SRM MADURAI

COLLEGE FOR ENGINEERING AND TECHNOLOGY

Approved by AICTE, New Delhi | Affiliated to Anna University, Chennai

CS3481- DATABASE MANAGEMENT SYSTEMS LABORATORY

RECORD

Name : _____

Roll No : _____

Branch : _____

Year/Sem : _____

SRM MADURAI
COLLEGE FOR ENGINEERING AND TECHNOLOGY
Approved by AICTE, New Delhi | Affiliated to Anna University, Chennai

BONAFIDE CERTIFICATE

Register No _____

Certified to be the bonafide record of work done by _____

of _____ Semester B.E / B.Tech _____

course in the **CS3481 - DATABASE MANAGEMENT SYSTEMS LABORATORY** in SRM MADURAI COLLEGE FOR ENGINEERING AND TECHNOLOGY, POTTAPALAYAM, during the academic year _____

Staff In-Charge **Head of the Department**

Submitted for the University Practical Examination held on _____at SRM MADURAI COLLEGE FOR ENGINEERING AND TECHNOLOGY, POTTAPALAYAM.

Internal Examiner **External Examiner**

CONTENTS

LIST OF EXPERIMENTS

1. Create a database table, add constraints (primary key, unique, check, Not null), insert rows, update and delete rows using SQL DDL and DML commands.
2. Create a set of tables, add foreign key constraints and incorporate referential integrity.
3. Query the database tables using different 'where' clause conditions and also implement aggregate functions.
4. Query the database tables and explore sub queries and simple join operations.
5. Query the database tables and explore natural, equi and outer joins.
6. Write user defined functions and stored procedures in SQL.
7. Execute complex transactions and realize DCL and TCL commands.
8. Write SQL Triggers for insert, delete, and update operations in a database table.
9. Create View and index for database tables with a large number of records.
10. Create an XML database and validate it using XML schema.
11. Create Document, column and graph based data using NOSQL database tools.
12. Develop a simple GUI based database application.
13. Case Study using any of the real life database applications

SOFTWARE:

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

Ex.No:1	DATA DEFINITION LANGUAGE (DDL) (WITHOUT CONSTRAINT)
Date:	

SQL BASIC COMMANDS

AIM:

Creation of a database and writing SQL commands to retrieve information from the database.

QUERIES:

//Create table

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

// Insert values

```
INSERT INTO emp(empno,empname, DOB, designation) VALUES(100,'John', '21-Apr-1996',  50000,'Manager');
INSERT INTO emp(empno,empname, DOB, designation) VALUES(101,'Greg', '20-July-1994',  2500,'Clerk');
-2 rows inserted
```

// Display values

```
SELECT * FROM emp;
```

EMPNO	EMPNAME	DOB	SALARY	DESIGNATION
100	John	21-Apr-1996	50000	Manager
101	Greg	20-July-1994	2500	Clerk

SELECT empname, salary FROM emp;

EMPNAME	SALARY
John	50000
Greg	25000

// Modify values

UPDATE emp SET salary = salary + 1000;

SELECT * FROM emp;

EMPNO	EMPNAME	DOB	SALARY	DESIGNATION
100	John	21-Apr-1996	51000	Manager
101	Greg	20-July-1994	3500	Clerk

// Delete values

DELETE FROM emp WHERE empno = 100;

-1 row deleted.
SELECT * FROM emp;

EMPNO	EMPNAME	DOB	SALARY	DESIGNATION
101	Greg	20-July-1994	3500	Clerk

DDL STATEMENTS

QUERIES:

1. CREATE THE TABLE

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

- Table Created

// Describe the table emp

DESC emp;

Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
EMPNO	INT	11	-	-	-	-	-	-
EMPNAME	VARCHAR	25	-	-	-	-	-	-
DOB	DATE	7	-	-	-	-	-	-
SALARY	INT	11	-	-	-	-	-	-
DESIGNATION	VARCHAR	20	-	-	-	-	-	-

2. ALTER THE TABLE

a. ADD

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

ALTER TABLE emp ADD department VARCHAR(50);

- Table Altered

DESC emp;

Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
EMPNO	INT	11	-	-	-	-	-	-
EMPNAME	VARCHAR	25	-	-	-	-	-	-
DOB	DATE	7	-	-	-	-	-	-
SALARY	INT	11	-	-	-	-	-	-
DESIGNATION	VARCHAR	20	-	-	-	-	-	-
DEPARTMENT	VARCHAR2	50	-	-	-	-	-	-

b. MODIFY

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

ALTER TABLE emp MODIFY department VARCHAR2(100);

- Table Altered

DESC emp;

Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
EMPNO	INT	11	-	-	-	-	-	-
EMPNAME	VARCHAR	25	-	-	-	-	-	-
DOB	DATE	7	-	-	-	-	-	-
SALARY	INT	11	-	-	-	-	-	-
DESIGNATION	VARCHAR	20	-	-	-	-	-	-
DEPARTMENT	VARCHAR	100	-	-	-	-	-	-

c. DROP

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

ALTER TABLE emp DROP(department);

- Table Altered

DESC emp;

Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
EMPNO	INT	11	-	-	-	-	-	-
EMPNAME	VARCHAR	25	-	-	-	-	-	-
DOB	DATE	7	-	-	-	-	-	-
SALARY	INT	11	-	-	-	-	-	-
DESIGNATION	VARCHAR	20	-	-	-	-	-	-

d. RENAME

// To alter the table name by using rename keyword

ALTER TABLE emp RENAME TO emp1 ;

- Table Altered

DESC emp1;

Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
EMPNO	INT	11	-	-	-	-	-	-
EMPNAME	VARCHAR	25	-	-	-	-	-	-
DOB	DATE	7	-	-	-	-	-	-
SALARY	INT	22	-	-	-	-	-	-
DESIGNATION	VARCHAR	20	-	-	-	-	-	-
DEPARTMENT	VARCHAR	100	-	-	-	-	-	-

3. DROP

//To delete the table from the database

DROP TABLE emp1;

- Table Dropped

DESC emp1;
Object to be described could not be found.

CONSTRAINT TYPES:

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

QUERIES:

1. CREATE THE TABLE

// To create a table student

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

// Describe the table student

```
DESC student;
```

Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
STUDENTID	INT	22	-	-	1	-	-	-
SNAME	VARCHAR	30	-	-	-	-	-	-
DEPARTMENT	VARCHAR	5	-	-	-	-	-	
SEM	INT	22	-	-	-	-	-	-
DOB	DATE	7	-	-	-	-	-	
EMAIL_ID	VARCHAR	20	-	-	-	-	-	-
COLLEGE	VARCHAR	20	-	-	-	-		

-

//To create a table exam

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

//Describe the table exam

```
DESC exam;
```

Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
EXAMID	INT	22	-	-	-	-	-	-
STUDENTID	INT	22	-	-	-	-	-	-
DEPARTMENT	VARCHAR	5	-	-	-	-	-	-
MARK1	INT	22	-	-	-	-	-	-
MARK2	INT	22	-	-	-	-	-	-
MARK3	INT	22	-	-	-	-	-	-
MARK4	INT	22	-	-	-	-	-	-
MARK5	INT	22	-	-	-	-	-	-
TOTAL	INT	22	-	-	-	-	-	-
AVERAGE	INT	22	-	-	-	-	-	-
GRADE	VARCHAR	1	-	-	-	-	-	-

1. ALTER THE TABLE

A. ADD

//To alter the table student by adding new attribute address

ALTER TABLE student ADD address VARCHAR2(100);

- Table Altered

DESC student;

Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
STUDENTID	INT	22	-	-	1	-	-	-
SNAME	VARCHAR	30	-	-	-	-	-	-
DEPARTMENT	VARCHAR	5	-	-	-		-	-
SEM	INT	22	-	-	-		-	-
DOB	DATE	7	-	-	-		-	-
EMAIL_ID	VARCHAR	20	-	-	-		-	-
COLLEGE	VARCHAR	20	-	-	-		'MEC'	-
ADDRESS	VARCHAR	100	-	-	-		-	-

B. MODIFY

//To alter the table student by modifying the size of the attribute address

ALTER TABLE student MODIFY address VARCHAR (150);

- Table Altered

DESC student;

Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
STUDENTID	INT	22	-	-	1	-	-	-
SNAME	VARCHAR	30	-	-	-	-	-	-
DEPARTMENT	VARCHAR	5	-	-	-	-	-	-

SEM	INT	22	-	-	-	-	-
-							
DOB	DATE	7	-	-	-	-	-
-							
EMAIL_ID	VARCHAR	20	-	-	-	-	-
-							
COLLEGE	VARCHAR	20	-	-	-	-	
'MEC' -							
ADDRESS	VARCHAR	150	-	-	-	-	-
-							

C. DROP

//To alter the table student by deleting the attribute address

ALTER TABLE student DROP(address);

- Table Altered

DESC student;

Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default
Comment							
STUDENTID	INT	22	-	-	1	-	
- -							
SNAME	VARCHAR	30	-	-	-	-	
- -							
DEPARTMENT	VARCHAR	5	-	-	-	-	
- -							
SEM	INT	22	-	-	-	-	
- -							
DOB	DATE	7	-	-	-	-	
- -							
EMAIL_ID	VARCHAR	20	-	-	-	-	
- -							
COLLEGE	VARCHAR	20	-	-	-	-	
'MEC' -							

D. RENAME

// To alter the table name by using rename keyword

ALTER TABLE student RENAME TO student1 ;

-Table Altered
DESC student1;

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

ALTER TABLE student1 RENAME TO student ;

- Table Altered

2. DROP

// To delete the table from the database

DROP TABLE exam;

- Table dropped

DESC exam;

Object to be described could not be found.

RESULT:

Databases are created and retrieved information successfully.

Ex.No:2	CREATION OF EMPLOYEE DATABASE WITH CONSTRAINTS
Date:	

AIM:

Creation of Employee Database with Constraints by using Structured Query Language.

QUERIES:

1. CREATE THE TABLE

// To create a table employee

```
CREATE TABLE employee
(
        empID INT PRIMARY  KEY,
        ename VARCHAR(30) NOT NULL,
        department VARCHAR(5),
        designation VARCHAR(50),
        dob DATE,
        email_id VARCHAR(20) UNIQUE,
        college VARCHAR(20) DEFAULT 'MEC'
);
```

// Describe the table student

```
DESC student;
```

Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
EMPID	INT	22	-	-	1	-	-	-
ENAME	VARCHAR	30	-	-	-	-	-	-
DEPARTMENT	VARCHAR	5	-	-	-	-	-	-
DESIGNATION	VARCHAR	50	-	-	-	-	-	-
DOB	DATE	7	-	-	-	-	-	-
EMAIL_ID	VARCHAR	20	-	-	-	-	-	-

COLLEGE VARCHAR 20 - - - -
'MEC' -

Create a table 'Employee' with the following details (Column level constraints)

Empno	Number(5)	Primary key
Empname	Varchar2(20)	
Designation	Varchar2(10)	
Date_of_join	Date	
Salary	Number(9,2)	NOT NULL
Depno	Number(2)	Foreign key(Reference 'Department' table)

QUERIES:

Create table Employee(Empno Number(5) Primary key, Empname Varchar2(20) ,Designation Varchar2(10), Date_of_join Date, Salary Number(9,2) constraint csk NOT NULL, Depno Number(2) Reference Department (Deptno));

SQL>descs employee;

NAME	Type	
Empno	Number(5)	
Empname	Varchar2(20)	
Designation	Varchar2(10)	
Date_of_join	Date	
Salary	Number(9,2)	NOTNULL
Depno	Number(2)	

SQL>insert into employee values(102,'saravanan','leturer','11-feb-2019',22);
1 row inserted

SQL> select* from employee;

EMPNO	ENAME	DESIGNATIN	SALARY	Dateofjoin
102	SARAVANAN	LECTURER	15000	11-03-2019 22
103	PANNERSELVAM	ASST. PROF	2000	12-05-2000 32
104	CHINNI	HOD, PROF	45000	13.05.2010 43

2. **Create another table called 'Department' with the following structure(Column level constraints)**

Depno	**Number(2)**	**Primary key**
Depname	**Varchar2(15)**	
Deplocation	**Varchar2(10)**	

Ans:

SQL>Create table department (deptno Number(2) Primary key , Depname Varchar2(15), Deplocation Varchar2(10));

SQL>desc department;

Name	Type	
Depno	Number(2)	Primary key
Depname	Varchar2(15)	
Deplocation	Varchar2(10)	

SQL>insert into department values(40,'cse' , 'krishnasamy');
1 row inserted

SQL>select* from department

DEPTNO	DEPNAME	Dlocation
40	cse	krishnasamy

i. Display the number of employees in each department with the department numbers in descending order

SQL>select deptno, count(empname)from department groupby department orderby desc;

ii. List the departments where there are employees functioning

SQL> Select deptname from department

iii. List the employees whose depno is not equal to '01'

SQL> Select ename from employee where not depno =01;

RESULT:

Thus the Employee database with constraints were performed and executed successfully.

Ex.No:3	QUERY THE DATABASE TABLES USING DIFFERENT 'WHERE CLAUSE CONDITIONS AND ALSO IMPLEMENT AGGREGATE FUNCTIONS
Date:	

AIM:

Creation of a database and execute Aggregate Functions to retrieve information from the database.

QUERIES:

- **ORDER BY**
- **GROUP BY**
- **AGGREGATE FUNCTIONS**

//Consider the following table of student

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

- Table Created

// Describe the table student

DESC student;

Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
STUDENTID	INT	22	-	-	1	-	-	-

SNAME VARCHAR 30 - - - - -
-
DEPARTMENT CHAR 5 - - - - -
-
SEM INT 22 - - - - -
-
DOB DATE 7 - - - - -
-
EMAIL_ID VARCHAR 20 - - - - -
-
COLLEGE VARCHAR 20 - - - -
'MEC' -

SELECT * FROM student;

\

STUDENTID	SNAME	DEPARTMENT	SEM	DOB	EMAIL_ID	COLLEGE
101	RUPESH	IT	5	04-DEC-1996	rupesh@gmail.com	MEC
102	BALA	CSE	7	10-JAN-1995	bala@gmail.com	IIT
104	HEMESH	IT	5	07-FEB-1996	hemesh@gmail.com	IIT
106	SAI VAISHNAVI	CSE	5	06-OCT-1996	vaishu@gmail.com	SMVEC
108	RISHA	IT	5	04-NOV-1996	risha@gmail.com	MEC

1. 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;

DEPARTMENT	SEM	SNAME
CSE	5	SAI VAISHNAVI
CSE	7	BALA
IT	5	RISHA
IT	5	RUPESH
IT	5	HEMESH

// 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;

DEPARTMENT	SEM	SNAME
IT	5	RUPESH

IT	5	RISHA
IT	5	HEMESH
CSE	7	BALA
CSE	5	SAI VAISHNAVI

2. GROUP BY

// To displays the total value group by department

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

DEPARTMENT	SUM_DEPARTMENT
CSE	430
IT	1346

3. AGGREGATE FUNCTIONS

// 1. COUNT - displays total number of rows

SELECT COUNT(examid) AS STUDENTS_REGISTERED FROM exam;

STUDENTS_REGISTERED
4

// 2. MAX - displays the maximum value

SELECT MAX(average) AS RANK_1 FROM exam;

RANK_1
90.8

// 3. MIN - displays the minimum value

SELECT MIN(average) AS LAST_RANK FROM exam;

LAST_RANK
86

// 4. SUM - displays the total value

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

DEPARTMENT	SUM_DEPARTMENT
CSE	430

IT	1346

// 5. AVG - displays the average value

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

DEPARTMENT	AVERAGE
IT	448.66667
CSE	430

RESULT:

Databases are created and SQL queries are retrieved information successfully.

Ex.No:4	QUERY THE DATABASE TABLES AND EXPLORE SUB QUERIES AND SIMPLE JOIN OPERATIONS
Date:	

AIM:

Creation of a database and execute SQL Joins to retrieve information from the database.

QUERIES:

JOIN TYPES

1. **INNER JOIN**
2. **SELF JOIN**

// Table Creation - cseitstudent

```
CREATE TABLE cseitstudent
(
        studentID INT PRIMARY KEY,
        sname VARCHAR(30),
        department VARCHAR(5),
        sem INT
);
```

-Table Created

// Table Creation - placement

```
CREATE TABLE placement
(
        PlacementID INT PRIMARY KEY,
        StudentID INT,
        department VARCHAR(5),
        Company VARCHAR(30),
        salary INT
);
```

-Table Created

// Inserting values into cseitstudent table

INSERT INTO cseitstudent (studentID, sname, department, sem) VALUES(101,'reema', 'IT',5);

INSERT INTO cseitstudent (studentID, sname, department, sem) VALUES(102,'reenu', 'IT',3);

INSERT INTO cseitstudent (studentID, sname, department, sem) VALUES(103,'sheela', 'CSE',3);

INSERT INTO cseitstudent (studentID, sname, department, sem) VALUES(104,'nirmal', 'IT',3);

INSERT INTO cseitstudent (studentID, sname, department, sem) VALUES(105,'eshwar', 'CSE',5);

- 5 row(s) inserted

// Inserting values into placement table

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

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

INSERT INTO placement VALUES(3, 204, 'MECH', 'HYUNDAI', 30000);

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

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

- 5 row(s) inserted

// Display the values in the table as rows

SELECT * FROM cseitstudent;

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

SELECT * FROM placement;

PLACEMENTID	STUDENTID	DEPARTMENT	COMPANY	SALARY

1	104	IT	infosys	25000
2	105	CSE	Wipro	22000
3	204	MECH	HYUNDAI	30000
4	102	IT	Infosys	25000
5	103	CSE	Infosys	25000

// INNER JOIN

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

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	Reenu	IT	3	4	102	IT	infosys	25000
103	Sheela	CSE	3	5	103	CSE	infosys	25000

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

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

//SELF JOIN

Returns rows by comparing the values of the same table.

//TABLE CREATION

```
CREATE TABLE employee
(
        empid INT,
        empname VARCHAR(25),
         reportingid INT
);
```

- Table Created

//INSERTING VALUES IN THE TABLE

```
INSERT INTO employee VALUES(1, 'Principal', null);
INSERT INTO employee VALUES(2, 'HOD', 1);
INSERT INTO employee VALUES(3, 'PO', 1);
INSERT INTO employee VALUES(4, 'Staff', 2);
INSERT INTO employee VALUES(5, 'Non Teaching Staff', 2);
```

- 5 row(s) inserted

//DISPLAYS VALUES IN THE TABLE

```
SELECT * FROM employee;
```

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

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

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

// EXISTS

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

PLACEMENTID	STUDENTID	DEPAR	COMPANY	SALARY

1	104	IT	infosys	25000
2	105	CSE	Wipro	22000
3	204	MECH	HYUNDAI	30000
4	102	IT	infosys	25000
5	103	CSE	infosys	25000

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

EMPNO	EMPNAME	SALARY	DESIGNATION
101	RAM	45000	Manager
102	LINGAM	35000	Asst. Manager
103	RUPESH	20000	Clerk
104	BALA	10000	Typist
105	PRAVEEN	15000	Cashier

// NOT EXISTS

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

no data found

RESULT:

Databases are created and SQL queries are retrieved information successfully.

Ex.No:5	QUERY THE DATABASE TABLES AND EXPLORE NATURAL ,
Date:	EQUI AND OUTER JOIN OPERATIONS

AIM:

Creation of a database and execute SQL Joins to retrieve information from the database.

QUERIES:

JOIN TYPES

1. OUTER JOIN

a. LEFT OUTER JOIN

b. RIGHT OUTER JOIN

c. CROSS OUTER JOIN

2. EQUI JOIN

3. NON EQUI JOIN

// Table Creation - cseitstudent

```
CREATE TABLE cseitstudent
(
        studentID INT PRIMARY KEY,
        sname VARCHAR(30),
        department VARCHAR(5),
        sem INT
);
```

-Table Created

// Table Creation - placement

```
CREATE TABLE placement
(
        PlacementID INT PRIMARY KEY,
        StudentID INT,
```

```
    department VARCHAR(5),
    Company VARCHAR(30),
    salary INT
);
```

-Table Created

// Inserting values into cseitstudent table

```
INSERT INTO cseitstudent (studentID, sname, department, sem) VALUES(101,'reema', 'IT',5);

INSERT INTO cseitstudent (studentID, sname, department, sem) VALUES(102,'reenu', 'IT',3);

INSERT INTO cseitstudent (studentID, sname, department, sem) VALUES(103,'sheela', 'CSE',3);

INSERT INTO cseitstudent (studentID, sname, department, sem) VALUES(104,'nirmal', 'IT',3);

INSERT INTO cseitstudent (studentID, sname, department, sem) VALUES(105,'eshwar', 'CSE',5);
```

- 5 row(s) inserted

// Inserting values into placement table

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

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

INSERT INTO placement VALUES(3, 204, 'MECH', 'HYUNDAI', 30000);

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

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

- 5 row(s) inserted

// Display the values in the table as rows

```
SELECT * FROM cseitstudent;
```

STUDENTID	SNAME	DEPARTMENT	SEM
101	reema	IT	5
102	reenu	IT	3
103	sheela	CSE	3

104	nirmal	IT	3
105	eshwar	CSE	5

SELECT * FROM placement;

PLACEMENTID	STUDENTID	DEPARTMENT	COMPANY	SALARY
1	104	IT	infosys	25000
2	105	CSE	Wipro	22000
3	204	MECH	HYUNDAI	30000
4	102	IT	Infosys	25000
5	103	CSE	Infosys	25000

//LEFT OUTER JOIN

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

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	reenu	IT	3	4	102	IT	infosys	25000
103	sheela	CSE	3	5	103	CSE	infosys	25000
101	reema	IT	5	-	-	-	-	-

SELECT cseitstudent.sname,placement.placementID, placement.company
FROM cseitstudent
LEFT OUTER JOIN placement
ON cseitstudent.studentID=placement.studentID;

SNAME	PLACEMENTID	COMPANY
nirmal	1	infosys
eshwar	2	Wipro
reenu	4	infosys

sheela	5	infosys
reema	-	-

//RIGHT OUTER JOIN

```
SELECT *
FROM cseitstudent
RIGHT OUTER JOIN placement
ON cseitstudent.studentID=placement.studentID;
```

STUDENTID	SNAME	DEPARTMENT	SEM	PLACEMENTID	STUDENTID	DEPARTMENT	COMPANY	SALARY
102	reenu	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

```
SELECT cseitstudent.sname,placement.placementID, placement.company
FROM cseitstudent
RIGHT OUTER JOIN placement
ON cseitstudent.studentID = placement.studentID;
```

SNAME	PLACEMENTID	COMPANY
reenu	4	infosys
sheela	5	infosys
nirmal	1	infosys
eshwar	2	Wipro
	3	HYUNDAI

//FULL OUTER JOIN

```
SELECT *
FROM cseitstudent
FULL OUTER JOIN placement
ON cseitstudent.studentID=placement.studentID;
```

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	reenu	IT	3	4	102	IT	infosys	25000
103	sheela	CSE	3	5	103	CSE	infosys	25000
101	reema	IT	5	-	-	-	-	-

```
SELECT cseitstudent.sname,placement.placementID, placement.company
FROM cseitstudent
FULL OUTER JOIN placement
ON cseitstudent.studentID=placement.studentID;
```

SNAME	PLACEMENTID	COMPANY
nirmal	1	infosys
eshwar	2	Wipro
-	3	HYUNDAI
reenu	4	infosys
sheela	5	infosys
reema	-	-

//EQUI JOIN

```
SELECT * FROM cseitstudent, placement WHERE cseitstudent.studentID=placement.studentID;
```

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	reenu	IT	3	4	102	IT	infosys	25000
103	sheela	CSE	3	5	103	CSE	infosys	25000

// NON EQUI JOIN

```
SELECT cseit.studentID, cseit.sname FROM cseitstudent cseit, placement placed
WHERE cseit.studentID>placed.studentID;
```

STUDENTID	SNAME
105	eshwar
105	eshwar
105	eshwar
104	nirmal
104	nirmal
103	sheela

// EXISTS

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

PLACEMENTID	STUDENTID	DEPAR	COMPANY	SALARY
1	104	IT	infosys	25000
2	105	CSE	Wipro	22000
3	204	MECH	HYUNDAI	30000
4	102	IT	infosys	25000
5	103	CSE	infosys	25000

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

EMPNO	EMPNAME	SALARY	DESIGNATION
101	RAM	45000	Manager
102	LINGAM	35000	Asst. Manager
103	RUPESH	20000	Clerk
104	BALA	10000	Typist
105	PRAVEEN	15000	Cashier

// NOT EXISTS

SELECT * FROM employee1 WHERE
NOT EXISTS (SELECT * FROM employee1 WHERE salary>20000);
no data found

// NATURAL JOIN

Syntax :
We will perform the natural join query by using the following syntax.
SELECT *

FROM TABLE1

NATURAL JOIN TABLE2;

Step-1:Creating Database :
create database saran;

Step-2: Using the database :
To use this database as follows.

use saran;

Step-3: Reference tables into the database :
This is our tables in the geeks database as follows.
Table-1: department –
Create Table department

```
(
  DEPT_NAME Varchar(20),
  MANAGER_NAME Varchar(255)
);
```

Table-2: employee –
Create Table employee

```
(
  EMP_ID  int,
  EMP_NAME Varchar(20),
  DEPT_NAME Varchar(255)
);
```

Step-4: Inserting values :
Add value into the tables as follows.

```
INSERT INTO DEPARTMENT(DEPT_NAME,MANAGER_NAME) VALUES ( "IT", "ROHAN");

INSERT INTO DEPARTMENT(DEPT_NAME,MANAGER_NAME) VALUES ( "SALES", "RAHUL");

INSERT INTO DEPARTMENT(DEPT_NAME,MANAGER_NAME) VALUES ( "HR", "TANMAY");

INSERT INTO DEPARTMENT(DEPT_NAME,MANAGER_NAME) VALUES ( "FINANCE", "ASHISH");

INSERT INTO DEPARTMENT(DEPT_NAME,MANAGER_NAME) VALUES ("MARKETING", "SAMAY");


INSERT INTO EMPLOYEE(EMP_ID, EMP_NAME, DEPT_NAME) VALUES (1, "SUMIT", "HR");

INSERT INTO EMPLOYEE(EMP_ID, EMP_NAME, DEPT_NAME) VALUES (2, "JOEL", "IT");

INSERT INTO EMPLOYEE(EMP_ID, EMP_NAME, DEPT_NAME) VALUES (3, "BISWA", "MARKETING");

INSERT INTO EMPLOYEE(EMP_ID, EMP_NAME, DEPT_NAME) VALUES (4, "VAIBHAV", "IT");
```

INSERT INTO EMPLOYEE(EMP_ID, EMP_NAME, DEPT_NAME) VALUES (5, "SAGAR", "SALES");

Step-5: Verifying inserted data :
This is our data inside the table as follows.
SELECT * FROM EMPLOYEE;

Output :

EMP_ID	EMP_NAME	DEPT_NAME
1	SUMIT	HR
2	JOEL	IT
3	BISWA	MARKETING
4	VAIBHAV	IT
5	SAGAR	SALES

SELECT * FROM DEPARTMENT;

Output :

DEPT_NAME	MANAGER_NAME
IT	ROHAN
SALES	RAHUL
HR	TANMAY
FINANCE	ASHISH
MARKETING	SAMAY

Step-6: Query to implement SQL Natural Join :

```
SELECT *
FROM EMPLOYEE
NATURAL JOIN DEPARTMENT;
```

Output :

EMP_ID	EMP_NAME	DEPT_NAME	MANAGER_NAME
1	SUMIT	HR	TANMAY
2	JOEL	IT	ROHAN
3	BISWA	MARKETING	SAMAY
4	VAIBHAV	IT	ROHAN
5	SAGAR	SALES	RAHUL

RESULT:

Databases are created and SQL queries are retrieved information successfully.

Ex.No:6	USER DEFINED FUNCTIONS AND STORED PROCEDURES IN SQL
Date:	

AIM:

Creation user defined functions and stored procedures by using Structures Query Language.

QUERY:

SQL UDFs

The following example creates a temporary SQL UDF named AddFourAndDivide and calls it from within a SELECT statement:

```
CREATE TEMP FUNCTION AddFourAndDivide(x INT64, y INT64)
RETURNS FLOAT64
AS (
  (x + 4) / y
);

SELECT
  val, AddFourAndDivide(val, 2)
FROM
  UNNEST([2,3,5,8]) AS val;
```

This example produces the following output:

```
+-----+-----+
| val | f0_ |
+-----+ ---- +
|   2 | 3.0 |
|   3 | 3.5 |
|   5 | 4.5 |
|   8 | 6.0 |
+-----+-----+
```

```
CREATE FUNCTION mydataset.AddFourAndDivide(x INT64, y INT64)
RETURNS FLOAT64
AS (
  (x + 4) / y
);
```

You must specify a dataset for the function (mydataset in this example). After you run the CREATE FUNCTION statement, you can call the function from a query:

```
SELECT
  val, mydataset.AddFourAndDivide(val, 2)
FROM
  UNNEST([2,3,5,8,12]) AS val;
```

Templated SQL UDF parameters

A parameter with a type equal to ANY TYPE can match more than one argument type when the function is called.

- If more than one parameter has type ANY TYPE, then BigQuery doesn't enforce any type relationship between these arguments.
- The function return type cannot be ANY TYPE. It must be either omitted, which means to be automatically determined based on sql_expression, or an explicit type.
- Passing the function arguments of types that are incompatible with the function definition will result in an error at call time.

The following example shows a SQL UDF that uses a templated parameter.

```
CREATE TEMP FUNCTION addFourAndDivideAny(x ANY TYPE, y ANY TYPE)
AS (
  (x + 4) / y
);

SELECT
  addFourAndDivideAny(3, 4) AS integer_input,
  addFourAndDivideAny(1.59, 3.14) AS floating_point_input;
```

This example produces the following output:

```
+---------------+-----------------------+
| integer_input  |  floating_point_input |
+---------------+-----------------------+
| 1.75          | 1.7802547770700636    |
+---------------+-----------------------+
```

The next example uses a templated parameter to return the last element of an array of any type:

```
CREATE TEMP FUNCTION lastArrayElement(arr ANY TYPE)
AS (
  arr[ORDINAL(ARRAY_LENGTH(arr))]
);

SELECT
  lastArrayElement(x) AS last_element
FROM (
  SELECT [2,3,5,8,13] AS x
);
```

This example produces the following output:

```
+--------------+
| last_element |
+--------------+
| 13           |
+--------------+
```

RESULT:

Databases are created user defined function and procedure in SQL queries are retrieved information successfully.

Ex.No:7	Execute Complex transactions and realize DCL and TCL Commands
Date:	

AIM:

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

DESCRIPTION:

The DCL language is used for controlling the access to the table and hence securing the Database. DCL is used to provide certain privileges to a particular user. Privileges are rights to be allocated. The privilege commands are namely,

- Grant
- Revoke
- Commit
- Savepoint
- Rollback

GRANT COMMAND: It is used to create users and grant access to the database. It requires database administrator (DBA) privilege, except that a user can change their password. A user can grant access to their database objects to other users.

REVOKE COMMAND: Using this command , the DBA can revoke the granted database privileges from the user.

COMMIT : It is used to permanently save any transaction into database.

SAVEPOINT: It is used to temporarily save a transaction so that you can rollback to that point whenever necessary

ROLLBACK: It restores the database to last commited state. It is also use with savepoint command to jump to a savepoint in a transaction.

COMMANDS

Consider the following tables namely “DEPARTMENTS” and “EMPLOYEES” Their schemas are as follows ,

Departments (dept _no , dept_ name , dept_location); Employees (emp_id , emp_name , emp_salary);

SQL> Grant all on employees to abcde; Grant succeeded.

SQL> Grant select , update , insert on departments to abcde with grant option; Grant succeeded.

SQL> Revoke all on employees from abcde; Revoke succeeded.

SQL> Revoke select , update , insert on departments from abcde; Revoke succeeded.

COMMIT, ROLLBACK and SAVEPOINT:

SQL> select * from class; NAME ID

anu 1

brindha 2

chinthiya 3

divya 4

ezhil 5

fairoz 7

SQL> insert into class values('gayathri',9); 1 row created.

SQL> commit; Commit complete.

SQL> update class set name='hema' where id='9'; 1 row updated.

SQL> savepoint A; Savepoint created.

SQL> insert into class values('indu',11); 1 row created.

SQL> savepoint B; Savepoint created.

SQL> insert into class values('janani',13); 1 row created.

SQL> select * from class;

NAME	ID
Anu	1
brindha	2
chinthiya	3
divya	4
ezhil	5
fairoz	7
hema	9
indu	11
janani	13

9 rows selected.
SQL> rollback to B; Rollback complete.

SQL> select * from class;

NAME	ID
Anu	1
Brindha	2
Chinthiya	3
Divya	4
Ezhil	5
Fairoz	7
Hema	9
Indu	11

8 rows selected.

SQL> rollback to A; Rollback complete.

SQL> select * from class;

NAME	ID
anu	1
brindha	2
chinthiya	3
divya	4
ezhil	5
fairoz	7
hema	9

RESULT:

Thus the creation of database with various constraints and the SQL queries to retrieve information from the database using DCL statements has been implemented and the output was verified successfully.

Ex.No:8	SQL TRIGGERS FOR INSERT, DELETE, AND UPDATE OPERATIONS IN DATABASE TABLE
Date:	

AIM:

To Create a SQL triggers for insert, delete and update operations in database table

TRIGGER

SYNTAX:

```
CREATE TRIGGER trigger_name
{BEFORE | AFTER} {INSERT | UPDATE| DELETE }
ON table_name FOR EACH ROW
trigger_body;
```

EXAMPLE-1:

```
mysql>select * from emp;
```

empno	empname	offcode
100	kalai	111
101	rama	222
102	began	333

```
mysql>create table emp_audit(id int auto_increment primary key,
        empno int,empname varchar(20),cdate datetime,action varchar(20));
```

TRIGGER CREATION :

```
mysql>create trigger before_emp_update
        before update on emp
        for each row
        insert into emp_audit
        set action = 'update',
        empno = old.empno,
        empname = old.empname,
        cdate = now();
```

UPDATE EMP TABLE:

```
mysql> update emp set empname = 'gokul' where empno = 100;
```

SELECT-EMP_AUDIT

mysql> select * from emp_audit;

id	empno	empname	cdate	action
1	100	gokul	2019-12-12 13-02-17	update

SELECT –EMP:

mysql> select * from emp1;

Empno	empname	offcode
100	gokul	111
101	rama	222
102	began	333

RESULT:

Thus, the Trigger was executed successfully.

Ex.No:9	CREATE VIEW AND INDEX FOR DATABASE TABLES WITH LARGE NUMBER OF RECORDS
Date:	

Aim:

To Create the View and index by using Structures Query Language.

Syntax :

CREATE VIEW view_name AS SELECT column_name(s)

FROM table_name WHERE condition

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>

QUERY:

create table student(sid int, sname varchar(50), dept varchar(5));

OUTPUT:

table STUDENT created.

QUERY:

insert into student values(1,'bala','IT');

insert into student values(2,'rupesh','IT');

insert into student values(3,'arthi','cse');

OUTPUT:

1 rows inserted.

1 rows inserted.

1 rows inserted.

QUERY:

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

OUTPUT:

view STUDVIEW created.

QUERY:

select * from studview;

OUTPUT:

SID	SNAME	DEPT
1	bala	IT
2	rupesh	IT

QUERY:

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

OUTPUT:

1 rows inserted.

QUERY:

select * from studview;

OUTPUT:

SID	SNAME	DEPT
1	bala	IT
2	rupesh	IT

7	anand	IT

QUERY:

update studview set sid=5 where sid=2;

OUTPUT:

1 rows updated.

QUERY:

select * from studview;

OUTPUT:

SID	SNAME	DEPT
1	bala	IT
5	rupesh	IT
7	anand	IT

QUERY:

delete from studview where sid = 5;

OUTPUT:

1 rows deleted

QUERY:

select * from studview;

OUTPUT:

SID	SNAME	DEPT
1	bala	IT
7	art	IT

QUERY:

drop view studview;

OUTPUT:

view STUDVIEW dropped.

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;

QUERY:

create index idx_stud on student(sid);

OUTPUT:

index IDX_STUD created.

QUERY:

drop index idx_stud;

OUTPUT:

index IDX_STUD dropped.

Result:

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

Ex.No:10	CREATE AN XML DATABASE AND VALIDATE IT USING XML SCHEMA.
Date:	

Aim:

To Create an xml database and validate it using XML Schema.

Query:

Register an XML Schema

First we needto register the schema using the DBMS_XMLSCHEMA.REGISTERSCHEMA procedure. This has a number of overloads, allowing you to specify the XML schema using a VARCHAR2, BFILE, BLOB, CLOB, XMLTYPE or URIType types. The parameter list is quite important. By default the REGISTERSCHEMA procedure will create database types and object tables, allowing documents to be shredded into object tables. For complex XSD documents you might get thousands of objects created in your database schema. For a basic document validation this is unnecessary and messy, so check you are using the correct settings.

```
DECLARE
  l_schema CLOB;
BEGIN

  l_schema := '<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<!-- definition of simple elements -->
<xs:element name="orderperson" type="xs:string"/>
<xs:element name="name" type="xs:string"/>
<xs:element name="address" type="xs:string"/>
<xs:element name="city" type="xs:string"/>
<xs:element name="country" type="xs:string"/>
<xs:element name="title" type="xs:string"/>
<xs:element name="note" type="xs:string"/>
<xs:element name="quantity" type="xs:positiveInteger"/>
<xs:element name="price" type="xs:decimal"/>

<!-- definition of attributes -->
<xs:attribute name="orderid" type="xs:string"/>

<!-- definition of complex elements -->
<xs:element name="shipto">
```

```
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="name"/>
      <xs:element ref="address"/>
      <xs:element ref="city"/>
      <xs:element ref="country"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="item">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="title"/>
      <xs:element ref="note" minOccurs="0"/>
      <xs:element ref="quantity"/>
      <xs:element ref="price"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="shiporder">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="orderperson"/>
      <xs:element ref="shipto"/>
      <xs:element ref="item" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute ref="orderid" use="required"/>
  </xs:complexType>
</xs:element>

</xs:schema>';

   DBMS_XMLSCHEMA.registerSchema(schemaurl        => 'my_schema.xsd',
                         schemadoc       => l_schema,
                         local           => TRUE,
                         gentypes        => FALSE,
                         gentables       => FALSE,
                         enablehierarchy => DBMS_XMLSCHEMA.enable_hierarchy_none);

END;
/
```

We can check the schema details using the USER_XML_SCHEMAS view.

```
SELECT schema_url FROM user_xml_schemas;

SCHEMA_URL
--------------------------------------------------------------------------------
my_schema.xsd

SQL>
```

With the schema registered, we can now validate XML documents against it.

The DELETESCHEMA procedure can be used to un-register the schema. The DELETE_OPTION parameter allows you to drop any dependent objects.

```
BEGIN
  DBMS_XMLSCHEMA.deleteschema(
    schemaurl     => 'my_schema.xsd',
    delete_option => DBMS_XMLSCHEMA.delete_cascade_force);
END;
/
```

Validate XML Document (SCHEMAVALIDATE)

In the following PL/SQL example, we create an XMLTYPE from some XML in a CLOB, then call the SCHEMAVALIDATE member procedure to test the XML against the XML schema.

```
DECLARE
  l_xml      CLOB;
l_xmltype XMLTYPE;
BEGIN

  l_xml := '<?xml version="1.0" encoding="UTF-8"?>
<shiporder orderid="889923">
  <orderperson>John Smith</orderperson>
  <shipto>
    <name>Ola Nordmann</name>
    <address>Langgt 23</address>
    <city>4000 Stavanger</city>
    <country>Norway</country>
  </shipto>
  <item>
    <title>Empire Burlesque</title>
    <note>Special Edition</note>
    <quantity>1</quantity>
    <price>10.90</price>
```

```
  </item>
  <item>
    <title>Hide your heart</title>
    <quantity>1</quantity>
    <price>9.90</price>
  </item>
</shiporder>';

  l_xmltype := XMLTYPE(l_xml, 'my_schema.xsd');
  l_xmltype.schemavalidate;

END;
/

PL/SQL procedure successfully completed.

SQL>
```

To see an example of a validation failure, rename the "name" tag to "name1". This is not part of the XML schema, so it will fail the validation.

```
DECLARE
  l_xml      CLOB;
l_xmltype XMLTYPE;
BEGIN

  l_xml := '<?xml version="1.0" encoding="UTF-8"?>
<shiporder orderid="889923">
  <orderperson>John Smith</orderperson>
  <shipto>
    <name1>Ola Nordmann</name1>
    <address>Langgt 23</address>
    <city>4000 Stavanger</city>
    <country>Norway</country>
  </shipto>
  <item>
    <title>Empire Burlesque</title>
    <note>Special Edition</note>
    <quantity>1</quantity>
    <price>10.90</price>
  </item>
  <item>
    <title>Hide your heart</title>
    <quantity>1</quantity>
    <price>9.90</price>
```

```
  </item>
</shiporder>';

  l_xmltype := XMLTYPE(l_xml, 'my_schema.xsd');
  l_xmltype.schemavalidate;

END;
/

DECLARE
*
ERROR at line 1:
ORA-30937: No schema definition for 'name1' (namespace '##local') in parent
'/shiporder/shipto'
ORA-06512: at "SYS.XMLTYPE", line 354
ORA-06512: at line 29


SQL>
```

Validate XML Document (XMLISVALID)

An alternative is to use the XMLISVALID function.

Create a table to hold the XML we used in the previous tests.

```
CREATE TABLE t1 (
  id NUMBER,
  xml XMLTYPE
);

INSERT INTO t1 VALUES (1, '<?xml version="1.0" encoding="UTF-8"?>
<shiporder orderid="889923">
  <orderperson>John Smith</orderperson>
  <shipto>
    <name>Ola Nordmann</name>
    <address>Langgt 23</address>
    <city>4000 Stavanger</city>
    <country>Norway</country>
  </shipto>
  <item>
    <title>Empire Burlesque</title>
    <note>Special Edition</note>
    <quantity>1</quantity>
    <price>10.90</price>
```

```
  </item>
  <item>
    <title>Hide your heart</title>
    <quantity>1</quantity>
    <price>9.90</price>
  </item>
</shiporder>');


INSERT INTO t1 VALUES (2, '<?xml version="1.0" encoding="UTF-8"?>
<shiporder orderid="889923">
  <orderperson>John Smith</orderperson>
  <shipto>
    <name1>Ola Nordmann</name1>
    <address>Langgt 23</address>
    <city>4000 Stavanger</city>
    <country>Norway</country>
  </shipto>
  <item>
    <title>Empire Burlesque</title>
    <note>Special Edition</note>
    <quantity>1</quantity>
    <price>10.90</price>
  </item>
  <item>
    <title>Hide your heart</title>
    <quantity>1</quantity>
    <price>9.90</price>
  </item>
</shiporder>');

COMMIT;
```

Now we can use the XMLISVALID function in SQL to test against the registered XML Schema. It returns a "1" if the XML is valid, and "0" if it isn't.

```
SELECT id,
       XMLISVALID(xml, 'my_schema.xsd') AS is_valid
FROM  t1;

        ID  IS_VALID
------------ ------------
         1          1
         2          0
```

```
SQL>
```

Result:

Thus, the XML database created and validated successfully.

Ex.No:11	CREATE DOCUMENT, COLUMN AND GRAPH BASED DATA USING NOSQL DATABASE TOOLS
Date:	

Aim:

To write NoSQL queries using Column Graph based data commands to manage the database.

Query:

The examples on this page use the `inventory` collection. Populate the `inventory` collection with the following documents:

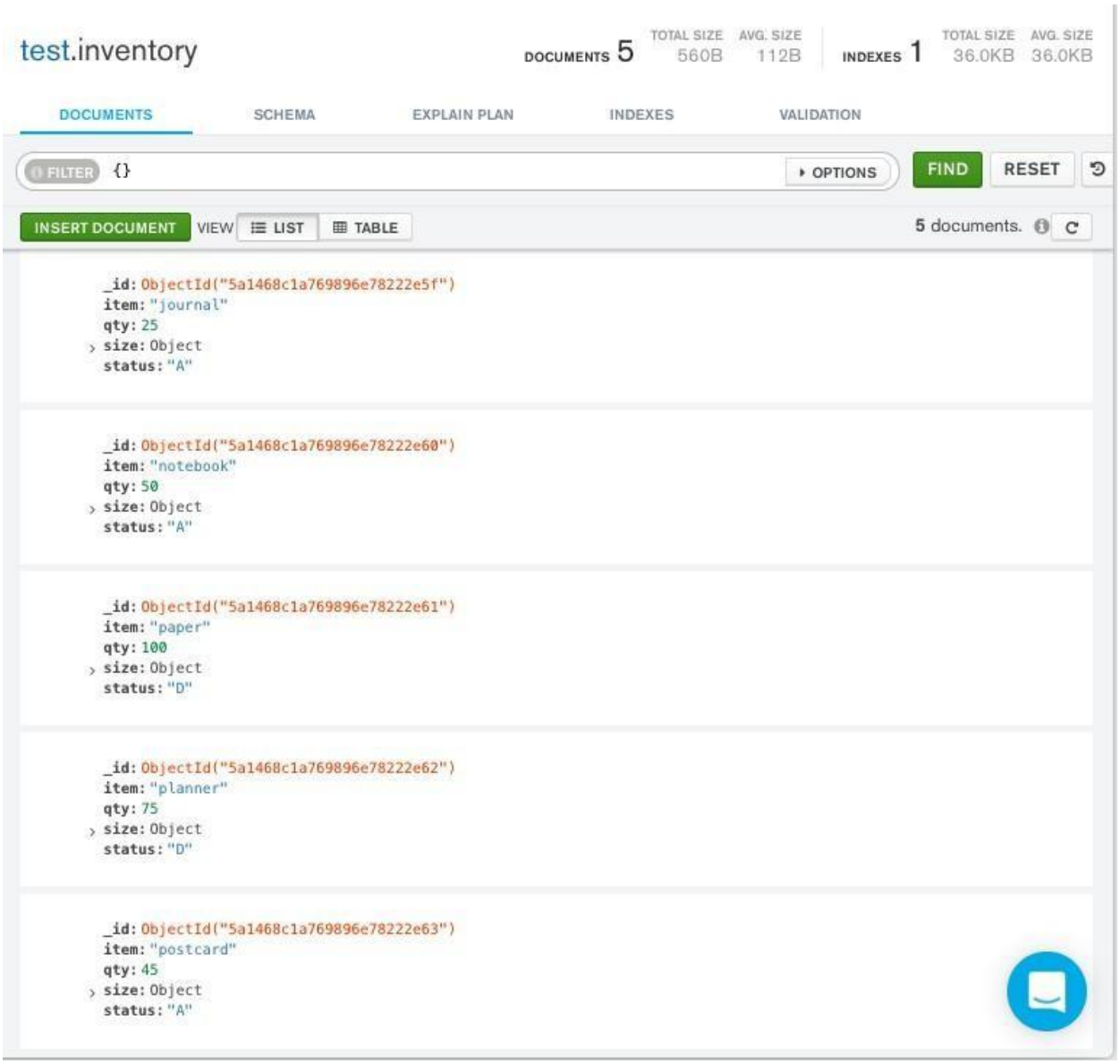
```
[
{ "item": "journal", "qty": 25, "size": { "h": 14, "w": 21, "uom": "cm" }, "status": "A" },
{ "item": "notebook", "qty": 50, "size": { "h": 8.5, "w": 11, "uom": "in" }, "status": "A" },
{ "item": "paper", "qty": 100, "size": { "h": 8.5, "w": 11, "uom": "in" }, "status": "D" },
{ "item": "planner", "qty": 75, "size": { "h": 22.85, "w": 30, "uom": "cm" }, "status": "D" },
{ "item": "postcard", "qty": 45, "size": { "h": 10, "w": 15.25, "uom": "cm" }, "status": "A" }
]
```

Compass

For instructions on inserting documents in MongoDB Compass, see Insert Documents.

Select All Documents in a Collection

To select all documents in the collection, pass an empty document as the query filter parameter to the query bar. The query filter parameter determines the select criteria:



This operation uses a filter predicate of `{}`, which corresponds to the following SQL statement:

```
SELECT * FROM inventory
```

For more information on the MongoDB Compass query bar, see Query Bar.

Specify Equality Condition

To specify equality conditions, use `<field>:<value>` expressions in the query filter document:

```
{ <field1>: <value1>, ... }
```

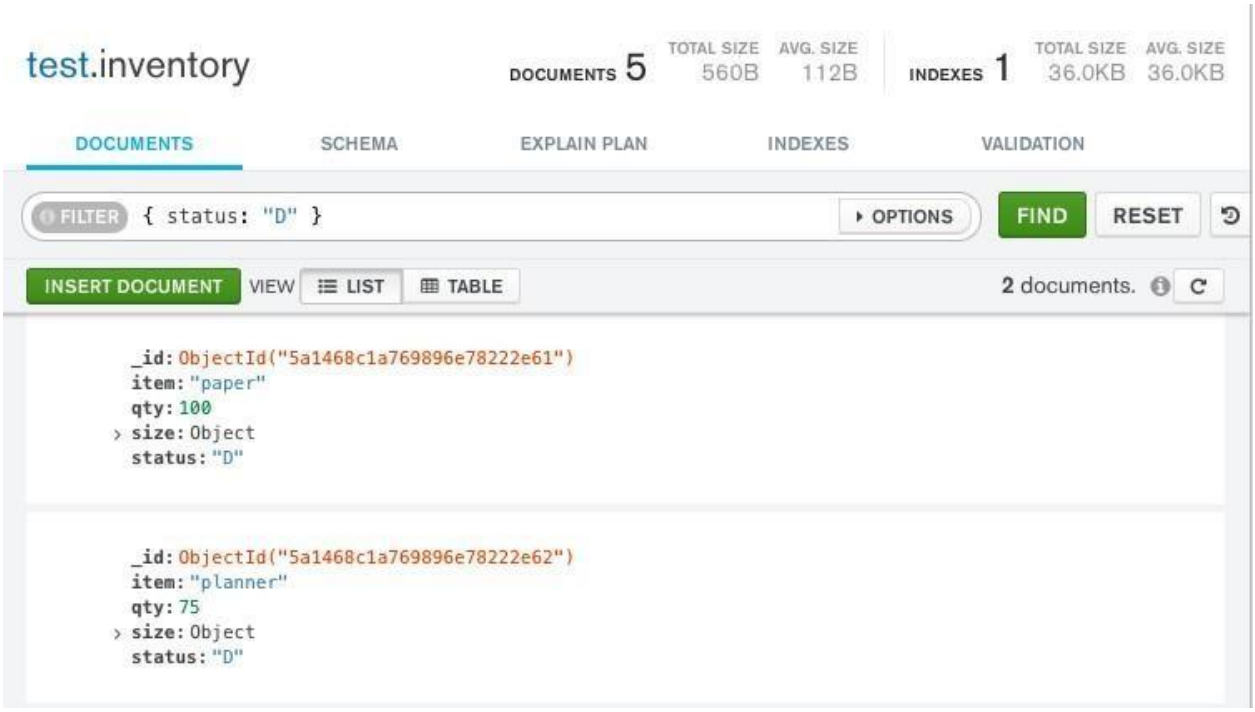
Compass

The following example selects from the `inventory` collection all documents where the `status` equals `"D"`:

Copy the following filter into the Compass query bar and click **Find**:

```
{ status: "D" }
```

Compass



This operation uses a filter predicate of `{ status: "D" }`, which corresponds to the following SQL statement:

```
SELECT * FROM inventory WHERE status = "D"
```

Specify Conditions Using Query Operators

A query filter document can use the query operators to specify conditions in the following form:

```
{ <field1>: { <operator1>: <value1> }, ... }
```

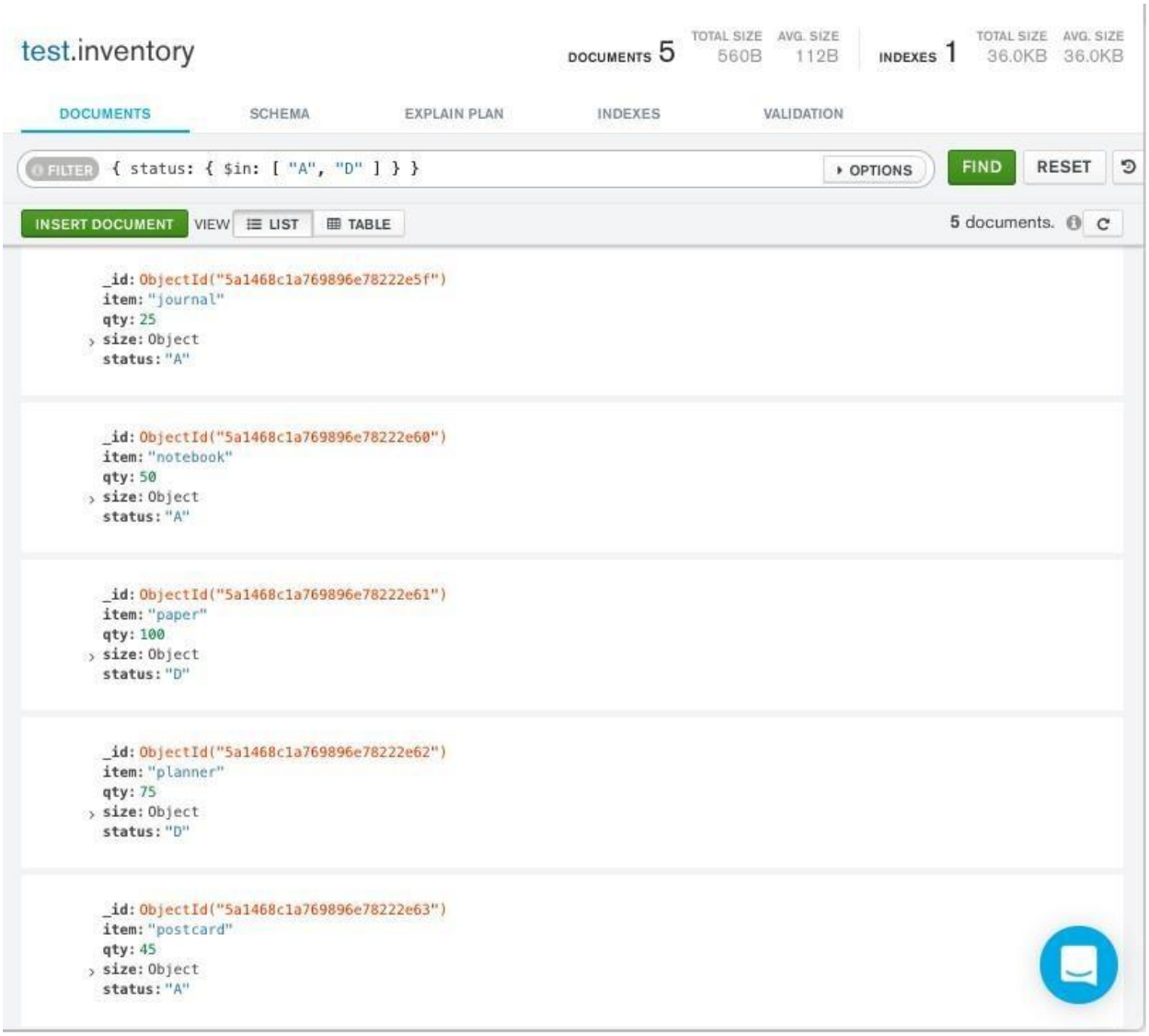
Compass

The following example retrieves all documents from the `inventory` collection where `status` equals either `"A"` or `"D"`:

Copy the following filter into the Compass query bar and click **Find**:

```
{ status: { $in: [ "A", "D" ] } }
```

Compass



NOTE

Although you can express this query using the $or operator, use the $in operator rather than the $or operator when performing equality checks on the same field.

The operation uses a filter predicate of which corresponds to the following SQL statement:

SELECT * FROM inventory WHERE status in ("A", "D")

Refer to the Query and Projection Operators document for the complete list of MongoDB query operators.

Specify AND Conditions

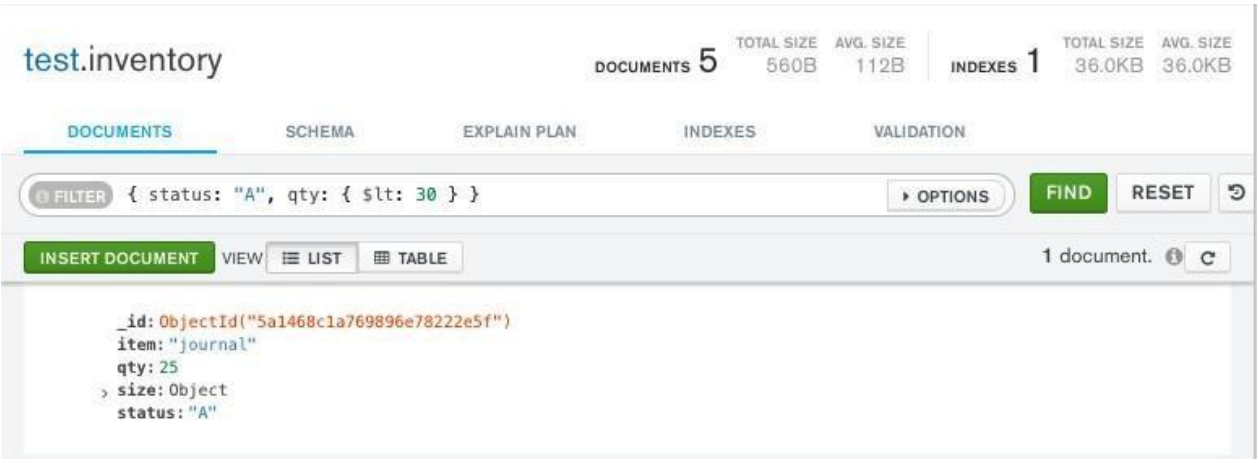
A compound query can specify conditions for more than one field in the collection's documents. Implicitly, a logical AND conjunction connects the clauses of a compound query so that the query selects the documents in the collection that match all the conditions.

The following example retrieves all documents in the inventory collection where the status equals "A" **and** qty is less than ($lt) 30:

Copy the following filter into the Compass query bar and click **Find**:

```
{ status: "A", qty: { $lt: 30 } }
```

Compass



The operation uses a filter predicate of { status: "A", qty: { $lt: 30 } }, which corresponds to the following SQL statement:

```
SELECT * FROM inventory WHERE status = "A" AND qty < 30
```

See comparison operators for other MongoDB comparison operators.

Specify `OR` Conditions

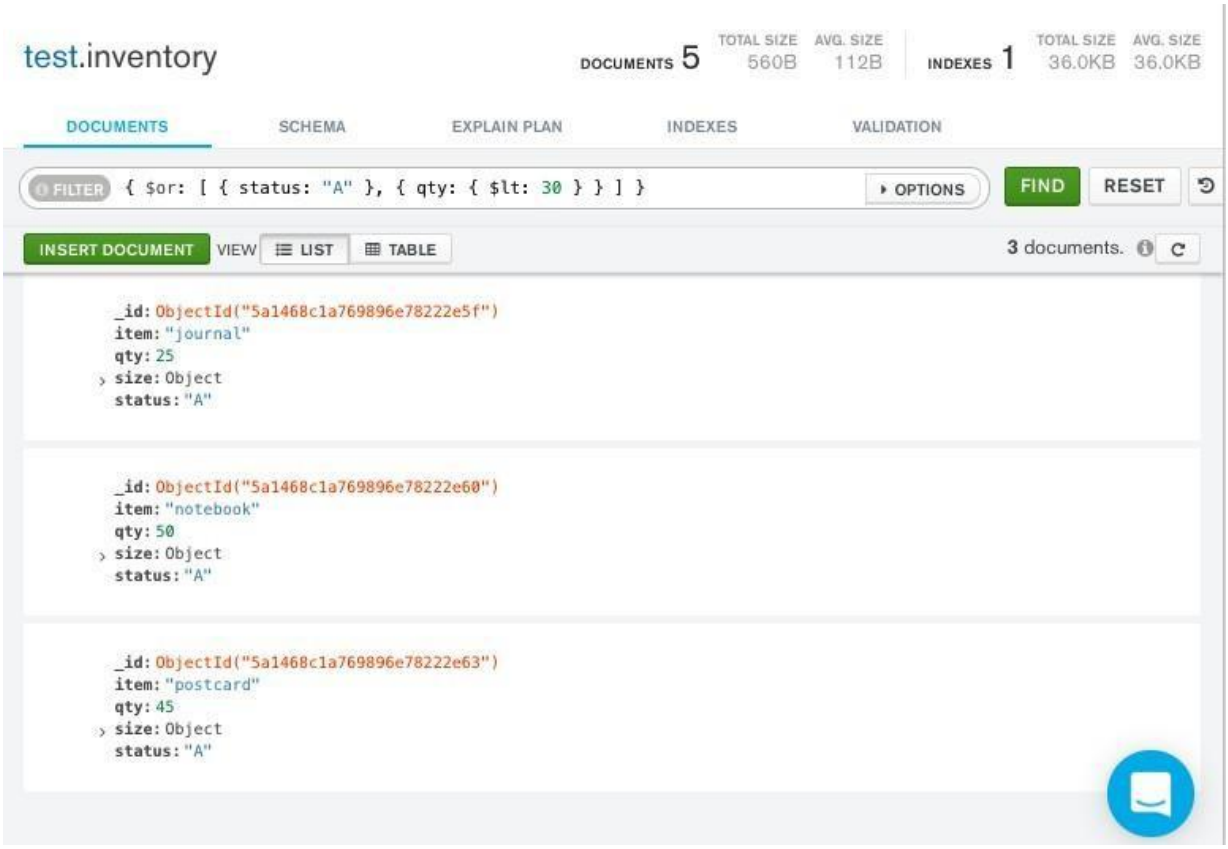
Using the `$or` operator, you can specify a compound query that joins each clause with a logical `OR` conjunction so that the query selects the documents in the collection that match at least one condition.

The following example retrieves all documents in the collection where the `status` equals `"A"` **or** `qty` is less than (`$lt`) `30`:

Copy the following filter into the Compass query bar and click **Find**:

```
{ $or: [ { status: "A" }, { qty: { $lt: 30 } } ] }
```

Compass



The operation uses a filter predicate of `{ $or: [ { status: 'A' }, { qty: { $lt: 30 } } ] }`, which corresponds to the following SQL statement:

```
SELECT * FROM inventory WHERE status = "A" OR qty < 30
```

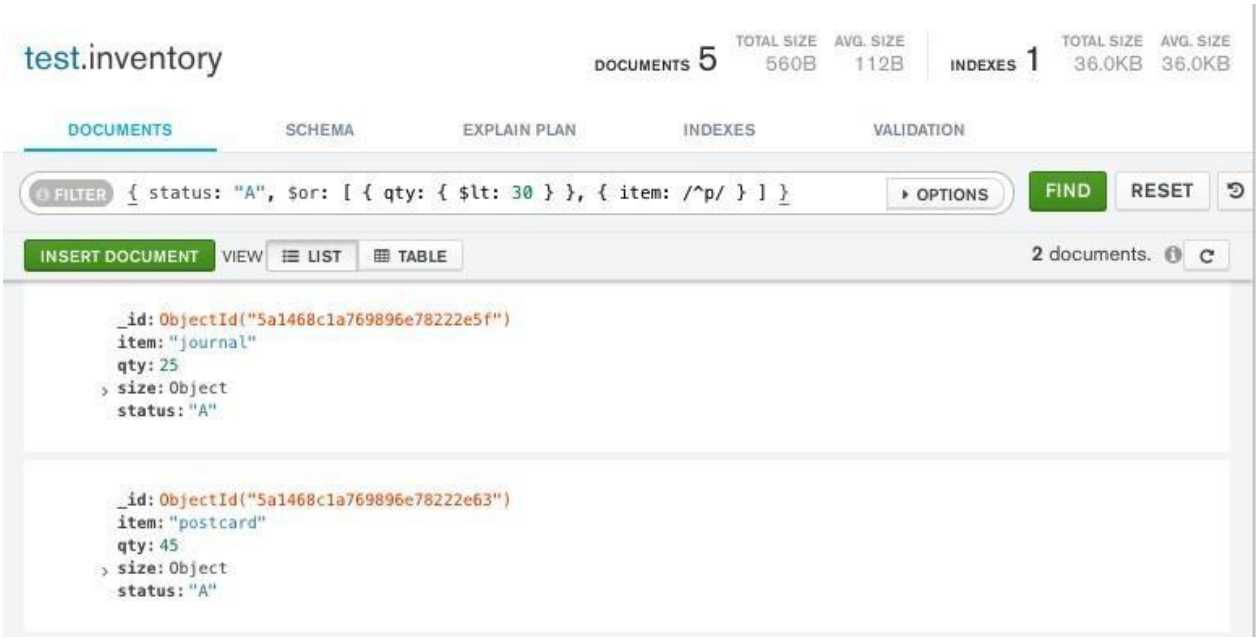
Specify `AND` as well as `OR` Conditions

In the following example, the compound query document selects all documents in the collection where the `status` equals `"A"` **and** *either* `qty` is less than (`$lt`) `30` *or* `item` starts with the character `p`:

Copy the following filter into the Compass query bar and click **Find**:

```
{ status: "A", $or: [ { qty: { $lt: 30 } }, { item: /^p/ } ] }
```

Compass



The operation uses a filter predicate of:

```
{
```

```
status: 'A',
$or: [
{ qty: { $lt: 30 } }, { item: { $regex: '^p' } }
]
}
```

which corresponds to the following SQL statement:

```
SELECT * FROM inventory WHERE status = "A" AND ( qty < 30 OR item LIKE "p%")
```

Result:

Thus, the graph-based data creation using NoSQL executed successfully

Ex.No:12	**DEVELOP A SIMPLE GUI BASED DATABASE APPLICATION**
Date:	

AIM:

To develop a simple GUI based database application for Personal Information System.

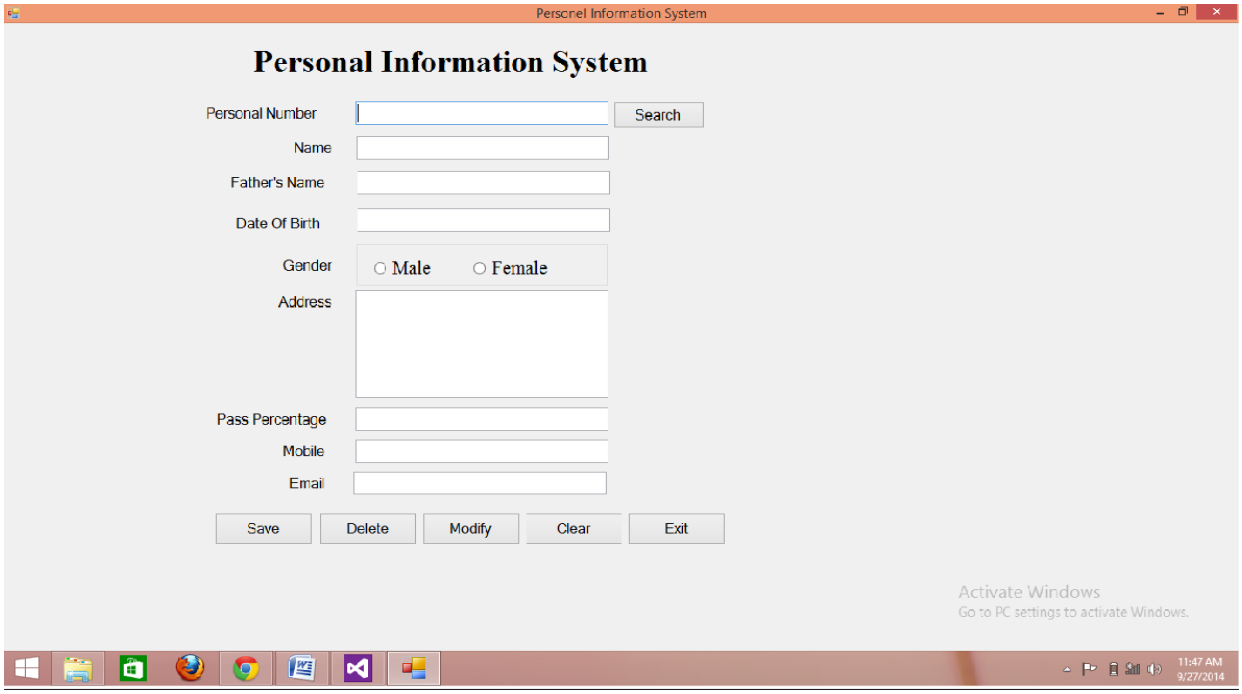
DESCRIPTION:

Personal Information System is a project which stores information about a particular person. It allows to insert, delete, modify and clear the records. Records are updated in the database through forms.

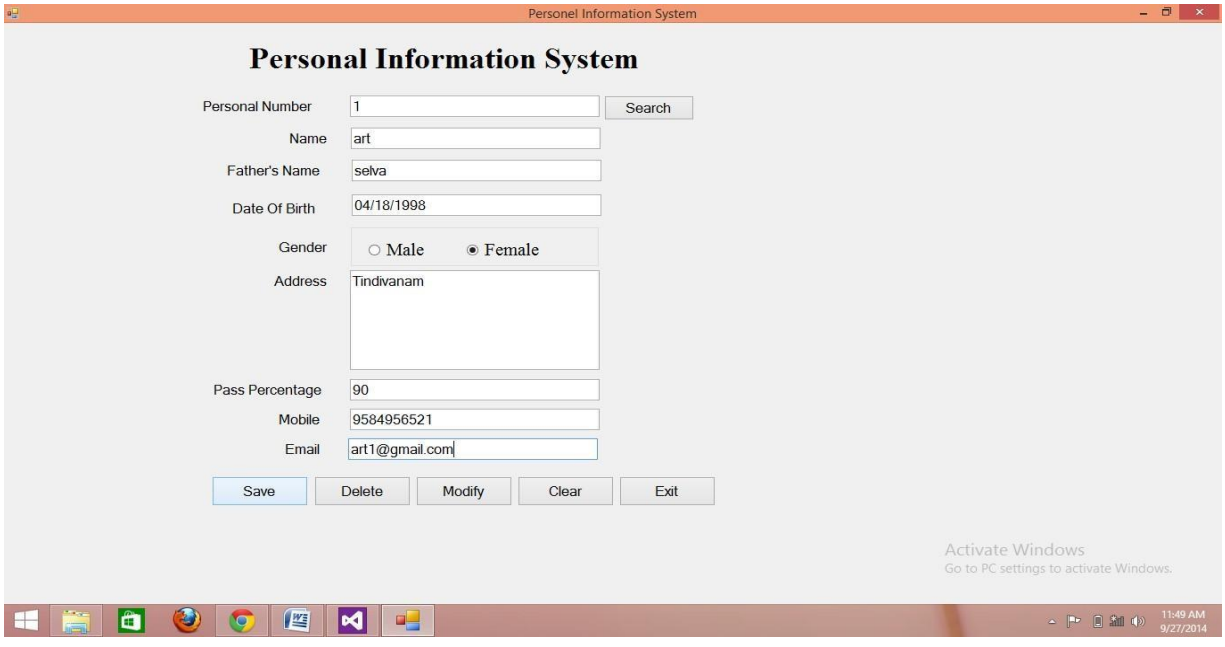
DATABASE:

```
CREATE TABLE personaldetail
(
        pid INT PRIMARY KEY,
        name VARCHAR2(40),
        father VARCHAR2(40),
        DOB DATE,
        gender VARCHAR2(6),
        address VARCHAR2(200),
        mark NUMBER(3,0),
        mobile NUMBER(10,0),
        email VARCHAR2(20)
)
```

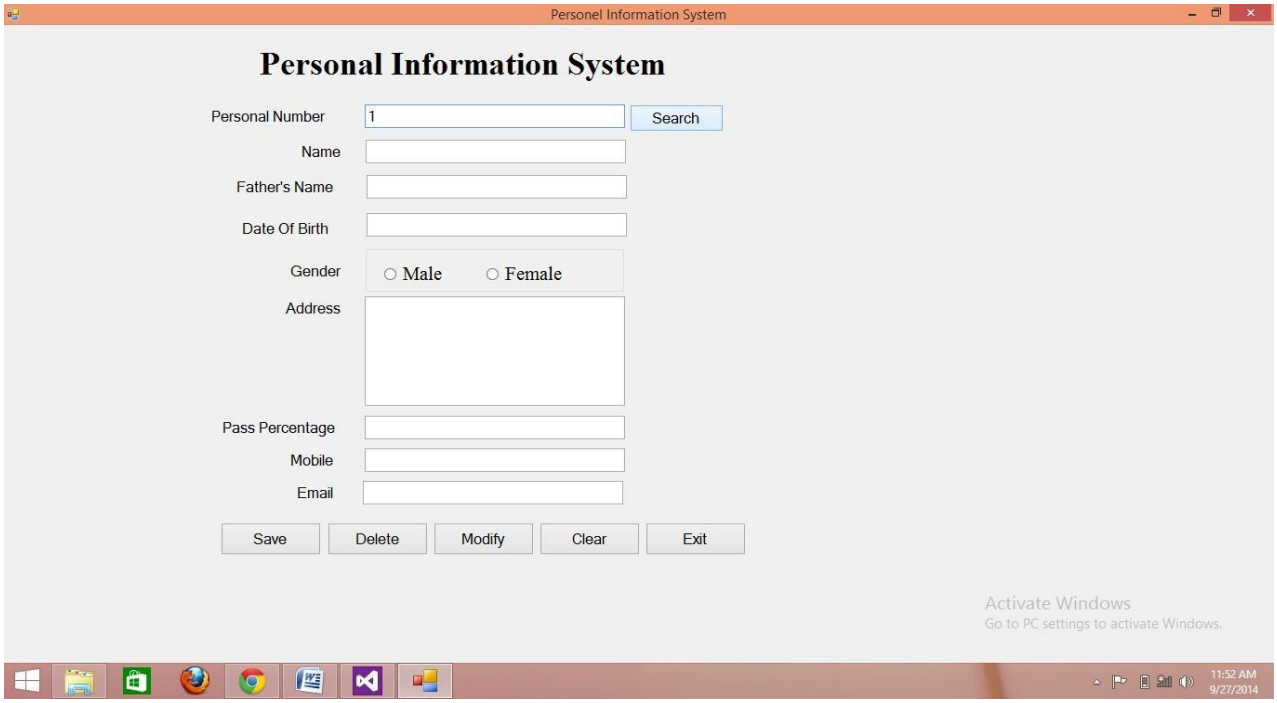
FORMS:



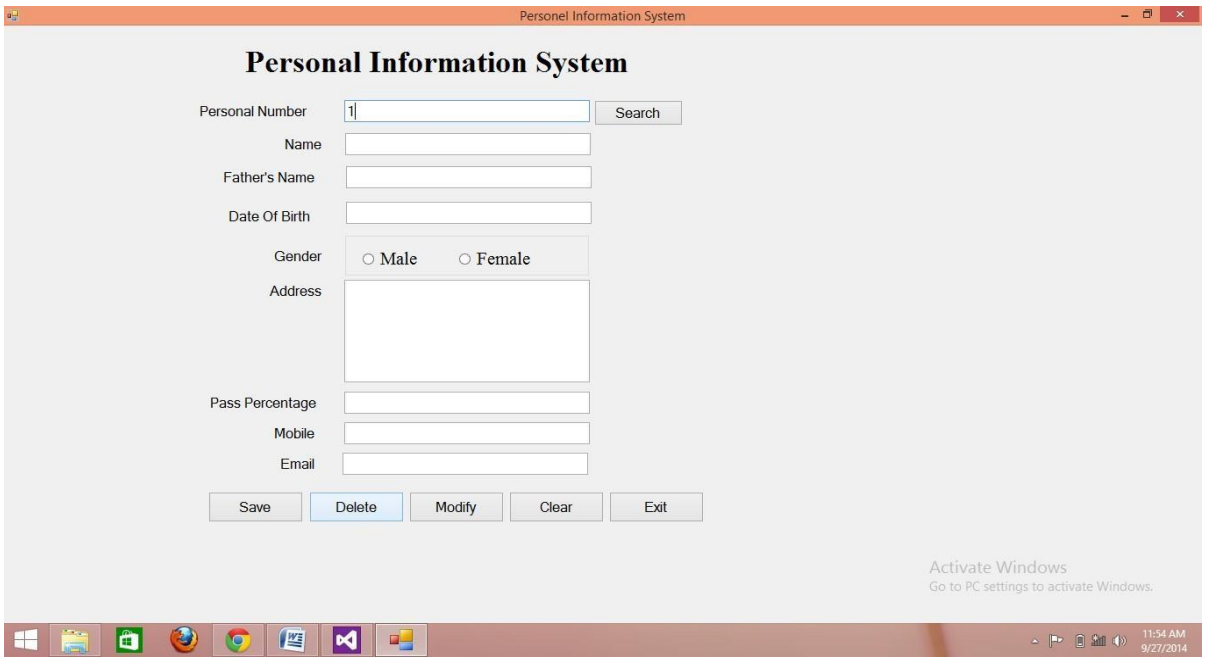
➔ TO INSERT THE RECORD CLICK SAVE



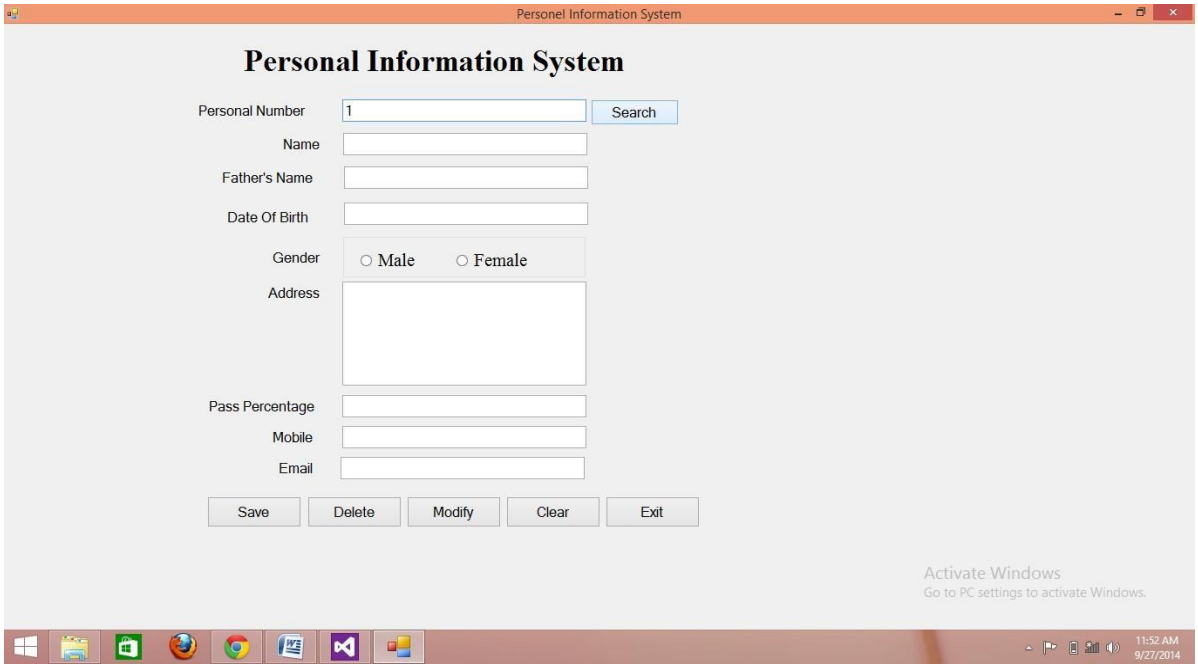
➔ TO SEARCH ENTER THE PERSONAL NUMBER AND CLICK SEARCH

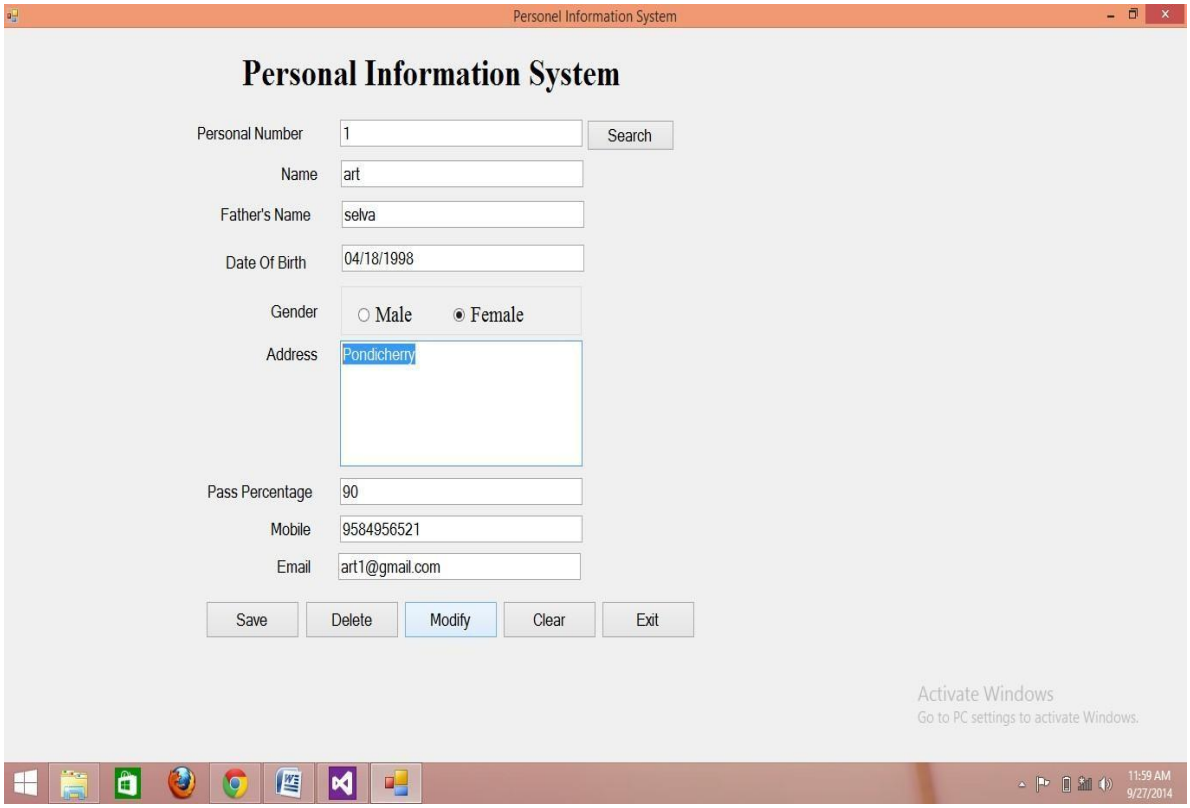


➔ TO DELETE ENTER THE PERSONAL NUMBER AND CLICK DELETE

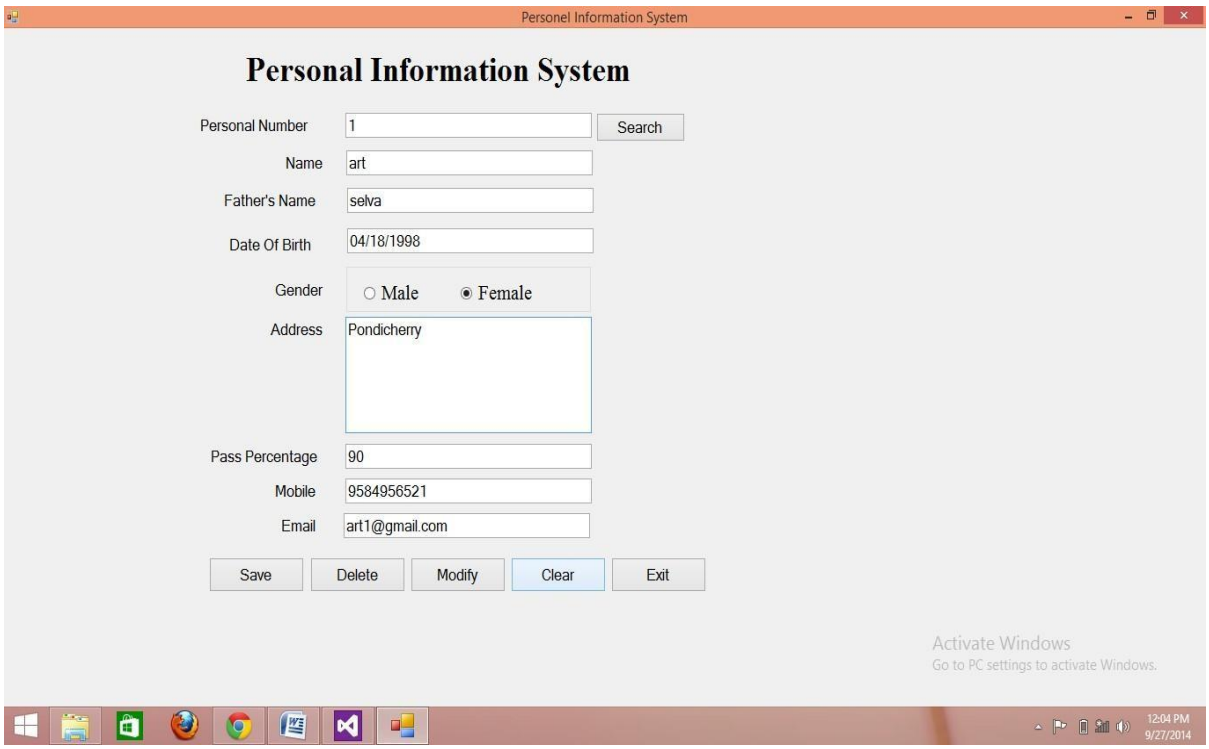


➔ TO MODIFY ENTER THE PERSONAL NUMBER AND CLICK SEARCH THEN PERFORM UPDATIONS AND CLICK MODIFY

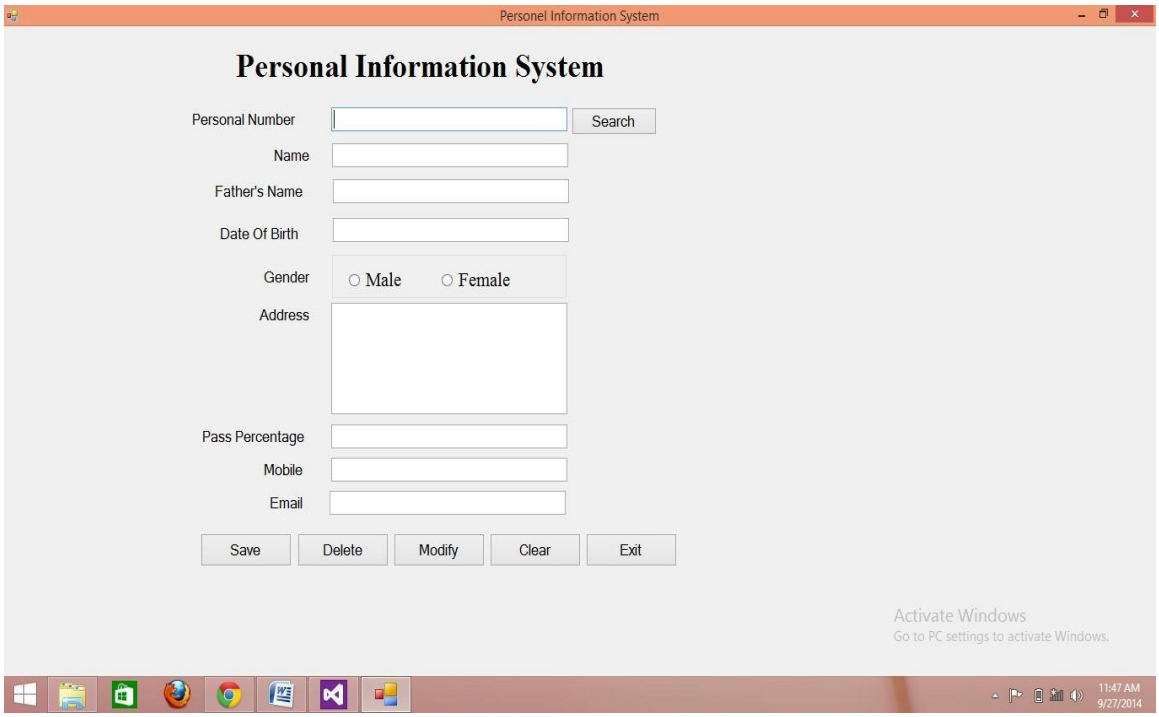




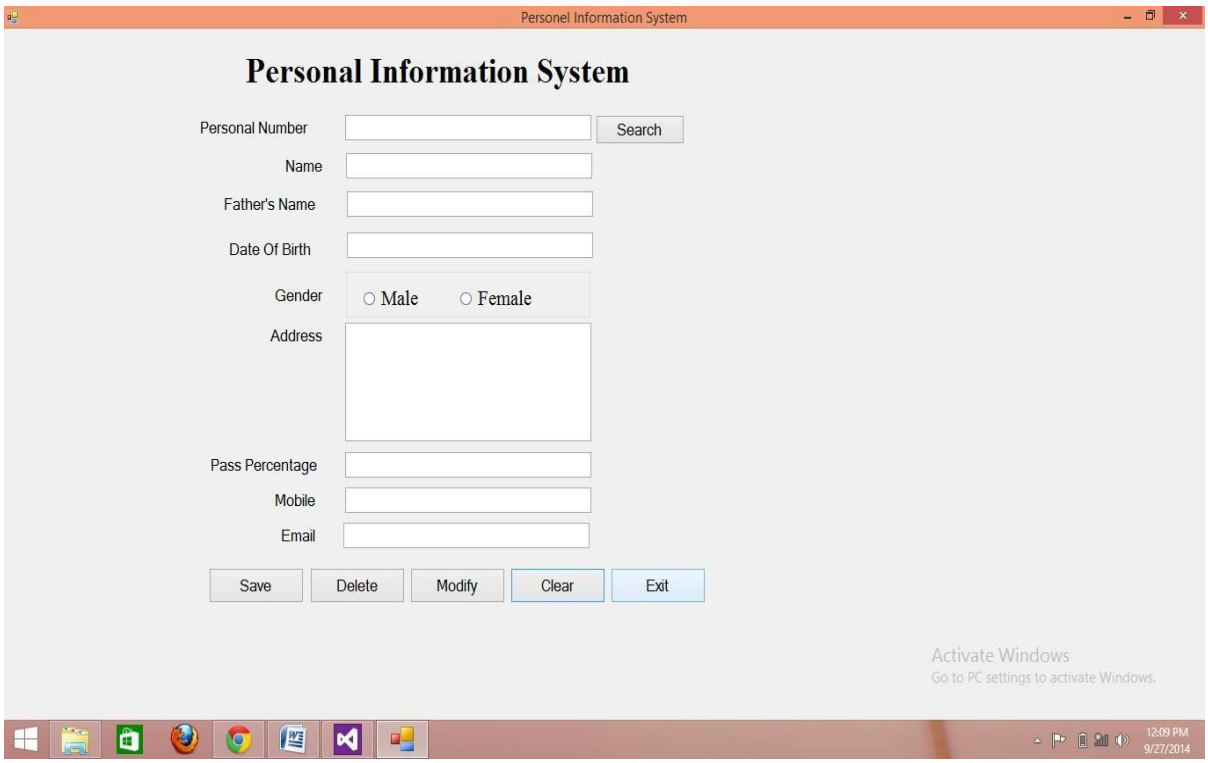
➔ TO CLEAR THE BOX CLICK THE CLEAR BUTTON



➔ AFTER CLICKING THE CLEAR BUTTON



➔ TO EXIT PRESS THE EXIT BUTTON



CODING:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using Oracle.DataAccess.Client;
namespace PersonelInformationSystem
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }
//CONNECTIVITY CODING
      OracleConnection conn;
      private void Form1_Load(object sender, EventArgs e)
```

```
        {
            string oradb = "Data Source=XE;User Id=art;Password=123;";
            conn = new OracleConnection(oradb);
            conn.Open();
        }
    //CLEAR FUNCTION
        private void Clear()
        {
            txtPID.Text = "";
            txtName.Text = "";
            txtFather.Text = "";
            txtDOB.Text = "";
            rdoMale.Checked = false;
            rdoFemale.Checked = false;
            txtAddress.Text = "";
            txtMark.Text = "";
            txtMobile.Text = "";
            txtEmail.Text = "";
        }
    //INSERTION CODING
        // Do NOT change the order of the parameter and values
        private void btnSave_Click(object sender, EventArgs e)
        {
            string gender = " ";
            if (rdoMale.Checked)
            {
                gender = "Male";
            }
            else if (rdoFemale.Checked)
            {
                gender = "Female";
            }
            DateTime dt = Convert.ToDateTime(txtDOB.Text);
                string sql = "INSERT INTO sankar.personaldetail
                (pid,name,father,dob,gender,address,mark,mobile,email) VALUES
                (:pid,:name,:father,:dob,:gender,:address,:mark,:mobile,:email)";
            OracleCommand cmd = new OracleCommand(sql, conn);
            cmd.Parameters.Add("pid", txtPID.Text);
            cmd.Parameters.Add("name", txtName.Text);
            cmd.Parameters.Add("father", txtFather.Text);
            cmd.Parameters.Add("dob", dt);
            cmd.Parameters.Add("gender", gender);
            cmd.Parameters.Add("address", txtAddress.Text);
            cmd.Parameters.Add("mark", txtMark.Text);
            cmd.Parameters.Add("mobile", txtMobile.Text);
            cmd.Parameters.Add("email", txtEmail.Text);
            cmd.ExecuteNonQuery();
            MessageBox.Show("inserted sucessfully");
            Clear();
        }
```

```
//UPDATION CODING
    // Do NOT change the order of the parameter and values
    private void btnEdit_Click(object sender, EventArgs e)
    {
       string gen = " ";
       if (rdoMale.Checked)
       {
          gen = rdoMale.Text;
       }
       else if (rdoFemale.Checked)
       {
          gen = rdoFemale.Text;
       }
       DateTime dt = Convert.ToDateTime(txtDOB.Text);

            string sql = "UPDATE sankar.personaldetail SET
            name=:name,father=:father,dob=:dob,gender=:gender,address=:address,mark=:mark,
            mobile=:mobile,email=:email WHERE pid=:pid";
       OracleCommand cmd = new OracleCommand(sql, conn);
       cmd.Parameters.Add("name", txtName.Text);
       cmd.Parameters.Add("father", txtFather.Text);
       cmd.Parameters.Add("dob", dt);
       cmd.Parameters.Add("gender", gen);
       cmd.Parameters.Add("address", txtAddress.Text);
       cmd.Parameters.Add("mark", txtMark.Text);
       cmd.Parameters.Add("mobile", txtMobile.Text);
       cmd.Parameters.Add("email", txtEmail.Text);
       cmd.Parameters.Add("pid", txtPID.Text);
       cmd.ExecuteNonQuery();
       MessageBox.Show("Updated successfully");
       Clear();
    }

//DELETION CODING
    private void btnDelete_Click(object sender, EventArgs e)
    {
       OracleCommand cmd = new OracleCommand("DELETE FROM sankar.personaldetail
WHERE
       pid=:pid", conn);
       cmd.Parameters.Add("pid", txtPID.Text);
       cmd.ExecuteNonQuery();
       MessageBox.Show("Message deleted succesfully");
       Clear();
    }
//CLEAR CODING
    private void btnClear_Click(object sender, EventArgs e)
    {
       Clear();
    }
```

```
//SEARCH CODING
      private void btnSearch_Click(object sender, EventArgs e)
      {
         OracleCommand cmd = new OracleCommand("SELECT * FROM sankar.personaldetail
WHERE
             pid=:pid1", conn);
         cmd.Parameters.Add("pid1", txtPID.Text);
         OracleDataReader dr = cmd.ExecuteReader();

         if (dr.Read())
         {
            txtPID.Text = dr["pid"].ToString();
            txtName.Text = dr["name"].ToString();
            txtFather.Text = dr["father"].ToString();
            txtDOB.Text = dr["dob"].ToString();
            if (dr["gender"].ToString() == "Male")
            {                rdoMale.Checked = true;              }
            else
            {                rdoFemale.Checked = true;               }
            txtAddress.Text = dr["address"].ToString();
            txtMark.Text = dr["mark"].ToString();
            txtMobile.Text = dr["mobile"].ToString();
            txtEmail.Text = dr["email"].ToString();
         }
         else
         {
            MessageBox.Show("Record not found ");
         }
      }
//EXIT CODING
      private void btnExit_Click(object sender, EventArgs e)
      {
         Close();       }          }  }
```

RESULT:

Thus the Coding was executed successfully.

Ex.No:13 Date:	**Case Study (Inventory Management for a EMart Grocery Shop)**

Aim:

Write the coding for Inventory Management System.

Description:

Inventory control system helps in monitoring the sales and various stocks available

DATABASE:

STOCK TABLE

CREATE TABLE stock

(

prodid INT PRIMARY KEY, prodname VARCHAR2(50), quantity INT, unitprice INT, reorderlevel INT

);

SALES TABLE

CREATE TABLE sales

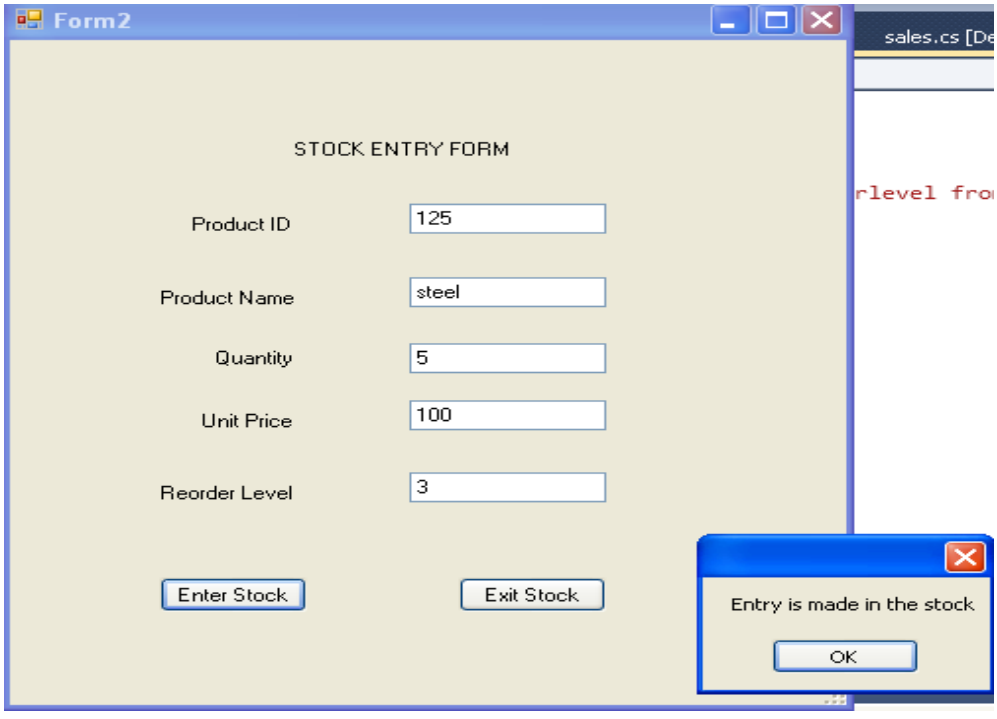
(

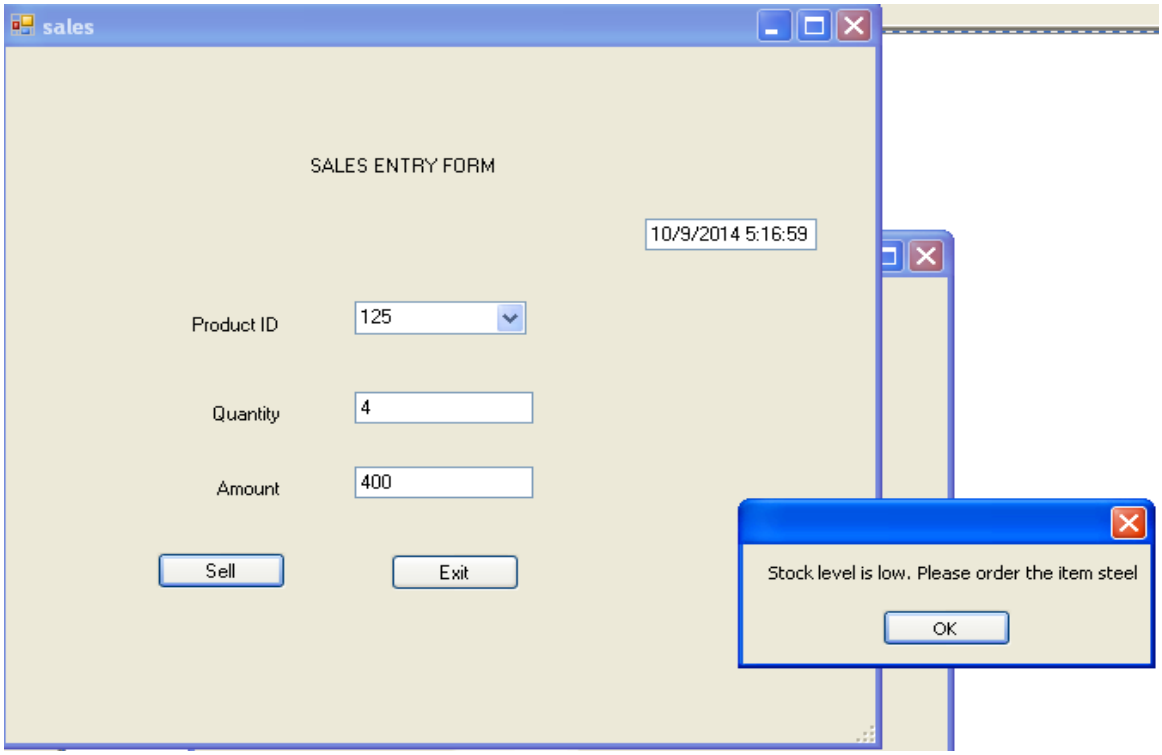
prodid INT REFERENCES stock(prodid), salesqty INT, amount INT, salesdate DATE

);

FORMS:

Inventory control system form:





CODING FOR SALES FORM:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using Oracle.DataAccess.Client;
namespace inventory1
{
   public partial class sales : Form
   {
      public sales()
      {
         InitializeComponent();
      }
      OracleConnection conn;
      private void sales_Load(object sender, EventArgs e)
      {
         string oradb = "Data Source=172.18.1.6;User Id=cselab;Password=cselab;";
         conn = new OracleConnection(oradb);
```

```
            conn.Open();

            string sel1 = "select prodid from cselab.stock";
            OracleCommand cmd = new OracleCommand(sel1, conn);
            OracleDataReader dr = cmd.ExecuteReader();
            while (dr.Read())
            {
               cmboPid.Items.Add(dr["prodid"].ToString());
            }
            DateTime dt = DateTime.Now;
            txtSalesdate.Text = dt.ToString();
        }
        private void btnSell_Click(object sender, EventArgs e)
        {
            Boolean result=validate();
            if (result==true)
            {
               MessageBox.Show("Insufficient Stock");
               return;
            }
            DateTime dt = DateTime.Now;
            String ins="Insert into cselab.sales(prodid,salesqty,amount,salesdate) values
(:prodid,:salesqty,:amount,:salesdate)";
            OracleCommand cmd = new OracleCommand(ins, conn);
            cmd.Parameters.Add("prodid", cmboPid.SelectedItem);
            cmd.Parameters.Add("salesqty", txtSalesqty.Text);
            cmd.Parameters.Add("amount", txtAmt.Text);
            cmd.Parameters.Add("salesdate", dt);
            cmd.ExecuteNonQuery();
            MessageBox.Show("Product sold successfully");
            updatestock();
            clear();
        }
         public Boolean validate()
        {
            String sel1 = "select quantity,reorderlevel,prodname from cselab.stock where
prodid=:prodid";
            OracleCommand cmd = new OracleCommand(sel1, conn);
            cmd.Parameters.Add("prodid", cmboPid.SelectedItem);
            OracleDataReader dr = cmd.ExecuteReader();
            int relevel = 0;
            int qty = 0;
```

```
            if (dr.Read())
            {
                qty = int.Parse(dr["quantity"].ToString());
                relevel = int.Parse(dr["reorderlevel"].ToString());
            }
            int sqty = int.Parse(txtSalesqty.Text);
            if (sqty > qty) // Insuficient stock
            {
                return true;
            }
            int afterSale = qty - sqty;
            if (afterSale < relevel)
            {
                MessageBox.Show("Stock level is low. Please order the item " +
dr["prodname"].ToString());
            }
            return false;
        }
         private void txtSalesqty_Validated(object sender, EventArgs e)
         {
             int amt = getAmount();
             txtAmt.Text = amt.ToString();
         }
        private int getAmount()
        {
            string sel = "SELECT unitprice FROM cselab.stock WHERE prodid=:prodid";
            OracleCommand cmd = new OracleCommand(sel, conn);
            cmd.Parameters.Add("prodid", cmboPid.SelectedItem);
            OracleDataReader dr = cmd.ExecuteReader();
            int uprice = 0;
            if (dr.Read())
            {
                uprice = int.Parse(dr["unitprice"].ToString());
            }
            int sqty = int.Parse((txtSalesqty.Text).ToString());
            int amt = uprice * sqty;
            return amt;
        }
```

```
        public void updatestock()
        {
            string upd = "update cselab.stock set quantity=quantity-:salesqty where prodid=:prodid";
            OracleCommand cmd = new OracleCommand(upd, conn);
            cmd.Parameters.Add("salesqty", txtSalesqty.Text);
            cmd.Parameters.Add("prodid", cmboPid.SelectedItem);
            cmd.ExecuteNonQuery();
        }
        void clear()
        {
            cmboPid.Text = "";
            txtSalesqty.Text = "";
            txtAmt.Text = "";
            txtSalesdate.Text = "";
        }

        private void btnXit1_Click(object sender, EventArgs e)
        {
            Close();
        }
    }
}
```

CODING FOR STOCK FORM:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using Oracle.DataAccess.Client;
namespace inventory1
{
    public partial class stock : Form
    {
        public stock()
        {
            InitializeComponent();
        }
```

```
OracleConnection conn;
Boolean IsExist;

private void stock_Load(object sender, EventArgs e)
{
   string oradb = "Data Source=172.18.1.6;User Id=cselab;Password=cselab;";
   conn = new OracleConnection(oradb);
   conn.Open();
}
void clear()
{
   txtPid.Text = "";
   txtPname.Text = "";
   txtPrice.Text = "";
   txtQty.Text = "";
   txtReorder.Text = "";
}
private void textBox3_TextChanged(object sender, EventArgs e)
{

}
private void btnStock1_Click(object sender, EventArgs e)
{
    if (IsExist)
    {
      String upd = "update cselab.stock set quantity=quantity + :qty where prodid=:prodid";
      OracleCommand cmd2 = new OracleCommand(upd, conn);
      cmd2.Parameters.Add("qty", txtQty.Text);
      cmd2.Parameters.Add("prodid", txtPid.Text);
      cmd2.ExecuteNonQuery();
      MessageBox.Show("Quantity is updated");
      clear();
    }
    else
    {
      String ins = "INSERT INTO cselab.stock(prodid, prodname, quantity, unitprice,
reorderlevel) values(:prodid, :prodname, :quantity, :unitprice, :reorderlevel)";
      OracleCommand cmd = new OracleCommand(ins, conn);
      cmd.Parameters.Add("prodid", txtPid.Text);
      cmd.Parameters.Add("prodname", txtPname.Text);
      cmd.Parameters.Add("quantity", txtQty.Text);
      cmd.Parameters.Add("unitprice", txtPrice.Text);
      cmd.Parameters.Add("reorderlevel", txtReorder.Text);
```

```
            cmd.ExecuteNonQuery();
            MessageBox.Show("Entry is made in the stock");
            clear(); ;
                }
        }
        private void btnXit_Click(object sender, EventArgs e)
        {
           Close();
        }
        private void txtPid_Validated(object sender, EventArgs e)
        {
           String sel = "Select  prodid,prodname,unitprice,reorderlevel from cselab.stock where
prodid=:prodid";
           OracleCommand cmd1 = new OracleCommand(sel, conn);
           cmd1.Parameters.Add("prodid", txtPid.Text);
           String pid = txtPid.Text;
           OracleDataReader dr = cmd1.ExecuteReader();
           IsExist = false;
           if (dr.Read())
           {
              txtPname.Text = dr["prodname"].ToString();
              txtPrice.Text = dr["unitprice"].ToString();
              txtReorder.Text = dr["reorderlevel"].ToString();
              txtPname.Enabled = false;
              txtPrice.Enabled = false;
              txtReorder.Enabled = false;
              IsExist = true;
           }
        }

    }
}
```

RESULT:

Thus, the Coding was executed successfully.

ADDITIONAL PROGRAM

Ex. No: 14.a **Even or ODD**

AIM:

To write a PL/SQL block to find whether the given number is odd or even.

PROGRAM:

```
declare
        n number:=&n;
begin
        if mod(n,2)=0
        then
                dbms_output.put_line('number is even');
        else
                dbms_output.put_line('number is odd');
        end if;
end;
/
```

Output

```
Enter value for n: 7
old  2:        n number:=&n;
new  2:         n number:=7;
number is odd
```

RESULT:

Thus the program to find whether the given number is odd or even by using PL/SQL has been executed successfully.

Ex. No: 14.b **Factorial of a Number**

AIM:

To write a PL/SQL block to find the factorial of a given number.

PROGRAM:

```
DECLARE
        fact number :=1;
        n number := &1;
BEGIN
        while n > 0 loop
                fact:=n*fact;
                n:=n-1;
        end loop;
         dbms_output.put_line(fact);
END;
```

Output

Consider 5 has given

120

RESULT:

Thus the program to find the factorial of a given number by using PL/SQL has been executed successfully.

Ex. No: 14.c **Leap Year or Not**

AIM:

To write a PL/SQL block to find whether the given year is leap year or not.

PROGRAM:

```
DECLARE
  year NUMBER := 2012;
BEGIN
  IF MOD(year, 4)=0
    AND
    MOD(year, 100)!=0
    OR
    MOD(year, 400)=0 THEN
    dbms_output.Put_line(year || ' is leap year ');
  ELSE
    dbms_output.Put_line(year || ' is not leap year.');
  END IF;
END;
```

Output

2012 is leap year

RESULT:

Thus, the program to find whether the given year is leap year or not by using PL/SQL has been executed successfully.

Ex. No: 14.d **Fibonacci Series**

AIM:

To write a PL/SQL block to generate Fibonacci series.

PROGRAM:

```
declare
a number := 0;
b number := 1;
temp number;
n number := 10;
i number;
begin
  dbms_output.put_line('fibonacci series is :');
  dbms_output.put_line(a);
  dbms_output.put_line(b);
  for i in 2..n
  loop
    temp:= a + b;
    a := b;
    b := temp;
    dbms_output.put_line(temp);
  end loop;
end;
```

Output

fibonacci series is : 0 1 1 2 3 5 8 13 21 34

RESULT:

Thus the program to generate Fibonacci series by using PL/SQL has been executed successfully.

Ex. No: 14.e **Reverse a Number**

AIM:

To write a PL/SQL block to reverse a number.

PROGRAM:

```
DECLARE
num number;
reverse_num number:=0;
begin
num:=98765;
while num>0
loop
reverse_num:=(reverse_num*10) + mod(num,10);
num:=trunc(num/10);
end loop;
dbms_output.put_line(' Reversed number is : '|| reverse_num);
```

Output

Reversed number is: 56789

RESULT:

Thus, the program to reverse a number by using PL/SQL has been executed successfully.