ST. ANNE'S

COLLEGE OF ENGINEERING AND TECHNOLOGY

ANGUCHETTYPALAYAM, PANRUTI – 607 110

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LAB OBSERVATION NOTE

NAME	**:**	**----------**
REGISTER NO.	**:**	**----------**
YEAR/ SEM	**:**	**II / IV**
SUBJECT	**:**	**CS8481 , Database Management Systems Laboratory**
DURATION	**:**	**Dec 2019 – Apr 2020**

LIST OF EXPERIMENTS

1. Data Definition Commands, Data Manipulation Commands for inserting, deleting, updating and retrieving Tables and Transaction Control statements
2. Database Querying – Simple queries, Nested queries, Sub queries and Joins
3. Views, Sequences, Synonyms
4. Database Programming: Implicit and Explicit Cursors
5. Procedures and Functions
6. Triggers
7. Exception Handling
8. Database Design using ER modeling, normalization and Implementation for any application
9. Database Connectivity with Front End Tools
10. Case Study using real life database applications

SOFTWARE:

- Front end: VB/VC ++/JAVA or Equivalent
- Back end: Oracle / SQL / MySQL/ PostGress / DB2 or Equivalent

TABLE OF CONTENTS

20.		Inventory Control System		
21.		Material Requirement Processing System		
22.		Hospital Management System		
23.		Railway Reservation System		
24.		Personal Information System		
25.		Hotel Management System		

Ex. No: 1 **SQL BASIC COMMANDS**

Date:

AIM:

To write SQL queries to execute basic SQL commands.

QUERIES:

1. Create table

Query:

```
CREATE TABLE emp
(
        empno NUMBER,
        empname VARCHAR2(255),
        DOB DATE,
        salary NUMBER,
        designation VARCHAR2(20)
);
```

Output:

Table created.

2. Insert values

Query:

```
INSERT INTO emp VALUES(100,'John','4.21.1994', 50000,'Manager');

INSERT INTO emp VALUES(101,'Greg','6.20.1994',25000,'Clerk');
```

Output:

2 rows inserted

3. Display values

Query:

```
SELECT * FROM emp;
```

Output:

EMPNO	EMPNAME	DOB	SALARY	DESIGNATION
100	John	04/21/1994	50000	Manager
101	Greg	06/20/1994	25000	Clerk

Query:

```
SELECT empname,salary FROM emp;
```

Output:

EMPNAME	SALARY
John	50000
Greg	25000

4. Modify values

Query:

UPDATE emp SET salary = salary + 1000;

Output:

2 row(s) updated.

Query:

SELECT * FROM emp;

Output:

EMPNO	EMPNAME	DOB	SALARY	DESIGNATION
100	John	04/21/1994	51000	Manager
101	Greg	06/20/1994	26000	Clerk

5. Delete values

Query:

DELETE FROM emp WHERE empno = 100;

Output:

1 row(s) deleted.

Query:

SELECT * FROM emp;

Output:

EMPNO	EMPNAME	DOB	SALARY	DESIGNATION
101	Greg	06/20/1994	26000	Clerk

RESULT:

Thus the basic SQL queries were successfully executed and verified.

Ex. No: 2 **DATA DEFINITION LANGUAGE (DDL)**

Date :

AIM:

To write the SQL queries using DDL Commands with and without constraints.

DDL STATEMENTS

- CREATE TABLE
- ALTER TABLE
- DROP TABLE

SYNTAX:

1. Create Table

The CREATE TABLE statement is used to create a relational table

```
CREATE TABLE table_name
(
        column_name1 data_type [constraints],
        column_name1 data_type [constraints],
        column_n
        ame1 data_type [constraints],
        ……..
);
```

2. Alter Table

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table

a. To Add a column

```
ALTER TABLE table_name ADD column_name datatype
```

b. To delete a column in a table

```
ALTER TABLE table_name DROP (column_name)
```

c. To change the data type of a column in a table

```
ALTER TABLE table_name MODIFY(column_name datatype )
```

3. Drop Table

Used to delete the table permanently from the storage

```
DROP TABLE table_name
```

QUERIES:

1. CREATE THE TABLE (with no constraint)

Query:

```
CREATE TABLE emp
(
        empno NUMBER,
        empname VARCHAR2(25),
        dob DATE,
        salary NUMBER,
        designation VARCHAR2(20)
);
```

Output:

Table Created

Query:

```
DESC emp;
```

Output:

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
EMP	EMPNO	NUMBER	22	-	-	-		-	-
	EMPNAME	VARCHAR2	255	-	-	-		-	-
	DOB	DATE	7	-	-	-		-	-
	SALARY	NUMBER	22	-	-	-		-	-
	DESIGNATION	VARCHAR2	20	-	-	-		-	-

2. ALTER THE TABLE

a. ADD

// To alter the table emp by adding new attribute department

Query:

```
ALTER TABLE emp ADD department VARCHAR2(50);
```

Output:

Table Altered

Query:

DESC emp;

Output:

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
EMP	EMPNO	NUMBER	22	-	-	-		-	-
	EMPNAME	VARCHAR2	255	-	-	-		-	-
	DOB	DATE	7	-	-	-		-	-
	SALARY	NUMBER	22	-	-	-		-	-
	DESIGNATION	VARCHAR2	20	-	-	-		-	-
	DEPARTMENT	VARCHAR2	50	-	-	-		-	-

b. MODIFY

//To alter the table emp by modifying the size of the attribute department

Query:

ALTER TABLE emp MODIFY (department VARCHAR2(100));

Output:

Table Altered

Query:

DESC emp;

Output:

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
EMP	EMPNO	NUMBER	22	-	-	-		-	-
	EMPNAME	VARCHAR2	255	-	-	-		-	-
	DOB	DATE	7	-	-	-		-	-
	SALARY	NUMBER	22	-	-	-		-	-
	DESIGNATION	VARCHAR2	20	-	-	-		-	-
	DEPAR	VARC	100	-	-	-		-	-

	TMENT	HAR2

c. DROP

// To alter the table emp by deleting the attribute department

Query:

ALTER TABLE emp DROP(department);

Output:

Table Altered

Query:

DESC emp;

Output:

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
EMP	EMPNO	NUMBER	22	-	-	-	-	-	
	EMPNAME	VARCHAR2	255	-	-	-	-	-	
	DOB	DATE	7	-	-	-	-	-	
	SALARY	NUMBER	22	-	-	-	-	-	
	DESIGNATION	VARCHAR2	20	-	-	-	-	-	

d. RENAME

// To alter the table name by using rename keyword

Query:

ALTER TABLE emp RENAME TO emp1 ;

Output:

Table Altered

Query:

DESC emp1;

Output:

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
EMP1	EMPNO	NUMBER	22	-	-	-	-	-	
	EMPNAME	VARCHAR2	255	-	-	-	-	-	
	DOB	DATE	7	-	-	-	-	-	

	SALARY	NUMBER	22	-	-	-	-	-	
	DESIGNATION	VARCHAR2	20	-	-	-	-	-	
	DEPARTMENT	VARCHAR2	100	-	-	-	-	-	

3. DROP

//To delete the table from the database

Query:

DROP TABLE emp1;

Output:

Table Dropped

Query:

DESC emp1;

Output:

Ouubject to be described could not be found.

CONSTRAINT TYPES:

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK
- DEFAULT

QUERIES:

1. CREATE THE TABLE

Query:

```
CREATE TABLE student
(
        studentID NUMBER PRIMARY KEY,
        sname VARCHAR2(30) NOT NULL,
        department CHAR(5),
        sem NUMBER,
        dob DATE,
        email_id VARCHAR2(20) UNIQUE,
        college VARCHAR2(20) DEFAULT 'MEC'
```

);

Output:

Table created.

Query:

DESC student;

Output:

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
STUDENT	STUDENTID	NUMBER	22	-	-	1	-	-	-
	SNAME	VARCHAR2	30	-	-	-	-	-	-
	DEPARTMENT	CHAR	5	-	-	-	✓	-	-
	SEM	NUMBER	22	-	-	-	✓	-	-
	DOB	DATE	7	-	-	-	✓	-	-
	EMAIL_ID	VARCHAR2	20	-	-	-	✓	-	-
	COLLEGE	VARCHAR2	20	-	-	-	✓	'MEC'	-

Query:

```
CREATE TABLE exam
(
        examID NUMBER ,
        studentID NUMBER REFERENCES student(studentID),
        department CHAR(5) NOT NULL,
        mark1 NUMBER CHECK (mark1<=100 and mark1>=0),
        mark2 NUMBER CHECK (mark2<=100 and mark2>=0),
        mark3 NUMBER CHECK (mark3<=100 and mark3>=0),
        mark4 NUMBER CHECK (mark4<=100 and mark4>=0),
        mark5 NUMBER CHECK (mark5<=100 and mark5>=0),
        total NUMBER,
        average NUMBER,
        grade CHAR(1)
);
```

Output:

Table created.

//To alter the table student by adding new constraint to the examID attribute

Query:

```
ALTER TABLE student ADD CONSTRAINT pr
                                    PRIMARY KEY (examid);
```

Output:

Table altered.

2. CREATE THE TABLE USING COMPOSITE PRIMARY KEY

Create the following table with the attributes reg_no and stu_name as primary key.

stu_details (reg_no, stu_name, DOB, address, city)

Query:

```
CREATE TABLE stu_details
(
    reg_no number,
    stu_name varchar2(30),
    DOB date,
    address varchar2(30),
    city char(30),
    primary key(reg_no, stu_name)
);
```

Output:

Table created.

Query:

```
DESCstu_details
```

Output:

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
STU_DETAILS	REG_NO	NUMBER	22	-	-	1	-	-	-
	STU_NAME	VARCHAR2	30	-	-	2	-	-	-
	DOB	DATE	7	-	-	-	✓	-	-
	ADDRESS	VARCHAR2	30	-	-	-	✓	-	-
	CITY	CHAR	30	-	-	-	✓	-	-

RESULT:

Thus the SQL queries using DDL Commands with and without constraints were successfully executed and verified.

Ex.No: 3 **DATA MANIPULATION LANGUAGE (DML)**

Date:

AIM:

To write SQL queries using DML commands to manage the database.

DML STATEMENTS

- INSERT
- UPDATE
- DELETE
- SELECT

SYNTAX:

1. Insert

// To insert a new row in a table.

a. Direct Substitution

```
INSERT INTO table_name VALUES(value1, value2,value3, ……...);
```

b. Specific Column Insertion

```
INSERT INTO table_name (columnane1, columnname2, columnname3)
VALUES (values1, value2,value3, ……) ;
```

2. Update

// To update new data into an existing table

```
UPDATE table_name SET column1=value, column2=value2,...
WHERE some_column=some_value;
```

3. Delete Query

//To delete rows in a table.

```
DELETE FROM table_name WHERE
```

QUERIES:

1. INSERT

Query:

```
INSERT INTO student
VALUES(101,'RUPESH','IT',5,'04/18/1996','rupesh@gmail.com','MEC');
```

INSERT INTO student VALUES
(102,'BALA','CSE',7,'10/7/1995','bala@gmail.com','IIT');

INSERT INTO student VALUES
(104,'HEMESH','IT',5,'7/23/1996','hemesh@gmail.com','IIT');

INSERT INTO student VALUES
(106,'SAIVAISHNAVI','CSE',5,'06/9/1996', 'vaishu@gmail.com','IFET');

//(For the purpose of default constraintSpecific Column Insertion)

INSERT INTO student(studentid,sname,department,sem,dob,email_id)
VALUES (108,'KAVINAYA','IT',5,'04/21/1996','kavin@gmail.com');

Output:

5 Rows Inserted

Query:

SELECT * FROM student;

Output:

STUDENTID	SNAME	DEPARTMENT	SEM	DOB	EMAIL_ID	COLLEGE
101	RUPESH	IT	5	04/18/1996	rupesh@gmail.com	MEC
102	BALA	CSE	7	10/07/1995	bala@gmail.com	IIT
103	HEMESH	IT	5	07/23/1996	hemesh@gmail.com	IIT
104	SAIVAISHNAVI	CSE	5	06/09/1996	vaishu@gmail.com	IFET
108	KAVINAYA	IT	5	04/21/1996	kavin@gmail.com	MEC

Query:

INSERT INTO exam(examid, studentid, department, mark1, mark2, mark3, mark4, mark5) VALUES (2222,101,'IT',98,87,83,99,87);

INSERT INTO exam(examid, studentid, department, mark1, mark2, mark3, mark4,mark5) VALUES(3333,104,'IT',99,82,84,89,100);

INSERT INTO exam(examid, studentid, department, mark1, mark2, mark3, mark4,mark5) VALUES(4444,108,'IT',92,85,83,91,87);

Output:

3 Rows inserted

Query:

SELECT * FROM exam;

Output:

EXAMID	STUDENTID	DEPARTMENT	MARK1	MARK2	MARK3	MARK4	MARK5	TOTAL	AVERAGE	GRADE
2222	101	IT	98	87	83	99	87	-	-	-
3333	104	IT	99	82	84	89	100	-	-	-
4444	108	IT	92	85	83	91	87	-	-	-

2. UPDATE

Query:

UPDATE exam SET total=(mark1+mark2+mark3+mark4+mark5);

Output:

1 row(s) updated

Query:

SELECT * FROM exam;

Output:

EXAMID	STUDENTID	DEPARTMENT	MARK1	MARK2	MARK3	MARK4	MARK5	TOTAL	AVERAGE	GRADE
2222	101	IT	98	87	83	99	87	454	-	-
3333	104	IT	99	82	84	89	100	454	-	-
4444	108	IT	92	85	83	91	87	438	-	-

Query:

UPDATE exam SET average=total/5;

Output:

3 row(s) updated

Query:

SELECT * FROM exam;

Output:

EXAMID	STUDENTID	DEPARTMENT	MARK 1	MARK 2	MARK 3	MARK 4	MARK 5	TOTAL	AVERAGE	GRADE
2222	101	IT	98	87	83	99	87	454	90.8	-
3333	104	IT	99	82	84	89	100	454	90.8	-
4444	108	IT	92	85	83	91	87	438	87.6	-

Query:

```
UPDATE exam SET grade='S' WHERE average>95;
```

Output:

0 row(s) updated

Query:

```
UPDATE exam SET grade='A' WHERE average<=95 AND average>90;
```

Output:

2 row(s) updated

Query:

```
UPDATE exam SET grade='B' WHERE average<=90 AND
average>85;
```

Output:

1 row(s) updated

Query:

```
UPDATE exam SET grade='C' WHERE average<=85 AND
average>80;
```

Output:

0 row(s) updated

Query:

```
UPDATE exam SET grade='D' WHERE average<=80 AND
average>75;
```

Output:

0 row(s) updated

Query:

```
UPDATE exam SET grade='F' WHERE average<75;
```

Output:

0 row(s) updated

Query:

SELECT * FROM exam;

Output:

EXAMID	STUDENTID	DEPARTMENT	MARK1	MARK2	MARK3	MARK4	MARK5	TOTAL	AVERAGE	GRADE
2222	101	IT	98	87	83	99	87	454	90.8	A
3333	104	IT	99	82	84	89	100	454	90.8	A
4444	108	IT	92	85	83	91	87	438	87.6	B

3. DELETE

Query:

DELETE FROM exam WHERE examid=2222;

Output:

1 row(s) deleted

Query:

SELECT * FROM exam;

Output:

EXAMID	STUDENTID	DEPARTMENT	MARK1	MARK2	MARK3	MARK4	MARK5	TOTAL	AVERAGE	GRADE
3333	104	IT	99	82	84	89	100	454	90.8	A
4444	108	IT	92	85	83	91	87	438	87.6	B

4. DATA QUERY PROJECTION/SELECTION STATEMENT

1. Select all columns

Query:

SELECT * FROM student;

Output:

STUDENTID	SNAME	DEPARTMENT	SEM	DOB	EMAIL_ID	COLLEGE
101	RUPESH	IT	5	04/18/1996	rupesh@gmail.com	MEC
102	BALA	CSE	7	10/07/1995	bala@gmail.com	IIT

103	HEMESH	IT	5	07/23/1996	hemesh@gmail.com	IIT
104	SAIVAISHNAVI	CSE	5	06/09/1996	vaishu@gmail.com	IFET
108	KAVINAYA	IT	5	04/21/1996	kavin@gmail.com	MEC

2. Select multicolumn

Query:

SELECT sname, department, sem FROM student;

Output:

SNAME	DEPARTMENT	SEM
RUPESH	IT	5
BALA	CSE	7
HEMESH	IT	5
SAIVAISHNAVI	CSE	5
KAVINAYA	IT	5

3. Selection with alias column name

Query:

SELECT sname as "Student Name" FROM student;

Output:

Student Name
RUPESH
BALA
HEMESH
SAIVAISHNAVI
KAVINAYA

4. Distinct record selection

Query:

SELECT DISTINCT college FROM student;

Output:

COLLEGE
MEC
IIT
IFET

5. Selection with concatenation

Query:

SELECT sname||studentid FROM student;

Output:

SNAME||STUDENTID
RUPESH101
BALA102
HEMESH103
SAIVAISHNAVI104
KAVINAYA108

6. Selection with where clause

// To display the records from the table student who belongs to mec college.

Query:

SELECT * FROM student WHERE college='MEC';

Output:

STUDENTID	SNAME	DEPARTMENT	SEM	DOB	EMAIL_ID	COLLEGE
101	RUPESH	IT	5	04/18/1996	rupesh@gmail.com	MEC
108	KAVINAYA	IT	5	04/21/1996	kavin@gmail.com	MEC

// To display the student id and student name from the table student who belongs

to IIT college

Query:

SELECT studentid, sname FROM student WHERE college='IIT';

Output:

STUDENTID	SNAME
102	BALA
103	HEMESH

// To display the student name and department from the table student who belongs to 5th sem

Query:

SELECT sname, department FROM student WHERE sem=5;

Output:

SNAME	DEPARTMENT
RUPESH	IT
HEMESH	IT
SAIVAISHNAVI	CSE
KAVINAYA	IT

// To display the student id, student name and department of the student whose the semester in between 5 and 6

Query:

SELECT studentid, sname, department FROM student WHERE sem BETWEEN 5 AND 6;

Output:

STUDENTID	SNAME	DEPARTMENT
101	RUPESH	IT
103	HEMESH	IT
104	SAIVAISHNAVI	CSE
108	KAVINAYA	IT

// To display the student id, student name and department of the student whose in CSE and IT department

Query:

SELECT studentid, sname, department FROM student WHERE department IN ('CSE','IT');

Output:

STUDENTID	SNAME	DEPARTMENT
101	RUPESH	IT
102	BALA	CSE
103	HEMESH	IT
104	SAIVAISHNAVI	CSE
108	KAVINAYA	IT

// To display the student id, student name and department of the student whose not in CSE department

Query:

SELECT studentid, sname, department FROM student WHERE department NOT IN ‘CSE’;

Output:

STUDENTID	SNAME	DEPARTMENT
101	RUPESH	IT
103	HEMESH	IT
108	KAVINAYA	IT

Query:

SELECT studentid, sname FROM student WHERE studentid > 105;

Output:

STUDENTID	SNAME
108	KAVINAYA

Query:

SELECT studentid, sname FROM student WHERE studentid > 105 AND department=’CSE’;

Output:

no data found

// To display the student id and student name of the student whose name starts letters 'R' from the table student

Query:

SELECT studentid, sname FROM student WHERE sname LIKE 'R%';

Output:

STUDENTID	SNAME
101	RUPESH

// To display the student id and student name of the student whose name end with 'H' from the table student

Query:

SELECT studentid, sname FROM student WHERE sname LIKE '%H';

Output:

STUDENTID	SNAME
101	RUPESH
103	HEMESH

DATETIME FUNCTIONS:

1. **Sysdate:**

SELECT sysdateFROM dual;

Output:

SYSDATE
06/17/2017

2. **Systimestamp:**

SELECT systimestamp FROM dual;

Output:

SYSTIMESTAMP
17-JUN-17 02.51.47.366000 PM -07:00

3. **Add_month**

SELECT add_months(sysdate,6) FROM dual;

Output:

ADD_MONTHS(SYSDATE,6)
12/17/2017

4. **Last_day**

SELECT last_day(sysdate) FROM dual;

Output:

LAST_DAY(SYSDATE)
06/30/2017

5. **Months_between**

SELECT months_between(sysdate,
to_date('08/15/1947')) FROM dual;

Output:

MONTHS_BETWEEN(SYSDATE,TO_DATE('08/15/1947'))
838.084636350059737156511350059737156511

6. **Next_day**

SELECT next_day(sysdate, 'TUESDAY') FROM dual;

Output:

NEXT_DAY(SYSDATE,'TUESDAY')
06/20/2017

7. **Years_between**

SELECT years_between(sysdate, to_date(10/07/1981'))
FROM dual;

Output:

NUMERICAL FUNCTIONS:

1. **Ceil**

SELECTceil(123.4567) FROM dual;

Output:

CEIL(12345.67)
12346

2. **Floor**

SELECTfloor(12345.67) FROM dual;

Output:

FLOOR(12345.67)
12345

3. **Round**

SELECTround(3.1415926, 4) FROM dual;

Output:

ROUND(3.1415926,4)
3.1416

STRING FUNCTIONS

1. **Initcap**

SELECTinitcap('information technology')FROM dual;

Output:

INITCAP('INFORMATIONTECHNOLOGY')
Information Technology

2. **Lower**

SELECTlower('INFORMATION TECHNOLOGY') FROM dual;

Output:

LOWER('INFORMATIONTECHNOLOGY')
information technology

3. **Upper**

SELECT upper('information technology') FROM dual;

Output:

UPPER('INFORMATIONTECHNOLOGY')
INFORMATION TECHNOLOGY

4. **Replace**

SELECT replace('So What', 'o', 'ay') FROM dual;

Output:

REPLACE('SOWHAT','O','AY')
Say What

ORDER BY

// To display the department, sem and student name from the table student based on department in ascending order.

SELECT department, sem, sname FROM student ORDER BY department;

Output:

DEPARTMENT	SEM	SNAME
CSE	7	BALA
CSE	5	SAIVAISHNAVI
IT	5	HEMESH
IT	5	RUPESH
IT	5	KAVINAYA

// To display the department, sem and student name from the table student based on department in descending order.

SELECT department, sem, sname FROM student ORDER BY department DESC, sem DESC, sname DESC;

Output:

DEPARTMENT	SEM	SNAME
IT	5	RUPESH
IT	5	KAVINAYA
IT	5	HEMESH
CSE	7	BALA
CSE	5	SAIVAISHNAV

RESULT:

Thus the SQL queries using DML Commands were successfully executed and verified.

Ex. No: 4 **AGGREGATE FUNCTIONS**
Date:

AIM:

To write SQL queries to perform aggregate functions.

AGGREGATE FUNCTIONS

1. Count
2. Max
3. Min
4. Sum
5. Avg

// 1. COUNT - displays total number of rows

SELECT COUNT(examid) AS STUDENTS_REGISTERED FROM exam;

Output:

STUDENTS_REGISTERED
3

// 2. MAX - displays the maximum value

SELECT MAX(average) AS RANK_1 FROM exam;

Output:

RANK_1
90.8

// 3. MIN - displays the minimum value

SELECT MIN(average) AS LAST_RANK FROM exam;

Output:

LAST_RANK
87.6

// 4. SUM - displays the total value

SELECT department, SUM(total) AS SUM_DEPARTMENT FROM exam GROUP BY department;

Output:

DEPARTMENT	SUM_DEPARTMENT
IT	1346

// 5. AVG - displays the average value

SELECT department, AVG(total) AS AVERAGE FROM exam GROUP BY department;

Output:

DEPARTMENT	AVERAGE
IT	448.66666666666666666666666666666666667

RESULT:

Thus the SQL queries using aggregate functions were successfully executed and verified.

Ex. No: 5 **SQL JOINS**

Date:

AIM:

To write SQL queries using joins to combine two or more tables.

JOIN TYPES

1. INNER JOIN
2. OUTER JOIN
    a. LEFT OUTER JOIN
    b. RIGHT OUTER JOIN
    c. FULL OUTER JOIN
3. SELF JOIN
4. EQUI JOIN
5. SELF JOIN

1. Table Creation

Query:

```
CREATE TABLE itstudent
(
        studentID NUMBER PRIMARY KEY,
        sname VARCHAR(30),
        department CHAR(5),
        sem NUMBER
);
```

Output:

Table Created

Query:

```
CREATE TABLE placement
(
        PlacementID NUMBER PRIMARY KEY,
        StudentID NUMBER references itstudent(studentid),
        department CHAR(5),
        Company VARCHAR2(30),
        salary NUMBER
);
```

Output:

Table Created

2. Insert into the table

Query:

INSERT INTO itstudent VALUES(101,'reema', 'IT',5);

Output:

1 row(s) inserted.

Query:

INSERT INTO itstudent VALUES(102,'reenu', 'IT',3);

Output:

1 row(s) inserted

Query:

INSERT INTO itstudent VALUES(103,'sheela', 'CSE',3);

Output:

1 row(s) inserted

Query:

INSERT INTO itstudent VALUES(104,'nirmal', 'IT',3);

Output:

1 row(s) inserted

Query:

INSERT INTO itstudent VALUES(105,'eshwar', 'CSE',5);

Output:

1 row(s) inserted

Query:

INSERT INTO placement VALUES(1, 104, 'IT', 'infosys', 25000);

Output:

1 row(s) inserted.

Query:

INSERT INTO placement VALUES(2, 105, 'CSE', 'Wipro', 22000);

Output:

1 row(s) inserted

Query:

INSERT INTO placement VALUES(4, 102, 'IT', 'infosys', 25000);

Output:

1 row(s) inserted

Query:

INSERT INTO placement VALUES(5, 103, 'CSE', 'infosys', 25000);

Output:

1 row(s) inserted

3. Select the table:

Query:

```
SELECT * FROM itstudent;
```

Output:

STUDENTID	SNAME	DEPARTMENT	SEM
101	rupesh	IT	5
102	bala	IT	3
103	sheela	CSE	3
104	nirmal	IT	3
105	eshwar	CSE	5

Query:

```
SELECT * FROM placement;
```

Output:

PLACEMENTID	STUDENTID	DEPARTMENT	COMPANY	SALARY
1	104	IT	infosys	25000
2	105	CSE	Wipro	22000
3	102	IT	infosys	25000
4	103	CSE	infosys	25000

4. INNER JOIN

Query:

```
SELECT *
FROM itstudent
INNER JOIN Placement
ON itstudent.studentID=placement.StudentID;
```

Output:

STUDENTID	SNAME	DEPARTMENT	SEM	PLACEMENTID	STUDENTID	DEPARTMENT	COMPANY	SALARY
104	nirmal	IT	3	1	104	IT	infosys	25000
105	eshwar	CSE	5	2	105	CSE	Wipro	22000
102	bala	IT	3	4	102	IT	infosys	25000

103	sheel a	CSE	3	5	103	CSE	infosy s	250 00

Query:

```
SELECT itstudent.studentID, itstudent.sname,placement.company,
placement.salary FROM itstudent
INNER JOIN placement
ON itstudent.studentID=placement.studentID;
```

Output:

STUDENTID	SNAME	COMPANY	SALARY
104	nirmal	infosys	25000
105	eshwar	Wipro	22000
102	bala	infosys	25000
103	sheela	infosys	25000

5. LEFT OUTER JOIN

Query:

```
SELECT *
FROM itstudent
LEFT OUTER JOIN placement
ON itstudent.studentID=placement.studentID;
```

Output:

STUDE NTID	SNA ME	DEPART MENT	SE M	PLACEM ENTID	STUDE NTID	DEPART MENT	COMP ANY	SAL ARY
104	nir mal	IT	3	1	104	IT	infosy s	2500 0
105	esh war	CSE	5	2	105	CSE	Wipro	2200 0
102	bala	IT	3	4	102	IT	infosy s	2500 0
103	shee la	CSE	3	5	103	CSE	infosy s	2500 0
101	rupe sh	IT	5	-	-	-	-	-

6. RIGHT OUTER JOIN

Query:

```
SELECT *
FROM itstudent
```

RIGHT OUTER JOIN placement

ON itstudent.studentID=placement.studentID;

Output:

STUDENTID	SNAME	DEPARTMENT	SEM	PLACEMENTID	STUDENTID	DEPARTMENT	COMPANY	SALARY
102	bala	IT	3	4	102	IT	infosys	25000
103	sheela	CSE	3	5	103	CSE	infosys	25000
104	nirmal	IT	3	1	104	IT	infosys	25000
105	eshwar	CSE	5	2	105	CSE	Wipro	22000
-	-	-	-	3	204	MECH	HYUNDAI	30000

7. FULL OUTER JOIN

Query:

SELECT *

FROM itstudent

FULL OUTER JOIN placement

ON itstudent.studentID=placement.studentID;

Output:

STUDENTID	SNAME	DEPARTMENT	SEM	PLACEMENTID	STUDENTID	DEPARTMENT	COMPANY	SALARY
104	nirmal	IT	3	1	104	IT	infosys	25000
105	eshwar	CSE	5	2	105	CSE	Wipro	22000
-	-	-	-	3	204	MECH	HYUNDAI	30000
102	bala	IT	3	4	102	IT	infosys	25000
103	sheela	CSE	3	5	103	CSE	infosys	25000
101	rupesh	IT	5	-	-	-	-	-

8. EQUI JOIN

Query:

SELECT * FROM itstudent, placement WHERE

itstudent.studentID=placement.studentID;

Output:

STUDENTID	SNAME	DEPARTMENT	SEM	PLACEMENTID	STUDENTID	DEPARTMENT	COMPANY	SALARY
104	nirmal	IT	3	1	104	IT	infosys	25000
105	eshwar	CSE	5	2	105	CSE	Wipro	22000
102	bala	IT	3	4	102	IT	infosys	25000
103	sheela	CSE	3	5	103	CSE	infosys	25000

9. SELF JOIN

Returns rows by comparing the values of the same table.

Query:

```
CREATE TABLE employee
(
        empid NUMBER,
        empname VARCHAR2(25),
        reportingid NUMBER
);
```

Output:

Table Created

Query:

```
INSERT INTO employee VALUES(1, 'Principal', null);
```

Output:

1 row(s) inserted.

Query:

```
INSERT INTO employee VALUES(2, 'HOD', 1);
```

Output:

1 row(s) inserted.

Query:

```
INSERT INTO employee VALUES(3, 'PO', 1);
```

Output:

1 row(s) inserted.

Query:

INSERT INTO employee VALUES(4, 'Staff', 2);

Output:

1 row(s) inserted.

Query:

INSERT INTO employee VALUES(5, 'Non Teaching Staff', 2);

Output:

1 row(s) inserted.

Query:

SELECT * FROM employee;

Output:

EMPID	EMPNAME	REPORTINGID
1	Principal	-
2	HOD	1
3	PO	1
4	Staff	2
5	Non Teaching Staff	2

Query:

```
SELECT e1.empid, e1.empname, e2.empname AS HeadName
FROM employee e1, employee e2
WHERE e1.reportingid=e2.empid;
```

Output:

EMPID	EMPNAME	HEADNAME
3	PO	Principal
2	HOD	Principal
5	Non Teaching Staff	HOD
4	Staff	HOD

RESULT:

Thus the SQL queries using SQL joins were successfully executed and verified.

Ex. No: 6 **NESTED QUERIES**

Date:

AIM:

To create and execute nested SQL queries .

//Create a table employee1

```
CREATE TABLE employee1
(
empno NUMBER,
empname VARCHAR2(255),
salary NUMBER,
designation VARCHAR2(20)
);
```

Table created

Query:

```
SELECT * FROM employee1 WHERE designation =
(SELECT designation FROM employee1 WHERE empname='RUPESH');
```

Output:

EMPNO	EMPNAME	SALARY	DESIGNATION
1	Rupesh	50000	MD

Query:

```
SELECT * FROM employee1 WHERE salary >(SELECT salary FROM
employee1 WHERE empname='Sai');
```

Output:

EMPNO	EMPNAME	SALARY	DESIGNATION
1	Rupesh	50000	MD
2	Bala	45000	Manager

Query:

```
SELECT studentID, sname FROM itstudent WHERE studentID IN
(SELECT studentID FROM placement);
```

STUDENTID	SNAME
102	reenu
103	sheela
104	nirmal

Output:

105	eshwar

Query:

SELECT studentID, sname FROM itstudent WHERE studentID NOT IN (SELECT studentID FROM placement);

Output:

STUDENTID	SNAME
101	reema

Query:

SELECT sname, department FROM itstudent WHERE studentID in (SELECT studentID FROM placement WHERE company='infosys');

Output:

SNAME	DEPARTMENT
reenu	IT
sheela	CSE
nirmal	IT

Query:

SELECT sname, department FROM itstudent WHERE studentID NOT IN (SELECT studentID FROM placement WHERE company='infosys');

Output:

SNAME	DEPARTMENT
eshwar	CSE
reema	IT

Query:

SELECT studentID, company, salary FROM placement WHERE SALARY < SOME (23000, 28000);

Output:

STUDENTID	COMPANY	SALARY

104	infosys	25000
105	Wipro	22000
102	infosys	25000
103	infosys	25000

Query:

SELECT studentID, company, salary FROM placement WHERE salary > SOME(SELECT salary FROM placement WHERE company='infosys');

Output:

no data found

Query:

SELECT studentID, company, salary FROM placement WHERE salary < ALL(23000,28000);

Output:

STUDENTID	COMPANY	SALARY
105	Wipro	22000

Query:

SELECT studentID, company, salary FROM placement WHERE salary >
ALL(SELECT salary FROM placement WHERE company='infosys');

Output:

no data found

// EXISTS

Query:

SELECT * FROM placement WHERE EXISTS (
SELECT * FROM placement WHERE salary>20000);

Output:

PLACEMENTID	STUDENTID	DEPARTMENT	COMPANY	SALARY
1	104	IT	infosys	25000
2	105	CSE	Wipro	22000
4	102	IT	infosys	25000

5	103	CSE	infosys	25000

Query:

SELECT * FROM employee1 WHERE EXISTS
(SELECT * FROM employee1 WHERE salary>20000);

Output:

EMPNO	EMPNAME	SALARY	DESIGNATION
1	Rupesh	50000	MD
2	Bala	45000	Manager
3	Sai	30000	Asst. Manager

// NOT EXISTS

Query:

SELECT * FROM employee1 WHERE
NOT EXISTS (SELECT * FROM employee1 WHERE salary>20000);

Output:

no data found

Result:

Thus all the above SQL nested queries has been executed successfully and the output was verified.

Ex. No: 7 **INDEXES**

Date :

AIM:

To write SQL queries using indexes.

DEFINITION:

An **index** is a performance-tuning method of allowing faster retrieval of records. An index creates an entry for each value that appears in the indexed columns.

SYNTAX:

Create index:

CREATE INDEX idx_name ON table_name(attribute);

Drop Index:

DROP INDEX index_name;

QUERIES:

Query:

create table student(sid number, sname varchar2(10), department varchar2(5));

Output:

Table created.

Query:

insert into student values(1, 'bala', 'IT');
insert into student values(2, 'rupesh', 'IT');
insert into student values(3, 'sai', 'CSE');
insert into student values(4, 'art', 'CSE');

Output:

4 row(s) inserted

Query:

select * from student;

Output:

SID	SNAME	DEPARTMENT
1	bala	IT
2	rupesh	IT
3	sai	CSE

4	art	CSE

Query:

create index idx_stud on student(sid);

Output:

index created.

Query:

drop index idx_stud;

Output:

index dropped.

RESULT

Thus the SQL queries to create an index was successfully executed and verified.

Ex. No: 8 **SYNONYM**

Date:

AIM:

To write SQL queries using synonym.

DEFINITION:

A synonym is an alternative name for objects such as tables, views, sequences, stored procedures, and other database objects.

SYNTAX

Create Synonym:

Create synonym syn_name for table_name;

Delete Synonym:

Drop synonym syn_name;

QUERIES:

Query:

create synonym stud_syn for student;

Output:

synonym created.

Query:

select * from stud_syn;

Output:

SID	SNAME	DEPARTMENT
1	bala	IT
2	rupesh	IT
3	sai	CSE
4	art	CSE

Query:

insert into stud_syn values(4,'harish','CSE');

Output:

1 row(s) inserted.

Query:

select * from stud_syn;

Output:

SID	SNAME	DEPARTMENT
1	bala	IT
2	rupesh	IT
3	sai	CSE
4	art	CSE
5	harish	CSE

Query:

drop synonym stud_syn;

Output:

synonym dropped.

RESULT:

Thus the SQL queries using synonymswas successfully executed and verified.

Ex. No: 9 **VIEW**

Date:

AIM:

To write SQL queries using views.

DEFINITION:

A **view**, is a logical table based on one or more tables or views. A view contains no data itself. The tables upon which a view is based are called **base tables**.

SYNTAX:

CREATE VIEW view_name

AS<Query Expression>

QUERIES:

Query:

create view studview as select * from student where department='IT';

Output:

view created.

Query:

select * from studview;

Output:

SID	SNAME	DEPARTMENT
1	bala	IT
2	rupesh	IT

Query:

insert into studview values(7,'art','IT');

Output:

1 row(s) inserted.

Query:

select * from studview;

Output:

SID	SNAME	DEPARTMENT

1	bala	IT
2	rupesh	IT
7	art	IT

Query:

update studview set sid=10 where sid=7;

Output:

1 row(s) updated.

Query:

select * from studview;

Output:

SID	SNAME	DEPARTMENT
1	bala	IT
2	rupesh	IT
10	art	IT

Query:

delete from studview where sid = 10;

Output:

1 row(s) deleted

Query:

select * from studview;

Output:

SID	SNAME	DEPARTMENT
1	bala	IT
2	rupesh	IT

Query:

drop view studview;

Output:

view dropped.

RESULT:

Thus the SQL queries using views were successfully executed and verified.

Ex. No: 10 **SEQUENCES**

Date:

AIM

To write SQL queries using sequences.

DEFINITION:

A**sequence**, is a database object from which multiple users may generate unique integers. Sequences can be used to automatically generate primary key values. When a sequence number is generated, the sequence is incremented, independent of the transaction committing or rolling back.

SYNTAX:

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

- [SEQUENCE NAME] – The Name of the sequence.
- [START WITH n] – This clause specifies the starting point of the sequence.
- [INCREMENT BY n] – This clause specifies the incremental difference between the two numbers.
- [MAXVALUE n] – This clause specifies the maximum value attainable by the sequence
- [NOMAXVALUE] – This clause is the default specification for Maximum value of the sequence.
- [MINVALUE n] – The clause specifies the minimum value of a sequence.
- [NOMINVALUE] – This clause is the default specification of Minimum value of the sequence.
- [CYCLE | NOCYCLE] – This clause adds cyclic behavior to a sequence. The sequence recycles the values once the MAXVALUE is reached. It is least relevant if sequence is used as Primary key generator. NOCYCLE is the default behavior.

- [CACHE n| NOCACHE] – This clause directs Oracle server to cache 'n' values of a sequence. Its value is 20 by default. NOCACHE is the default behavior of the sequence.
- [{ORDER n | NOORDER}] – This clause ensures that sequence values are generated in order. Generally, it is least usable clause in sequence definition but they are helpful in applications which use sequence as timestamp value. All sessions using same sequence share the same cache to fetch the values. NOORDER is the default behavior of the sequence.

QUERIES:

Query:

```
CREATE SEQUENCE seq_num
increment by 2
start with 11
minvalue 1
maxvalue 20
cache 20;
```

Output:

sequence created.

Query:

```
insert into student values(seq_num.nextval, 'Bala','IT');
```

Output:

1 row(s) inserted.

Query:

```
select * from student;
```

Output:

SID	SNAME	DEPARTMENT
1	bala	IT
2	rupesh	IT
3	sai	CSE
4	art	CSE
5	harish	CSE
11	Bala	IT

Query:

```
drop sequence seq_num;
```

Output:

sequence dropped.

RESULT:

Thus the SQL queries using sequences were successfully executed and verified.

Ex. No: 11 **SAVEPOINT**

Date:

AIM

To write SQL queries using savepoint.

DEFINITION:

- SAVEPOINT statement is to identify a point in a transaction to which you can later roll back.
- With the ROLLBACK TO statement, savepoints undo parts of a transaction instead of the whole transaction.
- The RELEASE SAVEPOINT statement removes the named savepoint from the set of savepoints of the current transaction.

SYNTAX:

```
SAVEPOINT identifier
ROLLBACK [WORK] TO [SAVEPOINT] identifier
RELEASE SAVEPOINT identifier
```

QUERIES:

Query:

SAVEPOINT s;

Output:

Savepoint created.

Query:

ROLLBACK TO s;

Query:

select * from emp;

RESULT

Thus the SQL queries using savepoints were successfully executed and verified.

Ex. No. 12 **EMPLOYEE DATABASE**

Date:

AIM

To create an employee database using various constraints .

INTRODUCTION:

SQL CREATE DATABASE

The CREATE DATABASE statement is used to create a new SQL database.

Syntax

create database databasename*;*

Example

create database empdb;

SQL CONSTRAINTS

- SQL constraints are used to specify rules for the data in a table.
- Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table.
- If there is any violation between the constraint and the data action, the action is aborted.
- Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.
- **Constraint types:**
  - **NOT NULL** - Ensures that a column cannot have a NULL value
  - **UNIQUE** - Ensures that all values in a column are different
  - **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
  - **FOREIGN KEY** - Uniquely identifies a row/record in another table
  - **CHECK** - Ensures that all values in a column satisfies a specific condition
  - **DEFAULT** - Sets a default value for a column when no value is specified

EXAMPLE 1:

Create employee table named EMP with the following fields, emp_id, emp_name, emp_designation and emp_mobile. Add constraint to the

emp_mobile such that, the value should not exceed 10 digits. And also add constraint to the emp_name such that, it should not accept numerical value.

Query:

```
CREATE TABLE emp1
(
        emp_id NUMBER PRIMARY KEY,
        emp_name CHAR(30) NOT NULL,
        emp_designation CHAR(5) DEFAULT 'AP' ,
        emp_mobile NUMBER CHECK(LENGTH(emp_mobile)=10)
);
```

Output:

Table created.

EXAMPLE 2:

Departments (dept _no , dept_ name , empid, dept_location)

Query:

```
CREATE TABLE dept
(
        dept_no NUMBER PRIMARY KEY,
        dept_name CHAR(30) NOT NULL,
        empid NUMBER REFERENCES emp1(emp_id),
        dept_location CHAR(50) DEFAULT 'chennai'
);
```

Output:

Table created.

RESULT

Thus the SQL queries for creating employee database using constraints were successfully executed and verified.

Ex.No.13 **STUDY OF PL/SQL**

Date:

AIM

To study the basics of PL/SQL.

INTRODUCTION:

The PL/SQL programming language was developed by Oracle Corporation in the late 1980s as procedural extension language for SQL and the Oracle relational database. Following are notable facts about PL/SQL:

- PL/SQL is a completely portable, high-performance transaction-processing language.
- PL/SQL provides a built-in interpreted and OS independent programming environment.
- PL/SQL can also directly be called from the command-line SQL*Plus interface.
- Direct call can also be made from external programming language calls to database.
- PL/SQL's general syntax is based on that of ADA and Pascal programming language.
- Apart from Oracle, PL/SQL is available in TimesTen in-memory database and IBM DB2.

PL/SQL - Basic Syntax

PL/SQL is a block-structured language, meaning that PL/SQL programs are divided and written in logical blocks of code.

Each block consists of three sub-parts:

1 Declarations

This section starts with the keyword DECLARE. It is an optional section and defines all variables, cursors, subprograms, and other elements to be used in the program.

2 Executable Commands

This section is enclosed between the keywords BEGIN and END and it is a mandatory section. It consists of the executable PL/SQL statements of the program. It should have at least one executable line of code, which may be just a NULL command to indicate that nothing should be executed.

3 Exception Handling

This section starts with the keyword EXCEPTION. This section is again optional and contains exception(s) that handle errors in the program.

Every PL/SQL statement ends with a semicolon (;). PL/SQL blocks can be nested within other PL/SQL blocks using BEGIN and END.

PL/SQL Basic Structure:

```
DECLARE
    <declarations section>
BEGIN
    <executable command(s)>
EXCEPTION
    <exception handling>
END;
```

Example:

```
DECLARE
    message  varchar2(20):= 'Hello, World!';
BEGIN
    dbms_output.put_line(message);
END;
/
```

Output:

```
Hello World
PL/SQL procedure successfully completed.
```

The end; line signals the end of the PL/SQL block. To run the code from SQL command line, you may need to type / at the beginning of the first blank line after the last line of the code. When the above code is executed at SQL prompt, it produces following result:

PL/SQL - Data Types

Category	Description
Scalar	Single values with no internal components, such as a NUMBER, DATE, or BOOLEAN.

Large Object (LOB)	Pointers to large objects that are stored separately from other data items, such as text, graphic images, video clips, and sound waveforms.
Composite	Data items that have internal components that can be accessed individually. For example, collections and records.
Reference	Pointers to other data items.

PL/SQL Scalar Data Types and Subtypes

Date Type	Description
Numeric	Numeric values on which arithmetic operations are performed.
Character	Alphanumeric values that represent single characters or strings of characters.
Boolean	Logical values on which logical operations are performed.
Datetime	Dates and times.

PL/SQL Numeric Data Types and Subtypes

Data Type	Description
PLS_INTEGER	Signed integer in range -2,147,483,648 through 2,147,483,647, represented in 32 bits
BINARY_INTEGER	Signed integer in range -2,147,483,648 through 2,147,483,647, represented in 32 bits
BINARY_FLOAT	Single-precision IEEE 754-format floating-point number
BINARY_DOUBLE	Double-precision IEEE 754-format floating-point number
NUMBER(prec, scale)	Fixed-point or floating-point number with absolute value in range 1E-130 to (but not including) 1.0E126. A NUMBER variable can also represent 0.

DEC(prec, scale)	ANSI specific fixed-point type with maximum precision of 38 decimal digits.
DECIMAL(prec, scale)	IBM specific fixed-point type with maximum precision of 38 decimal digits.
NUMERIC(pre, secale)	Floating type with maximum precision of 38 decimal digits.
DOUBLE PRECISION	ANSI specific floating-point type with maximum precision of 126 binary digits (approximately 38 decimal digits)
FLOAT	ANSI and IBM specific floating-point type with maximum precision of 126 binary digits (approximately 38 decimal digits)
INT	ANSI specific integer type with maximum precision of 38 decimal digits
INTEGER	ANSI and IBM specific integer type with maximum precision of 38 decimal digits
SMALLINT	ANSI and IBM specific integer type with maximum precision of 38 decimal digits
REAL	Floating-point type with maximum precision of 63 binary digits (approximately 18 decimal digits)

PL/SQL Character Data Types and Subtypes

Data Type	Description
CHAR	Fixed-length character string with maximum size of 32,767 bytes
VARCHAR2	Variable-length character string with maximum size of 32,767 bytes
RAW	Variable-length binary or byte string with maximum size of 32,767 bytes, not interpreted

	by PL/SQL
NCHAR	Fixed-length national character string with maximum size of 32,767 bytes
NVARCHAR2	Variable-length national character string with maximum size of 32,767 bytes
LONG	Variable-length character string with maximum size of 32,760 bytes
LONG RAW	Variable-length binary or byte string with maximum size of 32,760 bytes, not interpreted by PL/SQL
ROWID	Physical row identifier, the address of a row in an ordinary table
UROWID	Universal row identifier (physical, logical, or foreign row identifier)

PL/SQL Boolean Data Types

The BOOLEAN data type stores logical values that are used in logical operations. The logical values are the Boolean values TRUE and FALSE and the value NULL.

PL/SQL Datetime and Interval Types

Field Name	Valid Datetime Values	Valid Interval Values
YEAR	-4712 to 9999 (excluding year 0)	Any nonzero integer
MONTH	01 to 12	0 to 11
DAY	01 to 31 (limited by the values of MONTH and YEAR, according to the rules of the calendar for the locale)	Any nonzero integer
HOUR	00 to 23	0 to 23
MINUTE	00 to 59	0 to 59
SECOND	00 to 59.9(n), where 9(n) is the precision of time fractional seconds	0 to 59.9(n), where 9(n) is the precision of interval

		fractional seconds
TIMEZONE_ HOUR	-12 to 14 (range accommodates daylight savings time changes)	Not applicable
TIMEZONE_ MINUTE	00 to 59	Not applicable
TIMEZONE_ REGION	Found in the dynamic performance view V$TIMEZONE_NAMES	Not applicable
TIMEZONE_ ABBR	Found in the dynamic performance view V$TIMEZONE_NAMES	Not applicable

PL/SQL Large Object (LOB) Data Types

Data Type	Description	Size
BFILE	Used to store large binary objects in operating system files outside the database.	System-dependent. Cannot exceed 4 gigabytes (GB).
BLOB	Used to store large binary objects in the database.	8 to 128 terabytes (TB)
CLOB	Used to store large blocks of character data in the database.	8 to 128 TB
NCLOB	Used to store large blocks of NCHAR data in the database.	8 to 128 TB

RESULT: Thus PL/SQL was successfully studied.

Ex:No:14 **SIMPLE PL / SQL PROGRAMS**

AIM:

To write basic programs and SQL queries using PL/SQL.

1.SWAPPING OF TWO NUMBERS

```
DECLARE
        a number(2):=:a;
        b number(2):=:b;
        temp number(2);
BEGIN
        temp:=a;
        a:=b;
        b:=temp;
        dbms_output.put_line('a '||'= '||a||' and '||''||'b '||'= '||b);
END;
```

Output:

A = 20 AND B = 10

Statement processed.

2.FACTORIAL NUMBER

```
DECLARE
        n number;
        i number;
        fact number := 1;
BEGIN
        n := :n;
        i := 1;
        while i<=n loop
                fact := fact*i;
                I :=i+1;
            end loop;
```

```
        dbms_output.put_line('Factorial of '||n||' is '||fact);
    END;
```

Output:

Factorial of 5 is 120

Statement processed.

3. REVERSE NUMBER

```
DECLARE
    n number(5):=:n;
    rev number(5):=0;
    r number(5):=0;
BEGIN
    while n !=0
    loop
        r:=mod(n,10);
        rev:=rev*10+r;
        n:=trunc(n/10);
    end loop;
dbms_output.put_line('the reverse of a given number is '||rev);
END;
```

Output:

---

the reverse of a given number is 321

Statement processed.

4.PRIME NUMBER

```
DECLARE
        x number := 1;
        flag number := 0;
        n number;
        r number;
```

```
BEGIn
        x := :x;
        n := 2;
        while n<=x-1 loop
            r := mod(x,n);
            if r=0 then
         flag := 1;
            end if;
            n := n+1;
        end loop;
        if flag=0 then
            dbms_output.put_line('number is prime');
        else
            dbms_output.put_line('number  is not prime');
        end if;
END;
```

Output 1:

---

number is prime

Statement processed.

Output 2:

---

number is not prime

Statement processed.

5. LARGEST OF THREE NUMBERS

```
DECLARE
        a number;
        b number;
        c number;
```

```
BEGIN
        a:=:a;
        b:=:b;
        c:=:c;
        if  (a>b and a>c) then
                dbms_output.put_line('a is maximum ' || a);
        elsif (b>a and b>c) then
                dbms_output.put_line('b is maximum ' || b);
        else
                dbms_output.put_line('c is maximum ' || c);
        end if;
END;
```

Output:

---

c is maximum 70

Statement processed.

6.PALINDROME

```
DECLARE
        s varchar2(10):=:s;
        l varchar2(20);
        temp varchar2(10);
BEGIN
        for i in reverse 1..length(s)
                loop
                        l:=substr(s,i,1);
                        temp:=temp||''||l;
                end loop;
        if temp=s then
                dbms_output.put_line(temp ||''||' is palindrome');
        else
                dbms_output.put_line(temp ||''||' is not palindrome');
        end if;
END;
```

Output 1:

madam is palindrome

statement processed.

Output 2:

mamah is not palindrome

statement processed.

7.FIBONACCI SERIES

```
DECLARE
        n number := :n;
        t1 number := 0;
        t2 number := 1;
        t3 number;
BEGIN
        dbms_output.put_line(t1);
       dbms_output.put_line(t2);
       for i in 3..n loop
              t3 := t1 + t2;
              dbms_output.put_line(t3);
              t1 := t2;
              t2 := t3;
       end loop;
END;
```

Output:

0 1 1 2 3

Statement processed.

8. WRITE A PL/SQL PROGRAM FOR FINDING THE TOTAL, AVERAGE AND GRADE OF A STUDENT FOR SIX SUBJECTS.

```
DECLARE
        java   number(10);
        dbms          number(10);
        evs    number(10);
        maths number(10);
        pds    number(10);
        ca     number(10);
        total  number(10);
        avg    number(10);
        per    number(10);
BEGIN
        java   :=   :java;
        dbms :=   :dbms;
        evs    :=   :evs;
        maths:=    :maths;
        pds    :=   :pds;
        ca     :=   :ca;
        total  :=   (java + dbms + evs + maths + pds + ca);
        per    :=   (total / 600) * 100;
        if java<40 or dbms<40 or evs<40 or maths<40 or pds<40 or ca<40
then
                dbms_output.put_line('FAIL');
        elsif per>75 then
                dbms_output.put_line('GRADE A');
        elsif per>65 and per<75 then
                dbms_output.put_line('GRADE B');
        elsif per>55 and per<65 then
                dbms_output.put_line('GRADE C');
        else
                dbms_output.put_line('INVALID INPUT');
        end if;
        dbms_output.put_line('PERCENTAGE IS '||per);
END;
```

Output 1:

GRADE A
PERCENTAGE IS 86

Statement processed.

Output 2:

FAIL
PERCENTAGE IS 65

Statement processed.

9.WRITE A PL/SQL PROGRAM FOR FINDING THE SUM OF DIGITS IN GIVEN NUMBER

```
DECLARE
    n integer(10):=:n;
    temp1 integer(10);
    temp2 integer(10):=0;
    temp3 integer(10);
BEGIN
    temp3:=n;
    while n> 0
    loop
        temp1:=n mod 10;
        temp2:=temp2+temp1;
        n:=floor(n/10);
    end loop;
```

```
        dbms_output.put_line('The sum of digits of '||temp3||' is
'||temp2);
END;
```

Output:

The sum of digits of 123 is 6

Statement processed.

10.WRITE A PL/SQL PROGRAM TO FIND THE LARGEST OF TWO NUMBERS.

```
DECLARE
        n number;
        m number;
BEGIN
        n  := :n;
        m  := :m;
        if n<m then
                dbms_output.put_line(''|| m ||' is greater than '|| n ||'');
        else
                dbms_output.put_line(''|| n ||' is greater than '|| m ||'');
        end if;
END;
```

Output:

25 is greater than 10
Statement processed.

11. WRITE PL/SQL PROGRAM TO GENERATE EVEN NUMBERS.

Query:

```
DECLARE
        NUM1 number:=0;
```

```
BEGIN
    loop
        NUM1 := NUM1+2;
        dbms_output.put_line (NUM1||',');
        exit when NUM1=20;
    end loop;
END;
```

Output:

2,4,6,8,10,12,14,16,18,20,

Statement processed.

12. WRITE A PL/SQL CODE BLOCK TO CALCULATE THE AREA & CIRCUMFERENCE OF A CIRCLE

Query:

```
DECLARE
    radius float;
    area float;
    circum float;
BEGIN
    radius:=:radius;
    area:=3.14*radius*radius;
    circum := 2*3.14*radius;
    dbms_output.put_line('The area of circle is '||area);
    dbms_output.put_line('The circumference of circle is '||circum);
END;
```

OUTPUT:

The area of circle is 314
The circumference of circle is 62.8
Statement processed.

SQL QUERY RESULTS TO PL/SQL VARIABLES

Query:

```
CREATE TABLE employee
(
```

```
        eid INT NOT NULL,
        ename VARCHAR(10) NOT NULL,
        age INT NOT NULL,
        sal NUMBER
    );
```

Output:

Table created.

Query:

```
insert into employee values(1, 'Rupesh', 35, 30000);
```

Output:

1 row(s) inserted.

Query:

```
insert into employee values(2, 'Bala', 36, 25000);
```

Output:

1 row(s) inserted.

Query:

```
insert into employee values(3, 'Sai', 32, 24000);
```

Output:

1 row(s) inserted.

Query:

```
Select * from employee;
```

Output:

EID	ENAME	AGE	SAL
1	Rupesh	35	30000
2	Bala	36	25000
3	Sai	32	24000

PL/SQL Query:

```
DECLARE
    name VARCHAR(10);
    salary NUMBER;
```

```
BEGIN
    SELECT ename, sal INTO name, salary FROM employee
    WHERE eid = 2;
    dbms_output.put_line('Customer : ' ||name);
    dbms_output.put_line('Salary : ' ||salary);
END;
```

Output:

```
Customer : Bala
Salary : 25000

Statement processed.
```

RESULT:

Thus the basic PL/SQL statement and SQL queries using PL/SQL were successfully executed and verified.

Ex:No:15 **PL/SQL BLOCK THAT HANDLES EXCEPTIONS**

Date:

AIM:

To write PL/SQL program to handle exceptions.

PROGRAM - DIVIDE BY ZERO EXCEPTION

PL / SQL Query:

```
DECLARE
        a NUMBER;
        b NUMBER;
        c NUMBER;
BEGIN
        a:=:a;
        b:=:b;
        c:=a/b;
        dbms_output.put_line('Result : '||c);
EXCEPTION
    WHEN ZERO_DIVIDE THEN
    RAISE_APPLICATION_ERROR(-20003,'Invalid Input');
  END;  /
```

Output1:

Enter value for a: 10

Enter value for b: 5

Result : 2

Statement Processed.

Output2:

Enter value for a: 0

Enter value for b: 10

Result : 0

Statement Processed.

Output 3:

Enter value for a: 10

Enter value for b: 0

Invalid Input

Statement Processed.

PROGRAM-2: DUPLICATE VALUE EXCEPTION

TABLE CREATION:

Query:

```
CREATE TABLE employee
(
        eid INT PRIMARY KEY,
        ename VARCHAR(10),
        age INT,
        sal NUMBER
);
```

Output:

Table created.

PL / SQL Query:

```
DECLARE
        eno NUMBER;
        name VARCHAR2(10);
        age NUMBER;
        sal NUMBER;
BEGIN
        eno:=:eno;
        name:=:name;
        age:=:age;
        sal:=:sal;

        INSERT INTO employee VALUES(eno,name,age,sal);
        dbms_output.put_line('Inserted');

        EXCEPTION
                WHEN DUP_VAL_ON_INDEX THEN
                        dbms_output.put_line('Value already exists');
END;
```

Input 1:

Submit

:ENO 1

:NAME KIRTHEESH

:AGE 20

:SAL 45000

Output 1

Inserted

Statement processed.

Input 2:

Submit

:ENO 1

:NAME RUPESH

:AGE 25

:SAL 50000

Output 2

Inserted

Statement processed.

Input 3:

Submit

:ENO 2

:NAME BALA

:AGE 20

:SAL 45000

Output 3

Inserted

Statement processed.

Query:

Select * from employee;

Output:

EID	ENAME	AGE	SAL
1	BALA	25	50000
2	RUPESH	25	45000
3	SAI	20	45000

Input 4:

Submit

:ENO 3

:NAME SAI

:AGE 20

:SAL 45000

Output 3

Value already exists

Statement processed.

RESULT:

Thus the PL/SQL statement that handles exceptions were successfully executed and verified.

Ex:No: 16 **TRIGGERS**

Date:

AIM:

To write queries to execute triggers in SQL.

TRIGGER:

A trigger is a special kind of stored procedure that automatically executes when an event occurs in the database server. DML triggers execute when a

user tries to modify data through a data manipulation language (DML) event. DML events are INSERT, UPDATE, or DELETE statements on a table or view.

SYNTAX:

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

EMPLOYEE SALARY LOG

Query:

```
CREATE TABLE employ
(
    eid NUMBER,
    ename VARCHAR2(10),
    dept VARCHAR2(10),
    salary NUMBER
);
```

Output:

Table Created

Query:

Insert into employ values(101, 'John', 'IT', 25000);

Output:

1 row(s) inserted

Query:

Insert into employ values(102, 'Jeny', 'IT', 50000);

Output:

1 row(s) inserted

Query:

Insert into employ values(103, 'Joseph', 'CSE', 40000);

Output:

1 row(s) inserted

Query:

Insert into employ values(104, 'Rohit', 'IT', 55000);

Output:

1 row(s) inserted

Query:

select * from employ;

Output:

EID	ENAME	DEPT	SALARY
103	Joseph	CSE	40000
104	Rohit	IT	55000
101	John	IT	25000
102	Jeny	IT	50000

Query:

```
CREATE TABLE emplog
(
        eid NUMBER,
        ename VARCHAR2(10),
        idate DATE,
        nsal NUMBER
);
```

Output:

Table Created

TriggerQuery:

```
CREATE OR REPLACE TRIGGER logsal
        AFTER UPDATE OF salary ON employ
FOR EACH ROW
BEGIN
        INSERT INTO emplog(eid,ename,idate,nsal)
        VALUES(:NEW.eid,:NEW.ename,SYSDATE,:NEW.salary);
END;
```

Output:

Trigger created.

Query:

```
UPDATE employ SET salary = salary+5000 WHERE dept='IT';
```

Output:

3 rows updated.

Query:

```
select * from employ;
```

Output:

EID	ENAME	DEPT	SALARY
103	Joseph	CSE	40000
104	Rohit	IT	60000
101	John	IT	30000
102	Jeny	IT	55000

Query:

```
select * from emplog;
```

Output:

EID	ENAME	IDATE	NSAL
104	Rohit	08/20/2016	60000
101	John	08/20/2016	30000
102	Jeny	08/20/2016	55000

RESULT:

Thus the SQL queries to create triggers were successfully executed and verified.

Ex. No. 17 **FUNCTIONS AND PROCEDURES**

Date:

AIM:

To write PL/SQL queries to implement functions and procedures.

SYNTAX

```
CREATE OR REPLACE FUNCTION FUNCTION_NAME(ARGUMENTS)
RETURN DATATYPE IS
BEGIN
      ------
      ------
END;
```

EXAMPLE 1 :

Write a PL/SQL function called POW that takes two numbers as argument and return the value of the first number raised to the power of the second.

FUNCTION CREATION:

Query:

```
create or replace function pow (n1 number, n2 number)
return number
is
res number;
begin
select power( n1, n2) into res from dual;
return res;
end;
```

Qutput:

Function created.

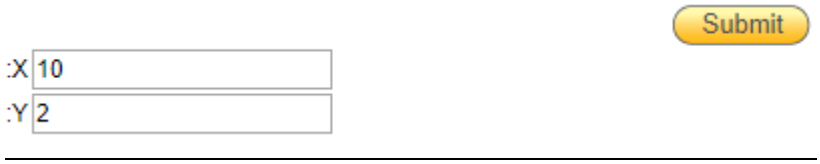
FUNCTION CALL:

Query:

```
DECLARE
```

```
        x   NUMBER;
        y  NUMBER;
ans number;
BEGIN
x := :x;
 y := :y;
ans := pow(x, y);
dbms_output.put_line('Power  :'||ans);
END;
```

OUTPUT:




```
Power  :100
Statement processed.
```

EXAMPLE 2 :

Implement a PL/SQL function ODDEVEN to return value TRUE if the numberpassed to it is EVEN else will return FALSE.

FUNCTION CREATION:

Query:

```
create or replace function oddeven (x in number) return boolean
is
begin
        if mod (x, 2)=0 then
                return true;
        else
                return false;
        end if;
end;
```

Qutput:

Function created.

FUNCTION CALL:

Query:

```
DECLARE
        num  NUMBER;
        ans   boolean;
BEGIN
        num:= :num;
        ans:= ODDEVEN(num);
 if ans = true then
                dbms_output.put_line('number is even');
 else
dbms_output.put_line('number is odd');
end if;
END;
```

OUTPUT 1:

number is even

Statement processed.

OUTPUT 2:

number is odd

Statement processed.

PROCEDURE

TABLE CREATION

```
CREATE TABLE stock
(
        stockid  NUMBER,
        productname  VARCHAR2(15),
        stockavail  VARCHAR2(15)
);
```

//Select

PROCEDURE CREATION STOCK_DETAILS:

```
CREATE or REPLACE PROCEDURE stock_details(x  NUMBER)  AS
sid   VARCHAR2(15);
sname   VARCHAR2(15);
BEGIN
        SELECT stockavail INTO sidFROM stock WHERE stockid=x;
        SELECT productname INTO snameFROM stock WHERE stockid=x;
        dbms_output.put_line('Stock Name  : '|| sname);
        dbms_output.put_line('Stock  :'||sid);
END;
/
```

OUTPUT:

```
EXEC stock_details(102);

Stock  Name  : Mustered Oil
Stock  :  52  lit
 procedure successfully completed.
```

RESULT:

Thus the PL/SQL queries to create functions and procedures were successfully executed and verified.

Ex. No:18 DATABASE DESIGN USING ER MODELING, NORMALIZATION

Date :

AIM:

To design a database Design using ER modelling and normalization to set various constraints.

Simple example of Relational model is as follows :

Student Details Table

Roll_no	Sname	S_Address
1	Rahul	Satelite
2	Sachin	Ambawadi
3	Saurav	Naranpura

Student Marksheet Table

Rollno	Sub1	Sub2	Sub3
1	78	89	94
2	54	65	77
3	23	78	46

Here, both tables are based on students details. Common field in both tables is Rollno. So we
can say both tables are related with each other through Rollno column.

Degree of Relationship

One to One (1:1)
One to Many or Many to One (1:M / M: 1)
Many to Many (M: M)

The Degree of Relationship indicates the link between two entities for a specified occurrence of each.

One to One Relationship: (1:1)

1 1

Student Has Roll No.

One student has only one Rollno. For one occurrence of the first entity, there can be, at the most one related occurrence of the second entity, and vice-versa.

One to Many or Many to One Relationship: (1:M/M: 1)

1 M

Course Contains Students

As per the Institutions Norm, One student can enroll in one course at a time however, in one course, there can be more than one student.

For one occurrence of the first entity there can exist many related occurrences of the second entity and for every occurrence of the second entity there exists only one associated occurrence of the first.

Many to Many Relationship: (M:M)

M M

Students Appears Tests

The major disadvantage of the relational model is that a clear-cut interface cannot be determined.

Reusability of a structure is not possible. The Relational Database now accepted model on which major database system are built.

Oracle has introduced added functionality to this by incorporated object-oriented capabilities.

Now it is known is as Object Relational Database Management System (ORDBMS). Object oriented concept is added in Oracle8.

Some basic rules have to be followed for a DBMS to be relational. They are known as Codd's
rules, designed in such a way that when the database is ready for use it encapsulates the
relational theory to its full potential. **E. F. Codd Rules**

1. The Information Rule

All information must be store in table as data values.

2. The Rule of Guaranteed Access

Every item in a table must be logically addressable with the help of a table name.

3. The Systematic Treatment of Null Values

The RDBMS must be taken care of null values to represent missing or inapplicable information.

4. The Database Description Rule

A description of database is maintained using the same logical structures with which data was defined by the RDBMS.

5. Comprehensive Data Sub Language

According to the rule the system must support data definition, view definition, data manipulation, integrity constraints, authorization and transaction management operations.

6. The View Updating Rule

All views that are theoretically updatable are also updatable by the system.

7. The Insert and Update Rule

This rule indicates that all the data manipulation commands must be operational on sets of rows having a relation rather than on a single row.

8. The Physical Independence Rule

Application programs must remain unimpaired when any changes are made in storage
representation or access methods.

9. The Logical Data Independence Rule

The changes that are made should not affect the user's ability to work with the data.The change can be splitting table into many more tables.

10. The Integrity Independence Rule

The integrity constraints should store in the system catalog or in the database.

11. The Distribution Rule

The system must be access or manipulate the data that is distributed in other systems.

12. The Non-subversion Rule

If a RDBMS supports a lower level language then it should not bypass any integrity constraints defined in the higher level.

ROADWAY TRAVELS

"Roadway Travels" is in business since 1977 with several buses connecting different places in India. Its main office is located in Hyderabad.

The company wants to computerize its operations in the following areas:

- Reservations
- Ticketing
- Cancellations

Reservations :

Reservations are directly handled by booking office. Reservations can be made 60 days in advance in either cash or credit. In case the ticket is not available, a wait listed ticket is issued to the customer. This ticket is confirmed against the cancellation.

Cancellation and modification:

Cancellations are also directly handed at the booking office. Cancellation charges
will be charged.

Wait listed tickets that do not get confirmed are fully refunded.

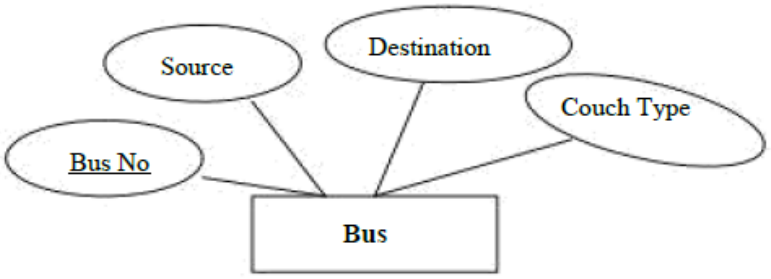
Example: Analyze the problem and come with the entities in it. Identify what Data has to be persisted in the databases.

The Following are the entities:

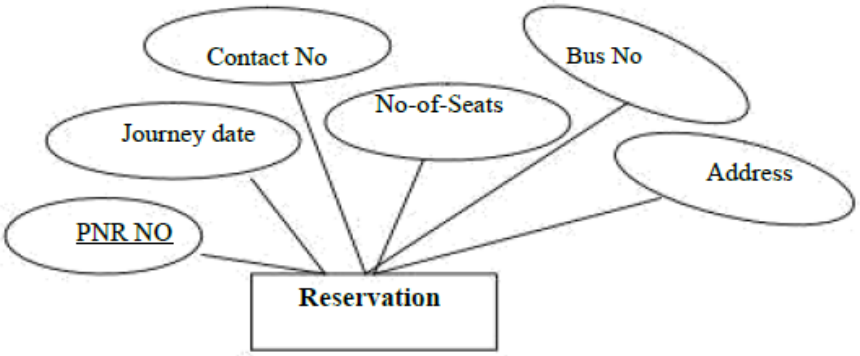
1 .Bus
2. Reservation
3. Ticket
4. Passenger
5. Cancellation

The attributes in the Entities:

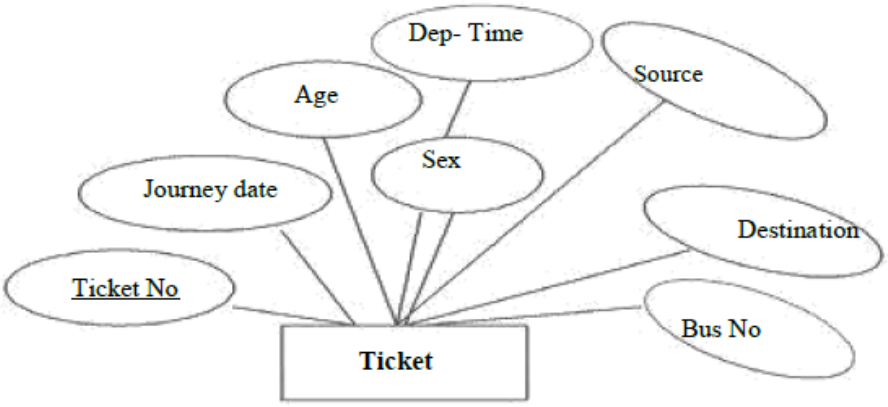
Bus:(Entity)



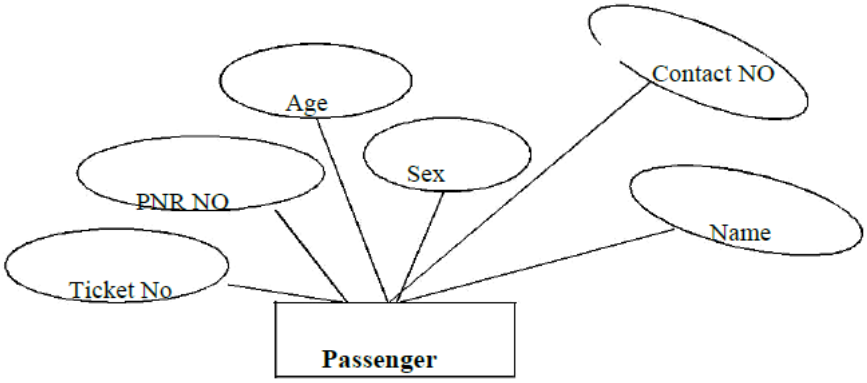
Reservation (Entity)



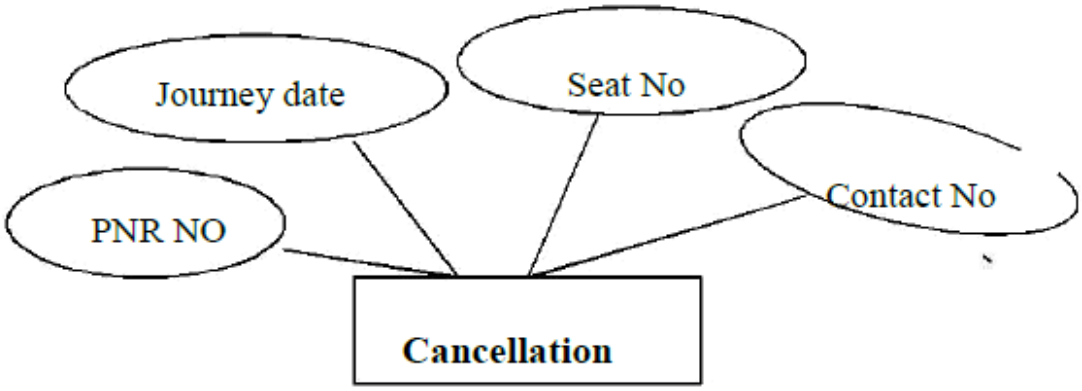
Ticket :(Entity)



Passenger:



Cancellation (Entity):



CONSTRAINTS:

The three types of constraints are Domain Integrity Constraints, Entity Integrity Constraints, and
Referential Integrity Constraints

Integrity Constraints are used to enforce rules that the columns in a table have to conform with. It is a mechanism used by Oracle to preempt invalid data entry into the table.

1. Domain Integrity Constraints

a. Not Null Constraint – The enforcement of Not Null Constraints in a table ensures that the
table contain values.

b. Check Constraint – Allow only a particular range of values

2. Entity Integrity Constraints

a. Unique Constraints – The unique constraint designates a Column or a group of columns as
unique key. This allows only unique value to be stored in the column. Rejects duplication.

b. Primary Key Constraints – Primary key similar to unique key. avoids duplication , relation
between two tables , does not allow not null values.

3. Referential Integrity Constraints

Enforces relationship between tables. It designates a column or group of columns as a foreign key

Sample Output

CONSTRAINTS

NOT NULL, UNIQUE AND CHECK CONSTRINTS

```
SQL> create table employee_master(empcode varchar(5) unique, empname
varchar(7) not null);
Table created.
SQL> desc employee_master;
Name Null? Type
----------------------------------------- -------- ---------
EMPCODE VARCHAR2(5)
EMPNAME NOT NULL VARCHAR2(7)
SQL> alter table employee_master add(productprice number(10)
check(productprice <10000 ) ) ;
Table altered.
SQL> insert into employee_master values('v001','Raj',9100);
1 row created.
```

UNIQUE CONSTRAINT VIOLATION

```
SQL> insert into employee_master values('&empcode','&empname',
&productprice);
Enter value for empcode: e001
Enter value for empname: Philips
Enter value for productprice: 6000
old 1: insert into emp_master values('&empcode','&empname', &productprice)
new 1: insert into employee_master values('e001','Philips',6000)
insert into employee_master values('e001','Philips',6000)
*
ERROR at line 1:
ORA-00001: unique constraint (USER06.SYS_C002807) violated
SQL> ed
Wrote file afiedt.buf
1* insert into employee_master values('&empcode','&empname',
```

```
&productprice)
SQL> /
Enter value for empcode: e002
Enter value for empname: Gabriel
Enter value for productprice: 6000
old 1: insert into employee_master
values('&empcode','&empname',&productprice)
new 1: insert into employee_master values('e002',Richard',6000)
1 row created.
NULL CONSTRAINT VIOLATION
SQL> /
Enter value for empcode: e003
Enter value for empname:
Enter value for productprice: 9000
old 1: insert into employee_master
values('&empcode','&empname',&productprice)
new 1: insert into employee_master values('e003','',9000)
insert into employee_master values('e003','',9000)
*
ERROR at line 1:
ORA-01400: cannot insert NULL into ("USER06"."EMPLOYEE_MASTER".
"EMPNAME")
SQL> /
Enter value for empcode: v003
Enter value for empname: Philips
Enter value for productprice: 8000
old 1: insert into employee_master
values('&empcode','&empname',&productprice)
new 1: insert into employee_master values('e003','Philips',8000)
1 row created.
CHECK CONSTRAINT VIOLATION
SQL> /
Enter value for empcode: e004
Enter value for empname: Akash
Enter value for productprice: 12000
old 1: insert into employee_master
values('&empcode','&empname',&productprice)
new 1: insert into employee_master values('e004','Akash',12000)
insert into employee_master values('e004','Akaah',12000)
*
ERROR at line 1:
ORA-02290: check constraint (USER06.SYS_C002809) violated
SQL> /
Enter value for empcode: e005
Enter value for empname: suresh
Enter value for productprice: 8500
old 1: insert into employee_master values('&empcode','&empname',
&productprice)
new 1: insert into employee_master values('e005','Akash',8500)
1 row created.
PRIMARY AND FOREIGN KEY CONSTRAINTS
SQL> create table order_master(orderno varchar(5) constraint order_prim
primary key,odate date,empcode varchar(5),o_status char(1) not null,deldate
date);
Table created.
```

```
SQL> desc order_master;
Name Null? Type
----------------------------------------- -------- ------------------
ORDERNO NOT NULL VARCHAR2(5)
ODATE DATE
EMPCODE VARCHAR2(5)
O_STATUS NOT NULL CHAR(1)
DELDATE DATE
SQL> create table order_detail(orderno varchar2(5) ,itemcode varchar(3),qtyorder
number(3),foreign key(orderno) references order_master on delete cascade on
update cascade);
Table created.
SQL> desc order_detail;
Name Null? Type
----------------------------------------- -------- ----------------
ORDERNO VARCHAR2(5)
ITEMCODE VARCHAR2(3)
QTYORDER NUMBER(3)
SQL> insert into order_master values('&oredrno','&odate','&empcode',
'&o_status',' &deldate');
Enter value for oredrno: o001
Enter value for odate: 12-aug-2009
Enter value for empcode: v001
Enter value for o_status: p
Enter value for deldate: 12-sep-2009
old 1: insert into order_master
values('&oredrno','&odate','&empcode','&o_status', '&deldate')
new 1: insert into order_master values('o001','12-aug-2009','v001','p',
'12-sep-2009')
1 row created.
PRIMARY KEY CONSTRAINT VIOLATION
SQL> /
Enter value for oredrno: o001
Enter value for odate: 11-nov-2008
Enter value for empcode: v002
Enter value for o_status: p
Enter value for deldate: 14-feb-2010
old 1: insert into order_master
values('&oredrno','&odate','&empcode','&o_status', '&deldate')
new 1: insert into order_master values('o001','11-nov-2008','v002','p',
'14-feb-2010')
insert into order_master values('o001','11-nov-2008','v002','p',
'14-feb-2010')
*
ERROR at line 1:
ORA-00001: unique constraint (USER06.ORDER_PRIM) violated
SQL> /
Enter value for oredrno: o002
Enter value for odate: 13-jan-2007
Enter value for empcode: v002
Enter value for o_status: p
Enter value for deldate: 16-oct-2009
old 1: insert into order_master
values('&oredrno','&odate','&empcode','&o_status', '&deldate')
new 1: insert into order_master values('o002','13-jan-2007','v002','p',
```

```
'16-oct-2009')
1 row created.
SQL> select * from order_master;
ORDER ODATE EMPCO O DELDATE
----- --------- ----- - ---------
o001 11-NOV-08 voo2 p 14-FEB-10
o002 13-JAN-07 v002 p 16-OCT-09
FOREIGN KEY CONSTRAINT VIOLATION
SQL> insert into order_detail values('&orderno','&itemcode',&qtyorder);
Enter value for orderno: o003
Enter value for itemcode: i01
Enter value for qtyorder: 25
old 1: insert into order_detail values('&orderno','&itemcode',&qtyorder)
new 1: insert into order_detail values('o003','i01',25)
insert into order_detail values('o003','i01',25)
*
ERROR at line 1:
ORA-02291: integrity constraint (USER06.SYS_C002814) violated - parent key
not
found
SQL> /
Enter value for orderno: o001
Enter value for itemcode: i01
Enter value for qtyorder: 25
old 1: insert into order_detail values('&orderno','&itemcode',&qtyorder)
new 1: insert into order_detail values('o001','i01',25)
1 row created.
SQL> select * from order_detail;
ORDER ITE QTYORDER
----- --- ----------
o001 i01 25
SQL>delete from order_master where orderno='o001'
*
ERROR at line 1:
ORA-02292: integrity constraint (USER06.SYS_C002814) violated – child record
found
USING on delete cascade with foreign key constraint
SQL> drop table order_master;
Table dropped.
SQL> drop table order_detail;
Table dropped.
SQL> create table order_master(orderno varchar(5) constraint order_prim
primary key, odate date, empcode varchar(5),o_status char(1) not null,deldate
date);
SQL>create table order_detail(orderno varchar2(5) ,itemcode varchar(3), qtyorder
number(3),
foreign key(orderno) references order_master on delete cascade);
Table created.
SQL> insert into order_master values('&oredrno','&odate','&empcode',
'&o_status','&deldate');
Enter value for oredrno: o001
Enter value for odate: 12-jan-2007
Enter value for empcode: voo1
Enter value for o_status: p
Enter value for deldate: 13-jun-2008
```

```
old 1: insert into order_master values('&oredrno','&odate','&empcode',
'&o_status','&deldate')
new 1: insert into order_master values('o001','12-jan-2007','voo1','p',
'13-jun-2008')
1 row created.
SQL> /
Enter value for oredrno: o002
Enter value for odate: 24-sep-2008
Enter value for empcode: voo2
Enter value for o_status: p
Enter value for deldate: 16-jan-2010
old 1: insert into order_master values('&oredrno','&odate','&empcode',
'&o_status','&deldate')
new 1: insert into order_master values('o002','24-sep-2008','voo2','p',
'16-jan-2010')
1 row created.
SQL> select * from order_master;
ORDER ODATE EMPCO O DELDATE
----- --------- ----- - ---------
o001 12-JAN-07 voo1 p 13-JUN-08
o002 24-SEP-08 voo2 p 16-JAN-10
SQL> insert into order_detail values('&orderno','&itemcode',&qtyorder);
Enter value for orderno: o001
Enter value for itemcode: i01
Enter value for qtyorder: 25
old 1: insert into order_detail values('&orderno','&itemcode',&qtyorder)
new 1: insert into order_detail values('o001','i01',25)
1 row created.
SQL> /
Enter value for orderno: o002
Enter value for itemcode: i02
Enter value for qtyorder: 50
old 1: insert into order_detail values('&orderno','&itemcode',&qtyorder)
new 1: insert into order_detail values('o002','i02',50)
1 row created.
SQL> select * from order_detail;
ORDER ITE QTYORDER
----- --- ----------
o001 i01 25
o002 i02 50
SQL> delete from order_master where orderno='o001';
1 row deleted.
SQL> select * from order_master;
ORDER ODATE EMPCO O DELDATE
----- --------- ----- - ---------
o002 24-SEP-08 voo2 p 16-JAN-10
SQL> select * from order_detail;
ORDER ITE QTYORDER
----- --- ----------
o002 i02 50
```

[Note: Deletion of a row in parent is reflected in child table also.]

RESULT:

Thus a database is designed using ER modelling and normalization to set various constraints.

Ex.No:19 DATABASE CONNECTIVITY WITH FRONT END TOOL
Date :

AIM:

To connect a database with front end tool Visual Basic.

PROCEDURE:

1. Create a database for payroll processing which request the using SQL
2. Establish ODBC connection
3. In the administrator tools open data source ODBC
4. Click add button and select oracle in ORA home 90, click finish
5. A window will appear given the data source home as oracle and select TNS source name as
   lion and give the used id as SWTT
6. ADODC CONTROL FOR SALARY FORM:-
7. The above procedure must be follow except the table , A select the table as salary
8. Write appropriate Program in form each from created in VB from each from created in VB
   form project.

SQL>create table emp(eno number primary key,enamr varchar(20),age number,addr varchar(20),DOB date,phno number(10));
Table created.
SQL>create table salary(eno number,edesig varchar(10),basic number,da number,hra number,pf number,mc number,met number,foreign key(eno) references emp);
Table created.

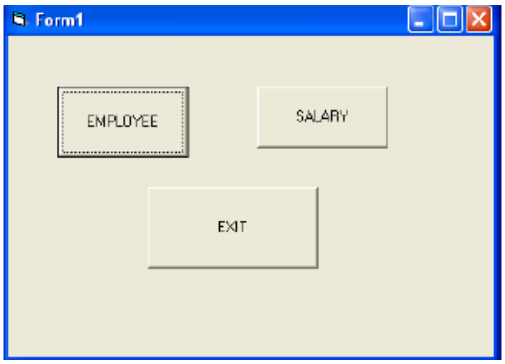
TRIGGER to calculate DA,HRA,PF,MC

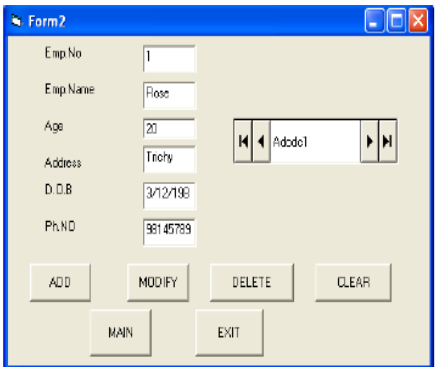
SQL> create or replace trigger employ
2 after insert on salary
3 declare
4 cursor cur is select eno,basic from salary;
5 begin
6 for cur1 in cur loop
7 update salary set
8 hra=basic*0.1,da=basic*0.07,pf=basic*0.05,mc=basic*0.03 where hra=0; 9 end loop;
10 end;
11 / Trigger created.

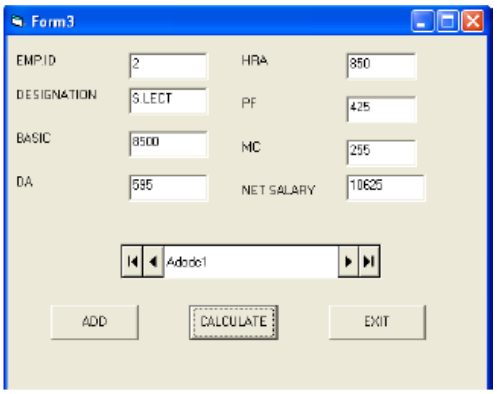
```
PROGRAM FOR FORM 1
Private Sub emp_Click() Form
2.Show
End Sub

Private Sub exit_Click()
Unload Me
End Sub
Private Sub salary_Click()
Form3.Show
End Sub

PROGRAM FOR FORM 2

Private Sub add_Click()
Adodc1.Recordset.AddNew MsgBox "Record added"
End Sub

Private Sub clear_Click()
Text1.Text = ""
Text2.Text = ""
Text3.Text = ""
Text4.Text = ""
Text5.Text = ""
Text6.Text = ""
End Sub

Private Sub delte_Click()
Adodc1.Recordset.Delete MsgBox "Record Deleted"
If Adodc1.Recordset.EOF = True
Then Adodc1.Recordset.MovePrevious
End If
End Sub

Private Sub exit_Click()
Unload Me
End Sub

Private Sub main_Click()
Form1.Show
End Sub

Private Sub modify_Click()
Adodc1.Recordset.Update
End Sub

PROGRAM FOR FORM 3
Private Sub add_Click()
Adodc1.Recordset.AddNew MsgBox "Record added"
End Sub

Private Sub
clear_Click()
Text1.Text = ""
Text2.Text = ""
```

```
Text3.Text = ""
Text4.Text = ""
Text5.Text = ""
Text6.Text = ""
End Sub

Private Sub delte_Click()
Adodc1.Recordset.Delete MsgBox "Record Deleted"
If Adodc1.Recordset.EOF = True
Then Adodc1.Recordset.MovePrevious
End If
End Sub

Private Sub exit_Click()
Unload Me
End Sub

Private Sub main_Click()
Form1.Show
End Sub

Private Sub
modify_Click()
Adodc1.Recordset.Update
End Sub
```

Output:







RESULT:

Thus a database in Oracle is connected with front end tool Visual Basic.

Ex.No:20 INVENTORY CONTROL SYSTEM

Date :

AIM:

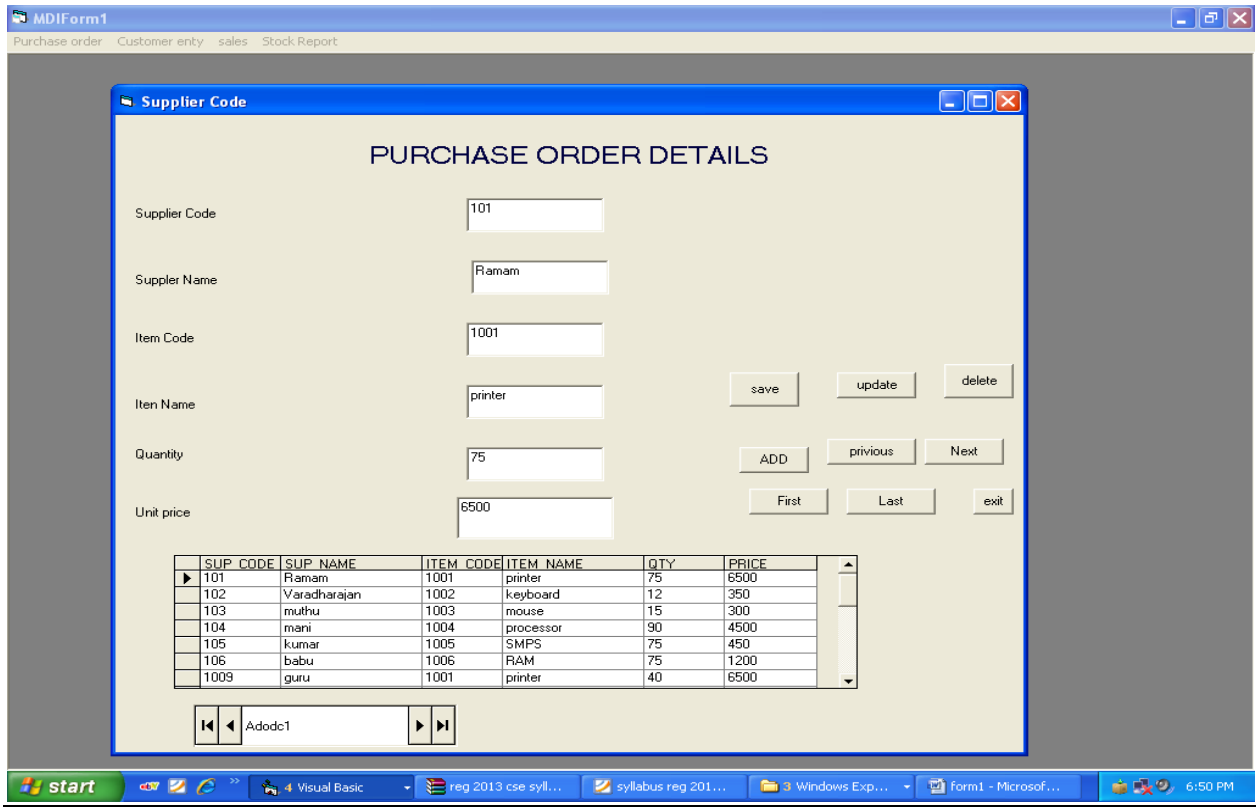
To create a inventory control system using VB as front end and oracle as back end.

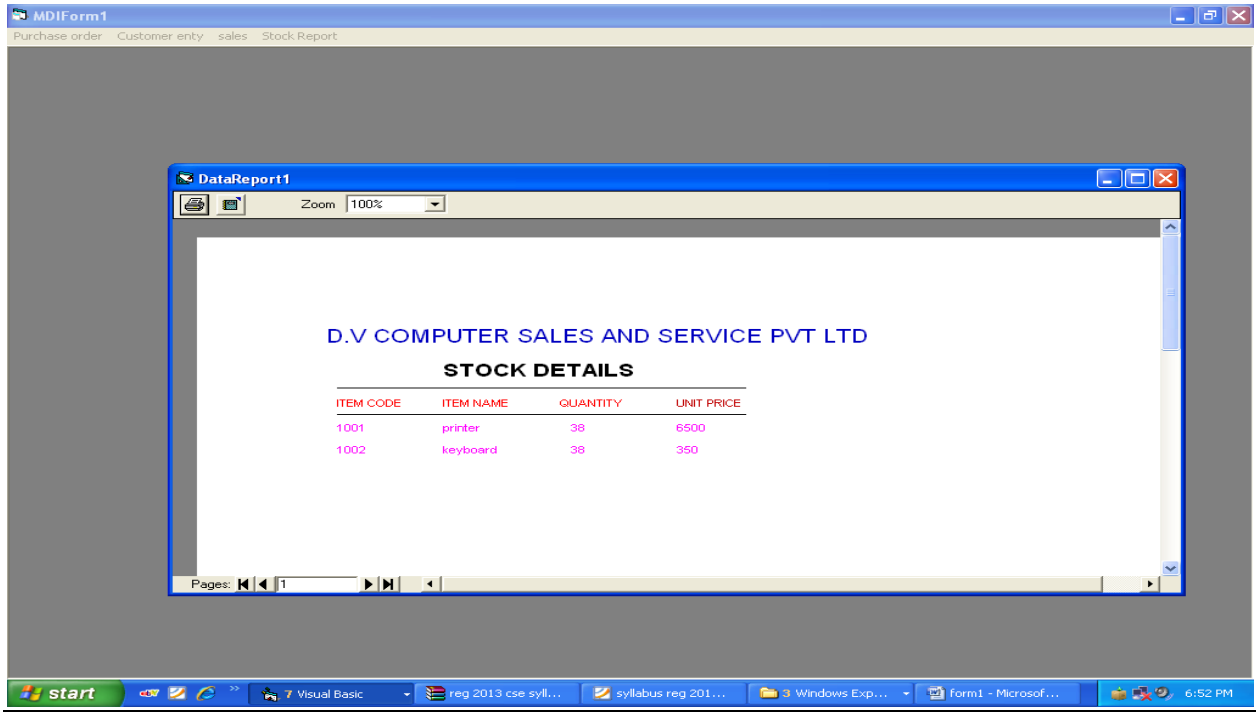
PROGRAM:

```
Dim conn As ADODB.Connection
Dim rs As ADODB.Recordset
Dim sto_cn As ADODB.Connection
Dim sto_rs As ADODB.Recordset

Private Sub add_Click()
Adodc1.Recordset.AddNew
Set DataGrid1.DataSource = Adodc1
DataGrid1.Refresh
End Sub

Private Sub delete_Click()
On Error Resume Next
If MsgBox("Data is not recoverable!", vbExclamation + vbOKCancel, "Confirm
Delete") = vbOK Then
rs.delete
End If
End Sub

Private Sub exit_Click()
If MsgBox("Close Applect?", vbQuestion + vbYesNo, "Confirm") = vbYes Then
   Unload Me
End If
End Sub

Private Sub First_Click()
rs.MoveFirst
Adodc1.Recordset.MoveFirst
Set DataGrid1.DataSource = rs
DataGrid1.Refresh
End Sub

Private Sub Form_Load()
 Set sto_cn = New ADODB.Connection
 sto_cn.Open "cust_dsn", "scott", "tiger"
 Set sto_rs = New ADODB.Recordset
 sto_rs.Open "select * from stock", sto_cn, adOpenDynamic, adLockPessimistic
Set conn = New ADODB.Connection
  conn.Open        "Provider=MSDAORA.1;Password=tiger;User        ID=scott;Data
Source=dvora;Persist Security Info=True"
Set rs = New ADODB.Recordset
   rs.ActiveConnection = conn
   rs.CursorLocation = adUseClient
   rs.CursorType = adOpenDynamic
   rs.LockType = adLockOptimistic
```

```
    rs.Source = "SELECT * FROM pur_order"
    rs.Open
 Set DataGrid1.DataSource = rs
DataGrid1.Refresh
End Sub

Private Sub Last_Click()
rs.MoveLast
Adodc1.Recordset.MoveLast
DataGrid1.Refresh
End Sub

Private Sub NEXT_Click()
If rs.EOF = True Then
    rs.MoveFirst
    Adodc1.Recordset.MoveFirst
     Else
    rs.MoveNext
    Adodc1.Recordset.MoveNext
   End If
DataGrid1.Refresh
End Sub

Private Sub PREVIOUS_Click()
If Adodc1.Recordset.BOF = True Then
    rs.MoveLast
    Adodc1.Recordset.MoveLast
Else
 rs.MovePrevious
Adodc1.Recordset.MovePrevious
End If
DataGrid1.Refresh
End Sub

Private Sub save_Click()
Adodc1.Recordset.update
Do Until sto_rs.EOF
If sto_rs.Fields(0) = Val(Text3.Text) Then
sto_rs.Fields(2) = sto_rs.Fields(2) + Val(Text5.Text)
sto_rs.update
sto_rs.MoveLast
End If
sto_rs.MoveNext
Loop
End Sub
```

OUTPUT





RESULT:

Thus the inventory control system was created successfully.

Ex.No: 21 **MATERIAL REQUIREMENT PROCESSING SYSTEM**

Date :

AIM:

To create a material requirement processing system using VB as front end and oracle as back end.

PROGRAM:

Form1

```
Dim cn As ADODB.Connection
Dim rs As ADODB.Recordset
Dim cn1 As ADODB.Connection
Dim rs1 As ADODB.Recordset

Private Sub add_Click()
rs.AddNew
rs.Fields(0) = Text1.Text
rs.Fields(1) = Text2.Text
rs.Fields(2) = Text3.Text
rs.Fields(4) = Text6.Text
rs.Fields(6) = Text4.Text
rs.Fields(7) = Text5.Text
rs.UpdateBatch
Text1.Text = ""
Text2.Text = ""
Text3.Text = ""
Text4.Text = ""
Text5.Text = ""
Text6.Text = ""
End Sub

Private Sub Combo1_click()
rs1.MoveFirst
Do Until rs1.RecordCount = 0
If rs1.Fields(0) = Val(Combo1.Text) Then
Text1.Text = rs1.Fields(0)
Text2.Text = rs1.Fields(1)
Text6.Text = rs1.Fields(2)
Text3.Text = ""
Text4.Text = ""
Text5.Text = ""
Exit Sub
Else
rs1.MoveNext
```

```
End If
Loop
rs1.Requery
End Sub

Private Sub DTPicker1_Change()
Text5.Text = DTPicker1.Value
End Sub
Private Sub Exit_Click()
Unload Me
End Sub

Private Sub Form_Load()
Set cn = New ADODB.Connection
Dim k As Integer
cn.Open "Req_dsn", "scott", "tiger"
 Set rs = New ADODB.Recordset
 rs.Open "select * from req_items", cn, adOpenDynamic, adLockPessimistic
Set cn1 = New ADODB.Connection
cn1.Open "Req_dsn", "scott", "tiger"
 Set rs1 = New ADODB.Recordset
 rs1.Open "select * from avail_stock", cn1, adOpenDynamic, adLockPessimistic
rs1.MoveFirst
Do Until rs1.EOF
Combo1.AddItem rs1.Fields(0)
rs1.MoveNext
Loop
End Sub
```

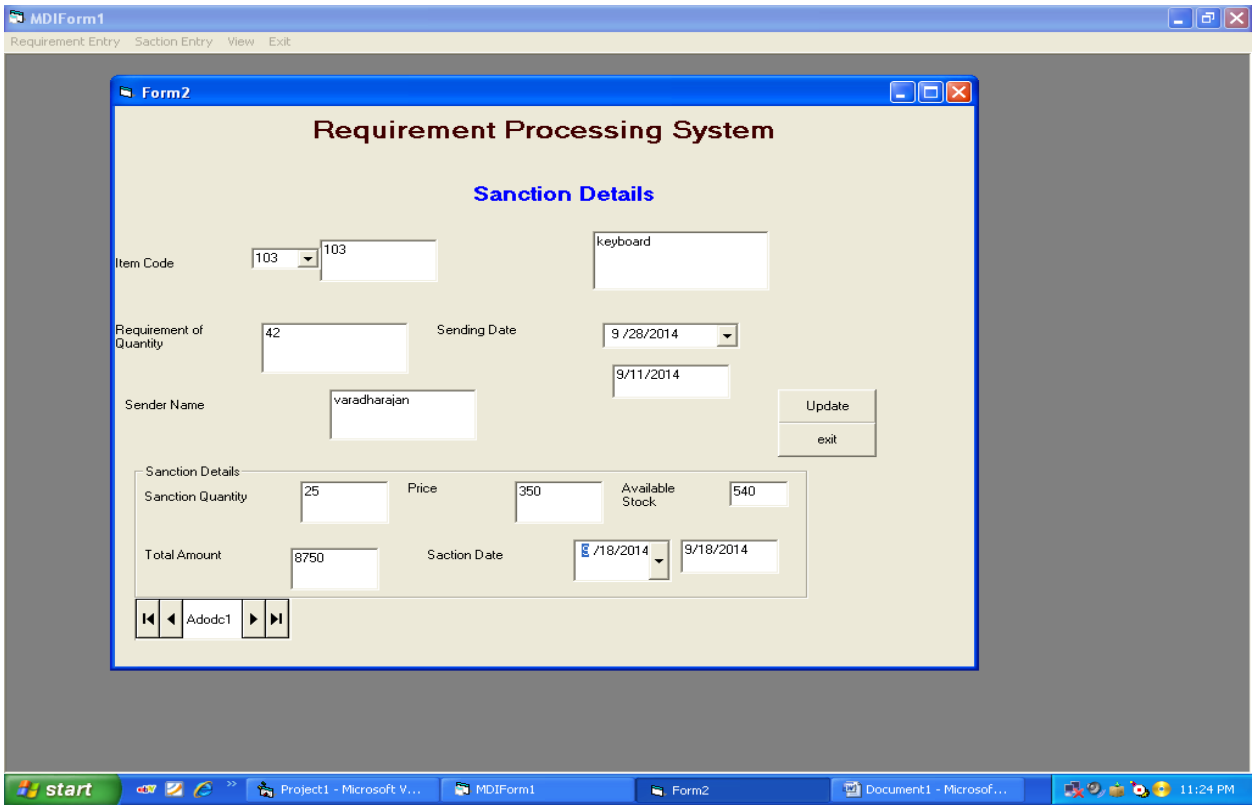
Form2

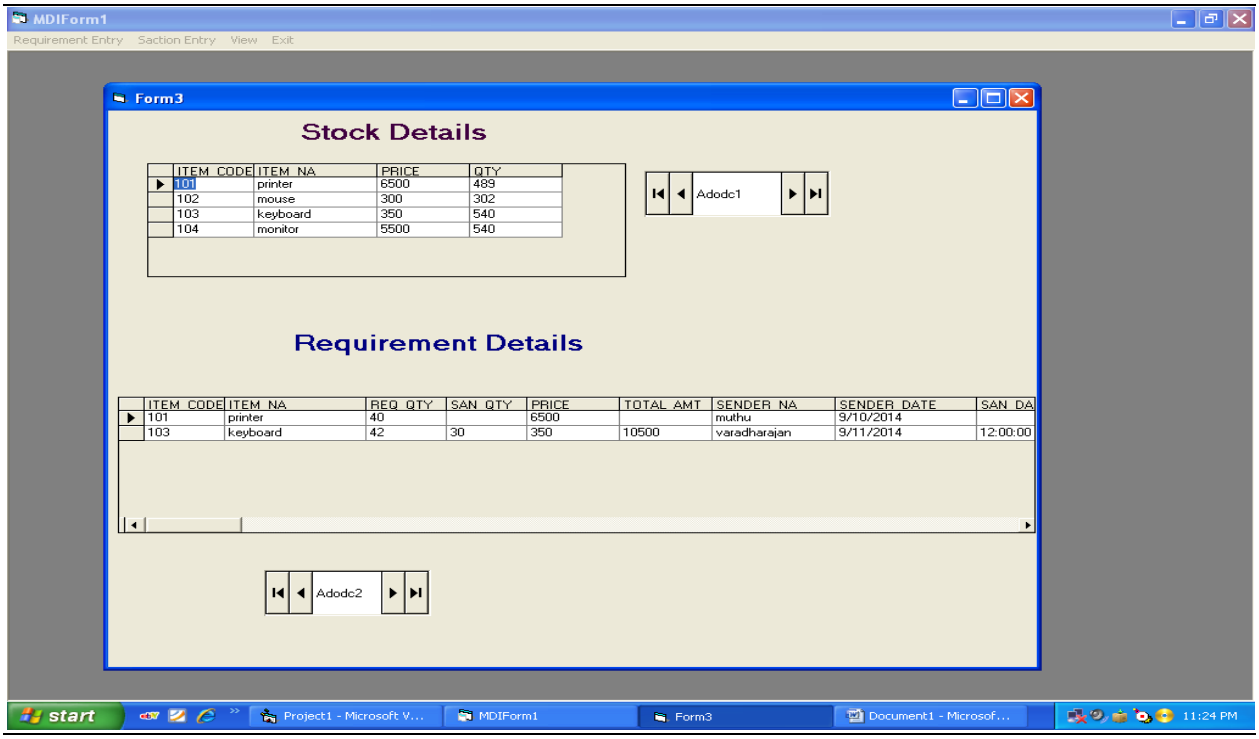
```
Dim cn As New ADODB.Connection
Dim rs As ADODB.Recordset
Dim cn1 As New ADODB.Connection
Dim rs1 As ADODB.Recordset
Dim cn2 As New ADODB.Connection
Dim rs2 As ADODB.Recordset

Private Sub Combo1_click()
rs.MoveFirst
Do Until rs.EOF
If rs.Fields(0) = Val(Combo1.Text) Then
Text1.Text = rs.Fields(0)
Text2.Text = rs.Fields(1)
Text3.Text = rs.Fields(2)
Text6.Text = rs.Fields(4)
Text4.Text = rs.Fields(6)
Text8.Text = rs.Fields(7)
rs1.MoveFirst
```

```
Do Until rs1.EOF
If rs1.Fields(0) = Val(Combo1.Text) Then
Text10.Text = rs1.Fields(3)
Exit Sub
Else
rs1.MoveNext
End If
Loop
Exit Sub
Else
rs.MoveNext
End If
Loop
rs1.MoveFirst
Do Until rs1.EOF
If rs1.Fields(0) = Val(Combo1.Text) Then
Text10.Text = rs1.Fields(3)
Exit Sub
Else
rs1.MoveNext
End If
Loop
End Sub

Private Sub DTPicker1_Change()
Text5.Text = DTPicker1.Value
End Sub
Private Sub DTPicker2_Change()
Text9.Text = DTPicker2.Value
End Sub

Private Sub Exit_Click()
Unload Me
End Sub

Private Sub Form_Load()
Set cn = New ADODB.Connection
Dim k As Integer
cn.Open "Req_dsn", "scott", "tiger"
 Set rs = New ADODB.Recordset
 rs.Open "select * from req_items", cn, adOpenDynamic, adLockPessimistic
cn1.Open "Req_dsn", "scott", "tiger"
 Set rs1 = New ADODB.Recordset
 rs1.Open "select * from avail_stock", cn1, adOpenDynamic, adLockPessimistic
cn2.Open "Req_dsn", "scott", "tiger"
Set rs2 = New ADODB.Recordset
rs.MoveFirst
Do Until rs.EOF
```

```
Combo1.AddItem rs.Fields(0)
rs.MoveNext
Loop
End Sub

Private Sub Update_Click()
rs.MoveFirst
Do Until rs.EOF
If rs.Fields(0) = Val(Text1.Text) Then
rs.Fields(3) = Val(Text5.Text)
rs.Fields(5) = Val(Text7.Text)
rs.Fields(8) = Val(Text9Text)
rs.Update
Text1.Text = ""
Text2.Text = ""
Text3.Text = ""
Text4.Text = ""
Text5.Text = ""
Text6.Text = ""
Text7.Text = ""
Text8.Text = ""
Text9.Text = ""
Text10.Text = ""
Combo1.Refresh
Exit Sub
Else
rs.MoveNext
End If
Loop
End Sub
```

OUTPUT:





RESULT:

Thus the material requirement processing system was created successfully.

Ex. No: 22 **HOSPITAL MANAGEMENT SYSTEM**

Date :

AIM:

To create a hospital management system using VB as front end and oracle as back end.

PROGRAM:

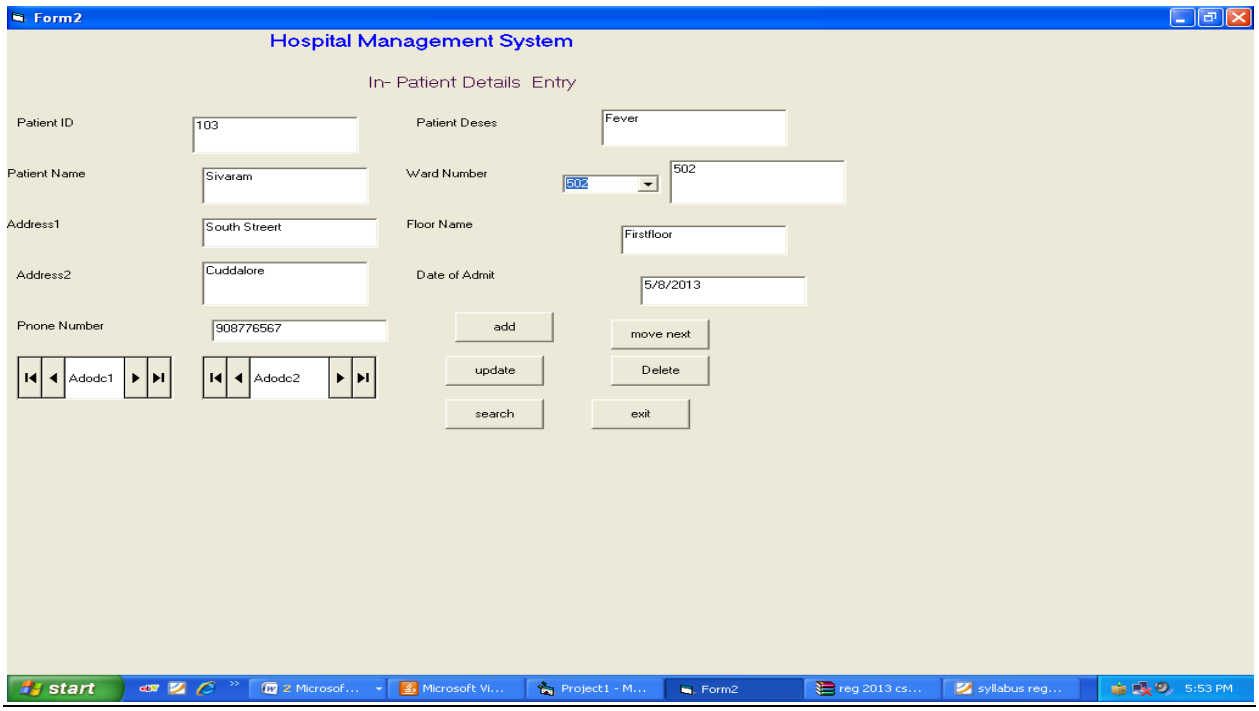
FORM 1

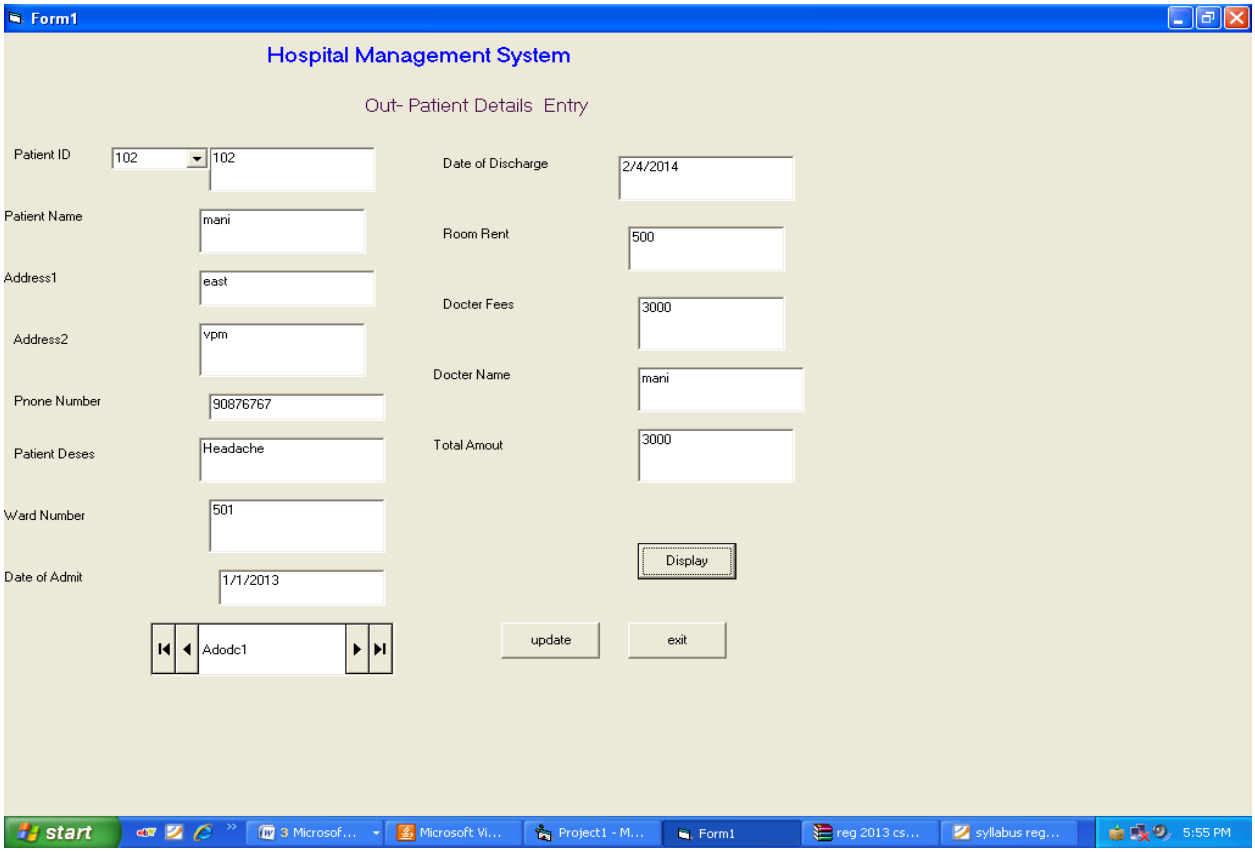
```
Private Sub disp_Click()
Dim k As Integer
Adodc1.Recordset.MoveFirst
Do Until Adodc1.Recordset.EOF
k = Val(Combo1.Text)
If Adodc1.Recordset.Fields(0) = k Then
Text1.Text = Val(Adodc1.Recordset.Fields(0))
Text2.Text = Adodc1.Recordset.Fields(1)
Text3.Text = Adodc1.Recordset.Fields(2)
Text4.Text = Adodc1.Recordset.Fields(3)
Text5.Text = Val(Adodc1.Recordset.Fields(4))
Text6.Text = Adodc1.Recordset.Fields(5)
Text7.Text = Val(Adodc1.Recordset.Fields(6))
Text8.Text = Adodc1.Recordset.Fields(7)
Exit Sub
Adodc1.Recordset.MoveLast
Else
Adodc1.Recordset.MoveNext
End If
Loop
End Sub

Private Sub Text11_Change()
Text13.Text = Val(Text10.Text) + Val(Text11.Text)
End Sub

Private Sub update_Click()
Adodc1.Recordset.update
End Sub
Private Sub exit_Click()
End
End Sub
Private Sub Form_Load()
Do Until Adodc1.Recordset.EOF
Combo1.AddItem Adodc1.Recordset.Fields(0)
Adodc1.Recordset.MoveNext
Loop
Text13.Text = 0
End Sub
```

FORM 2

```
Private Sub Combo1_Change()
Text7.Text = Combo1.Text
End Sub

Private Sub Combo1_Click()
Text7.Text = Combo1.Text
Adodc2.Recordset.MoveFirst
Do Until Adodc2.Recordset.EOF
If Adodc2.Recordset.Fields(0) = Val(Combo1.Text) Then
Text9.Text = Adodc2.Recordset.Fields(1)
Exit Sub
Else
Adodc2.Recordset.MoveNext
End If
Loop
End Sub

Private Sub Command1_Click()
Adodc1.Recordset.AddNew
End Sub

Private Sub Command2_Click()
Adodc1.Recordset.MoveNext
End Sub

Private Sub Command3_Click()
Adodc1.Recordset.Update
End Sub

Private Sub Command4_Click()
Adodc1.Recordset.Delete
End Sub

Private Sub Command6_Click()
End
End Sub

Private Sub Form_Load()
Adodc2.Recordset.MoveFirst
Do Until Adodc2.Recordset.EOF
Combo1.AddItem Adodc2.Recordset.Fields(0)
Adodc2.Recordset.MoveNext
Loop
End Sub
```

OUTPUT





RESULT:

Thus the hospital management system was created successfully.

Ex. No: 23 RAILWAY RESERVATION SYSTEM

Date :

AIM:

To create a railway reservation system using VB as front end and oracle as back end.

PROGRAM:

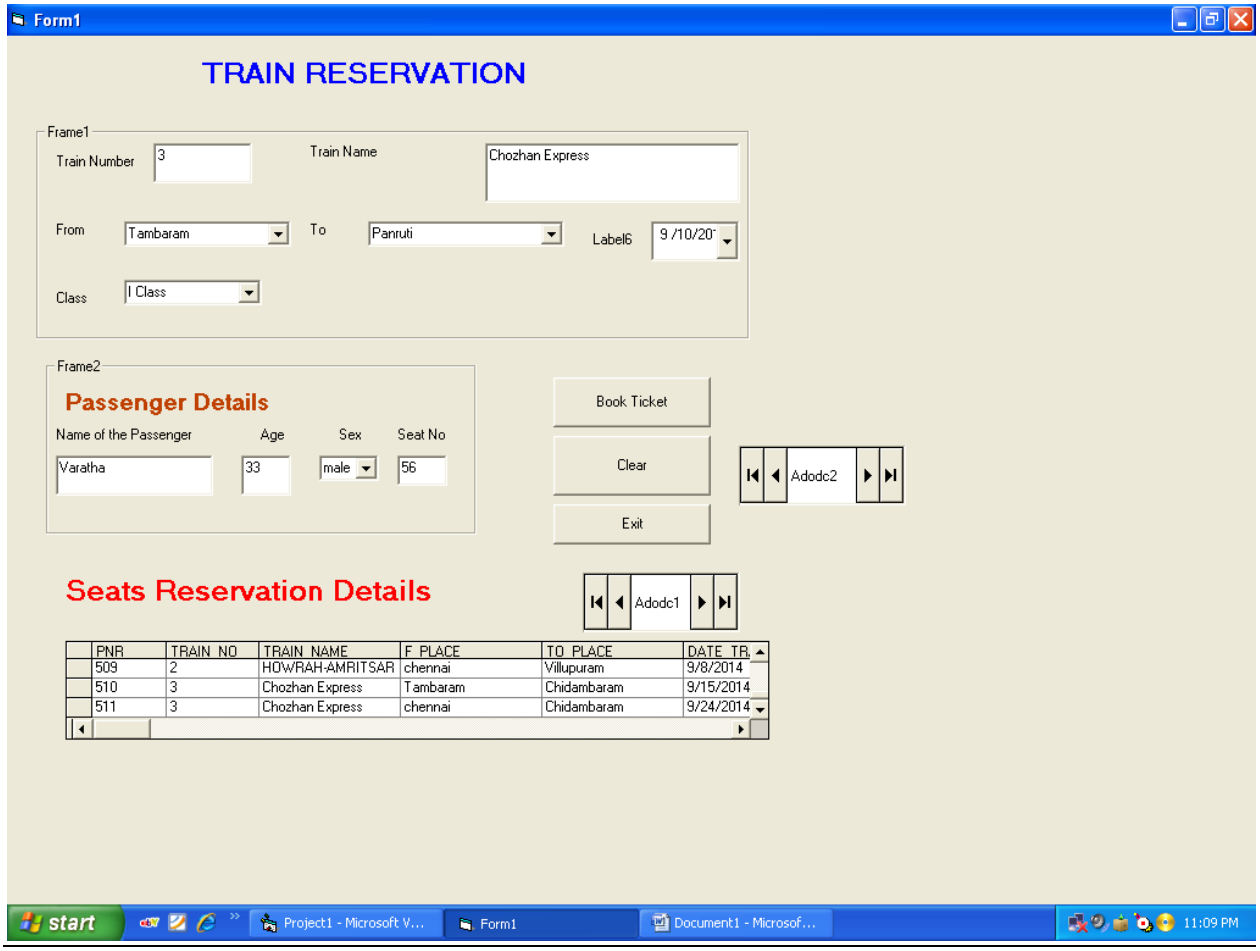
```
Dim cn As ADODB.Connection
Dim rs As ADODB.Recordset
Dim n As Integer
Private Sub book_ticket_Click()
rs.MoveLast
n = rs.RecordCount
If n < 0 Then
rs.MoveLast
k = rs.Fields(0)
k = k + 1
Else
k = 600
End If
rs.AddNew
rs.Fields(0) = k
rs.Fields(1) = Val(Text1.Text)
rs.Fields(2) = Text2.Text
rs.Fields(3) = Combo1.Text
rs.Fields(4) = Combo3.Text
rs.Fields(5) = DTPicker1.Value
rs.Fields(6) = Combo4.Text
rs.Fields(7) = Text5.Text
rs.Fields(8) = Val(Text6.Text)
rs.Fields(9) = Combo2.Text
rs.Fields(10) = Text7.Text
Adodc2.Recordset.Update
DataGrid1.Refresh
Text1.Text = ""
Text2.Text = ""
 Combo1.Text = ""
 Combo3.Text = ""
```

```
Combo4.Text = ""
 Text5.Text = ""
 Text6.Text = ""
 Combo2.Text = ""
 Text7.Text = ""
End Sub
Private Sub clear_Click()
Text1.Text = ""
Text2.Text = ""
 Combo1.Text = ""
 Combo3.Text = ""
Combo4.Text = ""
 Text5.Text = ""
 Text6.Text = ""
 Combo2.Text = ""
 Text7.Text = ""
End Sub
Private Sub Combo5_Change()
Do Until Adodc1.Recordset.EOF
If Adodc1.Recordset.Fields(0) = Val(Text1.Text) Then
Text2.Text = Adodc1.Recordset.Fields(1)
Exit Sub
Else
Adodc1.Recordset.MoveNext
End If
Loop
End Sub
Private Sub Combo5_Click()
Adodc1.Recordset.MoveFirst
Do Until Adodc1.Recordset.EOF
If Adodc1.Recordset.Fields(0) = Val(Text1.Text) Then
Text2.Text = Adodc1.Recordset.Fields(1)
Exit Sub
Else
Adodc1.Recordset.MoveNext
End If
Loop
End Sub
Private Sub Combo2_Click()
```

```
Combo2.AddItem "male"
Combo2.AddItem "female"
End Sub
Private Sub Form_Load()
Dim k As Integer
k = 600
Set cn = New ADODB.Connection
cn.Open "train_dsn", "scott", "tiger"
 Set rs = New ADODB.Recordset
 rs.Open "select * from train_reservation", cn, adOpenDynamic, adLockPessimistic
Set DataGrid1.DataSource = Adodc2
Adodc1.Recordset.MoveLast
k = Adodc1.Recordset.RecordCount
Adodc1.Recordset.MoveFirst
Do Until k = 0
Combo1.AddItem Adodc1.Recordset.Fields(2)
Combo3.AddItem Adodc1.Recordset.Fields(3)
k = k - 1
Adodc1.Recordset.MoveNext
Loop
Combo4.AddItem "I Class"
Combo4.AddItem "Second Class"
Combo4.AddItem "Slepper Class"
Set Text5.DataSource = Adodc2
Set DataGrid1.DataSource = Adodc2
Combo2.AddItem "male"
Combo2.AddItem "female"
End Sub
Private Sub Text1_Change()
Adodc1.Recordset.MoveFirst
Do Until Adodc1.Recordset.EOF
If Adodc1.Recordset.Fields(0) = Val(Text1.Text) Then
Text2.Text = Adodc1.Recordset.Fields(1)
Exit Sub
Else
Adodc1.Recordset.MoveNext
End If
Loop
```

End Sub

OUTPUT



RESULT:

Thus the railway reservation system was created successfully.

Ex. No: 24 **PERSONAL INFORMATION SYSTEM**

Date :

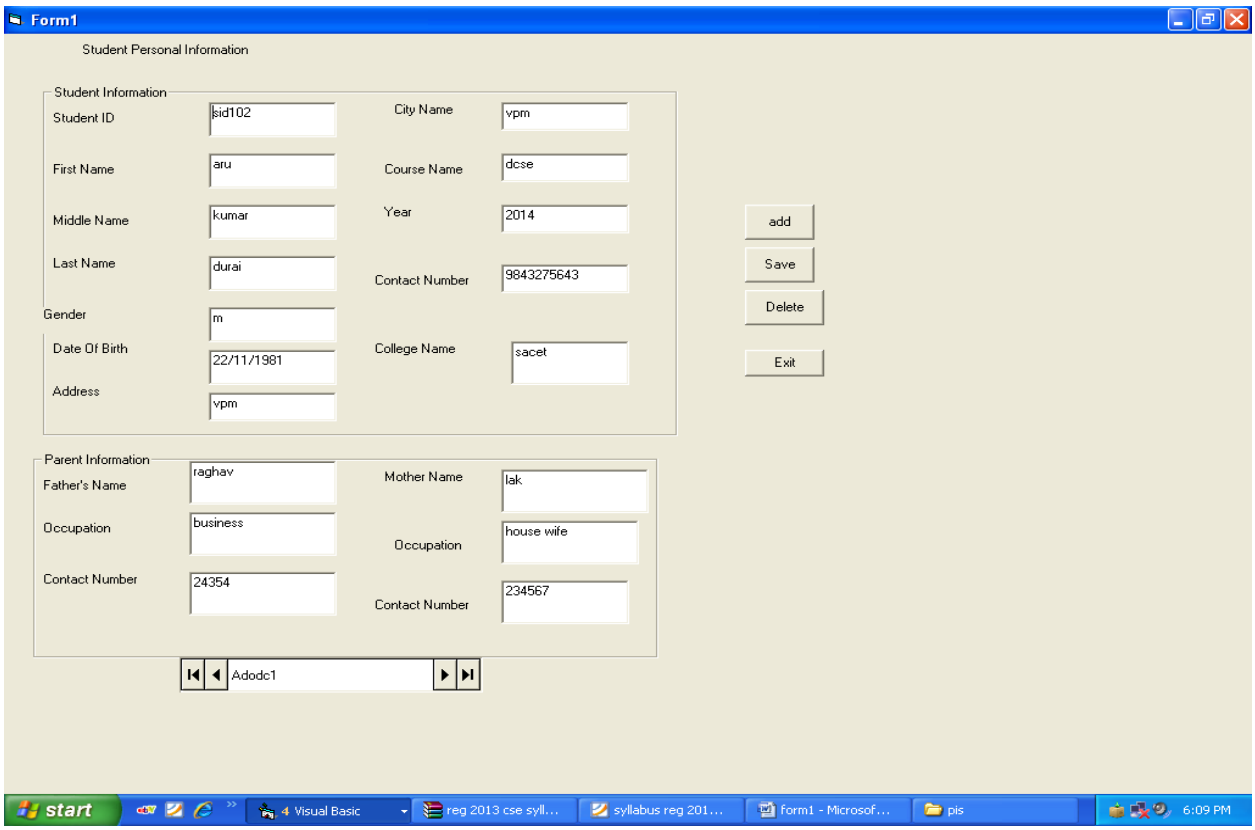
AIM:

To create a personal information system using VB as front end and oracle as back end.

PROGRAM:

```
Private Sub add_Click()
Adodc1.Recordset.AddNew
End Sub
Private Sub Delete_Click()
Adodc1.Recordset.Delete
End Sub
Private Sub exit_Click()
End
End Sub
Private Sub Save_Click()
Adodc1.Recordset.Update
End Sub
```

OUTPUT



RESULT:

Thus the personal information system was created successfully.

Ex. No: 25 HOSPITAL MANAGEMENT SYSTEM

Date :

AIM:

To create a hospital management system using VB as front end and oracle as back end.

PROGRAM:

FORM 1

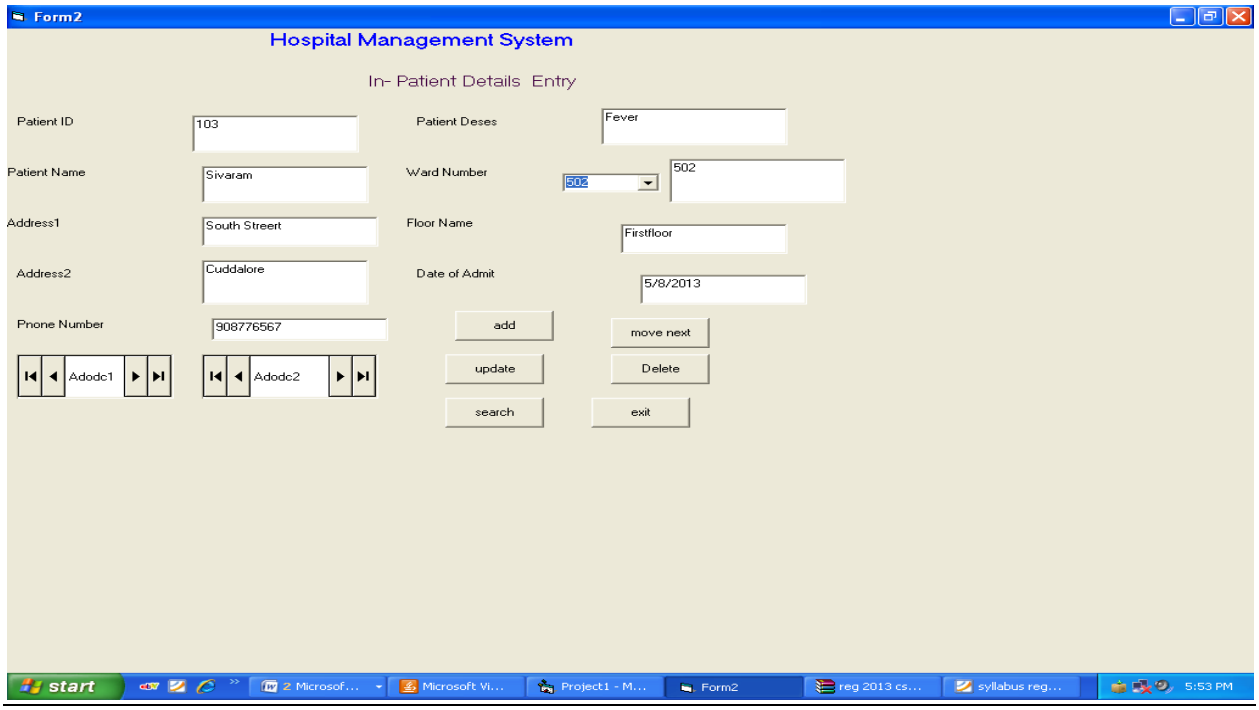
```
Private Sub disp_Click()
Dim k As Integer
Adodc1.Recordset.MoveFirst
Do Until Adodc1.Recordset.EOF
k = Val(Combo1.Text)
If Adodc1.Recordset.Fields(0) = k Then
Text1.Text = Val(Adodc1.Recordset.Fields(0))
Text2.Text = Adodc1.Recordset.Fields(1)
Text3.Text = Adodc1.Recordset.Fields(2)
Text4.Text = Adodc1.Recordset.Fields(3)
Text5.Text = Val(Adodc1.Recordset.Fields(4))
Text6.Text = Adodc1.Recordset.Fields(5)
Text7.Text = Val(Adodc1.Recordset.Fields(6))
Text8.Text = Adodc1.Recordset.Fields(7)
Exit Sub
Adodc1.Recordset.MoveLast
Else
Adodc1.Recordset.MoveNext
End If
Loop
End Sub

Private Sub Text11_Change()
Text13.Text = Val(Text10.Text) + Val(Text11.Text)
End Sub

Private Sub update_Click()
Adodc1.Recordset.update
End Sub
Private Sub exit_Click()
End
End Sub
Private Sub Form_Load()
Do Until Adodc1.Recordset.EOF
Combo1.AddItem Adodc1.Recordset.Fields(0)
Adodc1.Recordset.MoveNext
Loop
Text13.Text = 0
End Sub
```

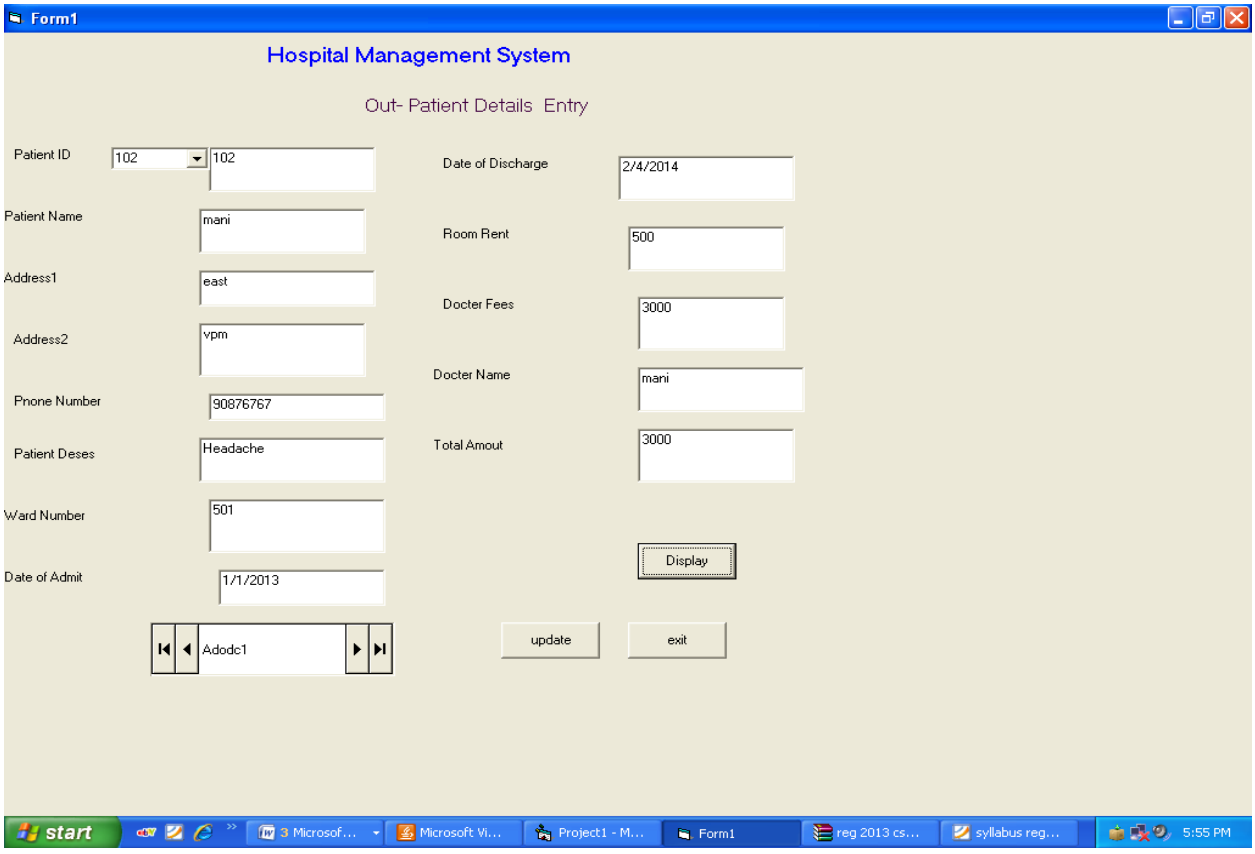
FORM 2

```
Private Sub Combo1_Change()
```

```
Text7.Text = Combo1.Text
End Sub

Private Sub Combo1_Click()
Text7.Text = Combo1.Text
Adodc2.Recordset.MoveFirst
Do Until Adodc2.Recordset.EOF
If Adodc2.Recordset.Fields(0) = Val(Combo1.Text) Then
Text9.Text = Adodc2.Recordset.Fields(1)
Exit Sub
Else
Adodc2.Recordset.MoveNext
End If
Loop
End Sub

Private Sub Command1_Click()
Adodc1.Recordset.AddNew
End Sub

Private Sub Command2_Click()
Adodc1.Recordset.MoveNext
End Sub

Private Sub Command3_Click()
Adodc1.Recordset.Update
End Sub

Private Sub Command4_Click()
Adodc1.Recordset.Delete
End Sub

Private Sub Command6_Click()
End
End Sub

Private Sub Form_Load()
Adodc2.Recordset.MoveFirst
Do Until Adodc2.Recordset.EOF
Combo1.AddItem Adodc2.Recordset.Fields(0)
Adodc2.Recordset.MoveNext
Loop
End Sub
```

OUTPUT





RESULT:

Thus the hospital management system was created successfully.