VALLIAMMAI ENGINEERING COLLEGE

SRM NAGAR

KACHEEPURAM DISTRICT, TAMIL NADU.

LAB MANUAL

SUB NAME: DATABASE MANAGEMENT SYSTEMS LAB SUB CODE: CS2258

PREPARED BY:

NAGARAJAN .M, M.Tech (CSE)

PROGRAMMER / CSE

VALLIAMMAI ENGINEERING COLLEGE

CHENNAI 603 203

LIST OF EXPERIMENTS

1. To implement Data Definition language

1.1. Create, alter, drop, truncate

1.2. To implement Constraints.

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

2. To implementation on DML, TCL and DRL

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

3. To implement Nested Queries & Join Queries

3.1.(a). To implementation of Nested Queries

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

4. To implement Views

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

5(a). Control Structure

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

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

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

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

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

5. (b).Procedures

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

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

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

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

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

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

5. (c). Functions:

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

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

6. Front End Tools

6.1. To design a form using different tools in Visual Basic

7. Form

7.1. To design a Single Document Interface and Multiple Document Interface forms using Visual Basic.

8. Trigger:

8.1. To write a Trigger to pop-up the DML operations

8.2. To write a Trigger to check the age valid or not Using Message Alert.

8.3. Create a Trigger for Raise appropriate error code and error message.

8.4. Create a Trigger for a table it will update another table while inserting values

9. Menu Design

9.1. To design a Note Pad Application menu using Visual Basic.

10. Report design

10.1. To design a report using Visual Basic.

11. To design the Database and Implement it by using VB (Mini Project).

11.1. PASSPORT AUTOMATION SYSTEM

DBMS MANUAL

EX: NO: 1

DATA DEFINITION LANGUAGE (DDL) COMMANDS IN RDBMS

AIM:

To execute and verify the Data Definition Language commands and constraints

DDL (DATA DEFINITION LANGUAGE)

- CREATE
- ALTER
- DROP
- TRUNCATE
- COMMENT
- RENAME

PROCEDURE

STEP 1: Start

STEP 2: Create the table with its essential attributes.

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

STEP 4: Stop

SQL COMMANDS

1. COMMAND NAME: **CREATE**
   COMMAND DESCRIPTION: **CREATE** command is used to create objects in the database.
2. COMMAND NAME: **DROP**
   COMMAND DESCRIPTION: **DROP** command is used to delete the object from the database.
3. COMMAND NAME: **TRUNCATE**
   COMMAND DESCRIPTION: **TRUNCATE** command is used to remove all the records from the table

4. COMMAND NAME: **ALTER**

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

5. COMMAND NAME: **RENAME**

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

QUERY: 01

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

Syntax for creating a table:

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

QUERY: 01

```
SQL>CREATE TABLE EMP (EMPNO NUMBER (4),
                      ENAME VARCHAR2 (10),
                      DESIGNATIN VARCHAR2 (10),
                      SALARY NUMBER (8,2));
Table created.
```

QUERY: 02

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

Syntax for describe the table:

SQL: DESC <TABLE NAME>;

```
SQL> DESC EMP;

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

QUERY: 03

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

Syntax For Create A from An Existing Table With All Fields

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

QUERY: 03

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

SQL> DESC EMP1

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

QUERY: 04

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

Syntax For Create A from An Existing Table With Selected Fields

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

QUERY: 04

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

SQL> DESC EMP2

Name	Null?	Type
EMPNO		NUMBER (4)

ENAME **VARCHAR2 (10)**

QUERY: 05

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

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

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

QUERY: 05

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

Table created.

SQL> DESC EMP3;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------
 EMPNO                                              NUMBER(4)
 ENAME                                              VARCHAR2(10)
 DESIGNATIN                                         VARCHAR2(10)
 SALARY                                             NUMBER(8,2);
```

ALTER & MODIFICATION ON TABLE

QUERY: 06

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

Syntax for Alter & Modify on a Single Column:

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

QUERY: 06

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

Table altered.
```

SQL> DESC EMP;

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

QUERY: 07

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

Syntax for alter table with multiple column:

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

QUERY: 07

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

SQL> DESC EMP;

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

QUERY: 08

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

Syntax for add a new column:

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

QUERY: 08

SQL> ALTER TABLE EMP ADD QUALIFICATION VARCHAR2(6);
Table altered.
SQL> DESC EMP;

Name	Null?	Type
EMPNO		NUMBER(7)
ENAME		VARCHAR2(12)
DESIGNATIN		VARCHAR2(10)
SALARY		NUMBER(8,2)
QUALIFICATION		VARCHAR2(6)

QUERY: 09

Q9. Write a query to add multiple columns in to employee

Syntax for add a new column:

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

QUERY: 09

SQL>ALTER TABLE EMP ADD (DOB DATE, DOJ DATE);
Table altered.
SQL> DESC EMP;

Name	Null?	Type
EMPNO		NUMBER(7)
ENAME		VARCHAR2(12)
DESIGNATIN		VARCHAR2(10)
SALARY		NUMBER(8,2)
QUALIFICATION		VARCHAR2(6)
DOB		DATE
DOJ		DATE

REMOVE / DROP

QUERY: 10

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

Syntax for add a new column:

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

QUERY: 10

SQL> ALTER TABLE EMP DROP COLUMN DOJ;

Table altered.
SQL> DESC EMP;

Name	Null?	Type
EMPNO		NUMBER(7)
ENAME		VARCHAR2(12)
DESIGNATIN		VARCHAR2(10)
SALARY		NUMBER(8,2)
QUALIFICATION		VARCHAR2(6)
DOB		DATE

QUERY: 11

Q10. Write a query to drop multiple columns from employee

Syntax for add a new column:

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

QUERY: 11

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

SQL> DESC EMP;

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

REMOVE

QUERY: 12

Q10. Write a query to rename table emp to employee

Syntax for add a new column:

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

QUERY: 12

```
SQL> ALTER TABLE EMP RENAME EMP TO EMPLOYEE;
SQL> DESC EMPLOYEE;
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------
 EMPNO                                              NUMBER(7)
 ENAME                                              VARCHAR2(12)
 DESIGNATIN                                         VARCHAR2(10)
 SALARY                                             NUMBER(8,2)
```

CONSTRAINTS

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

TYPES OF CONSTRAINTS:

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

CONSTRAINTS CAN BE CREATED IN THREE WAYS:

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

OPERATION ON CONSTRAINT:

i) ENABLE
ii) DISABLE
iii) DROP

Column level constraints Using Primary key

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

Primary key

Syntax for Column level constraints Using Primary key:

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

QUERY:13

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

Column level constraints Using Primary key with naming convention

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

Syntax for Column level constraints Using Primary key:

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

QUERY:14

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

Table Level Primary Key Constraints

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

Syntax for Table level constraints Using Primary key:

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

QUERY: 15

```
SQL>CREATE TABLE EMPLOYEE (EMPNO NUMBER(6),
                           ENAME VARCHAR2(20),
                           JOB VARCHAR2(6),
                           SAL NUMBER(7),
                           DEPTNO NUMBER(5),
                          CONSTRAINT EMP_EMPNO_PK PRIMARY
                          KEY(EMPNO));
```

Table level constraint with alter command (primary key):

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

Syntax for Column level constraints Using Primary key:

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

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

QUERY: 16

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

DEPTNO NUMBER(6));

SQL>ALTER TABLE EMP3 ADD CONSTRAINT **EMP3_EMPNO_PK PRIMARY KEY (EMPNO);**

Reference /foreign key constraint

Column level foreign key constraint:

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

Parent Table:

Syntax for Column level constraints Using Primary key:

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

Child Table:

Syntax for Column level constraints Using foreign key:

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

QUERY: 17

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

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

Column level foreign key constraint with naming conversions:

Parent Table:

Syntax for Column level constraints Using Primary key:

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

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

Child Table:

Syntax for Column level constraints using foreign key:

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

QUERY:18

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

SQL>CREATE TABLE EMP4A
      (EMPNO NUMBER(3),
      DEPTNO NUMBER(2)CONSTRAINT EMP4A_DEPTNO_FK
       REFERENCES DEPT(DEPTNO),
       DESIGN VARCHAR2(10));
```

Table Level Foreign Key Constraints

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

Parent Table:

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

Child Table:

Syntax for Table level constraints using foreign key:

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

QUERY: 19

SQL>CREATE TABLE DEPT

```
(DEPTNO NUMBER(2) PRIMARY KEY,
DNAME VARCHAR2(20),
LOCATION VARCHAR2(15));

SQL>CREATE TABLE EMP5
(EMPNO NUMBER(3),
DEPTNO NUMBER(2),
DESIGN VARCHAR2(10)CONSTRAINT ENP2_DEPTNO_FK FOREIGN
KEY(DEPT NO)REFERENCESDEPT(DEPTNO));
```

Table Level Foreign Key Constraints with Alter command

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

Parent Table:

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

Child Table:

Syntax for Table level constraints using foreign key:

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

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

QUERY:20

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

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

```
    DESIGN VARCHAR2(10));
```

```
SQL>ALTER TABLE EMP6 ADD CONSTRAINT EMP6_DEPTNO_FK FOREIGN
KEY(DEPTNO)REFERENCES DEPT(DEPTNO);
```

Check constraint

Column Level Check Constraint

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

Syntax for clumn level constraints using Check:

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

QUERY:21

```
SQL>CREATE TABLE EMP7(EMPNO NUMBER(3),
        ENAME VARCHAR2(20),
        DESIGN VARCHAR2(15),
        SAL NUMBER(5)CONSTRAINT EMP7_SAL_CK CHECK(SAL>500 AND
        SAL<10001),
        DEPTNO NUMBER(2));
```

Table Level Check Constraint:

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

Syntax for Table level constraints using Check:

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

QUERY:22

```
SQL>CREATE TABLE EMP8(EMPNO NUMBER(3),
        ENAME VARCHAR2(20),
```

```
DESIGN VARCHAR2(15),
SAL NUMBER(5),DEPTNO NUMBER(2),
CONSTRAINTS EMP8_SAL_CK CHECK(SAL>500 AND
SAL<10001));
```

Check Constraint with Alter Command

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

Syntax for Table level constraints using Check:

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

QUERY:23

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

SQL>ALTER TABLE EMP9 ADD CONSTRAINTS EMP9_SAL_CK
CHECK(SAL>500 AND SAL<10001);
```

Unique Constraint

Column Level Constraint

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

Syntax for Column level constraints with Unique:

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

QUERY:24

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

Table Level Constraint

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

Syntax for Table level constraints with Unique:

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

QUERY:25

```
SQL>CREATE TABLE EMP11(EMPNO NUMBER(3),
        ENAME VARCHAR2(20),
        DESIGN VARCHAR2(15),
        SAL NUMBER(5),CONSTRAINT EMP11_DESIGN_UK UNIGUE(DESIGN));
```

Table Level Constraint Alter Command

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

Syntax for Table level constraints with Check Using Alter

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

SQL> ALTER TABLE ADD <CONSTRAINTS> <CONSTRAINTS NAME>

<CONSTRAINTS TYPE>(COLUMN NAME);

QUERY:26

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

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

Not Null

Column Level Constraint

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

Syntax for Column level constraints with Not Null:

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

QUERY: 27

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

Null

Column Level Constraint

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

Syntax for Column level constraints with Null:

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

QUERY:28

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

Constraint Disable \ Enable

Constraint Disable

Q.29. Write a query to disable the constraints

Syntax for disabling a single constraint in a table:
SQL>ALTER TABLE <TABLE-NAME> DISABLE CONSTRAINT <CONSTRAINT-NAME>

Constraint Enable

QUERY:29

SQL>ALTER TABLE EMP13 DISABLE CONSTRAINT EMP13_ENAME_NN

NULL;

Q.30. Write a query to enable the constraints

Syntax for disabling a single constraint in a table:
SQL>ALTER TABLE <TABLE-NAME> DISABLE CONSTRAINT <CONSTRAINT-NAME>

QUERY:30

SQL>ALTER TABLE EMP13 ENABLE CONSTRAINT EMP13_ENAME_NN

NULL;

EX: NO: 2
To implementation on DML and DCL Commands in RDBMS

AIM:
To execute and verify the DML and TCL Language commands

DML (DATA MANIPULATION LANGUAGE)

- SELECT
- INSERT
- DELETE
- UPDATE

TCL (TRANSACTION CONTROL LANGUAGE)

- COMMIT
- ROLL BACK
- SAVE POINT

PROCEDURE

STEP 1: Start

STEP 2: Create the table with its essential attributes.

STEP 3: Insert the record into table

STEP 4: Update the existing records into the table

STEP 5: Delete the records in to the table

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

STEP 7: use rollback for completely undo the records

STEP 6: use commit for permanently save the records.

SQL COMMANDS

1. COMMAND NAME: **INSERT**
   COMMAND DESCRIPTION: INSERT command is used to Insert objects in the database.
2. COMMAND NAME: **SELECT**

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

3. COMMAND NAME: **UPDATE**

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

4. COMMAND NAME: **DELETE**

   COMMAND DESCRIPTION: DELETE command is used to DELETE the Records form the table

5. COMMAND NAME: **COMMIT**

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

6. COMMAND NAME: **ROLLBACK**

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

6. COMMAND NAME: **SAVE POINT**

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

INSERT

QUERY: 01

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

Syntax for Insert Records in to a table:

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

QUERY: 01

INSERT A RECORD FROM AN EXISTING TABLE:

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

1 row created.

SELECT

QUERY: 02

Q3. Write a query to display the records from employee.
Syntax for select Records from the table:

SQL> SELECT * FROM <TABLE NAME>;

QUERY: 02

DISPLAY THE EMP TABLE:

SQL> SELECT * FROM EMP;

EMPNO	ENAME	DESIGNATIN	SALARY
101	NAGARAJAN	LECTURER	15000

INSERT A RECORD USING SUBSITUTION METHOD

QUERY: 03

Q3. Write a query to insert the records in to employee using substitution method.
Syntax for Insert Records into the table:

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

QUERY: 03

SQL> INSERT INTO EMP

VALUES(&EMPNO,'&ENAME','&DESIGNATIN','&SALARY');

Enter value for empno: 102

Enter value for ename: SARAVANAN

Enter value for designatin: LECTURER

Enter value for salary: 15000

```
old   1: INSERT INTO EMP
VALUES(&EMPNO,'&ENAME','&DESIGNATIN','&SALARY')
new   1: INSERT INTO EMP VALUES(102,'SARAVANAN','LECTURER','15000')
1 row created.
SQL> /
Enter value for empno: 103
Enter value for ename: PANNERSELVAM
Enter value for designatin: ASST. PROF
Enter value for salary: 20000
old   1: INSERT INTO EMP
VALUES(&EMPNO,'&ENAME','&DESIGNATIN','&SALARY')
new   1: INSERT INTO EMP VALUES(103,'PANNERSELVAM','ASST.
PROF','20000')
1 row created.

SQL> /
Enter value for empno: 104
Enter value for ename: CHINNI
Enter value for designatin: HOD, PROF
Enter value for salary: 45000
old   1: INSERT INTO EMP
VALUES(&EMPNO,'&ENAME','&DESIGNATIN','&SALARY')
new   1: INSERT INTO EMP VALUES(104,'CHINNI','HOD, PROF','45000')
1 row created.
SQL> SELECT * FROM EMP;

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

UPDATE

QUERY: 04

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

Syntax for update Records from the table:

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

QUERY: 04

```
SQL> UPDATE EMP SET SALARY=16000 WHERE EMPNO=101;
1 row updated.
SQL> SELECT * FROM EMP;
    EMPNO ENAME       DESIGNATIN     SALARY
---------- ---------- ----------     ----------
      101 NAGARAJAN    LECTURER       16000
      102 SARAVANAN    LECTURER       15000
      103 PANNERSELVAM ASST. PROF             20000
      104 CHINNI       HOD, PROF              45000
```

UPDATE MULTIPLE COLUMNS

QUERY: 05

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

Syntax for update multiple Records from the table:

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

QUERY: 05

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

1 row updated.

SQL> SELECT * FROM EMP;

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

DELETE

QUERY: 06

Q5. Write a query to delete records from employee.

Syntax for delete Records from the table:

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

QUERY: 06

SQL> DELETE EMP WHERE EMPNO=103;

1 row deleted.

SQL> SELECT * FROM EMP;

EMPNO	ENAME	DESIGNATIN	SALARY
101	NAGARAJAN	LECTURER	16000
102	SARAVANAN	ASST. PROF	16000

104 CHINNI HOD, PROF 45000

TCL(TRNSACTION CONTROL LANGUAGE)

SAVEPOINT:

QUERY: 07

Q5. Write a query to implement the save point.

Syntax for save point:

SQL> SAVEPOINT <SAVE POINT NAME>;

QUERY: 07

SQL> SAVEPOINT S1;

Savepoint created.

SQL> SELECT * FROM EMP;

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

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

1 row created.

SQL> SELECT * FROM EMP;

EMPNO	ENAME	DESIGNATIN	SALARY
105	PARTHASAR	STUDENT	100

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

ROLL BACK

QUERY: 08

Q5. Write a query to implement the Rollback.

Syntax for save point:

SQL> ROLL BACK <SAVE POINT NAME>;

QUERY: 08

SQL> ROLL BACK S1;
Rollback complete.

SQL> SELECT * FROM EMP;

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

COMMIT

QUERY: 09

Q5. Write a query to implement the Rollback.

Syntax for commit:

SQL> COMMIT;

QUERY: 09

SQL> COMMIT;

Commit complete.

DCL (DATA CONTROL LANGUAGE)

CREATING A USER

SQL>CONNECT SYSTEM/MANAGER;

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

SQL>GRANT DBA TO "USERNAME"

SQL>CONNECT "USERNAME"/"PASSWORD";

EXAMPLE

CREATING A USER

SQL>CONNECT SYSTEM/MANAGER;

SQL>CREATE USER CSE2 IDENTIFIED BY CSECSE;

SQL>GRANT DBA TO CSE2;

SQL>CONNECT CSE2/CSECSE;

SQL>REVOKE DBA FROM CSE2;

DRL-DATA RETRIEVAL IMPLEMENTING ON SELECT COMMANDS

SQL> select * from emp;

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

7566 JONES MANAGER 7839 02-APR-81 2975 2000

4 rows selected.

SQL> select empno,ename,sal from emp;

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

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

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

6 rows selected.

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

EMPNO	ENAME	SAL
7369	SMITH	800
7839	KING	5000

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

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

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

10 rows selected.

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

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

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

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

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

    EMPNO ENAME      JOB              SAL
---------- ---------- --------- ----------
     7369 SMITH      CLERK           800
     7566 JONES      MANAGER          2975
     7698 BLAKE      MANAGER          2850
     7782 CLARK      MANAGER          2450
     7788 SCOTT      ANALYST         3000
SQL> select ename,job,sal from emp where sal>2500;

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


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

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

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

9 rows selected.
```

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

EMPNO	ENAME	JOB	SAL
7369	SMITH	CLERK	800
7900	JAMES	CLERK	950
7876	ADAMS	CLERK	1100
7521	WARD	SALESMAN	1250
7654	MARTIN	SALESMAN	1250
7934	MILLER	CLERK	1300
7844	TURNER	SALESMAN	1500
7499	ALLEN	SALESMAN	1600
7782	CLARK	MANAGER	2450
7698	BLAKE	MANAGER	2850
7566	JONES	MANAGER	2975

EMPNO	ENAME	JOB	SAL
7788	SCOTT	ANALYST	3000
7902	FORD	ANALYST	3000
7839	KING	PRESIDENT	5000

14 rows selected.

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

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

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

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

14 rows selected.

EX: NO: 3 NESTED QUERIES AND JOIN QUERIES

EX: NO: 3 A Nested Queries

AIM

To execute and verify the SQL commands for Nested Queries.

OBJECTIVE:

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

PROCEDURE

STEP 1: Start

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

STEP 3: Insert attribute values into the table.

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

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

STEP 6: Stop

SQL COMMANDS

1. COMMAND NAME: **SELECT**

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

2. COMMAND NAME: **WHERE**

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

3. COMMAND NAME: **HAVING**

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

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

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

Table -1

SYNTAX FOR CREATING A TABLE:

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

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

SYNTAX FOR INSERT RECORDS IN TO A TABLE:

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

INSERTION

```
SQL> INSERT INTO EMP2 VALUES(1001,'MAHESH','PROGRAMMER',15000,1560,200);
1 ROW CREATED.
SQL> INSERT INTO EMP2 VALUES(1002,'MANOJ','TESTER',12000,1560,200);
1 ROW CREATED.
SQL> INSERT INTO EMP2 VALUES(1003,'KARTHIK','PROGRAMMER',13000,1400,201);
1 ROW CREATED.
 SQL> INSERT INTO EMP2 VALUES(1004,'NARESH','CLERK',1400,1400,201);
1 ROW CREATED.
SQL> INSERT INTO EMP2 VALUES(1005,'MANI','TESTER',13000,1400,200);
1 ROW CREATED.
SQL> INSERT INTO EMP2 VALUES(1006,'VIKI','DESIGNER',12500,1560,201);
1 ROW CREATED.
SQL> INSERT INTO EMP2 VALUES(1007,'MOHAN','DESIGNER',14000,1560,201);
```

1 ROW CREATED.

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

1 ROW CREATED.

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

1 ROW CREATED.

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

1 ROW CREATED.

SYNTAX FOR SELECT RECORDS FROM THE TABLE:

SQL> SELECT * FROM <TABLE NAME>;

SQL> SELECT *FROM EMP2;

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

TABLE- 2

SYNTAX FOR CREATING A TABLE:

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

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

Table created.

SYNTAX FOR INSERT RECORDS IN TO A TABLE:

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

INSERTION

```
SQL> INSERT INTO DEPT2 VALUES(107,'DEVELOP','ADYAR');
1 ROW CREATED.
SQL> INSERT INTO DEPT2 VALUES(201,'DEBUG','UK');
1 ROW CREATED.
SQL> INSERT INTO DEPT2 VALUES(200,'TEST','US');
SQL> INSERT INTO DEPT2 VALUES(201,'TEST','USSR');
1 ROW CREATED.
SQL> INSERT INTO DEPT2 VALUES(108,'DEBUG','ADYAR');
1 ROW CREATED.
SQL> INSERT INTO DEPT2 VALUES(109,'BUILD','POTHERI');
1 ROW CREATED.
```

SYNTAX FOR SELECT RECORDS FROM THE TABLE:

SQL> SELECT * FROM <TABLE NAME>;

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

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

6 rows selected.

GENERAL SYNTAX FOR NESTED QUERY:

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

SYNTAX NESTED QUERY STATEMENT:

```
SQL> SELECT <COLUMN_NAME> FROM FRORM <TABLE _1> WHERE
                <COLUMN_NAME> <RELATIONAL _OPERATION> 'VALUE'
      (SELECT (AGGRECATE FUNCTION) FROM <TABLE_1> WHERE <COLUMN
                NAME> = 'VALUE'
      (SELECT <COLUMN_NAME> FROM <TABLE_2> WHERE <COLUMN_NAME=
                'VALUE'));
```

NESTED QUERY STATEMENT:

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

                (SELECT MIN(SAL) FROM EMP2 WHERE DPTNO=

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

Nested Query Output:

ENAME

MAHESH
MANOJ
KARTHIK
MANI
VIKI
MOHAN
NAVEEN
PRASAD
AGNESH

EX: NO: 3 B - JOINS

AIM

To execute and verify the SQL commands using Join queries.

OBJECTIVE:

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

PROCEDURE

STEP 1: Start

STEP 2: Create the table with its essential attributes.

STEP 3: Insert attribute values into the table

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

STEP 5: Stop

SQL COMMANDS

COMMAND NAME: **INNER JOIN**

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

COMMAND NAME LEFT JOIN

COMMAND DESCRIPTION: The LEFT JOIN keyword returns all rows from the left table (table_name1), even if there are no matches in the right table (table_name2).

COMMAND NAME : **RIGHT JOIN**

COMMAND DESCRIPTION: The RIGHT JOIN keyword Return all rows from the right table (table_name2), even if there are no matches in the left table (table_name1).

COMMAND NAME : **FULL JOIN**

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

LEFT JOIN or LEFT OUTTER JOIN

Table:1 - ORDERS

```
SQL> CREATE  table orders(O_Id number(5),
                Orderno number(5),
                P_Id number(3));
Table created.
SQL> DESC orders;
 Name                      Null?    Type
 ----------------------    -------- -----------
 O_ID                               NUMBER(5)
 ORDERNO                            NUMBER(5)
 P_ID                               NUMBER(3)
```

INSERTING VALUES INTO ORDERS

```
SQL> INSERT into orders values(&O_Id,&Orderno,&P_Id);
Enter value for o_id: 1
Enter value for orderno: 77895
Enter value for p_id: 3
old   1: INSERT into orders values(&O_Id,&Orderno,&P_Id)
new   1: INSERT into orders values(1,77895,3)
1 row created.

SQL> INSERT into orders values(&O_Id,&Orderno,&P_Id);
Enter value for o_id: 2
```

Enter value for orderno: 44678

Enter value for p_id: 3

old 1: INSERT into orders values(&O_Id,&Orderno,&P_Id)

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

1 row created.

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

Enter value for o_id: 3

Enter value for orderno: 22456

Enter value for p_id: 1

old 1: INSERT into orders values(&O_Id,&Orderno,&P_Id)

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

1 row created.

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

Enter value for o_id: 4

Enter value for orderno: 24562

Enter value for p_id: 1

old 1: INSERT into orders values(&O_Id,&Orderno,&P_Id)

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

1 row created.

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

Enter value for o_id: 5

Enter value for orderno: 34764

Enter value for p_id: 15

old 1: INSERT into orders values(&O_Id,&Orderno,&P_Id)

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

1 row created.

TABLE SECTION:

SQL> SELECT * FROM orders;

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

TABLE -2: PERSONS

SQL> CREATE table persons(p_Id number(5),

LASTNAME varchar2(10),

Firstname varchar2(15), Address varchar2(20),

city varchar2(10));

Table created.

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

```
Enter value for p_id: 1
Enter value for lastname: Hansen
Enter value for firstname: Ola
Enter value for address: Timoteivn 10
Enter value for city: sadnes
old   1: INSERT into persons values(&p_Id,'&Lastname','&firstname','&Address','&city')
new   1: INSERT into persons values(1,'Hansen','Ola','Timoteivn 10','sadnes')

1 row created.

SQL> INSERT into persons values(&p_Id,'&Lastname','&firstname','&Address','&city');
Enter value for p_id: 2
Enter value for lastname: Svendson
Enter value for firstname: Tove
Enter value for address: Borgn 23
Enter value for city: Sandnes
old   1: INSERT into persons values(&p_Id,'&Lastname','&firstname','&Address','&city')
new   1: INSERT into persons values(2,'Svendson','Tove','Borgn 23','Sandnes')

1 row created.

SQL> INSERT into persons values(&p_Id,'&Lastname','&firstname','&Address','&city');
Enter value for p_id: 3
Enter value for lastname: Pettersen
Enter value for firstname: Kari
```

Enter value for address: Storgt 20

Enter value for city: Stavanger

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

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

1 row created.

SQL> SELECT * FROM persons;

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

LEFT JOIN SYNTAX

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

LEFT JOIN EXAMPLE

```
SQL> SELECT persons.lastname,persons.firstname,orders.orderno
  FROM persons
  LEFT JOIN orders
  ON persons.p_Id = orders.p_Id
```

ORDER BY persons.lastname;

OUTPUT

LASTNAME FIRSTNAME ORDERNO

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

Hansen Ola 22456

Hansen Ola 24562

Pettersen Kari 77895

Pettersen Kari 44678

Svendson Tove

FULL OUTTER JOIN

SQL> SELECT * FROM persons;

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

SQL> SELECT * FROM orders;

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

4	24562	1
5	34764	15

FULL OUTER JOIN SYNTAX

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

FULL OUTER JOIN EXAMPLE

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

RIGHT OUTTER JOIN

RIGHT OUTTER JOIN SYNTAX

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

RIGHT OUTTER JOIN EXAMPLE

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

LASTNAME FIRSTNAME ORDERNO

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

Hansen Ola 24562

Hansen Ola 22456

Pettersen Kari 44678

Pettersen Kari 77895

INNER JOIN

INNTER JOIN SYNTAX

SQL>SELECT column_name(s)

FROM table_name1
INNER JOIN table_name2
ON table_name1.column_name=table_name2.column_name

INNTER JOIN EXAMPLE

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

2 FROM persons

3 INNER JOIN orders

4 ON persons.p_Id = orders.p_Id

5 ORDER BY persons.lastname;

LASTNAME FIRSTNAME ORDERNO

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

Hansen Ola 22456

Hansen Ola 24562

Pettersen Kari 77895

Pettersen Kari 44678

LASTNAME	FIRSTNAME	ORDERNO
Hansen	Ola	22456
Hansen	Ola	24562
Pettersen	Kari	77895
Pettersen	Kari	44678
Svendson	Tove	34764

6 rows selected.

EX: NO: 4 VIEWS

AIM

To execute and verify the SQL commands for Views.

OBJECTIVE:

- Views Helps to encapsulate complex query and make it reusable.
- Provides user security on each view - it depends on your data policy security.
- Using view to convert units - if you have a financial data in US currency, you can create view to convert them into Euro for viewing in Euro currency.

PROCEDURE

STEP 1: Start

STEP 2: Create the table with its essential attributes.

STEP 3: Insert attribute values into the table.

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

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

STEP 6: Stop

.SQL COMMANDS

1. COMMAND NAME: **CREATE VIEW**

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

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

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

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

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

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

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

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

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

COMMANDS EXECUTION

CREATION OF TABLE

```
SQL> CREATE TABLE EMPLOYEE (
                                EMPLOYEE_NAMEVARCHAR2(10),
                                EMPLOYEE_NONUMBER(8),
                                DEPT_NAME VARCHAR2(10),
                                DEPT_NO NUMBER (5),DATE_OF_JOIN DATE);
```

Table created.

TABLE DESCRIPTION

```
SQL> DESC EMPLOYEE;
 NAME                          NULL?          TYPE
 ----------------------------- -------- ----------------------
 EMPLOYEE_NAME                          VARCHAR2(10)
 EMPLOYEE_NO                            NUMBER(8)
 DEPT_NAME                              VARCHAR2(10)
 DEPT_NO                                NUMBER(5)
 DATE_OF_JOIN                           DATE
```

SUNTAX FOR CREATION OF VIEW

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

CREATION OF VIEW

```
SQL> CREATE VIEW EMPVIEW AS SELECT
EMPLOYEE_NAME,EMPLOYEE_NO,DEPT_NAME,DEPT_NO,DATE_OF_JOIN FROM
EMPLOYEE;
```

VIEW CREATED.

DESCRIPTION OF VIEW

```
SQL> DESC EMPVIEW;

NAME                          NULL?    TYPE
 ----------------------------------------- -------- ----------------------------
 EMPLOYEE_NAME                                VARCHAR2(10)
 EMPLOYEE_NO                                NUMBER(8)
 DEPT_NAME                                 VARCHAR2(10)
 DEPT_NO                                  NUMBER(5)
```

DISPLAY VIEW:

SQL> SELECT * FROM EMPVIEW;

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

INSERTION INTO VIEW

INSERT STATEMENT:
SYNTAX:
SQL> INSERT INTO <VIEW_NAME> (COLUMN NAME1,…….)
VALUES(VALUE1,….);

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

1 ROW CREATED.

SQL> SELECT * FROM EMPVIEW;

```
EMPLOYEE_N EMPLOYEE_NO DEPT_NAME     DEPT_NO
---------- ----------- ---------- ----------
RAVI             124 ECE              89
VIJAY            345 CSE              21
RAJ               98 IT               22
GIRI             100 CSE              67
SRI              120 CSE              67
```

SQL> SELECT * FROM EMPLOYEE;

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

DELETION OF VIEW:

DELETE STATEMENT:
SYNTAX:
SQL> DELETE <VIEW_NMAE>WHERE <COLUMN NMAE> =’VALUE’;

SQL> DELETE FROM EMPVIEW WHERE EMPLOYEE_NAME='SRI';

1 ROW DELETED.

SQL> SELECT * FROM EMPVIEW;

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

UPDATE STATEMENT:
SYNTAX:
AQL>UPDATE <VIEW_NAME> SET< COLUMN NAME> = <COLUMN NAME>
+<VIEW> WHERE <COLUMNNAME>=VALUE;

SQL> UPDATE EMPKAVIVIEW SET EMPLOYEE_NAME='KAVI' WHERE
EMPLOYEE_NAME='RAVI';

1 ROW UPDATED.

SQL> SELECT * FROM EMPKAVIVIEW;

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

DROP A VIEW:
SYNTAX:
SQL> DROP VIEW <VIEW_NAME>
EXAMPLE

SQL>DROP VIEW EMPVIEW;

VIEW DROPED

CREATE A VIEW WITH SELECTED FIELDS:
SYNTAX:

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

EXAMPLE-2:
SQL> CREATE OR REPLACE VIEW EMPL_VIEW1 AS SELECT EMPNO, ENAME,
SALARY FROM EMPL;

SQL> SELECT * FROM EMPL_VIEW1;

EXAMPLE-3:

SQL> CREATE OR REPLACE VIEW EMPL_VIEW2 AS SELECT * FROM EMPL WHERE DEPTNO=10;

SQL> SELECT * FROM EMPL_VIEW2;

Note:

→Replace is the keyboard to avoid the error "ora_0095:name is already used by an existing abject".

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

TYPE-1:

SQL> CREATE OR REPLACE VIEW EMP_TOTSAL(EID,NAME,SAL) AS SELECT EMPNO,ENAME,SALARY FROM EMPL;

View created.

EMPNO	ENAME	SALARY
7369	SMITH	1000
7499	MARK	1050
7565	WILL	1500
7678	JOHN	1800
7578	TOM	1500
7548	TURNER	1500

6 rows selected.

View created.

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

View created.

SQL> SELECT * FROM EMP_TOTSAL;

TYPE-2:

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

SQL> SELECT * FROM EMP_TOTSAL;

EXAMPLE FOR JOIN VIEW:

TYPE-3:

```
SQL> CREATE OR REPLACE VIEW DEPT_EMP AS SELECT A.EMPNO "EID",A.ENAME
"EMPNAME",A.DEPTNO "DNO",B.DNAM
E "D_NAME",B.LOC "D_LOC" FROM EMPL A,DEPMT B WHERE
A.DEPTNO=B.DEPTNO;

SQL> SELECT * FROM DEPT_EMP;
EID     NAME                 SAL
---------- -------------------- ---------- ----------
     7369 SMITH                 1000
     7499 MARK                  1050
     7565 WILL                 1500
     7678 JOHN                 1800
     7578 TOM                  1500
     7548 TURNER                1500
6 rows selected.

View created.

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

View created.

   EID EMPNAME       DNO       D_NAME    D_LOC
---------- -------------------- ---------- ---------- ----------
     7578 TOM              10      ACCOUNT    NEW YORK
     7548 TURNER           10      ACCOUNT    NEW YORK
     7369 SMITH            20       SALES      CHICAGO
     7678 JOHN             20       SALES      CHICAGO
     7499 MARK             30    RESEARCH      ZURICH
     7565 WILL             30    RESEARCH      ZURICH
```

VIEW READ ONLY AND CHECK OPTION:

READ ONLY CLAUSE:

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

EXAMPLE-4:

```
SQL>CREATE OR REPLACE VIEW EMP_NO_DML AS SELECT * FROM EMPL WITH
READ ONLY;
```

WITH CHECK OPTION CLAUSE

EXAMPLE-4:

```
SQL> CREATE OR REPLACE VIEW EMP_CK_OPTION AS SELECT
EMPNO,ENAME,SALARY,DEPTNO FROM EMPL WHERE DEPTNO
=10 WITH CHECK OPTION;


SQL> SELECT * FROM EMP_CK_OPTION;
```

JOIN VIEW:

EXAMPLE-5:

```
SQL> CREATE OR REPLACE VIEW DEPT_EMP_VIEW AS SELECT A.EMPNO,
A.ENAME, A.DEPTNO, B.DNAME, B.LOC FROM EMPL
 A,DEPMT B WHERE A.DEPTNO=B.DEPTNO;

SQL> SELECT * FROM DEPT_EMP_VIEW;
View created.


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


View created.

    EMPNO ENAME       DEPTNO    DNAME   LOC
----------------------------- ---------- ---------- --------------------
7578 TOM                   10    ACCOUNT    NEW YORK
7548 TURNER                10    ACCOUNT     NEW YORK
 7369 SMITH                20     SALES        CHICAGO
 7678 JOHN                 20      SALES          CHICAGO
 7499 MARK                 30  RESEARCH       ZURICH
 7565 WILL                 30  RESEARCH    ZURICH
6 rows selected.
```

FORCE VIEW

EXAMPLE-6:

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


SQL> SELECT * FROM MYVIEW;

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

SQL> SELECT * FROM XYZ;
```

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

SQL> SELECT * FROM MYVIEW;

Warning: View created with compilation errors.

SELECT * FROM MYVIEW
       *
ERROR at line 1:
ORA-04063: view "4039.MYVIEW" has errors


Table created.

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


View created.

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

COMPILING A VIEW

SYNTAX:

ALTER VIEW <VIEW_NAME> COMPILE;

EXAMPLE:

```
SQL> ALTER VIEW MYVIEW COMPILE;
```

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

EX: NO: 5 A CONTROL STRCTURE

AIM

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

OBJECTIVE:

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

Addition of Two Numbers:

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

PROCEDURE

STEP 1: Start

STEP 2: Initialize the necessary variables.

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

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

STEP 5: Execute the statements.

STEP 6: Stop.

PL/ SQL General Syntax

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

PL/SQL CODING FOR ADDITION OF TWO NUMBERS

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

INPUT:

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

OUTPUT:

```
sum of23and12is35

PL/SQL procedure successfully completed.
```

PL/ SQL Program for IF Condition:

2. Write a PL/SQL Program using if condition

PROCEDURE

STEP 1: Start

STEP 2: Initialize the necessary variables.

STEP 3: invoke the if condition.

STEP 4: Execute the statements.

STEP 5: Stop.

PL/ SQL GENERAL SYNTAX FOR IF CONDITION:

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

Coding for If Statement:

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

OUTPUT:

C is maximum

PL/SQL procedure successfully completed.

PL/ SQL GENERAL SYNTAX FOR IF AND ELSECONDITION:

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

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

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

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

Input

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

Output:

entered number is less than 5

PL/SQL procedure successfully completed.

PL/ SQL GENERAL SYNTAX FOR NESTED IF:

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

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

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

INPUT:

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

OUTPUT:

```
C is maximum

PL/SQL procedure successfully completed.
```

PL/ SQL GENERAL SYNTAX FOR LOOPING STATEMENT:

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

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

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

INPUT:

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

OUTPUT:

sum =4

PL/SQL procedure successfully completed.

PL/ SQL GENERAL SYNTAX FOR LOOPING STATEMENT:

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

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

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

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

INPUT:

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

OUTPUT:

```
sum of odd no. bt 1 and4is4


PL/SQL procedure successfully completed.
```

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

EX: NO:5B PROCEDURES

AIM

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

ALGORITHM

STEP1:Start

STEP2:Create a table with table name stud_exam

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

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

STEP5: Display the total and average of student

STEP6: End

EXECUTION

SETTING SERVEROUTPUT ON:

SQL> SET SERVEROUTPUT ON

I) PROGRAM:

PROCEDURE USING POSITIONAL PARAMETERS:

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

Output:

Procedure created.

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

PL/SQL procedure successfully completed.

II) PROGRAM:

PROCEDURE USING NOTATIONAL PARAMETERS:

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

Output:

Procedure created.

```
SQL> VARIABLE T NUMBER

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

PL/SQL procedure successfully completed.

```
SQL> PRINT T

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

PROCEDURE FOR GCD NUMBERS

III) PROGRAM:

```
SQL> create or replace procedure pro
        is
        a number(3);
        b number(3);
        c number(3);
        d number(3);
        begin
        a:=&a;
        b:=&b;
        if(a>b) then
        c:=mod(a,b);
        if(c=0) then
        dbms_output.put_line('GCD is');
        dbms_output.put_line(b);
        else
        dbms_output.put_line('GCD is');
        dbms_output.put_line(c);
        end if;
        else
        d:=mod(b,a);
        if(d=0) then
        dbms_output.put_line('GCD is');
        dbms_output.put_line(a);
        else
        dbms_output.put_line('GCD is');
        dbms_output.put_line(d);
        end if;
        end if;
        end;
        /
Enter value for a: 8
old   8: a:=&a;
new   8: a:=8;
Enter value for b: 16
old   9: b:=&b;
new   9: b:=16;
Procedure created.

SQL> set serveroutput on;
SQL> execute pro;
GCD is
```

8
PL/SQL procedure successfully completed.

PROCEDURE FOR CURSOR IMPLEMENATION

IV) PROGRAM:

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

     Table created

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

     1 row created.
SQL> insert into student values (102,'surya', 99,77,69,81,99);

     1 row created.
SQL> insert into student values (103,'suryapriya', 100,90,97,89,91);

     1 row created.
SQL> select * from student;

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

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

SAMPLE OUTPUT

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

PL/SQL procedure successfully completed.

EXPLICIT CURSORS AND EXPLICIT CURSORS IMPLEMENTATION

CREATING A TABLE EMP IN ORACLE

V) PROGRAM

SQL> select * from EMP;

```
    EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM
---------- ---------- --------- ---------- --------- ---------- ----------
    DEPTNO
----------
     7369 SMITH      CLERK           7902 17-DEC-80        800
        20

     7499 ALLEN      SALESMAN        7698 20-FEB-81       1600        300
        30

     7521 WARD       SALESMAN        7698 22-FEB-81       1250        500
        30


    EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM
---------- ---------- --------- ---------- --------- ---------- ----------
    DEPTNO
----------
     7566 JONES      MANAGER         7839 02-APR-81       2975
        20

     7654 MARTIN     SALESMAN        7698 28-SEP-81       1250        1400
        30

     7698 BLAKE      MANAGER         7839 01-MAY-81       2850
        30


    EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM
---------- ---------- --------- ---------- --------- ---------- ----------
    DEPTNO
----------
     7782 CLARK      MANAGER         7839 09-JUN-81       2450
        10

     7788 SCOTT      ANALYST         7566 09-DEC-82       3000
        20
```

```
     7839 KING       PRESIDENT            17-NOV-81       5000
       10


    EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM
---------- ---------- --------- ---------- --------- ---------- ----------
    DEPTNO
----------
     7844 TURNER     SALESMAN         7698 08-SEP-81       1500          0
       30

     7876 ADAMS      CLERK            7788 12-JAN-83       1100
       20

     7900 JAMES      CLERK            7698 03-DEC-81        950
       30


    EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM
---------- ---------- --------- ---------- --------- ---------- ----------
    DEPTNO
----------
     7902  FORD      ANALYST          7566 03-DEC-81       3000
       20

     7934 MILLER     CLERK            7782 23-JAN-82       1300
       10


14 rows selected.
```

Implicit curscors:

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

```
14 /
```

Output:

```
Enter value for empno: 7844
old   6:                  WHERE EMPNO = &EMPNO;
new   6:                   WHERE EMPNO = 7844;

PL/SQL procedure successfully completed.
```

Explicit Cursors:

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

Output:

```
SMITH salry is $ 800
ALLEN salry is $ 1600
WARD salry is $ 1250
```

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

EX: NO: 5C FUNCTIONS

AIM

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

PROCEDURE

STEP 1: Start

STEP 2: Create the table with essential attributes.

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

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

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

STEP 6: Stop

EXECUTION

SETTING SERVEROUTPUT ON:

```
SQL> SET SERVEROUTPUT ON
```

IMPLEMENTATION OF FACTORIAL USING FUNCTION

I) PROGRAM:

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

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

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

Function created.

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

PL/SQL procedure successfully completed.

II) PROGRAM

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

Table created.

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

1 row created.

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

1 row created.

SQL> select * from phonebook;

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

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

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

Function created.

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

OUTPUT 1:
Vijay,120/5D,bharathi street,NGO colony,629002

PL/SQL procedure successfully completed.

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

OUTPUT2:
Address not found

PL/SQL procedure successfully completed.

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

EX:NO:6 FRONT END TOOLS

AIM

To design a form using different tools in Visual Basic.

PROCEDURE

STEP 1: Start

STEP 2: Create the form with essential controls in tool box.

STEP 3: Write the code for doing the appropriate functions.

STEP 4: Save the forms and project.

STEP 5: Execute the form .

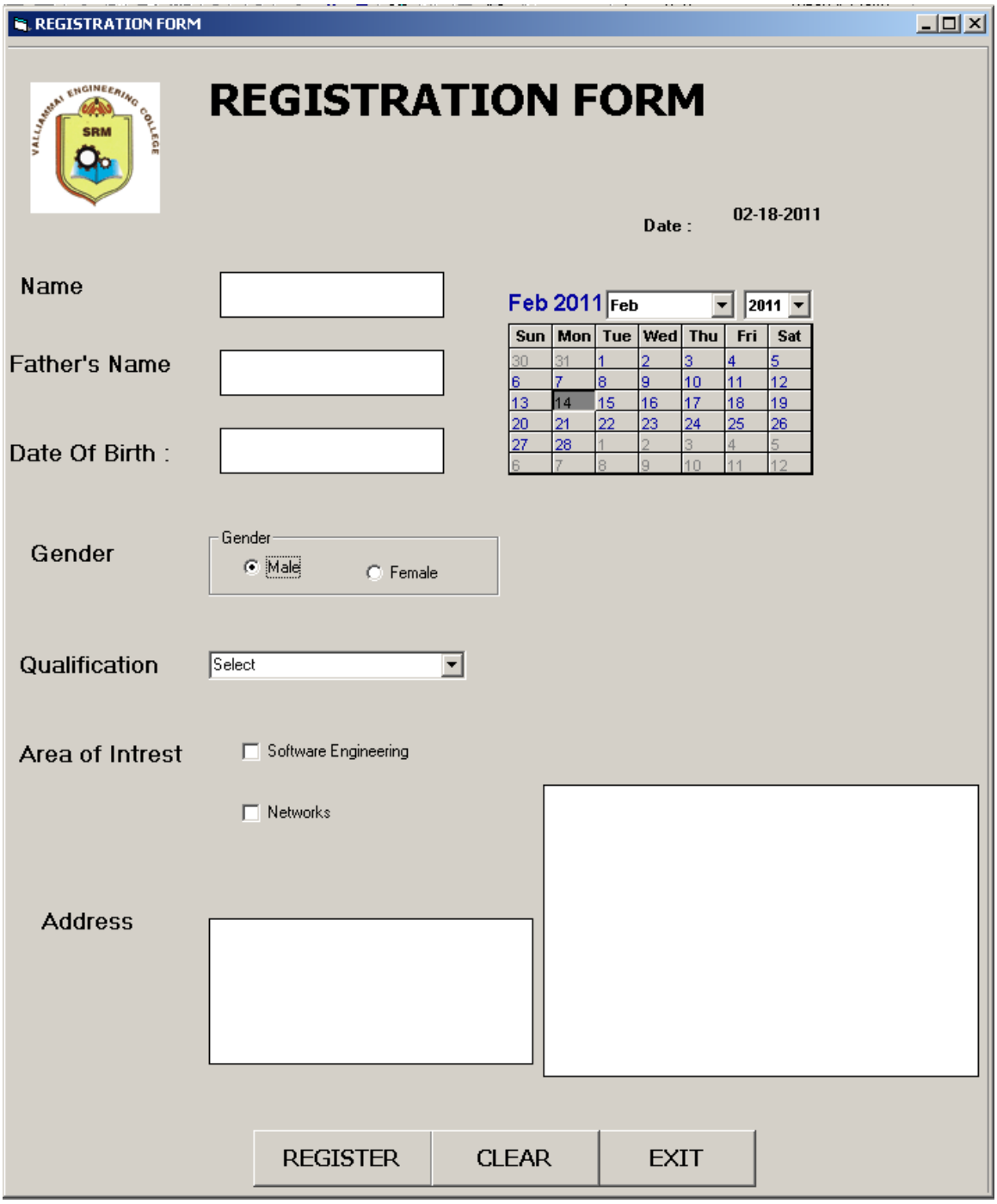
STEP 6: Stop

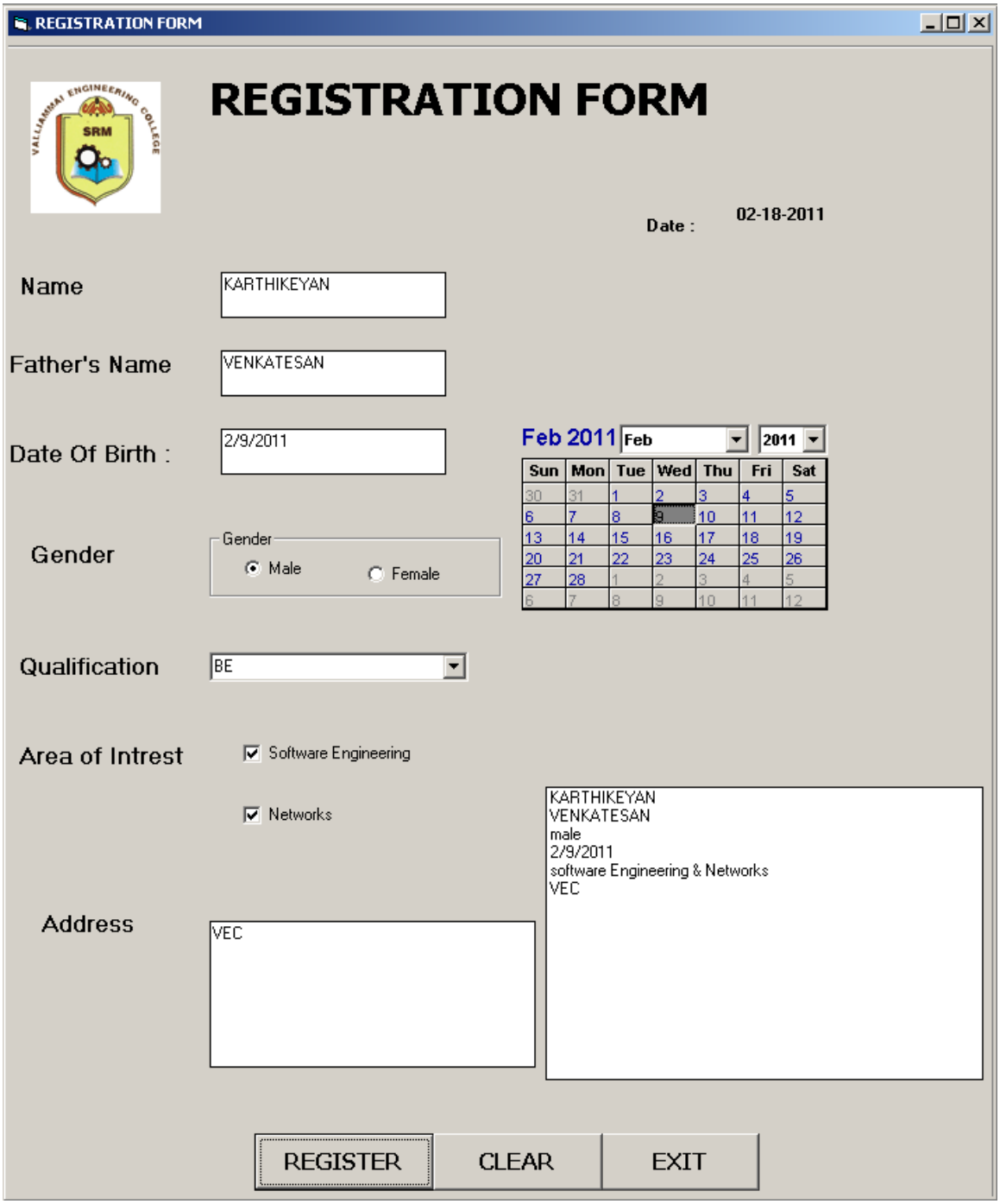
CODING:

```
Private Sub Calendar1_Click()
Text3.Text = Calendar1.Value
End Sub

Private Sub Combo1_Change()
Combo1.AddItem "BSC"
Combo1.AddItem "MSC"
Combo1.AddItem "BE"
Combo1.AddItem "ME"
End Sub

Private Sub Command1_Click()
List1.AddItem Text1.Text
List1.AddItem Text2.Text
If Option1.Value = True Then
gender = "male"
End If
If Option2.Value = True Then
gender = "female"
End If
List1.AddItem gender
```

```
List1.AddItem Text3.Text

If Check1.Value = 1 And Check2.Value = 1 Then
area = "software Engineering & Networks"
End If
If Check1.Value = 0 And Check2.Value = 1 Then
area = " Networks"
End If
List1.AddItem area
List1.AddItem Text4.Text

End Sub


Private Sub Command2_Click()
End
End Sub

Private Sub Command3_Click()
If List1.ListIndex <> 0 Then
List1.RemoveItem (0)
End If
End Sub

Private Sub Form_Load()
Label10.Caption = Date$
MsgBox "Welcome to Registration"

End Sub

Private Sub Option1_Click()
If (Option1.Value = True) Then
MsgBox ("You have selected Male")
ElseIf (Option2.Value = True) Then
MsgBox ("You have selected Female")
End If
End Sub

Private Sub Option2_Click()
If (Option1.Value = True) Then
MsgBox ("You have selected Male")
ElseIf (Option2.Value = True) Then
MsgBox ("You have selected Female")
End If
End Sub
```

REGISTRATION FORM:





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

EX: NO:7 FORM DESIGN

AIM

To design a Single Document Interface and Multiple Document Interface forms using Visual Basic.

PROCEDURE

STEP 1: Start

STEP 2: Create the form with essential controls in tool box.

STEP 3: Write the code for doing the appropriate functions.

STEP 4: Save the forms and project.

STEP 5: Execute the form.

STEP 6: Stop

EXECUTION

Code for Dialog Menu:

```
Private Sub OKButton_Click()
If (Option1.Value = True) Then
SDI.Show
Unload Me
Else
MDIForm1.Show
Unload Me
End If
End Sub
```

Code for MDI Menu:

```
Private Sub ADD_Click()
MDIADD.Show
End Sub

Private Sub DIV_Click()
MDIDIV.Show
End Sub
```

```
Private Sub EXIT_Click()
End
End Sub

Private Sub MUL_Click()
MDIMUL.Show
End Sub

Private Sub SUB_Click()
MDISUB.Show
End Sub
```

Code for MDI ADD:

```
Private Sub Command1_Click()
Dim a As Integer
a = Val(Text1.Text) + Val(Text2.Text)
MsgBox ("Addition of Two numbers is" + Str(a))
End Sub

Private Sub Command5_Click()
MDIForm1.Show
End Sub
```

Code for MDI DIV:

```
Private Sub Command1_Click()
Dim a As Integer
a = Val(Text1.Text) / Val(Text2.Text)
MsgBox ("Addition of Two numbers is" + Str(a))
End Sub

Private Sub Command5_Click()
MDIForm1.Show
End Sub
```

Code for MDI MUL:

```
Private Sub Command1_Click()
Dim a As Integer
a = Val(Text1.Text) * Val(Text2.Text)
MsgBox ("Addition of Two numbers is" + Str(a))
End Sub

Private Sub Command5_Click()
MDIForm1.Show
End Sub
```

Code for MDI SUB:

```
Private Sub Command1_Click()
Dim a As Integer
a = Val(Text1.Text) - Val(Text2.Text)
MsgBox ("Addition of Two numbers is" + Str(a))
End Sub

Private Sub Command5_Click()
MDIForm1.Show
End Sub
```

Code for SDI MENU:

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

Private Sub Command2_Click()
SDIMUL.Show
End Sub

Private Sub Command3_Click()
SDIDIV.Show
End Sub

Private Sub Command4_Click()
SDISUB.Show
End Sub

Private Sub Command5_Click()
Dialog.Show
Unload Me
End Sub
```

Code for SDI ADD:

```
Private Sub Command1_Click()
Dim a As Integer
a = Val(Text1.Text) + Val(Text2.Text)
MsgBox ("Addition of Two numbers is" + Str(a))
Unload Me
End Sub
```

```
Private Sub Command5_Click()
SDI.Show
Unload Me
End Sub
```

Code for SDI DIV:

```
Private Sub Command2_Click()
a = Val(Text1.Text) / Val(Text2.Text)
MsgBox ("Addition of Two numbers is" + Str(a))
Unload Me
End Sub

Private Sub Command5_Click()
SDI.Show
Unload Me
End Sub
```

Code for SDI MUL:

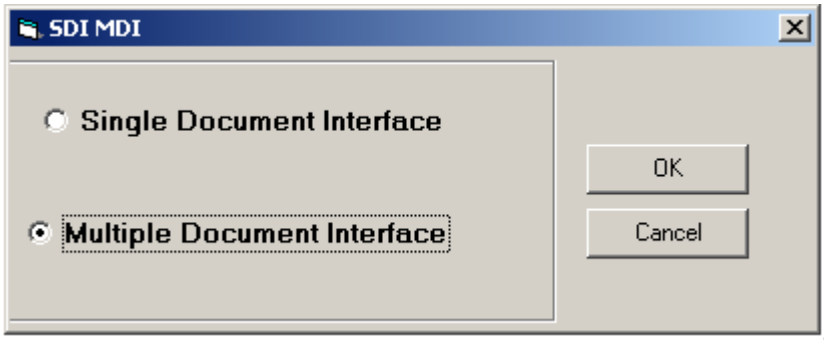
```
Private Sub Command2_Click()
a = Val(Text1.Text) * Val(Text2.Text)
MsgBox ("Addition of Two numbers is" + Str(a))
Unload Me
End Sub

Private Sub Command5_Click()
SDI.Show
Unload Me
End Sub
```

Code for SDI SUB:

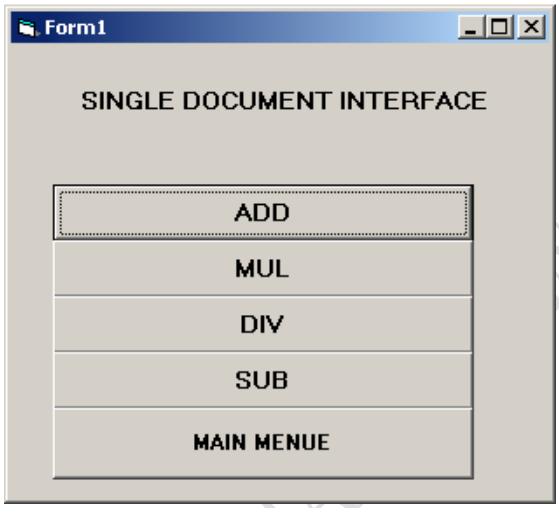
```
Private Sub Command2_Click()
a = Val(Text1.Text) - Val(Text2.Text)
MsgBox ("Addition of Two numbers is" + Str(a))
Unload Me
End Sub

Private Sub Command5_Click()
SDI.Show
Unload Me
End Sub
```

Output:

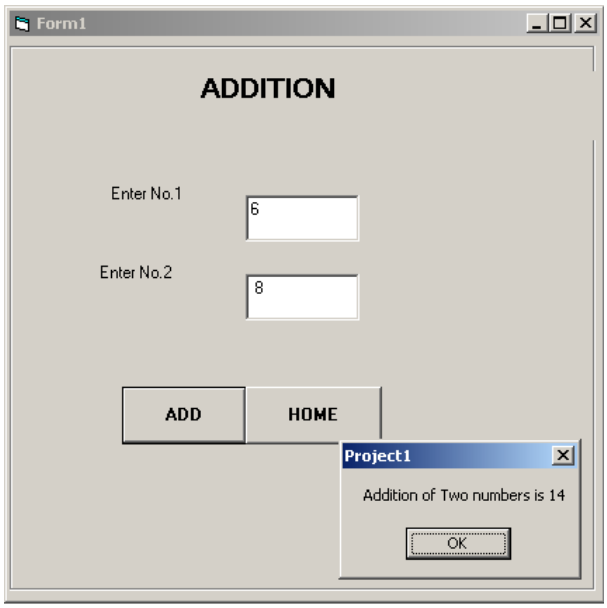
SDI MDI MENU:



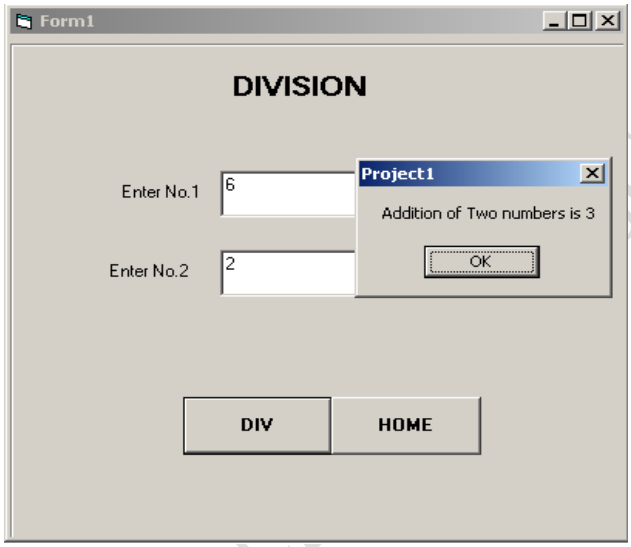
SDI MENU:



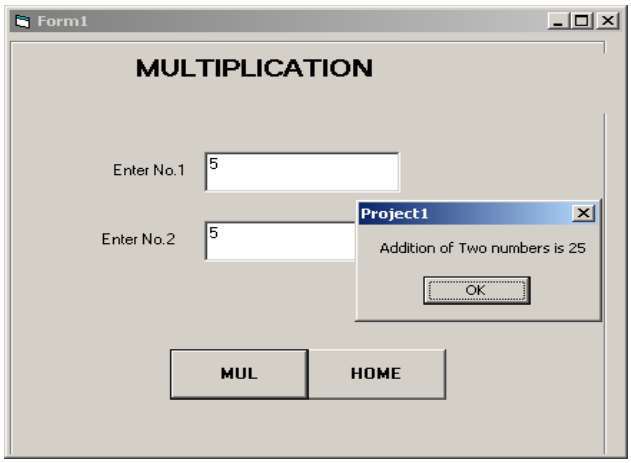
SDI ADDITION:



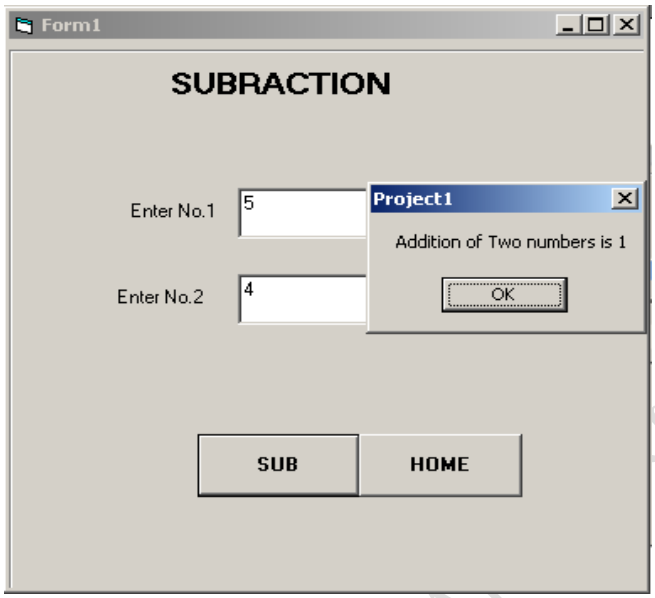
SDI DIVISION



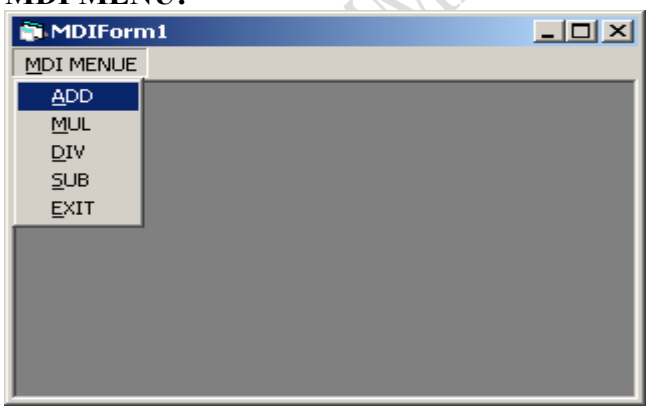
SDI MULTIPLICATION



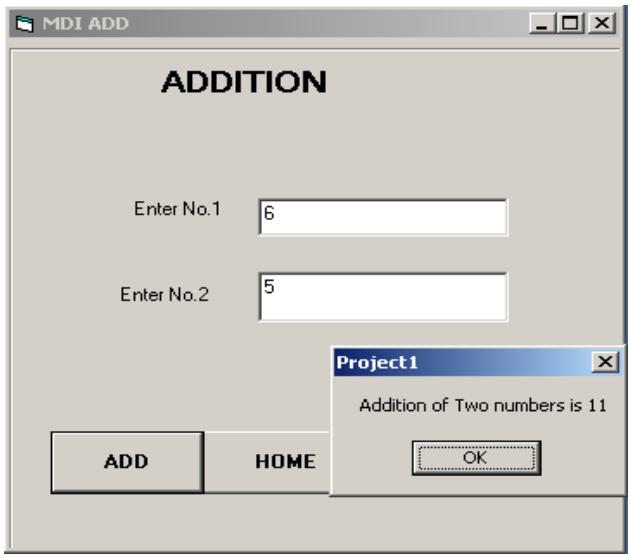
SDI SUBRACTION



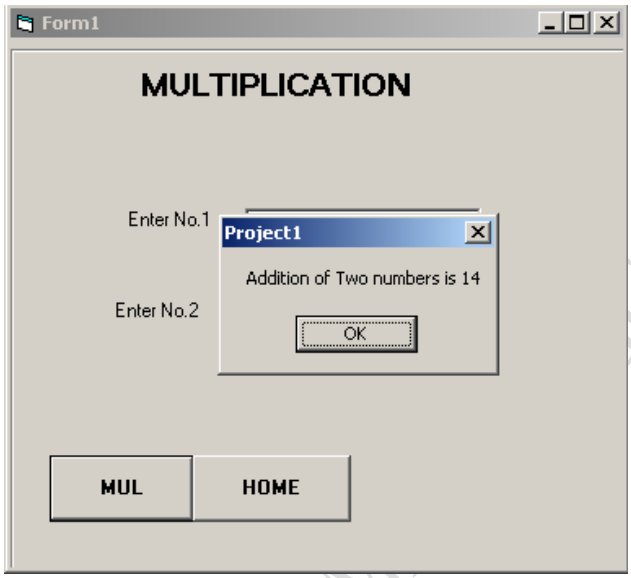
MDI MENU:



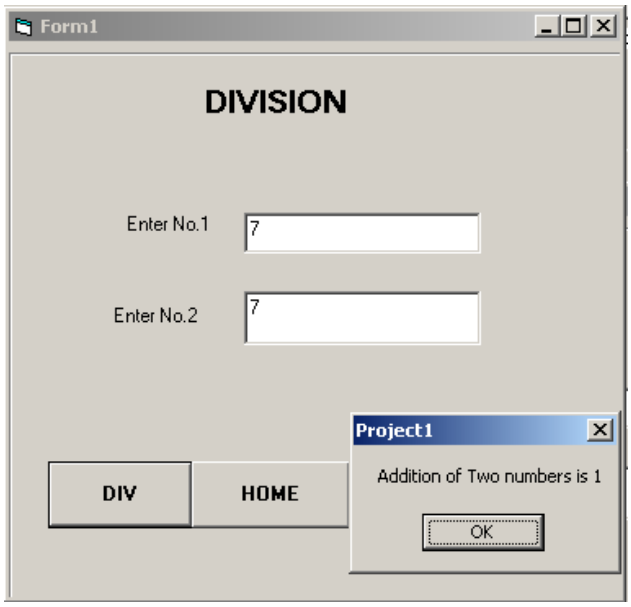
MDI ADD:



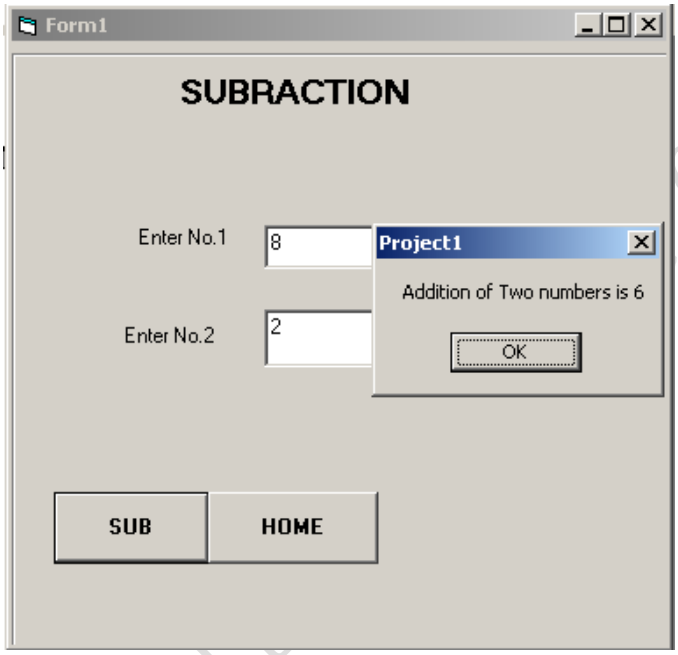
MDI MUL:



MDI DIV



MDI SUB



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

EX: NO: 8 TRIGGER

AIM

To develop and execute a Trigger for Before and After update, Delete, Insert operations on a table.

PROCEDURE

STEP 1: Start

STEP 2: Initialize the trigger with specific table id.

STEP 3:Specify the operations (update, delete, insert) for which the trigger has to be executed.

STEP 4: Execute the Trigger procedure for both Before and After sequences

STEP 5: Carryout the operation on the table to check for Trigger execution.

STEP 6: Stop

EXECUTION

1. **Create a Trigger to pop-up the DML operations**

SQL> create table empa(id number(3),name varchar2(10),income number(4),expence number(3),savings number(3));

Table created.

SQL> insert into empa values(2,'kumar',2500,150,650);
1 row created.

SQL> insert into empa values(3,'venky',5000,900,950);

1 row created.

SQL> insert into empa values(4,'anish',9999,999,999);

1 row created.

```
SQL> select * from empa;

ID         NAME       INCOME    EXPENCE    SAVINGS
---------- ---------- ---------- ---------- ----------------------------------
 2          kumar      2500       150          650
 3          venky      5000       900          950
 4          anish      9999       999          999
```

TYPE 1- TRIGGER AFTER UPDATE

```
SQL> CREATE OR REPLACE TRIGGER VIJAY
      AFTER UPDATE OR INSERT OR DELETE ON EMP
      FOR EACH ROW
      BEGIN
     IF UPDATING THEN
      DBMS_OUTPUT.PUT_LINE('TABLE IS UPDATED');
     ELSIF INSERTING THEN
     DBMS_OUTPUT.PUT_LINE('TABLE IS INSERTED');
     ELSIF DELETING THEN
     DBMS_OUTPUT.PUT_LINE('TABLE IS DELETED');
     END IF;
     END;
     /
Trigger created.

SQL> update emp set income =900 where empname='kumar';
TABLE IS UPDATED
1 row updated.

SQL> insert into emp values ( 4,'Chandru',700,250,80);
TABLE IS INSERTED
1 row created.

SQL> DELETE FROM EMP WHERE EMPID = 4;
TABLE IS DELETED
1 row deleted.

SQL> select * from emp;

   EMPID EMPNAME     INCOME   EXPENSE   SAVINGS
   ------ ---------- ---------- ---------- ----------
     2        vivek       830        150        100
     3        kumar      5000        550         50
     9        vasanth     987       6554        644
```

TYPE 2 - TRIGGER BEFORE UPDATE

```
SQL> CREATE OR REPLACE TRIGGER VASANTH
        BEFORE UPDATE OR INSERT OR DELETE ON EMPLOYEE
        FOR EACH ROW
        BEGIN
        IF UPDATING THEN
        DBMS_OUTPUT.PUT_LINE('TABLE IS UPDATED');
        ELSIF INSERTING THEN
        DBMS_OUTPUT.PUT_LINE('TABLE IS INSERTED');
        ELSIF DELETING THEN
        DBMS_OUTPUT.PUT_LINE('TABLE IS DELETED');
        END IF;
        END;
        /

Trigger created.

SQL> INSERT INTO EMP VALUES (4,'SANKAR',700,98,564);
TABLE IS INSERTED

1 row created.

SQL> UPDATE EMP SET EMPID = 5 WHERE EMPNAME = 'SANKAR';
TABLE IS UPDATED

1 row updated.

SQL> DELETE EMP WHERE EMPNAME='SANKAR';
TABLE IS DELETED

1 row deleted.
```


2. **Create a Trigger to check the age valid or not Using Message Alert**

```
SQL> CREATE TABLE TRIG(NAME CHAR(10),AGE NUMBER(3));

SQL> DESC TRIG;

Table created.

Name                            Null?    Type
```

```
----------------------------------------- -------- -------------------------
NAME                                       CHAR(10)
AGE                                        NUMBER(3)
```

PROGRAM:

```
SQL> SET SERVEROUTPUT ON;
SQL>  CREATE TRIGGER TRIGNEW
        AFTER INSERT OR UPDATE OF AGE ON TRIG
        FOR EACH ROW
        BEGIN
        IF(:NEW.AGE<0) THEN
        DBMS_OUTPUT.PUT_LINE('INVALID AGE');
        ELSE
        DBMS_OUTPUT.PUT_LINE('VALID AGE');
        END IF;
        END;
        /

Trigger created.

SQL> insert into trig values('abc',15);
Valid age
1 row created.

SQL> insert into trig values('xyz',-12);
Invalid age
1 row created.

NAME           AGE
---------- ----------
abc            15
xyz            -12
```

3. **Create a Trigger to check the age valid and Raise appropriate error code and error message.**

```
SQL> create table data(name char(10),age number(3));
Table created.

SQL> desc data;

Name                                      Null?    Type
 ----------------------------------------- -------- ------------------------
```

```
NAME                            CHAR(10)
AGE                             NUMBER(3)
```

```
SQL>  CREATE TRIGGER DATACHECK
        AFTER INSERT OR UPDATE OF AGE ON DATA
        FOR EACH ROW
        BEGIN
        IF(:NEW.AGE<0) THEN
        RAISE_APPLICATION_ERROR(-20000,'NO NEGATIVE AGE ALLOWED');
        END IF;
        END;
        /
Trigger created.

SQL> INSERT INTO DATA VALUES('ABC',10);
1 ROW CREATED.


SQL> INSERT INTO DATA VALUES ('DEF',-15)
            *
ERROR at line 1:
ORA-20000: No negative age allowed
ORA-06512: at "4039.DATACHECK", line 3
ORA-04088: error during execution of trigger '4039.DATACHECK'

NAME          AGE
----------    ----------
abc             10
```

4. **Create a Trigger for EMP table it will update another table SALARY while inserting values.**

```
SQL> CREATE TABLE SRM_EMP2(INAME VARCHAR2(10),
                           IID NUMBER(5),
                           SALARY NUMBER(10));
Table created.

SQL>  CREATE TABLE SRM_SAL2(INAME VARCHAR2(10),
                           TOTALEMP NUMBER(5),
                           TOTALSAL NUMBER(10));
Table created.
```

```
SQL> CREATE OR REPLACE TRIGGER EMPTRIGR22 AFTER INSERT ON SRM_EMP2
       FOR EACH ROW
         DECLARE
         A VARCHAR2(10);
         BEGIN
         A:=:NEW.INAME;
         UPDATE SRM_SAL2 SET
         TOTALSAL=TOTALSAL+:NEW.SALARY,TOTALEMP=TOTALEMP+1 WHERE
         INAME=A;
 END;
/
Trigger created.


SQL> INSERT INTO SRM_SAL2 VALUES('VEC',0,0);

1 row created.

SQL>  INSERT INTO SRM_SAL2 VALUES('SRM',0,0);

1 row created.

SQL> INSERT INTO SRM_EMP2 VALUES('VEC',100,1000);

1 row created.

SQL> SELECT * FROM SRM_SAL2;


INAME      TOTALEMP   TOTALSAL
---------- ---------- ----------
VEC             1       1000
SRM             0          0

SQL> INSERT INTO SRM_EMP2 VALUES('SRM',200,3000);

1 row created.

SQL>  SELECT * FROM SRM_SAL2;

INAME      TOTALEMP   TOTALSAL
---------- ---------- ----------
VEC             1       1000
SRM             1       3000

SQL>  INSERT INTO SRM_EMP2 VALUES('VEC',100,5000);
```

```
1 row created.

SQL>  SELECT * FROM SRM_SAL2;

INAME      TOTALEMP   TOTALSAL
---------- ---------- ----------
VEC             2       6000
SRM             1       3000

SQL> INSERT INTO SRM_EMP2 VALUES('VEC',100,2000);

1 row created.

SQL> SELECT * FROM SRM_SAL2;

INAME      TOTALEMP   TOTALSAL
---------- ---------- ----------
VEC             3       8000
SRM             1       3000

SQL> INSERT INTO SRM_EMP2 VALUES('SRM',200,8000);

1 row created.


SQL> SELECT * FROM SRM_SAL2;

INAME      TOTALEMP   TOTALSAL
---------- ---------- ----------
VEC             3       8000
SRM             2      11000
```

RESULT: Thus the Trigger procedure has been executed successfully for both before and after sequences.

EX:NO:9 MENU DESIGN

AIM

To design a Note Pad Application menu using Visual Basic.

PROCEDURE

STEP 1: Start

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.

STEP 4: Save the forms and project.

STEP 5: Execute the form.

STEP 6: Stop

EXECUTION

Coding:

```
Private Sub ab_Click()
RichTextBox1.SelFontName = "Arial Black"
End Sub

Private Sub al_Click()

End Sub

Private Sub bold_Click()
RichTextBox1.SelBold = True
End Sub

Private Sub cb_Click()
RichTextBox1.SelColor = vbblue
End Sub

Private Sub cl_Click()
RichTextBox1.SelColor = vbred
End Sub

Private Sub copy_Click()
'Clipboard.SetText "richtextbox1.seltext", 1
```

```
'MsgBox Clipboard.GetText
Clipboard.SetText RichTextBox1.SelText, 1
RichTextBox1.SelText = Clipboard.GetText
MsgBox Clipboard.GetText
End Sub

Private Sub eighteen_Click()
RichTextBox1.SelFontSize = 18
End Sub

Private Sub exit_Click()
End
End Sub

Private Sub fcg_Click()
RichTextBox1.SelColor = vbgreen
End Sub

Private Sub fourteen_Click()
RichTextBox1.SelFontSize = 14
End Sub

Private Sub helpp_Click()
ans = MsgBox("visual basic sample notepad .....!", vbYes + vbinforamtion, "Help")
If ans = vbYes Then
Unload Me
End If
End Sub

Private Sub italic_Click()
RichTextBox1.SelItalic = True
End Sub

Private Sub MC_Click()
RichTextBox1.SelFontName = "Monotype Corsiva"
End Sub

Private Sub new_Click()
RichTextBox1 = ""
End Sub

Private Sub open_Click()
RichTextBox1.LoadFile ("C:\Notepad\Document.rtf")
End Sub

Private Sub paste_Click()
RichTextBox1.SelText = Clipboard.GetText
```

```
End Sub

Private Sub save_Click()
RichTextBox1.SaveFile ("C:\Notepad\Document.rtf")
End Sub

Private Sub sixteen_Click()
RichTextBox1.SelFontSize = 16
End Sub

Private Sub Th_Click()
RichTextBox1.SelFontName = "Tahoma"
End Sub

Private Sub tn_Click()
RichTextBox1.SelFontName = "Times New Roman"
End Sub

Private Sub twele_Click()
RichTextBox1.SelFontSize = 12
End Sub

Private Sub underline_Click()
RichTextBox1.SelUnderline = True
End Sub

Private Sub vbblue_Click()
RichTextBox1.SelColor = vbblue
End Sub

Private Sub vbgreen_Click()
RichTextBox1.SelColor = vbgreen
End Sub

Private Sub vbred_Click()
RichTextBox1.SelColor = vbred
End Sub
```

Output:

File Menu:

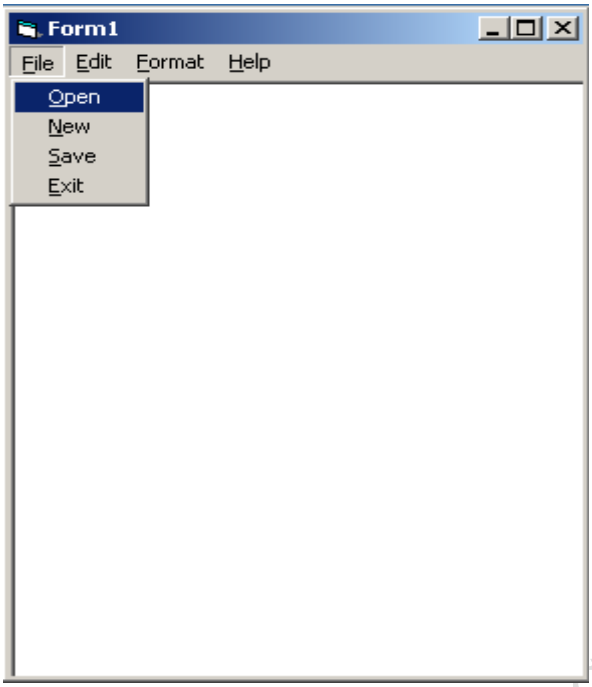


Fig.1. File Menu

Edit Menu

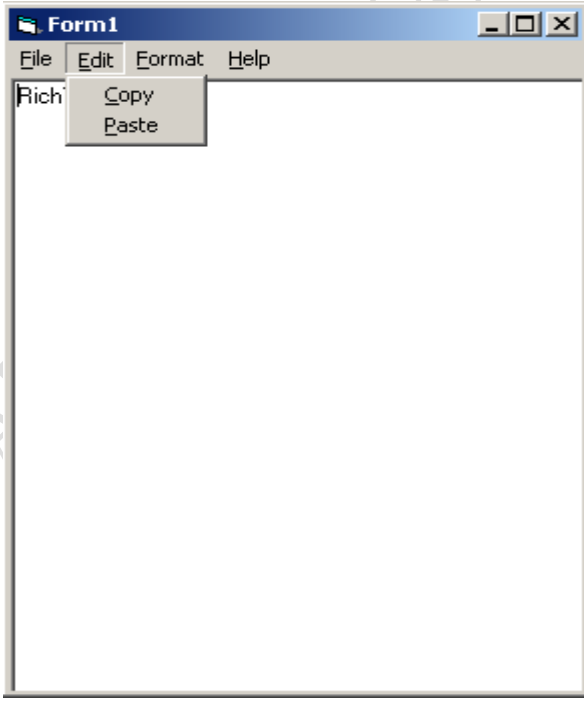


Fig.2. Edit Menu

Format Menu:

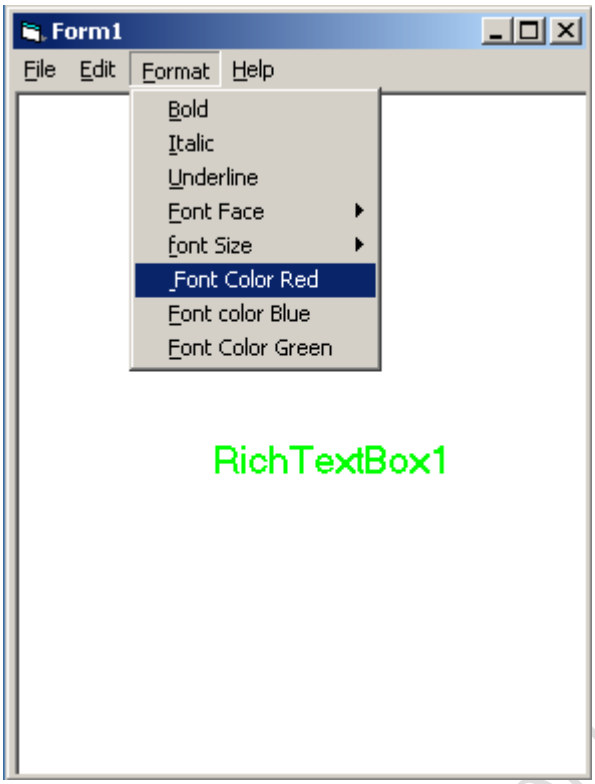


Fig.3. Format Menu

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

EX: NO: 10 REPORT DESIGN

AIM

To design a report design using Visual Basic.

PROCEDURE

STEP 1: Start

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.

STEP 4: Save the forms and project.

STEP 5: Execute the form and generate report

STEP 6: Stop

EXECUTION

Code for progress bar:

```
Private Sub Form_KeyPress(KeyAscii As Integer)
   Unload Me
End Sub
Private Sub Frame1_Click()
Unload Me
frmLogin.Show
End Sub
Private Sub Timer1_Timer()
On Error Resume Next
ProgressBar1.Value = ProgressBar1.Value + 1
If ProgressBar1.Value = 100 Then
```

```
login.Show
Unload Me
End If
End Sub
```

```
Private Sub Timer2_Timer()
On Error Resume Next
ProgressBar2.Value = ProgressBar2.Value + 1
If ProgressBar2.Value = 100 Then
MsgBox ("welcome")
login.Show
Unload Me
End If
End Sub
```

Code for login form:

```
Private Sub Command1_Click()
If (LCase(Text1.Text)) = "nagraaj" And (LCase(Text2.Text)) = "nagraaj" Then
   Unload Me
   Stock.Show
Else
   MsgBox "Please Enter Correct Username and Password"
End If
```

```
End Sub


Private Sub Command2_Click()
End
End Sub




Private Sub Text2_KeyPress(KeyAscii As Integer)
If (LCase(Text1.Text)) = "nagraaj" And (LCase(Text2.Text)) = "nagraaj" Then
    frmDataEnv.Show
    Unload Me
End If
End Sub
```

Stock Form:

```
Private Sub cmadd_Click(Index As Integer)
Adodc1.Recordset.AddNew
a = InputBox("ENTER THE PRODUCT CODE")
Text1.Text = a
B = InputBox("ENTER THE PRODUCT NAME")
```

```
Text2.Text = B

C = InputBox("ENTER THE MAKE")

Text3.Text = C

D = InputBox("ENTER SUPPLIER")

Text4.Text = D

e = InputBox("ENTER THE QUANTITY")

Text5.Text = e

F = InputBox("ENTER THE PURCHASE DATE")

Text6.Text = F

G = InputBox("ENTER THE PRICE")

Text7.Text = G

H = InputBox("ENTER THE VAT %")

Text8.Text = H

Text8.Text = Val(Text7.Text) / 14

Text9.SetFocus

Text9.Text = Val(Text7.Text) + Val(Text8.Text)

'Adodc1.Recordset.Save

'MsgBox ("UPDATED")

End Sub


Private Sub cmddelete_Click(Index As Integer)

Dim s As String

a = InputBox("Enter The product name")

a = Trim(a)
```

```
s = "product_TNAME='" & a "' "

Adodc1.Recordset.Delete


MsgBox ("deleted")

Text1.Text = ""

Text2.Text = ""

Text3.Text = ""

Text4.Text = ""

Text5.Text = ""

Text6.Text = ""

Text7.Text = ""

Text8.Text = ""

Text9.Text = ""

End Sub


Private Sub cmdmovl_Click(Index As Integer)

Adodc1.Recordset.MoveNext

End Sub


Private Sub cmdmovn_Click(Index As Integer)

'dodc1.Recordset.MoveNext

If (Adodc1.Recordset.EOF) = True Then

Adodc1.Recordset.MoveNext

Else
```

```
Adodc1.Recordset.MovePrevious
MsgBox ("THIS IS YOUR LAST RECORD")
End If

End Sub

Private Sub cmdmovp_Click(Index As Integer)
'dodc1.Recordset.MovePrevious
If Adodc1.Recordset.BOF = True Then
   Adodc1.Recordset.MoveFirst
Else '
Adodc1.Recordset.MoveNext
MsgBox ("THISIS YOUR FIRST RECORD")
End If
End Sub

Private Sub cmdsearch_Click(Index As Integer)

Dim a As String
a = InputBox("Enter Item Code")
s = "Item_code = '" + a + "'"
Adodc1.Recordset.MoveFirst
Adodc1.Recordset.Find s
If Adodc1.Recordset.EOF Then
```

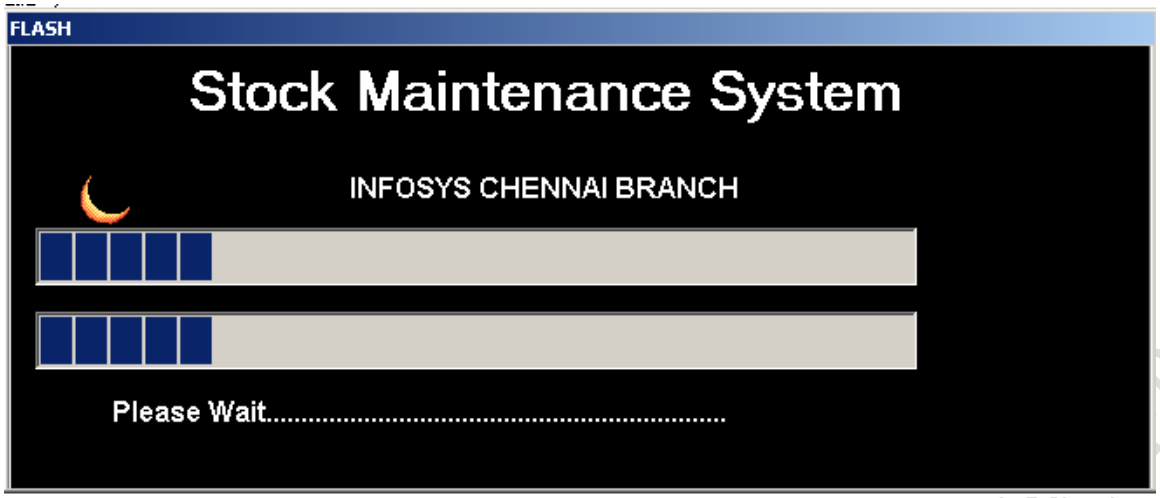
```
MsgBox ("INVALID RECORD")

End If

End Sub

Private Sub cmdupadte_Click(Index As Integer)

Adodc1.Recordset.Update

MsgBox ("UPDATED")

Text1.Text = ""

Text2.Text = ""

Text3.Text = ""

Text4.Text = ""

Text5.Text = ""

Text6.Text = ""

Text7.Text = ""

Text8.Text = ""

Text9.Text = ""

End Sub

Private Sub commov_Click(Index As Integer)

Adodc1.Recordset.MoveFirst

End Sub

Private Sub Command1_Click()

Dim a As String

a = InputBox("Enter Item Code")

s = "Item_code = '" + a + "'"

Adodc1.Recordset.MoveFirst
```

```
Adodc1.Recordset.Find s

'Adodc1.Recordset ("select * from t1 where [ITEM_CODE] = " & Text1.Text(0) & "")

DataReport1.Show

If Adodc1.Recordset.EOF Then

MsgBox ("INVALID RECORD")

End If

End Sub

Private Sub Command2_Click()

Adodc1.Recordset.Update

MsgBox ("UPDATED SUCCESSFULY")

End Sub

Private Sub Command3_Click()

Dim s As String

a = InputBox("Enter The student name")

a = Trim(a)

s = "STUDENTNAME='" & a "' "

Adodc1.Recordset.Delete

MsgBox ("deleted")

Text1.Text = ""

Text2.Text = ""

Text3.Text = ""

Text4.Text = ""

Text5.Text = ""

Text6.Text = ""
```

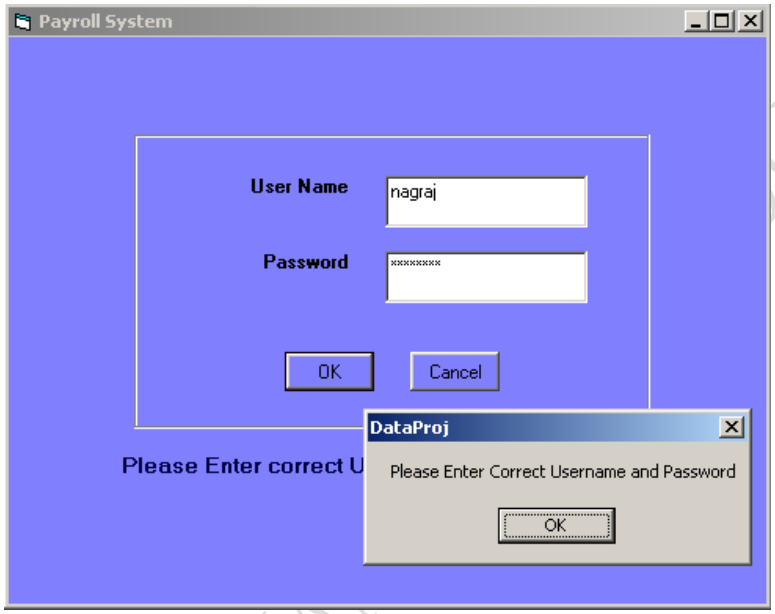
```
Text7.Text = ""

Text8.Text = ""

Text9.Text = ""

End Sub

Private Sub Command5_Click()

DataReport1.Show

End Sub

Private Sub EXIT_Click()

End

End Sub

Private Sub Image2_Click()

'Adodc1.Recordset (" * from t1 where [ITEM_CODE] = " & Text1.Text(0) & "")

DataReport1.Show

End Sub

Private Sub MSHFlexGrid1_Click()

'Adodc2.Refresh

End Sub

Private Sub VIEW_Click()

DataReport1.Show

End Sub
```

Output:

Progress Bar

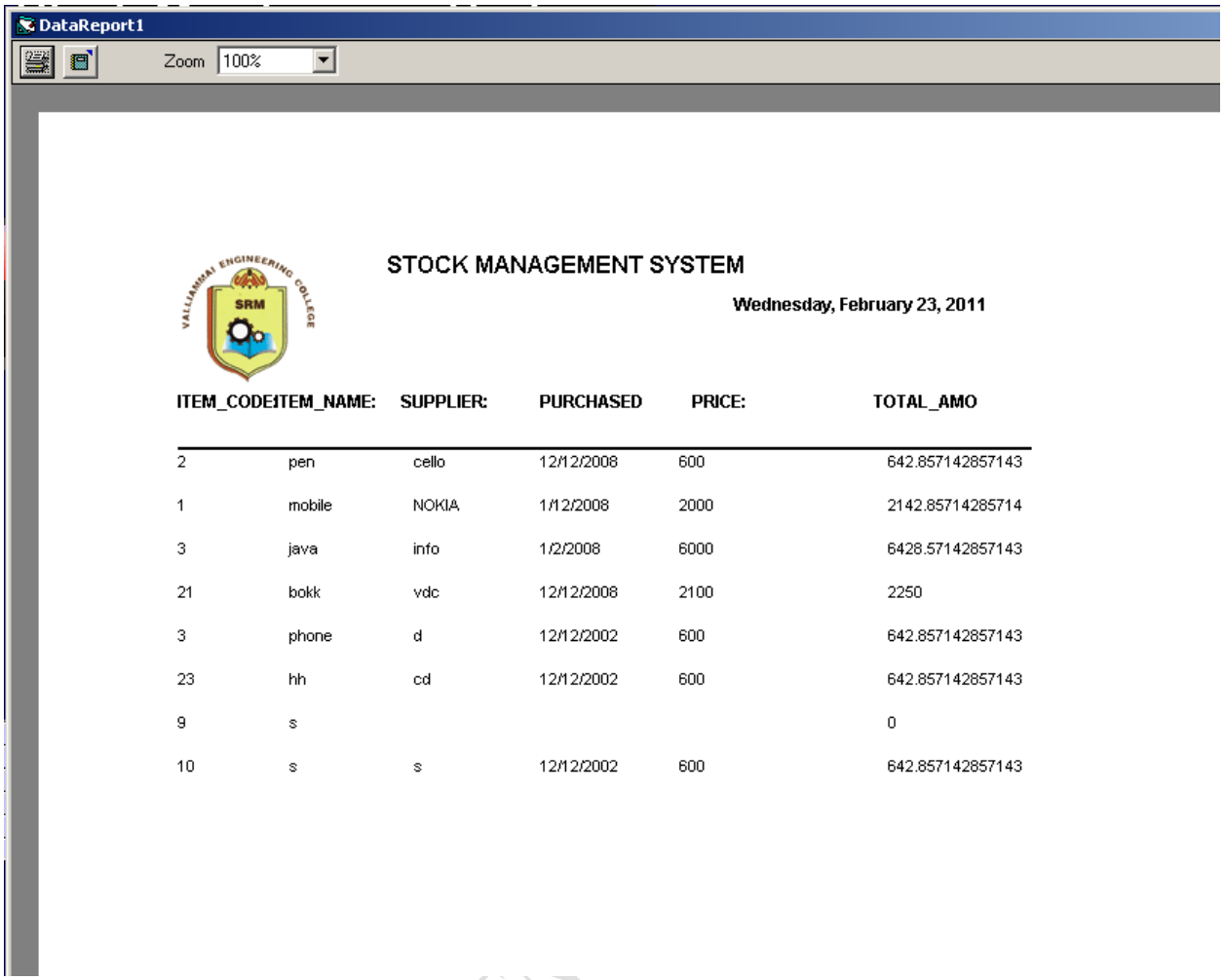


Login



Stock Form:

Report Design:



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

Ex. No. 11. MINI PROJECT

PASSPORT AUTOMATION SYSTEM

A PROJECT MINI REPORT

Submitted by

NAME:
REG.NO:

COMPUTER SCIENCE AND ENGINEERING
VALLIAMMAI ENGINEERING COLLEGE,
KATTANKULATHUR

ANNA UNIVERSITY: CHENNAI 600 025
MAY 2011

VALLIAMMAI ENGINEERING COLLEGE,
KATTANKULATHUR
ANNA UNIVERSITY: CHENNAI 600 025

ABSTRACT

Earlier the issue of passport was done manually which eventually became hectic as the number of applications saw an exponential rise. Also it took several months for applicants to obtain the passport. Only after this, the concept of computerization in processing of passport applications came into consideration. This software Passport Automation System is used in the effective dispatch of passport. It adopts a comprehensive approach to minimize the manual work and schedule resources and time in a cogent manner. It adopts different strategies to tackle the rise in the number of passport applications. The main objective of the system is to ensure speedy dispatch of passport to the applicants and to minimize the manual work. Technical and practical problems encountered during the development are discussed in this paper, and a thorough performance evaluation of the developed prototype is also presented.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

PAS	Passport Automation System
VB	Visual Basic
SDLC	Software Development Life Cycle
ER	Entity - Relationship

CHAPTER 1

INTRODUCTION

1.1 ABOUT PASSPORT AUTOMATION SYSTEM

Passport Automation System is an interface between the Applicant and the Authority responsible for the issue of passport. If the entire process of 'Issue of Passport' is done in a manual manner then it would take several months for the passport to reach the applicant. Considering the fact that the number of applicants for passport is increasing every year, an Automated System becomes essential to meet the demand. So this system uses several programming and database techniques to elucidate the work involved in this process. As this is a matter of National Security, the system has been carefully verified and validated in order to satisfy it. It aims at improving the efficiency in the issue of passport and reduces the complexities involved in it to the maximum possible extent

1.2 PROBLEM DEFINITION

- Passport Automation System is used in the effective dispatch of passport to all of the applicants.
- The core of the system is to get the online registration form(with details such as name, address etc.,) filled by the applicant whose testament is verified for its genuineness by the Passport Automation System with respect to the already existing information in the database. This forms the first and foremost step in the processing of passport application.

- After the first round of verification done by the system, the information is in turn forwarded to the administrator.
- The scanned documents are verified by the administrator whose decision is regarded as final.
- The administrator will be provided with an option to display the current status of application to the applicant, which they can view in their status checking interface.
- After all the necessary criteria have been met, the original information is added to the database and the passport is sent to the applicant.

1.3 EXISTING SYSTEM

- In the existing system the processing of passport applications is done manually.
- The applicant has to fill in a printed application form and is expected to submit it.
- The application submitted actually takes a long time to reach the Administrator's desk due to the long list of applications already pending.
- The verification documents have to be submitted manually by the applicant after waiting in the queue for a long time.
- The Administrator will have to verify the documents and have to notify the police for personal verification which again is a time consuming process.

- The number of applications that are verified and dispatched in a single day is very less in the existing system.
- It becomes very difficult to tackle the ever increasing applications for passport, and as a result of which the applicants face trouble.

1.4 PROPOSED SYSTEM

- In Passport Automation System, the processing of applications is done with the help of computer.
- The applicant need not go all the way to passport office to submit his printed form; instead he can fill the online application form from the comfort of his home.
- The Administrator is notified instantly about the submitted application.
- The verification documents can be scanned along with electronic signatures whose validity is thoroughly checked and taken into consideration. The applicant need not stand in the queue for a long time.
- The Administrator after verifying documents can easily proceed with other formalities to dispatch the passport to the applicant.
- The number of applications processed and dispatched in a single day is very high when compared to the existing system.
- It becomes very easy to manage the increase in applications.

.

1.5 ENVIRONMENT SPECIFICATION

1.5.1 HARDWARE SPECIFICATION

1. Hard Disk: 40 GB and above.
2. RAM : 512 MB and above.
3. Processor : Pentium 4 and above, or any other equivalent processor.

1.5.2 SOFTWARE SPECIFICATION

1. Operating System : Windows XP and above.
2. Documentation tool : Microsoft Word 2003.

1.6 TECHNOLOGIES USED

1. Windows XP.
2. Visual Basic 6.0
3. Oracle for backend.

1.7 SOFTWARE ENGINEERING PARADIGM USED

This Passport Automation System uses a **waterfall model**. The simplest, oldest and most widely used process model for software designing is the waterfall model. The essence of this software paradigm is that the process of software designing consists of linear set of distinct phases.

The various stages in this model are depicted in the **Fig 1.1**,

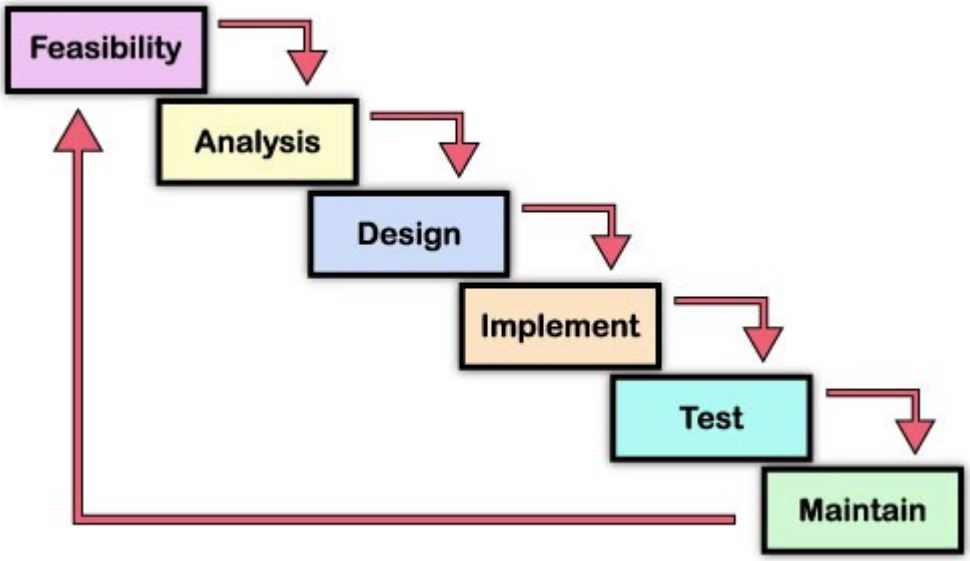


Fig 1.1: Waterfall model in PAS

The various stages involved in the development of our Passport Automation System are given below.

Stage 1: Feasibility Study

Here the most feasible and desirable system for designing of PAS was identified. Five types of feasibility were addressed in this study.

1. Technical feasibility
2. Economic Feasibility
3. Motivational Feasibility
4. Schedule Feasibility
5. Operational Feasibility

Stage 2: Requirement Analysis and Specification

The exact requirements of the PAS were identified and were documented properly. The main was to document all functions, performance and interfacing requirements for the software designing

and management. The requirements describe "what" of a system. This phase produced a large document containing a description of what the system will do without describing how it will be done. That document is considered to be the software requirement specification (SRS) document.

Stage 3: Design

Here the requirement specification produced in the requirement analysis phase was transformed into a structure that is suitable for implementation in Visual Basic. The overall software architecture was defined; and the product design and detailed design work was performed. This was well documented and is called Software Design Description (SDD).

Stage 4: Implementation

In this phase the designs were translated into code. Computer programs are written using Visual Basic. Programming tools like Compilers, Interpreters, and Debuggers were used to generate the code.

Stage 5: Testing

Stage 5.1: Unit Testing

Each of these program modules is unit tested. The purpose of unit testing was to determine the correct working of individual modules.

Stage 5.2: Integration and System Testing:

During this phase the different program modules were integrated in a planned way and then tested as a complete system to ensure that the designed system functions according to its

requirements as specified in the SRS document. After testing, the software was delivered to the customer.

Stage 6: Software Maintenance

This is the last phase of software designing which includes a broad set of activities such as error correction, enhancement of capabilities, deletion of obsolete capabilities and optimization. This is the longest SDLC phase.

1.8 SYSTEM PLANNING

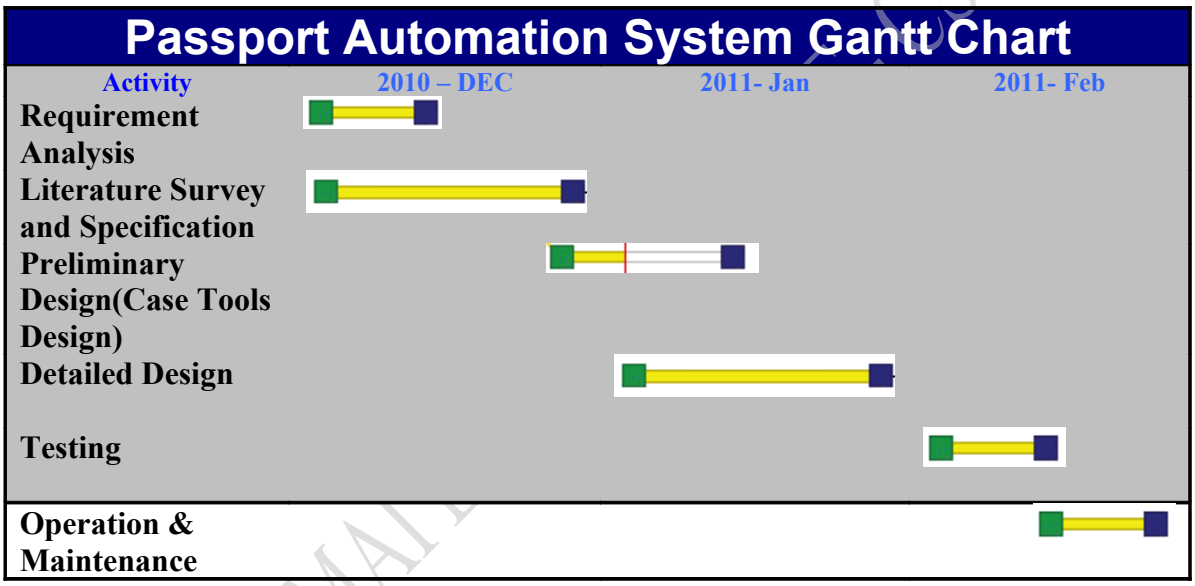


Fig 1.2 Gantt chart

CHAPTER 2

SYSTEM DESIGN

2.1 INPUT DESIGN

The input from the user is obtained through various forms that contain the details to be filled by the applicant.

- New Registration
- Check Status
- Authentication
- Admin Panel

2.1.1 INPUT DESCRIPTION

New Registration

This module is used to develop a GUI design for an application form that is to be filled by the applicant. The module requires the applicant to **fill in all the details and upload his photo.** Finally he can submit the application which is in turn added to the database. The module also provides the user with a unique applicant id.

Checking Status

This module allows the user to know about the status of his submitted application which is either 'Waiting' or 'Dispatched'. The applicant can know his status by typing his **applicant id**.

Authentication

This is a module that takes care of authentication to the Admin Panel; it verifies if the person who is trying to enter into the administrator panel is authorized personnel or not. The administrator will have to

provide his **username and password**, only if it matches he can gain access to the Admin Panel.

Admin Panel

This module takes care of the interface that is displayed after a successful authentication. It provides the administrator the details of applications which are yet to be processed and a list of all dispatched details. The administrator can enter any **applicant id** to process that application.

2.2 DATABASE DESIGN

The Passport Automation System uses Oracle, relational database management software as a backend for the system. The database for this consists of a single table to store all the details of the application. The table is named as "**REGISTRATIONS**". The Structure of the table is given below:

Name	Null?	Type
ID	NOT NULL	NUMBER(5)
NAME		CHAR(50)
AGE		CHAR(2)
DOB		DATE
POB		CHAR(50)
FATHER		CHAR(50)
MOTHER		CHAR(50)
GENDER		CHAR(7)
ADDRESS		CHAR(150)
STATUS		CHAR(20)
PICTURE		VARCHAR2(250)

Table 2.1 Registrations Table

2.3 ER DIAGRAM

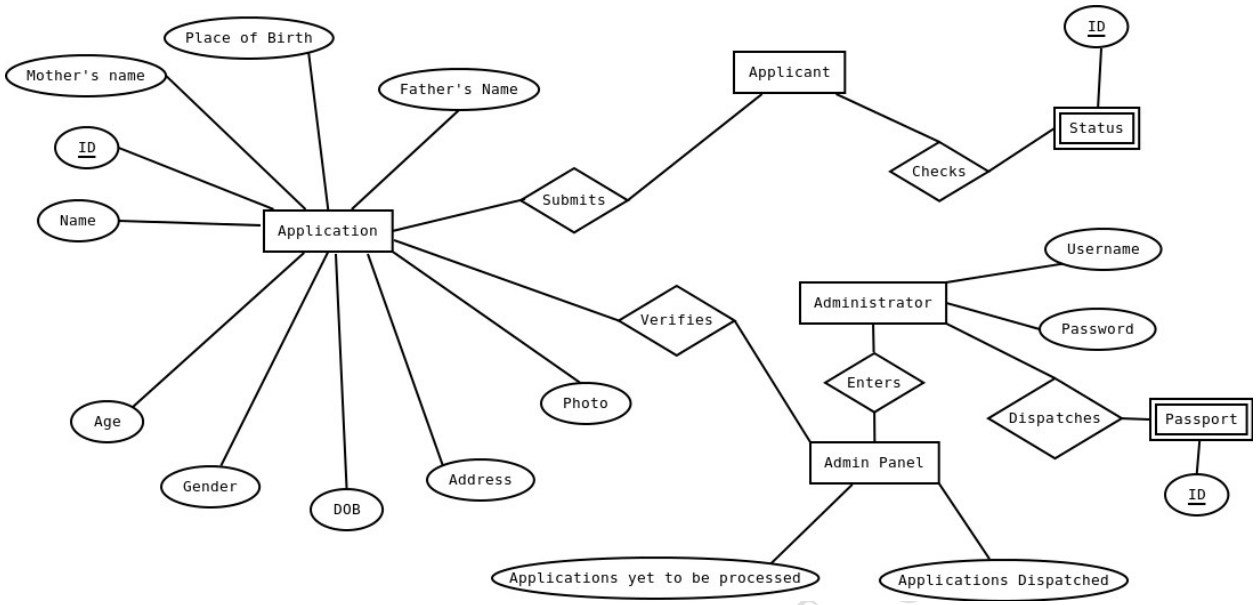


Fig 2.1 ER Diagram for PAS

2.4 USE CASE DIAGRAM

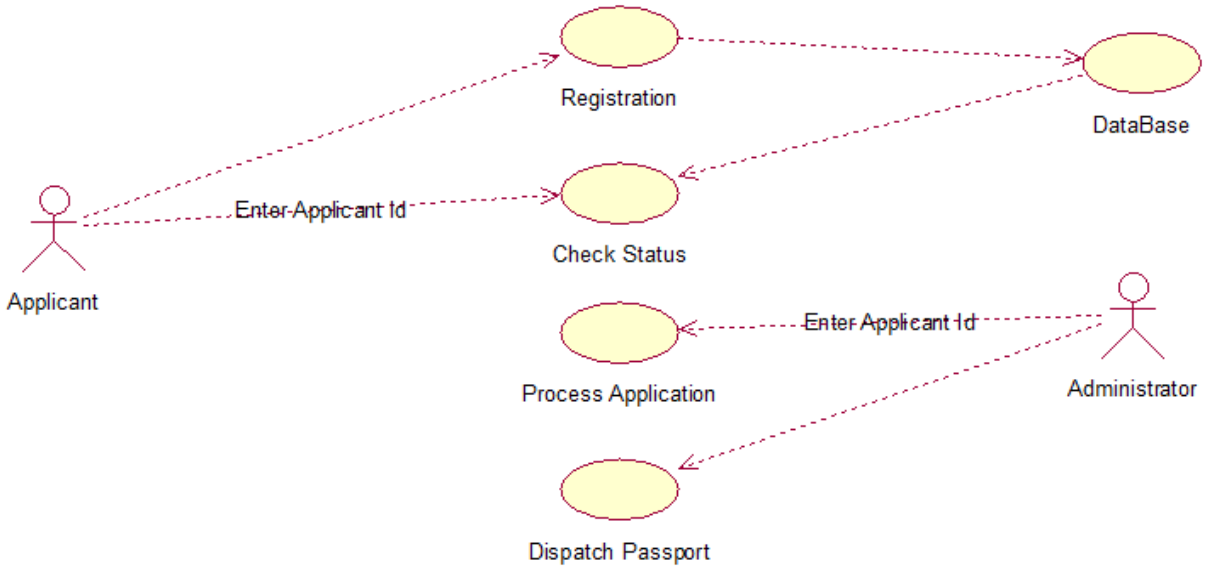


Fig 2.2 Use case diagram for PAS

DESCRIPTION

The use case model consist of the following actors,

- Applicant
- Administrator

The various actions that take place in this system are,

- Registration
- Check status
- Process application
- Dispatch passport
- Database

The applicant can perform various activities such as register and check the status of the process. The applicant has to fill in certain forms during the process of registration and he has to provide hid id and certain details in order to check his passport status. The applicant is involved in processing the applications from the applicants and the dispatch of passport to eligible candidates.

2.5 CLASS DIAGRAM

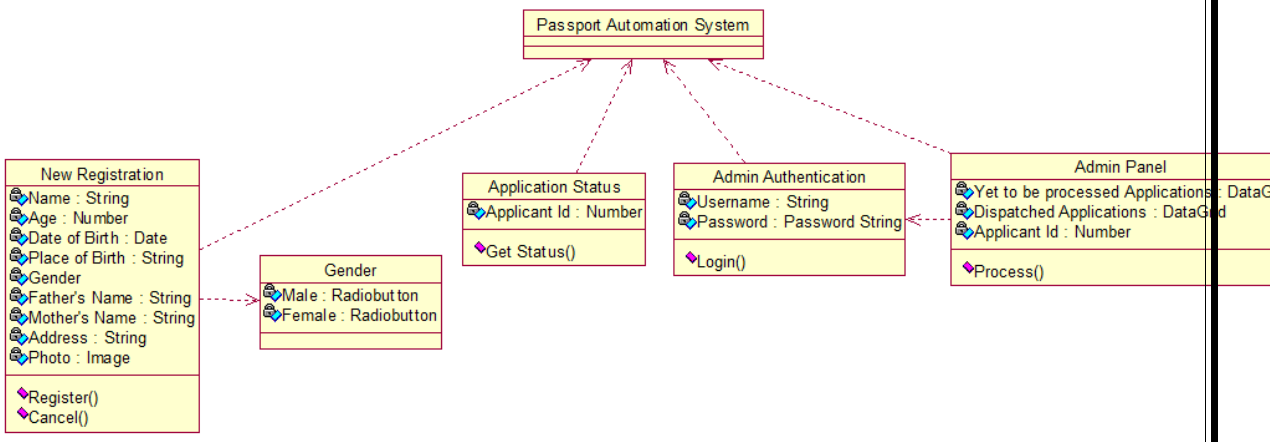


Fig 2.3 Class Diagram for PAS

DESCRIPTION

The UML Class diagram shows the various actions that are to be performed in a system. The various actions are,

- Registration
- Application status
- Admin authentication
- Admin panel

The class diagram shows the various attributes and the operations related to those activities.

2.6 ACTIVITY DIAGRAM

Activity Diagram for 'New Registration'

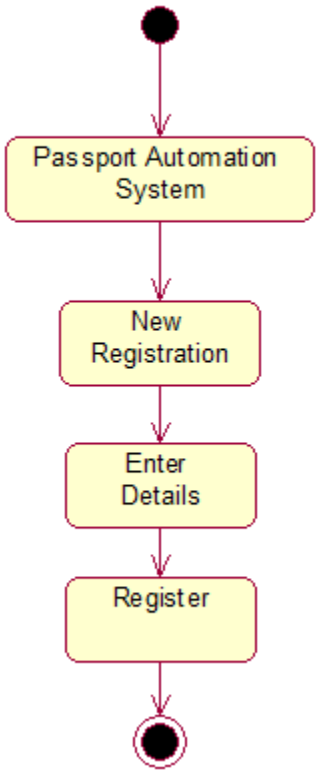


Fig 2.4 Activity Diagram for New Registration

Description

The various activities involved in the registration process involve filling the details which are required by the administrator. Once all the details are correctly filled by the applicant the applicant clicks the register button to register.

Activity diagram for 'Checking status'

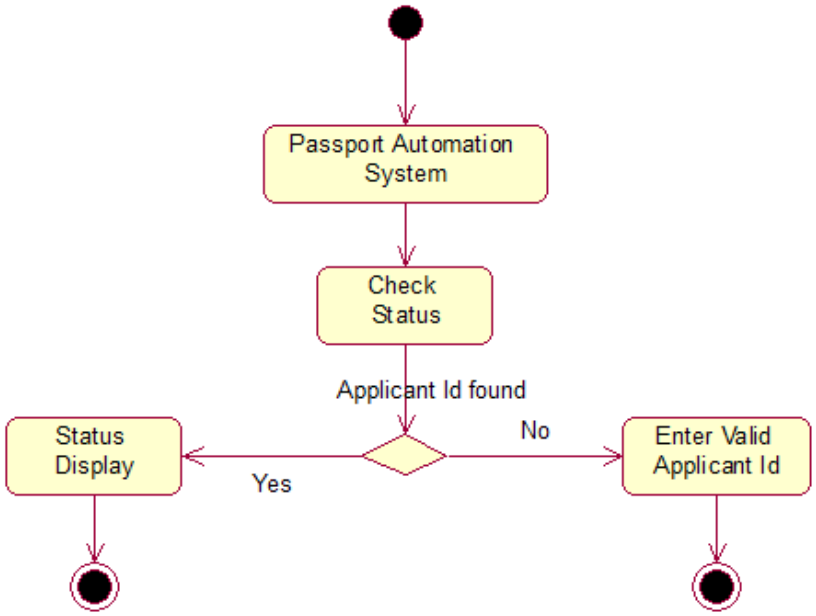


Fig 2.5 Activity Diagram for Checking Status

Description

The activity diagram for checking status involves the following operations,

- Check status
- Display status

For checking the status of the passport the applicant has to enter his id and then log into the system. The applicant enters his applicant id and then clicks the check status button. If the details for the specified id are present then it displays the information about the status of the

passport and if the id is not present then the system prompts the applicant to enter proper id.

Activity Diagram for 'Admin Panel'

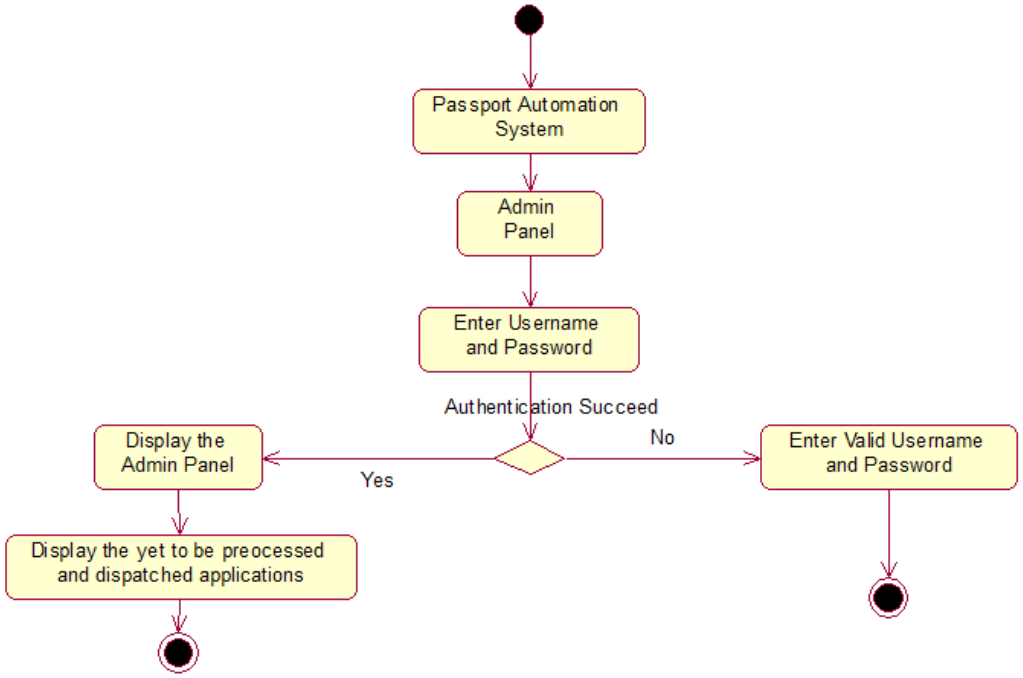


Fig 2.6 Activity Diagram for Admin Panel

Description

The administrator enters the id of the applicant whose application is yet to be processed and then dispatches it if he/she is eligible. The admin panel contains two lists, one contains the list of applications which are not processed and the other contains the details of the dispatched passport.

2.7 SEQUENCE DIAGRAM

Sequence diagram for 'New Registration'

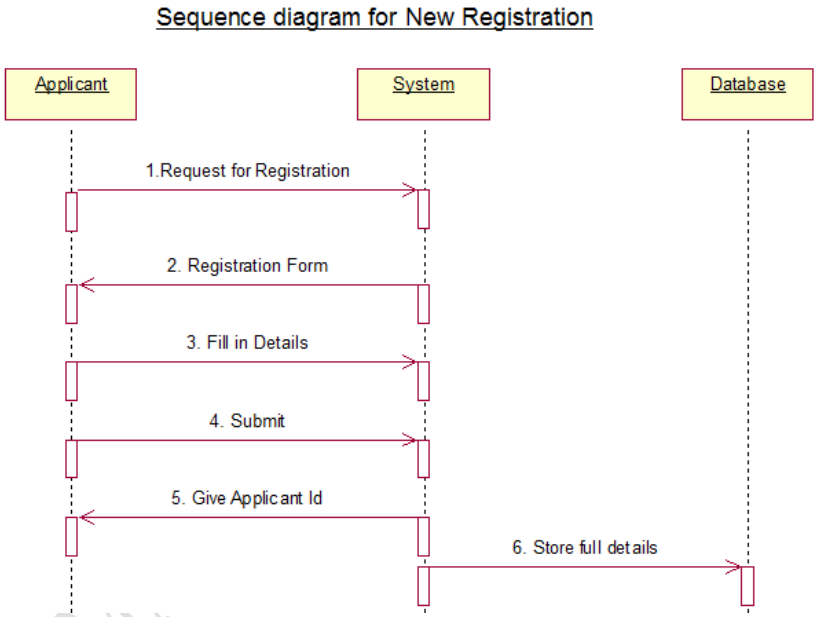


Fig 2.7 Sequence Diagram for New Registration

Description

The sequence diagram shows the sequence of activities that are performed in the process of registration. First the applicant requests the system for registration and system provides the application form in reply. The applicant fills the registration form and then submits it to the system. After the successful receipt of the application the system provides a unique applicant id to the applicant. Finally the details of the applicant are stored in the database by the system.

Sequence Diagram for 'Checking status'

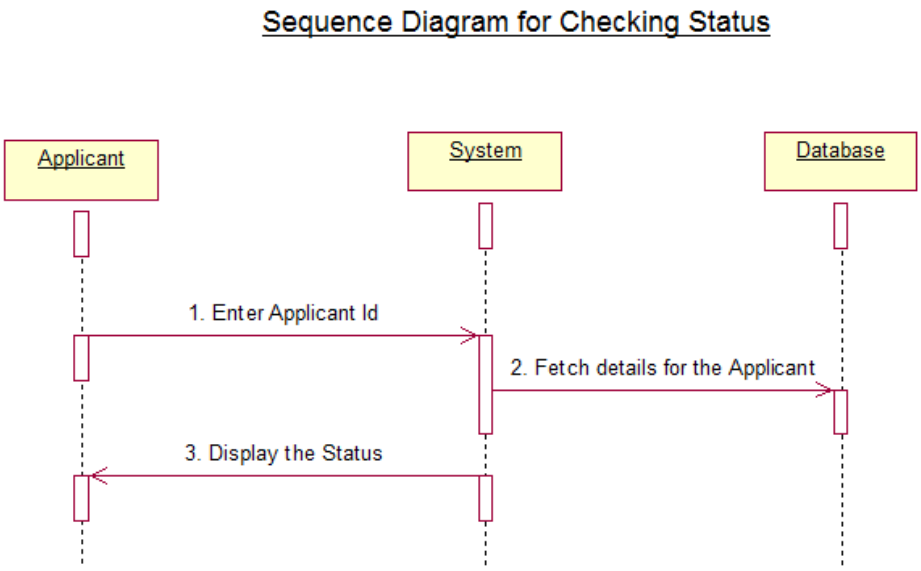


Fig 2.8 Sequence Diagram for Checking status

Description

The sequence diagram for status shows the sequence of actions that take place. First the applicant enters the applicant id. The system then fetches the details of the applicant from the database and displays the status information to the applicant.

Sequence Diagram for Admin Panel

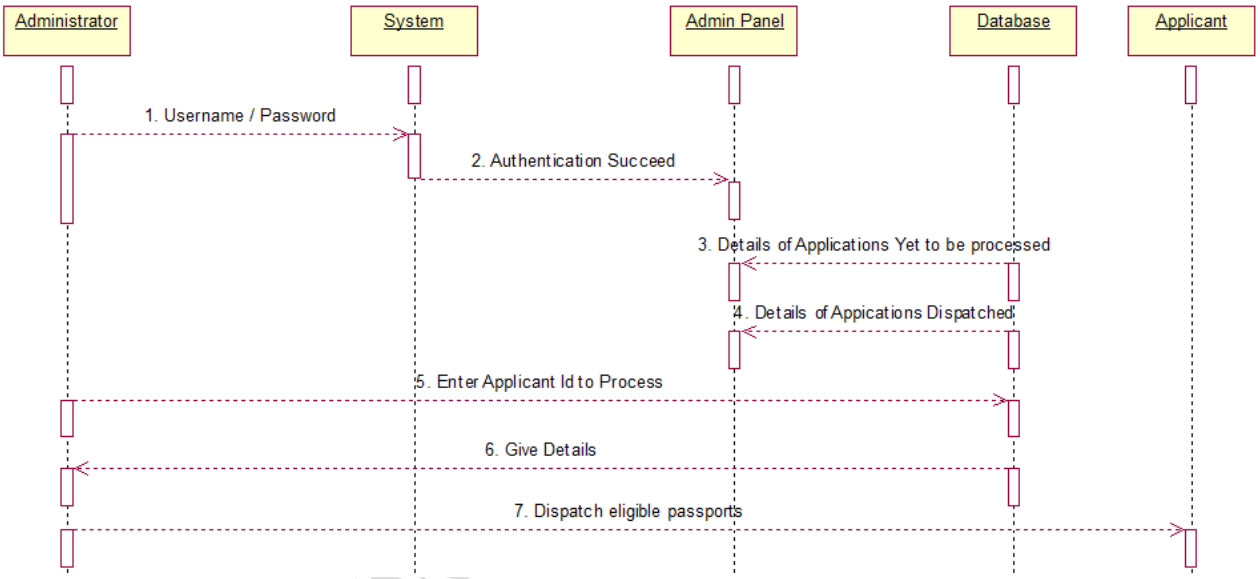


Fig 2.9 Sequence Diagram for Admin Panel

Description

The sequence diagram for the admin panel clearly visualizes the various actions that take place. The administrator enters the user name and password and submits it to the system. The authentication succeed is given to the admin panel. The details of the applications yet to process or the details of the application dispatched are fetched by the admin panel on the request from the administrator. The administrator enters the id to be processed and the database in turn returns the details. The administrator orders the database to dispatch eligible passports.

2.8 COLLABORATION DIAGRAM

Collaboration diagram for 'New registration'

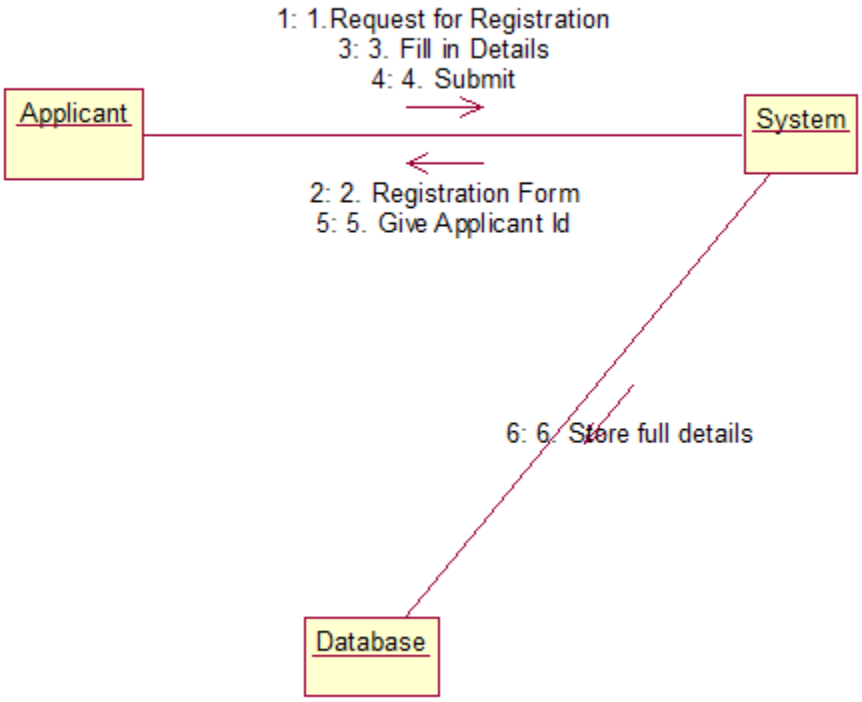


Fig 2.10 Collaboration diagram for new registration

Description

The collaboration diagram is a one which is formed from the sequence diagram. The collaboration diagram shows the sequence of activities in an collaborated matter. The various activities that appear

between the applicant and the system are shown in a collaborated manner

Collaboration diagram for 'Checking Status'

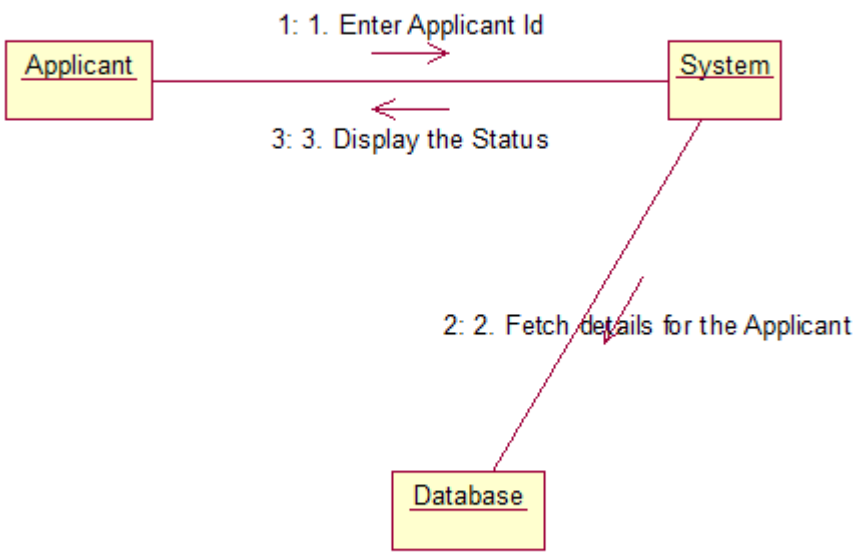


Fig 2.11 Collaboration diagram for Checking Status

Description

The various actions between the applicant, system and the database in the process of checking status are clearly depicted in the collaboration diagram. Enter applicant id and display status are the two activities that are done between the applicant and the system. Fetch details in

the only activity which takes place between the system and the database.

Collaboration diagram for 'Admin Panel'

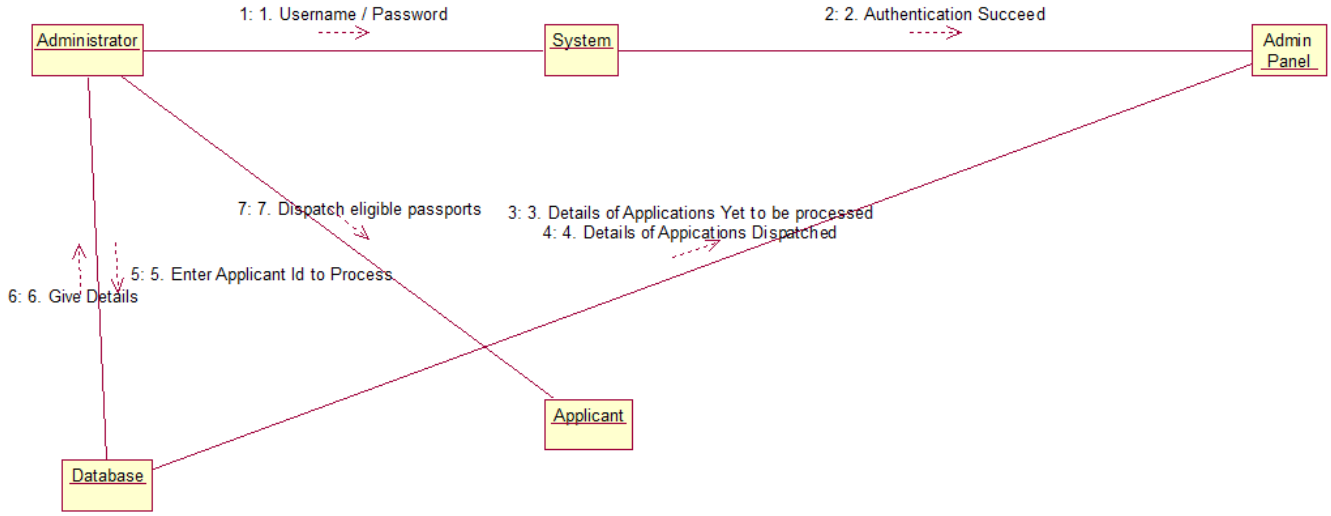


Fig 2.12 Collaboration diagram for Admin Panel

Description

The collaboration diagram for the admin panel makes it simple to understand the various activities that are taking place in the admin panel. We can easily understand the activities between each actor in the admin panel.

2.9 COMPONENT DIAGRAM

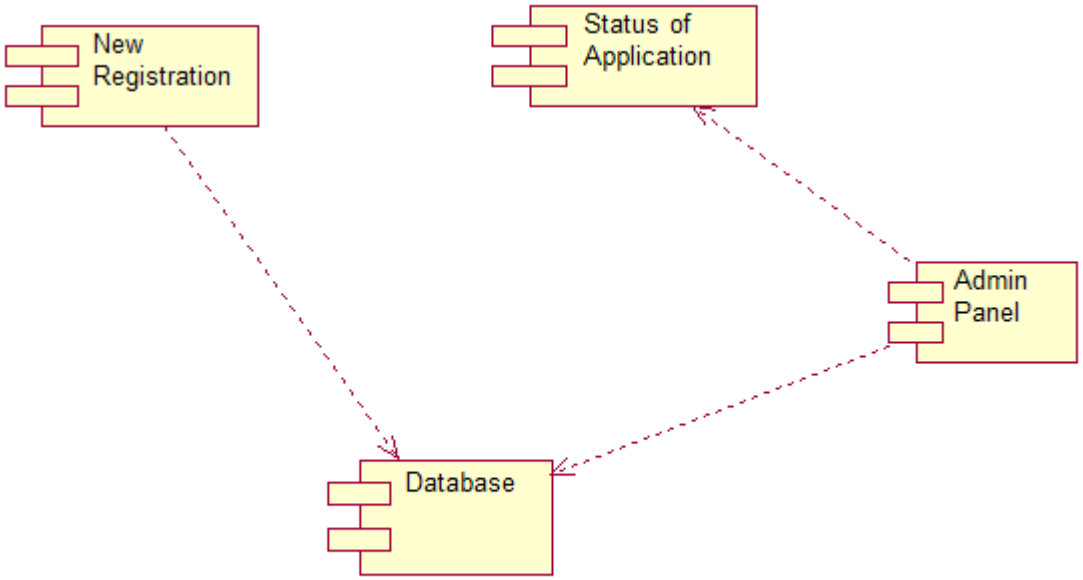


Fig 2.13 Component Diagram for PAS

A component diagram is a graph of the design's components connected by dependency relationships.

2.10 DEPLOYMENT DIAGRAM

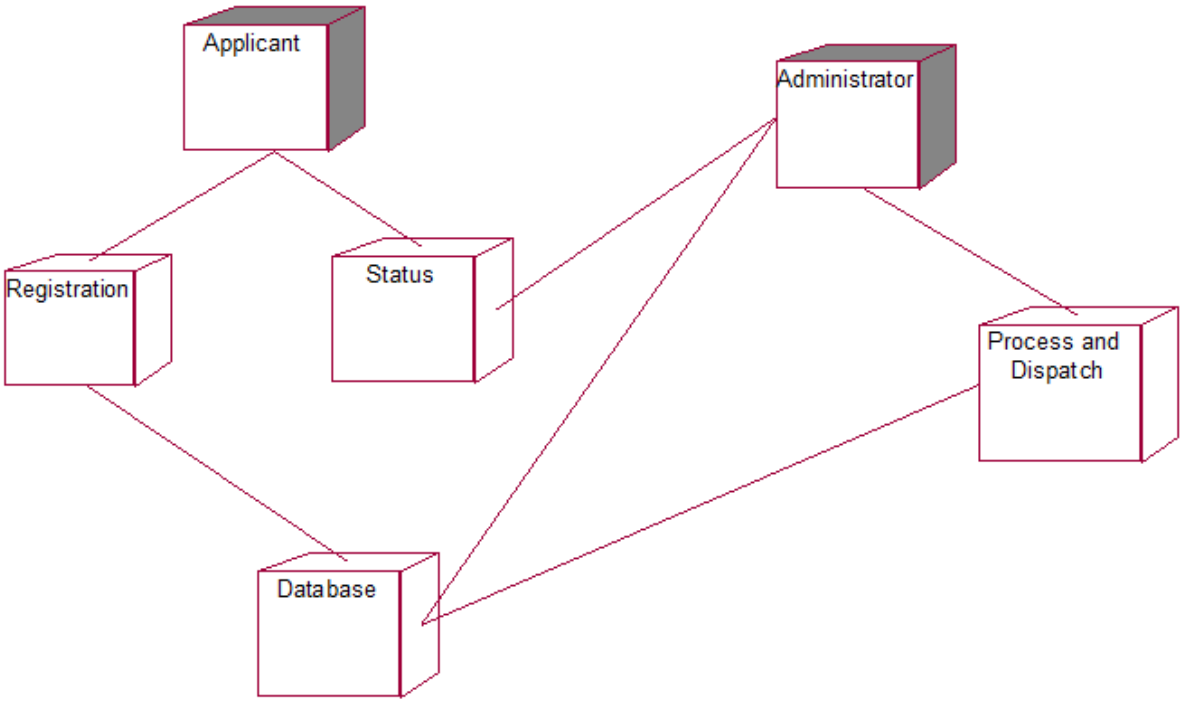


Fig 2.14 Deployment Diagram for PAS

Deployment diagram shoes the configuration of run-time processing elements and the software components, processors and the objects that live in them.

CHAPTER 3

SYSTEM DEVELOPMENT

3.1 SYSTEM ARCHITECTURE

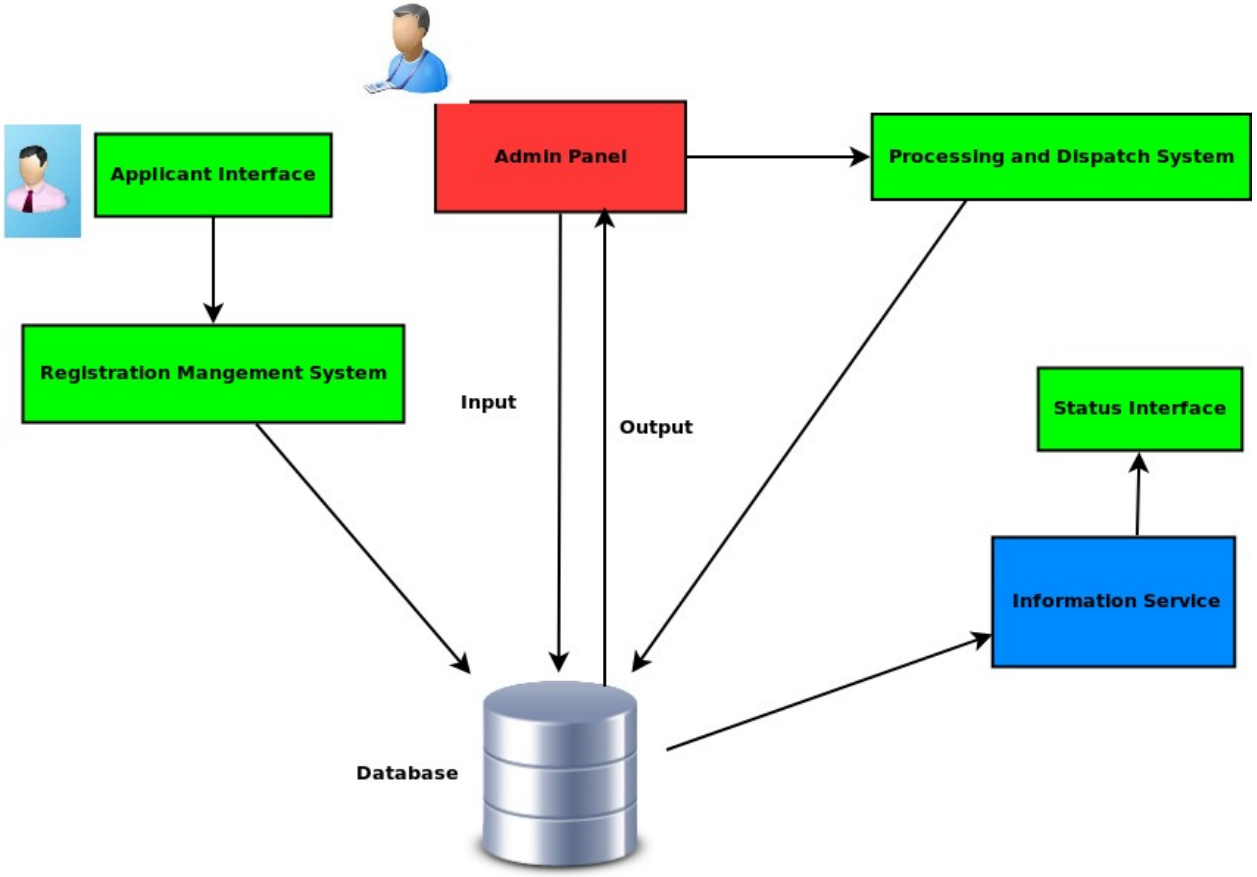


Fig 3.1 Passport Automation System architecture

This architecture gives the conceptual model that defines the structure, behavior, and views of the Passport Automation System. The four main components of the system architecture: processing

power, storage, connectivity, and user experience. The Applicant interface and status interface is responsible for user experience, database takes care of storage that are required in the passport automation system. The processing power of applications is vested on the Administrator who uses the Admin panel to connect various components of the system. The Information Service acts as a bridge between the database and the status interface.

CHAPTER 5

IMPLEMENTATION OF PASSPORT AUTOMATION SYSTEM

5.1 SCREENSHOTS

STARTUP SCREEN

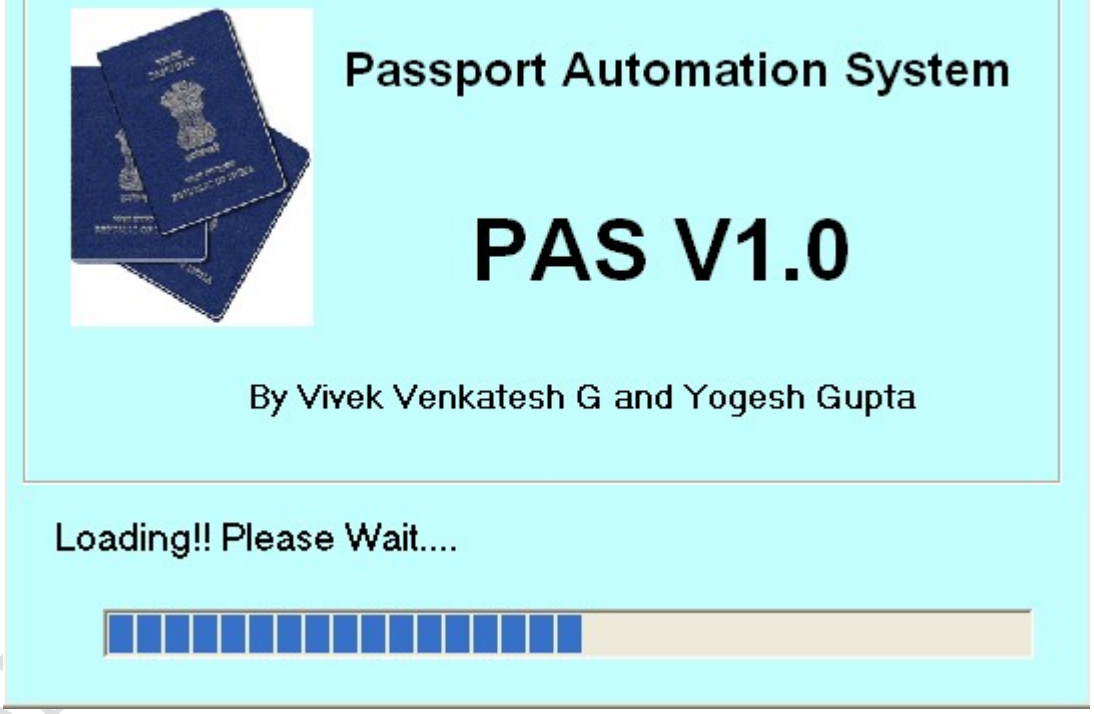


Fig 5.1 Startup Screen for PAS

MAIN MENU SCREEN

Fig 5.2 Main Menu

NEW REGISTRATION SCREEN

Fig 5.3 New Registration

CHECK STATUS SCREEN

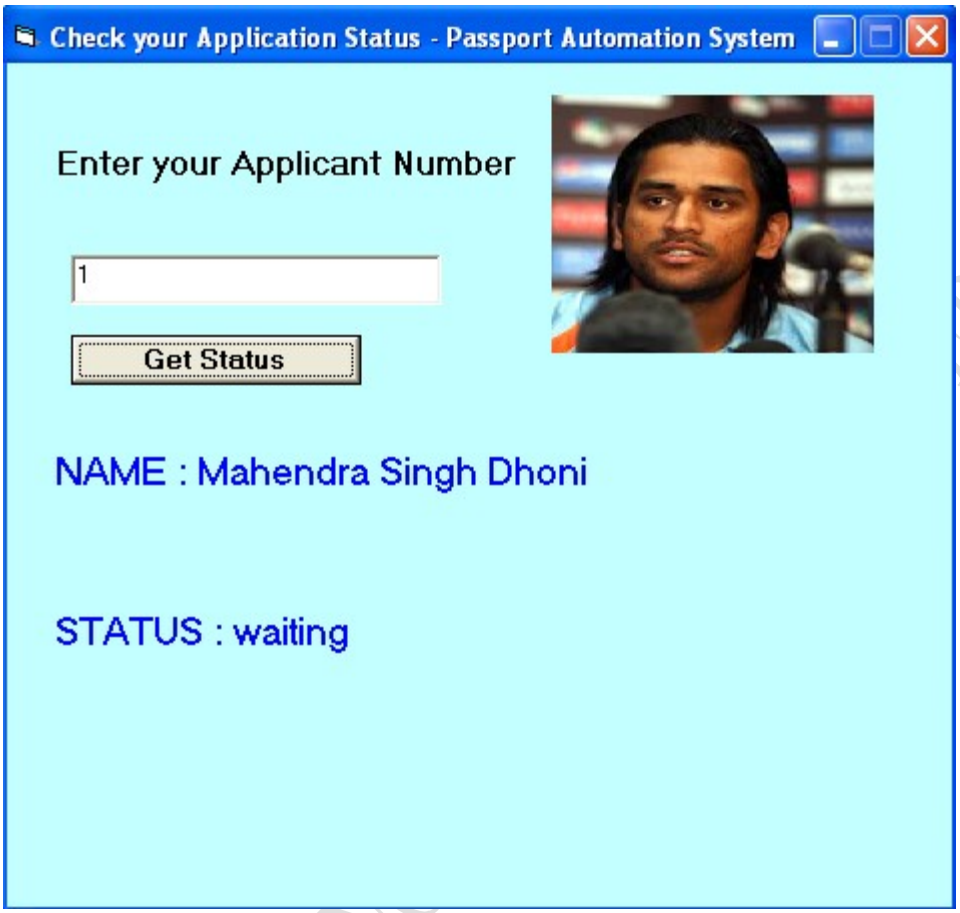


Fig 5.4 Check Status

AUTHENTICATION SCREEN

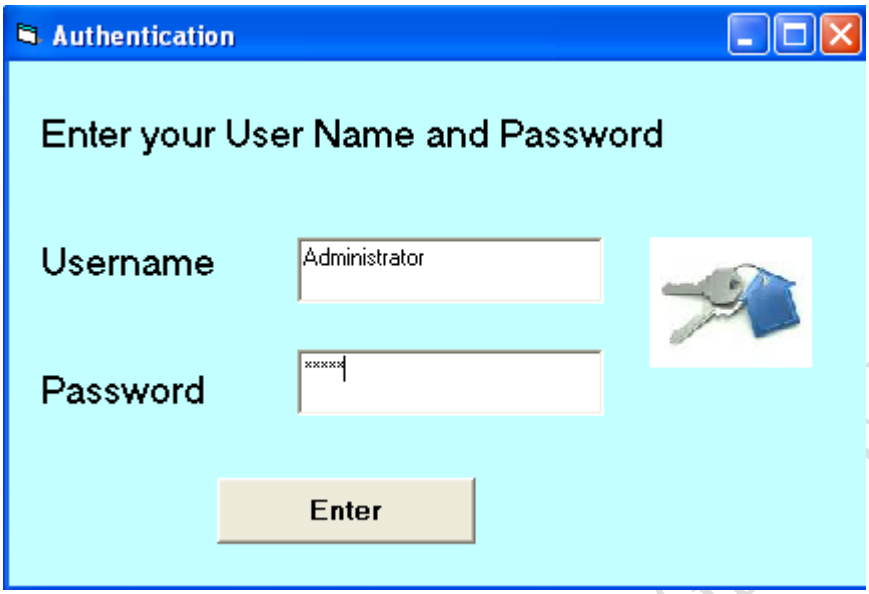


Fig 5.5 Authentication Screen

ADMIN PANEL SCREEN

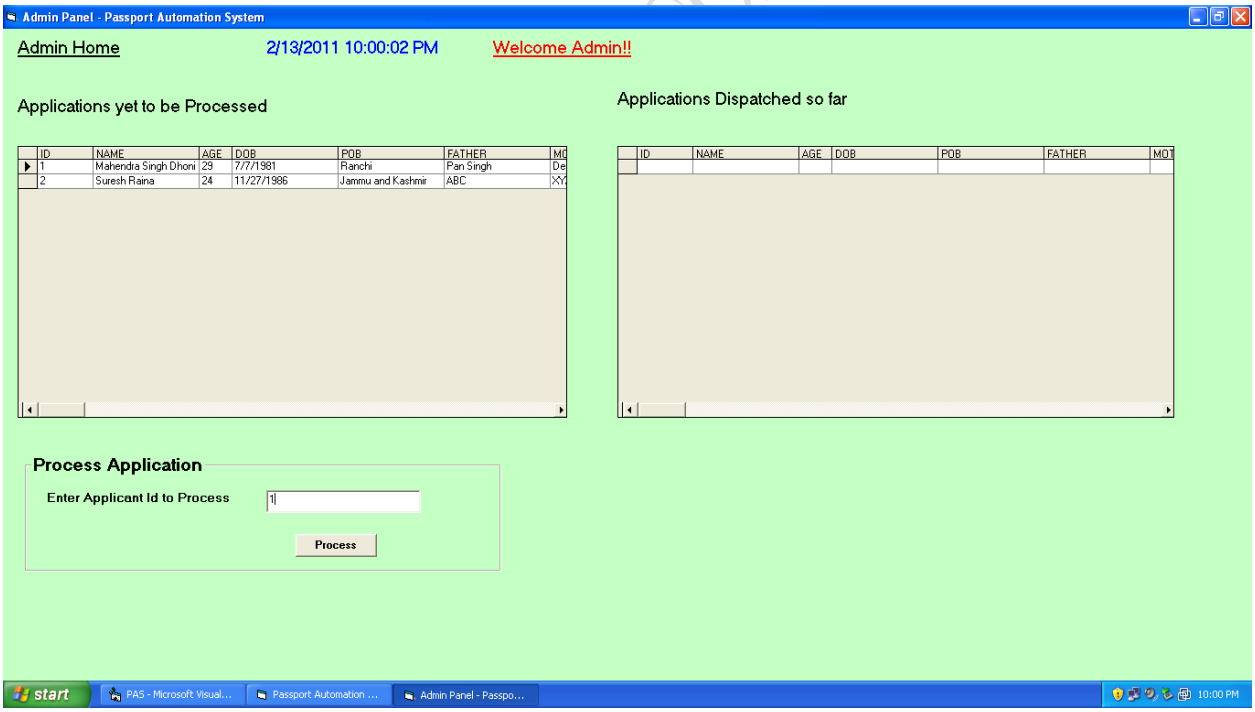


Fig 5.6 Admin Panel

PROCESS APPLICATION SCREEN

Fig 5.7 Process Application

5.2 CODING

5.2.1 Code for Opening Screen

```
Dim A As Integer
Option Explicit
Private Sub Form_KeyPress(KeyAscii As Integer)
        Unload Me
End Sub
Private Sub Frame1_Click()
  Unload Me
End Sub
Private Sub Timer1_Timer()
```

```
ProgressBar1.Value = ProgressBar1.Value + 1
If ProgressBar1.Value = 10 Then
Timer1.Enabled = False
ProgressBar1.Value = 0
Unload Me
Main.Show
End If
End Sub
```

5.2.2 Code for Main Page

Here is the user is provided with three options

1. New Registrations.
2. Check Status
3. Admin Panel

According to their choice, the related page is displayed

```
Private Sub Command1_Click()
        Register.Show
End Sub
Private Sub Command2_Click()
        Check_Status.Show
End Sub
Private Sub Command3_Click()
        Authenticate.Show
End Sub
Private Sub Image4_Click()
        Unload Me
End Sub
```

```
Private Sub mnu_admin_Click()
        Authenticate.Show
End Sub
Private Sub mnu_register_Click()
        Register.Show
End Sub
Private Sub mnuExit_Click()
        Unload Me
End Sub
Private Sub mnuPAS_Click()
        ABout_us.Show
End Sub
Private Sub mnuStatus_Click()
        Check_Status.Show
End Sub
Private Sub Timer1_Timer()
        lbl_time.Caption = Now
End Sub
```

5.2.3 Code for New Registration

Here the user will have to type in all the registration details and then select register.

After selecting register the System will give a Applicant number which they will have to note down. It is used for checking status. This applicant number is used by administrator for dispatching the passport.

```
Private Sub Command1_Click()
```

```
On Error Resume Next
txt_id = Adodc1.Recordset.RecordCount
txt_status = Trim("Waiting")
MsgBox Adodc1.Recordset.RecordCount, , "Your Applicant Number
is"
Adodc1.Recordset.Update
Unload Me
End Sub
Private Sub Command2_Click()
Unload Me
End Sub
Private Sub Command3_Click()
CommonDialog1.InitDir = App.Path
CommonDialog1.Filter = "JPG Image (*.jpg)|*.jpg|GIF Image(*.gif)|
*.gif|PNG Image(*.png)|*.png|Bitmap Image(*.bmp)|*.bmp"
app_pic.Stretch = True
CommonDialog1.ShowOpen
If Not CommonDialog1.FileName = "" Then
txt_imagepath = CommonDialog1.FileName
app_pic.Picture = LoadPicture(CommonDialog1.FileName)
End If
End Sub
Private Sub Command4_Click()
Me.PrintForm
End Sub
Private Sub Form_Load()
Adodc1.Recordset.AddNew
```

```
male.Value = True
End Sub
Private Sub Timer1_Timer()
If male.Value = True Then
   hid_gender = "Male"
Else
   hid_gender = "Female"
End If

If (Not txt_name.Text = "") And (Not txt_age.Text = "") And (Not
txt_dob.Text = "") And (Not txt_pob.Text = "") And (Not
txt_father.Text = "") And (Not txt_mother.Text = "") And (Not
txt_address.Text = "") And (Not txt_imagepath.Text = "") Then
Command1.Enabled = True
Else
Command1.Enabled = False
End If
End Sub
```

5.2.4 Code for Checking Status

Here the user will have to type in his/her applicant id to know his/ her status.

```
Private Sub Command1_Click()
On Error Resume Next
If Not IsNumeric(Text1.Text) Then
MsgBox "Applicant Id should be a number", vbCritical, "PAS -
Error"
Unload Me
```

```
Check_Status.Show
Else
Dim oconn As New ADODB.Connection
Dim rs As New ADODB.Recordset
Dim strSQL As String
strSQL = "SELECT * FROM Registrations where id =" + Text1.Text
+ ""
strStatus = "Select status from registrations wher id = " + Text1.Text
+ ""
Set oconn = New ADODB.Connection
oconn.Open "Provider=msdaora;Data Source=localhost;User
Id=vivek;Password=vivek;"
rs.CursorType = adOpenStatic
rs.CursorLocation = adUseClient
rs.LockType = adLockOptimistic
rs.Open strSQL, oconn, , , adCmdText
If rs(9).Value = Null Then
MsgBox "Please Verify the Id you have entered", vbCritical, "PAS -
Problem"
Unload Me
Check_Status.Show
End If
lbl_status.Caption = "STATUS : " + rs(9).Value
lbl_name.Caption = "NAME : " + rs(1).Value
app_pic.Stretch = True
app_pic.Picture = LoadPicture(rs(10).Value)
End If
```

```
End Sub
```

5.2.5 Code for Authentication to Admin Panel

Here the system administrator will have to type the username and password, only after authentication the admin panel will be displayed.

```
Private Sub Command1_Click()
If txt_usr = "Administrator" And txt_pass = "admin" Then
Admin.Show
Unload Me
Else
MsgBox "Authentication Failed..Retry"
End If
End Sub
```

5.2.6 Code for Admin Panel

After the admin panel is displayed the administrator can process an application by typing its id.

```
Private Sub Command1_Click()
On Error Resume Next
Dim oconn As New ADODB.Connection
Dim rs As New ADODB.Recordset
Dim strSQL As String
strSQL = "SELECT * FROM Registrations where id=" + app_id.Text
+ ""
```

```
Set oconn = New ADODB.Connection
oconn.Open "Provider=msdaora;Data Source=localhost;User
Id=vivek;Password=vivek;"
rs.CursorType = adOpenStatic
rs.CursorLocation = adUseClient
rs.LockType = adLockOptimistic
rs.Open strSQL, oconn, , , adCmdText
If Not Trim(rs(9).Value) = "Waiting" Then
MsgBox "Passport for the applicant Id you have entered has already
been dispatched"
Else
Process_app.lbl_id = rs(0).Value
Process_app.lbl_name = rs(1).Value
Process_app.lbl_age = rs(2).Value
Process_app.lbl_dob = rs(3).Value
Process_app.lbl_pob = rs(4).Value
Process_app.lbl_father = rs(5).Value
Process_app.lbl_mother = rs(6).Value
Process_app.lbl_gender = rs(7).Value
Process_app.lbl_address = rs(8).Value
Process_app.app_pic.Stretch = True
Process_app.app_pic.Picture = LoadPicture(rs(10).Value)
Process_app.Show
Unload Me
End If
End Sub
```

```
Private Sub Form_Load()
Dim oconn As New ADODB.Connection
Dim rs As New ADODB.Recordset
Dim rs1 As New ADODB.Recordset
Dim strSQL As String
strSQL = "SELECT * FROM Registrations where status='Waiting'"
strSQL1 = "SELECT * FROM Registrations where
status='Dispatched'"
Set oconn = New ADODB.Connection
oconn.Open "Provider=msdaora;Data Source=localhost;User
Id=vivek;Password=vivek;"
rs.CursorType = adOpenStatic
rs.CursorLocation = adUseClient
rs.LockType = adLockOptimistic
rs.Open strSQL, oconn, , , adCmdText
Set DataGrid1.DataSource = rs
rs1.CursorType = adOpenStatic
rs1.CursorLocation = adUseClient
rs1.LockType = adLockOptimistic
rs1.Open strSQL1, oconn, , , adCmdText
Set DataGrid2.DataSource = rs1
End Sub
Private Sub Timer1_Timer()
Label2.Caption = Now
End Sub
```

5.2.7 Code for Processing Application

Here the desired application is processed and the passport is dispatched.

```
Private Sub Command1_Click()
If Check1.Value = vbChecked And Check2.Value = vbChecked And
Check3.Value = vbChecked Then
Dim oconn As New ADODB.Connection
Dim rs As New ADODB.Recordset
Dim strSQL As String
strSQL = "update registrations set status='Dispatched' where id=" +
lbl_id.Caption + ""
Set oconn = New ADODB.Connection
oconn.Open "Provider=msdaora;Data Source=localhost;User
Id=vivek;Password=vivek;"
rs.CursorType = adOpenStatic
rs.CursorLocation = adUseClient
rs.LockType = adLockOptimistic
rs.Open strSQL, oconn, , , adCmdText
MsgBox "Dispatch Successful.."
Unload Me
Admin.Show
Else
MsgBox "Please make sure you selected all conditions", , "Verify!!"
End If
```

```
End Sub
Private Sub Command2_Click()
Unload Me
Admin.Show
End Sub
```

CHAPTER 4

SYSTEM TESTING

Testing is one of the important steps in the software development phase. Testing is performed to identify errors and is an integral part of the entire development and maintenance process. The Passport Automation System has been put under rigorous testing so as ensure the correctness of its design. The two basic testing strategies that were used:

1. **Unit Testing**.
2. **Integration Testing.**

4.1 UNIT TESTING

Unit testing was conducted to verify the functional performance of each modular component of the software. Unit testing focuses on the smallest unit of software design i.e. the module. White-box testing were heavily employed for unit testing.

4.1.1 NEW REGISTRATION TEST CASE

Step No	Test Step	Expected Result	Actual Result	Test Result
1	Enter an the date of birth as "20.05.1991"	An error "Enter a valid date of birth format".	Nothing actually popped up and the improper format was added to the database.	FAIL

2	Enter the age as "AB" (some characters rather that number)	An error message "Age should be a number"		PASS

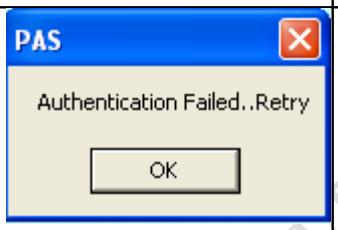
Tabel 4.1 New Registration Test Case

4.1.2 CHECK STATUS TEST CASE

Step No	Test Step	Expected Result	Actual Result	Test Result
1	Enter an applicant Id not existing in the database.	An error "Please verify the id you have entered".		PASS
2	Enter invalid data type as an input Applicant Id.	An error message "**Applicant Id should be a number**"	 It gave message which was not expected.	FAIL

Tabel 4.2 Check status Test Case

4.1.3 AUTHENTICATION TEST CASE

Step No	Test Step	Expected Result	Actual Result	Test Result
1	Enter the username as "Administrator" and password as "**********" (a wrong password)	An error "Authentication Failed. Retry".		PASS

Tabel 4.3 Authentication Test Case

4.1.4 ADMIN PANEL TEST CASE

Step No	Test Step	Expected Result	Actual Result	Test Result
1	Enter an applicant Id of an already dispatched applicant.	It must not show the 'Process application' form.	It showed the 'Process application form' despite the fact that the application has already been dispatched.	FAIL

Tabel 4.4 Admin Panel Test Case

4.2 INTEGRATION TESTING

"Unit testing" focuses on testing a unit of the code."Integration testing" is the next level of testing. This 'level of testing' focuses on testing the integration of "units of code" or components. **The Passport Automation System was tested as a whole**.

Step No	Test Step	Expected Result	Actual Result	Test Result
1	Enter all the details of the applicant and then click Register"	It must display that "Registration Successful" with the Applicant Id.	 The Applicant id was displayed but the message "Registration Successful" was not displayed.	**Partial Success.**
2	Check the status by typing the applicant ID as "3".	It must display the name and the status as waiting.	NAME : DHONI STATUS : Waiting	**PASS**
3	The Administrator enters the correct Authentication details.	Successful Login into the Admin Panel.	Admin Panel was opened successfully.	**PASS**
4	The Administrato	Process Application	The page was displayed.	**PASS**

	r enters the Applicant Id to process i.e. "3"	page to be displayed.		
5	The Administrator without checking the contents of the "Checklist" clicked "Dispatch Passport"	Error to prompt the administrator to verify the checklist.	The passport was successfully dispatched.	**FAIL**
6	The administrator enters the id of an already dispatched pasport.	Error stating that "Passport Already dispatched. "	An error message stating "The Passport for applicant id you have entered is already dispatched" occured	**PASS**

Tabel 4.5 Integration Testing

CHAPTER 6

CONCLUSION

PAS simplifies the manual work load and is helpful in the effective dispatch of passport. The applicant of passport is benefited by the fact that he need not go all the way to the passport office to submit application but can do this from the comfort of his own. The job of the administrator is simplified because of this system. The applicant can check his passport status anytime, anywhere and he need not stand in the enquiry queue. Furthermore, the time taken for the passport to reach the applicant is considerably reduced.

REFERENCES

[1] Ali Bahrami, "Object oriented Systems development", using the unified modeling language, Tata McGraw Hill edition, pp.400-420, 2008

[2] Gary Cornell, "Visual basic 6: from the GROUND UP" Build windows and web applications step by step, pp. 950-1000

[3] Kevin Loney, "Oracle 10g: The complete reference", Master the revolutionary features of oracle, pp. 940-950, 2004

[4] Steven Holzner, "Visual Basic black book", Comprehensive problem solver, pp. 1050-1100, 1998.