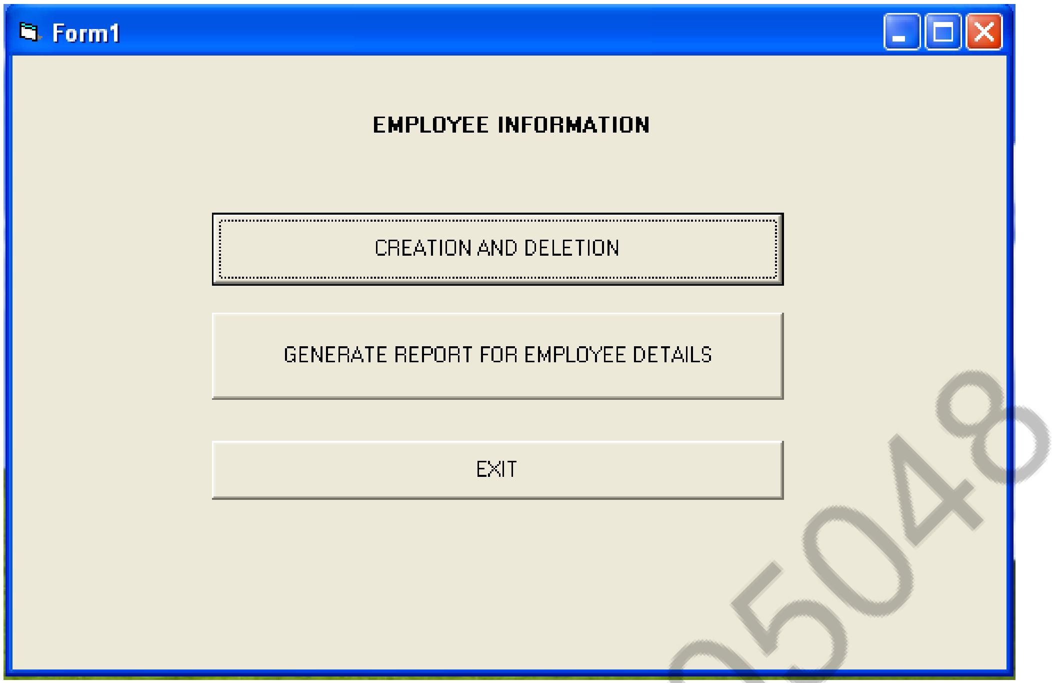
```
Adodc1.Recordset.MoveFirst
Else
MsgBox "Record Not deleted"
End If
End Sub

Private Sub Form_Load()
Form1.Visible = False
End Sub
```

REPORT FORM

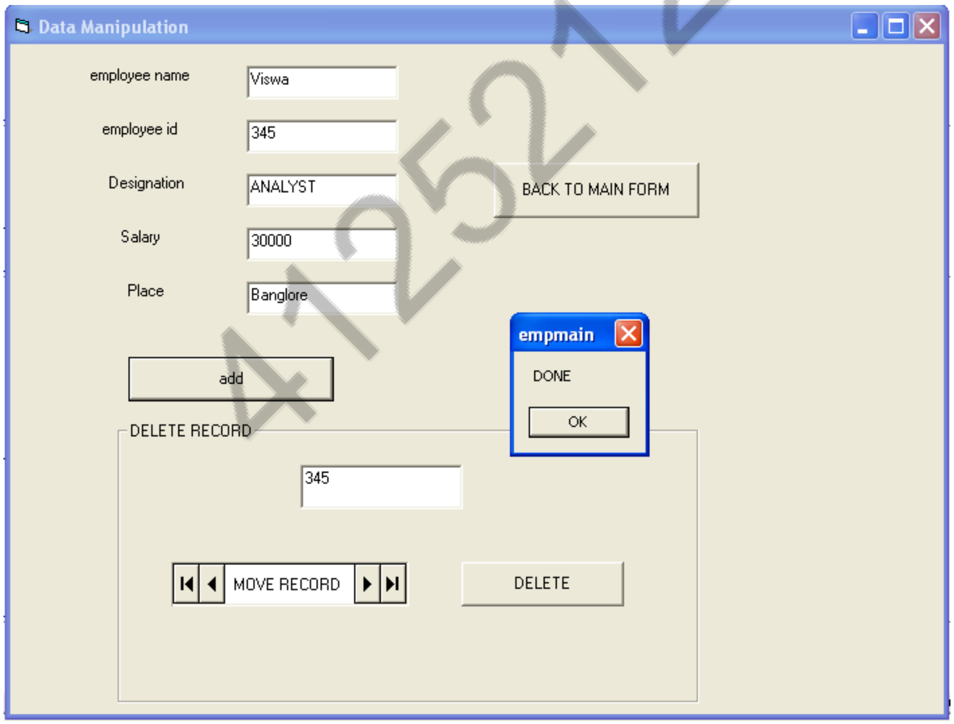
```
Private Sub cmdback_Click()
Unload Me
Form1.Show
End Sub

Private Sub cmdclr_Click()
Txteno.Text = ""
End Sub

Private Sub cmdfullrep_Click()
Adodc1.Refresh
DataReport1.Show
End Sub

Private Sub cmdview_Click()
Adodc1.CommandType = adCmdText
Adodc1.RecordSource = "select * from employee where eno= ' " & Val(Txteno.Text) & " ' "
Adodc1.Refresh
End Sub

Private Sub Form_Load()
Form1.Visible = False
End Sub
```

SCREEN SHOTS:

MAIN FORM



ADD FORM



employee name: Viswa

employee id: 345

Designation: ANALYST

Salary: 30000

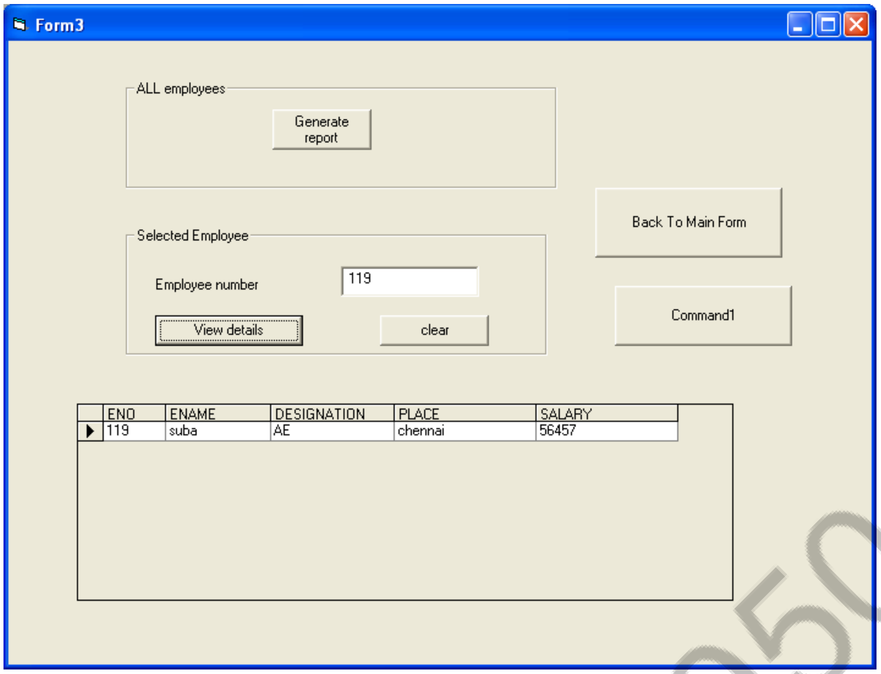
Place: Banglore

empmain

DONE

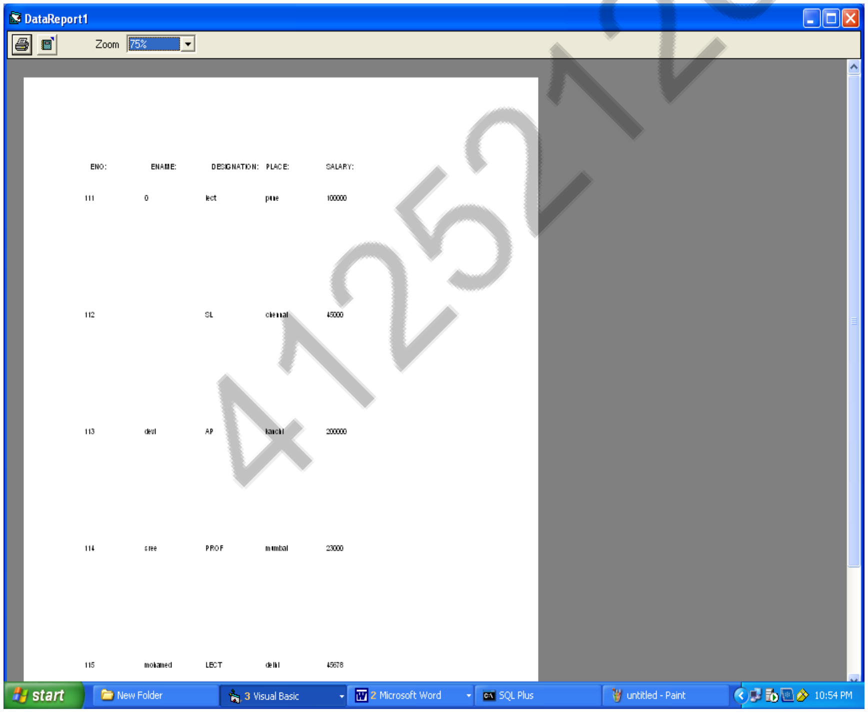
345

REPORT FORM



ENO	ENAME	DESIGNATION	PLACE	SALARY
119	suba	AE	chennai	56457

DATA REPORT



ENO:	ENAME:	DESIGNATION:	PLACE:	SALARY:
111	0	lect	pune	100000
112		SL	chennai	45000
113	devi	AP	kanchi	200000
114	sree	PROF	mumbai	23000
115	mohamed	LECT	delhi	45678

RESULT:

Thus the mini project for managing the details of employee was successfully created using VB6 and Oracle as per the constraints given and the report was generated.