**SRI CHANDRASEKHARENDRA SARASWATHI VISWA MAHAVIDYALAYA**

**(UNIVERSITY ESTABLISHED UNDER SECTION 3 OF UGC ACT 1956)**

**ENATHUR,KANCHIPURAM – 631561**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

****

**DATABASE MANAGEMENT SYSTEM**

**LAB RECORD**

**Name : S.Krishnama Naidu**

**Reg.No : 11239A080**

**Class : 3rd year**

**Course : BE-CSE**

**Faculty in charge : D.JEEVAN**

**SRI CHANDRASEKHARENDRA SARASWATHI VISWA MAHAVIDYALAYA**

**(UNIVERSITY ESTABLISHED UNDER SECTION 3 OF UGC ACT 1956)**

**ENATHUR,KANCHIPURAM – 631561**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

****

**BONAFIDE CERTIFICATE**

This is to certify that this is the bonafide record of work done by Mr/Mrs.S.Krishnama Naidu with Reg.no 11239A080 of III-B.E-CSE in the DATABASE MANAGEMENT SYSTEM FOR ENGINEERS during the academic year 2025-2026

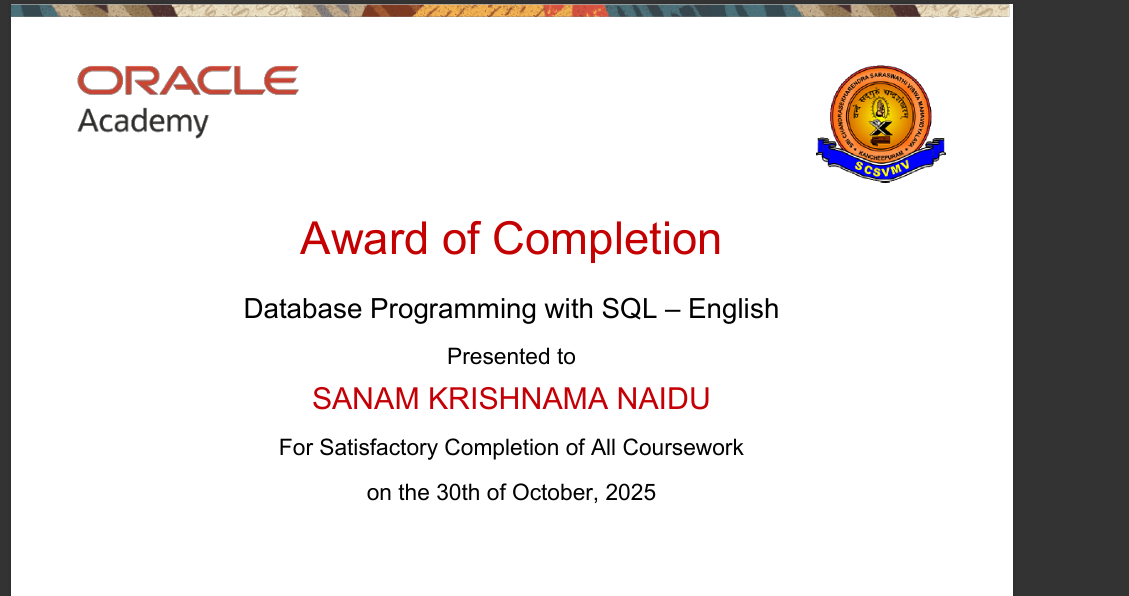
Station: **Enathur**

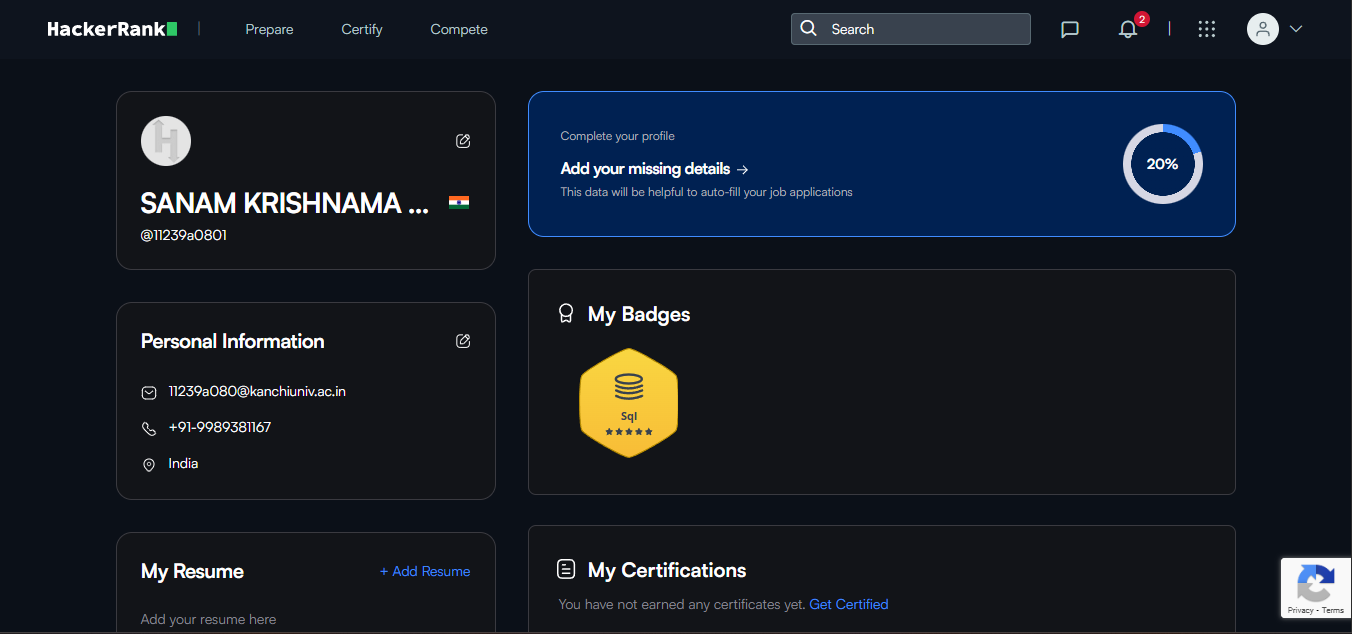
Date :

**Staff-in-charge Head of the department**

**Submitted for the practical examination held on\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Examiner-1 Examiner-2**

****

****

**Experiment 1 : Create Table with Constraints**

**A. NOT NULL**

Ensures a column cannot have NULL (empty) values.

**B. UNIQUE**

Ensures all values in a column are different.

**C. PRIMARY KEY**

Uniquely identifies each record in a table (NOT NULL + UNIQUE)

**D. FOREIGN KEY**

Links one table to another using a referenced key.

**E. CHECK**

Ensures that all values in a column satisfy a specific condition.

**F. DEFAULT**

Assigns a default value to a column if no value is provided.

**G. CREATE INDEX**

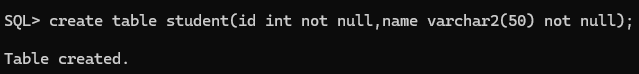
Improves the speed of data retrieval from a table.

**Experiments:**

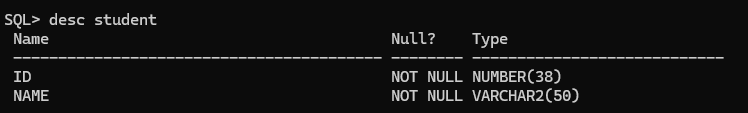
**A. NOT NULL**

**Code:**

create table student(id int not null,name varchar2(50) not null);



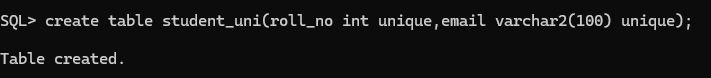
**Output:**

****

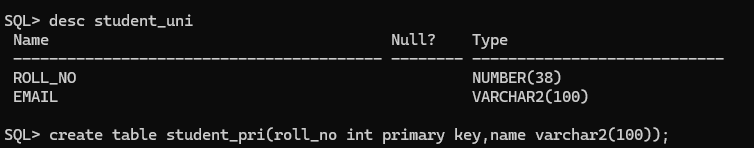
**B. UNIQUE**

**Code:**

create table student\_uni(roll\_no int unique,email varchar2(100) unique);



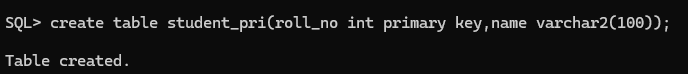
**Output:**

****

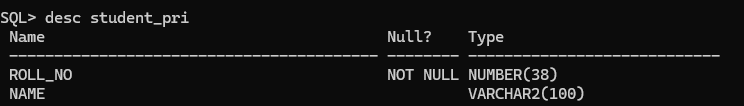
**C. PRIMARY KEY**

**Code:**

create table student\_pri(roll\_no int primary key,name varchar2(100));

****

**Output:**

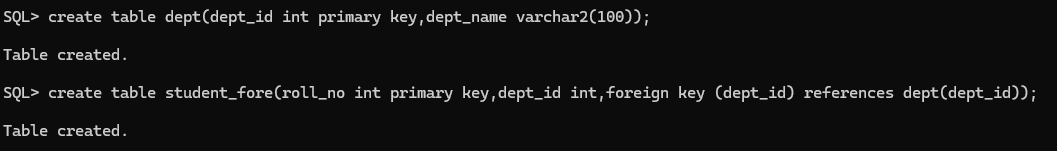


**D. FOREIGN KEY**

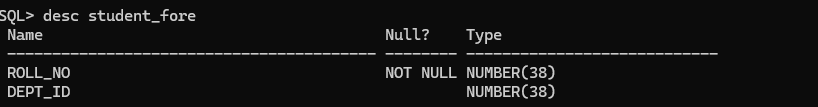
**Code:**

create table dept(dept\_id int primary key,dept\_name varchar2(100));

create table student\_fore(roll\_no int primary key,dept\_id int,foreign key (dept\_id) references dept(dept\_id));



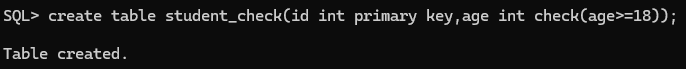
**Output:**

****

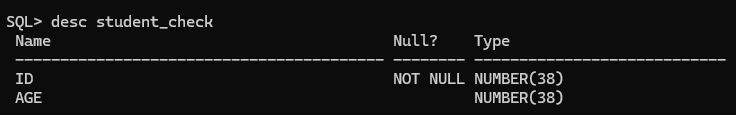
**E. CHECK**

**Code:**

create table student\_check(id int primary key,age int check(age>=18));



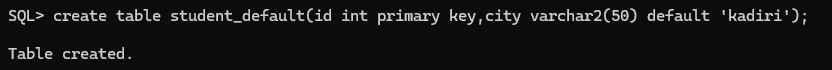
**Output:**

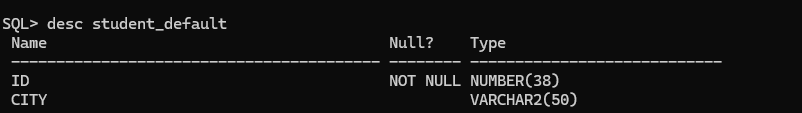
****

**F. DEFAULT**

**Code:**

create table student\_default(id int primary key,city varchar2(50) default 'kadiri');

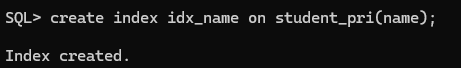


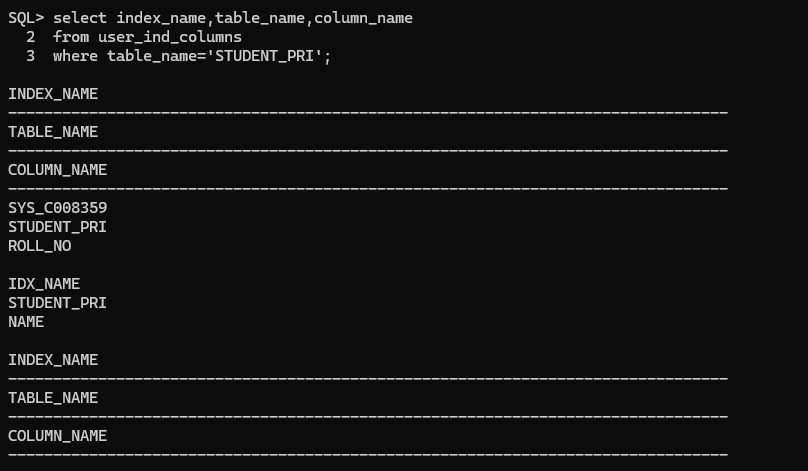
**Output:**

**G. CREATE INDEX**

**Code:**

create index idx\_name on student\_pri(name);



**Output:**

**Experiment 2 : INSERT Command**

**1. Insert values with single entry**

**2. Insert values with multiple entires**

**3. ALTER Table Structure**

**4. VIEW Table structure**

**5. UPDATE table**

**6. DELETE Rows in table**

**Source Code:**

-- =========================================

-- EXPERIMENT 2: INSERT COMMAND

-- =========================================

-- Step 1: Create and use database

CREATE DATABASE IF NOT EXISTS DBMS\_Experiments;

USE DBMS\_Experiments;

-- Step 2: Create necessary tables (so INSERT has targets)

CREATE TABLE IF NOT EXISTS Students (

StudentID INT PRIMARY KEY,

FirstName VARCHAR(50) NOT NULL,

LastName VARCHAR(50),

Email VARCHAR(100) UNIQUE,

Age INT CHECK (Age >= 18),

City VARCHAR(50) DEFAULT 'Unknown'

);

CREATE TABLE IF NOT EXISTS Courses (

CourseID INT PRIMARY KEY,

CourseName VARCHAR(100) NOT NULL

);

CREATE TABLE IF NOT EXISTS Enrollments (

EnrollmentID INT PRIMARY KEY,

StudentID INT,

CourseID INT,

FOREIGN KEY (StudentID) REFERENCES Students(StudentID),

FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)

);

-- =========================================

-- INSERT COMMANDS

-- =========================================

-- 1️⃣ Insert values with single entry

INSERT INTO Students (StudentID, FirstName, LastName, Email, Age, City)

VALUES (1, 'John', 'Doe', 'john.doe@example.com', 20, 'New York');

-- 2️⃣ Insert values with multiple entries

INSERT INTO Students (StudentID, FirstName, LastName, Email, Age, City)

VALUES

(2, 'Alice', 'Smith', 'alice.smith@example.com', 22, 'Boston'),

(3, 'Bob', 'Brown', 'bob.brown@example.com', 21, 'Chicago'),

(4, 'Mary', 'Johnson', 'mary.johnson@example.com', 19, DEFAULT);

-- 3️⃣ Insert into Courses table

INSERT INTO Courses (CourseID, CourseName)

VALUES

(101, 'Database Systems'),

(102, 'Operating Systems'),

(103, 'Computer Networks');

-- 4️⃣ Insert into Enrollments table

INSERT INTO Enrollments (EnrollmentID, StudentID, CourseID)

VALUES

(1, 1, 101),

(2, 2, 102),

(3, 3, 103),

(4, 4, 101);

-- =========================================

-- VIEW INSERTED DATA

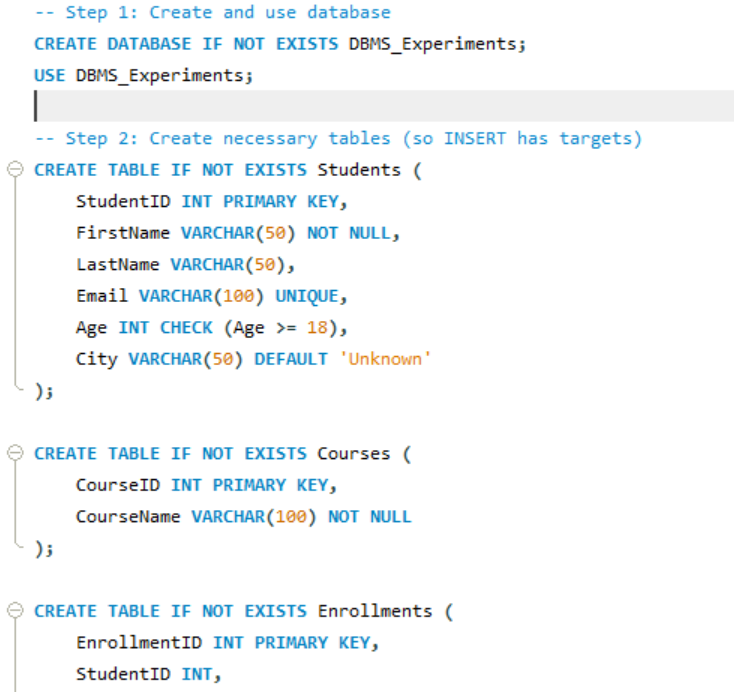
-- =========================================

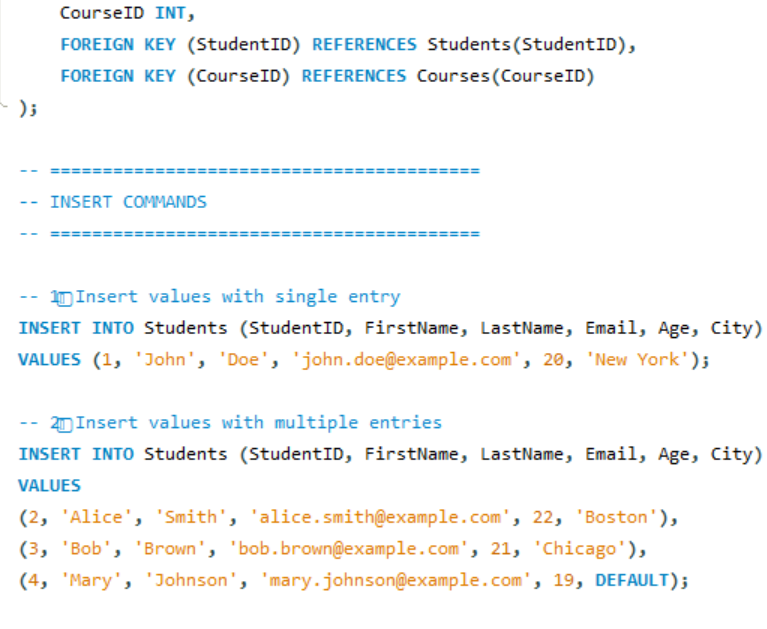
SELECT \* FROM Students;

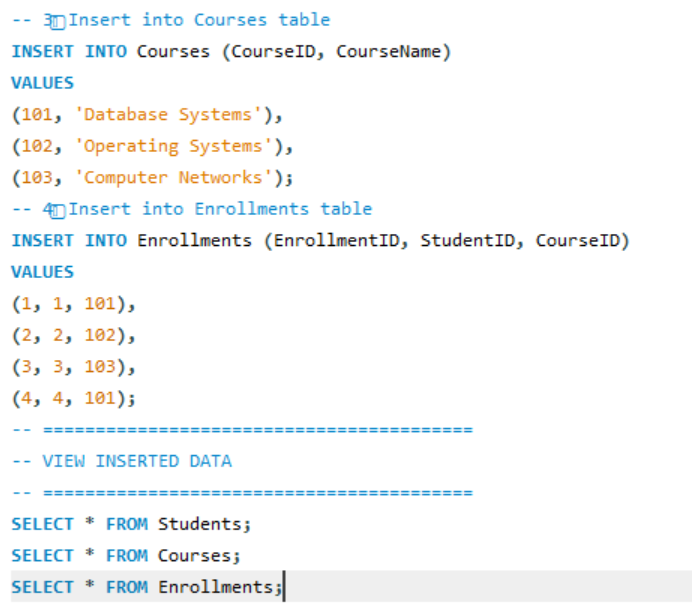
SELECT \* FROM Courses;

SELECT \* FROM Enrollments;

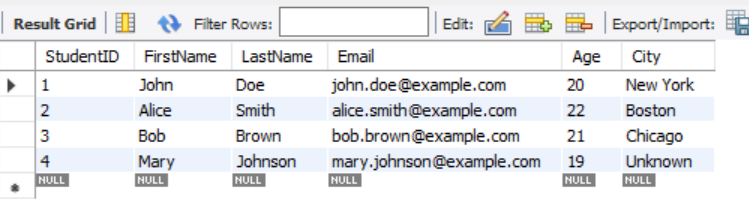
**Program:**

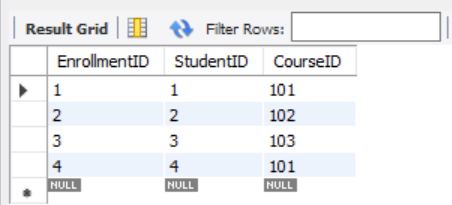
****

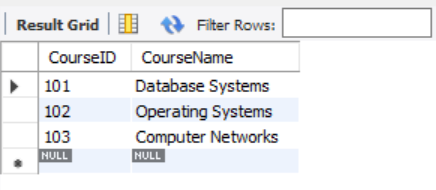
****

****

**Output:**

****



****

**Experiment 3 :Aggregate Functions**

**Source Code:**

DROP DATABASE IF EXISTS college;

CREATE DATABASE college;

USE college;

-- Create table

CREATE TABLE marks (

student\_name VARCHAR(50),

subject VARCHAR(50),

score INT

);

-- Insert sample data

INSERT INTO marks VALUES

('Manasa', 'DBMS', 90),

('Ravi', 'DBMS', 80),

('Priya', 'Math', 95),

('Kiran', 'Math', 65),

('Meena', 'Math', 60);

-- Display all records

SELECT \* FROM marks;

-- Aggregate functions demonstration

SELECT

MIN(score) AS min\_score,

MAX(score) AS max\_score,

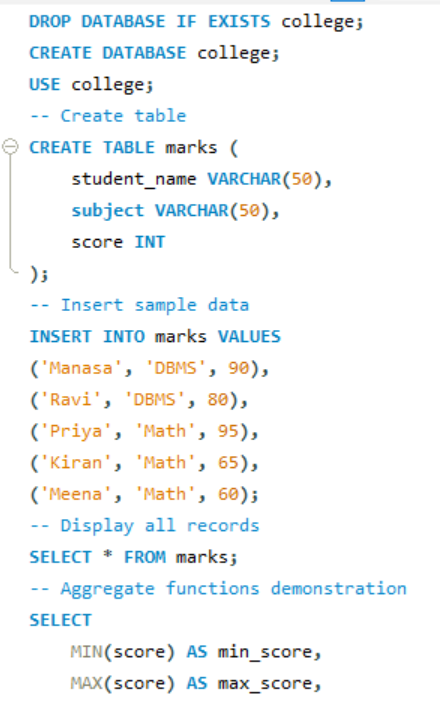
COUNT(\*) AS total\_students,

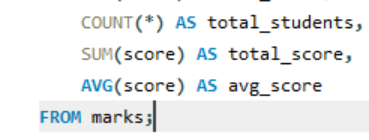
SUM(score) AS total\_score,

AVG(score) AS avg\_score

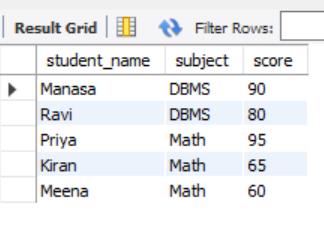
FROM marks;

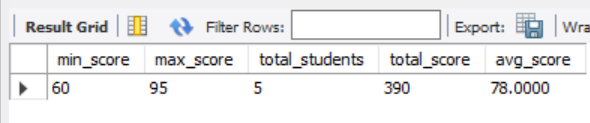
**Program:**





**Output:**





**Experiment 4 :Group By and Order By**

**Source Code:**

DROP DATABASE IF EXISTS college;

CREATE DATABASE college;

USE college;

CREATE TABLE marks (

student\_name VARCHAR(50),

subject VARCHAR(50),

score INT

);

INSERT INTO marks VALUES

('Manasa', 'DBMS', 90),

('Ravi', 'DBMS', 80),

('Priya', 'Math', 95),

('Kiran', 'Math', 65),

('Meena', 'Math', 60);

-- Average score per subject

SELECT subject, AVG(score) AS avg\_score

FROM marks

GROUP BY subject;

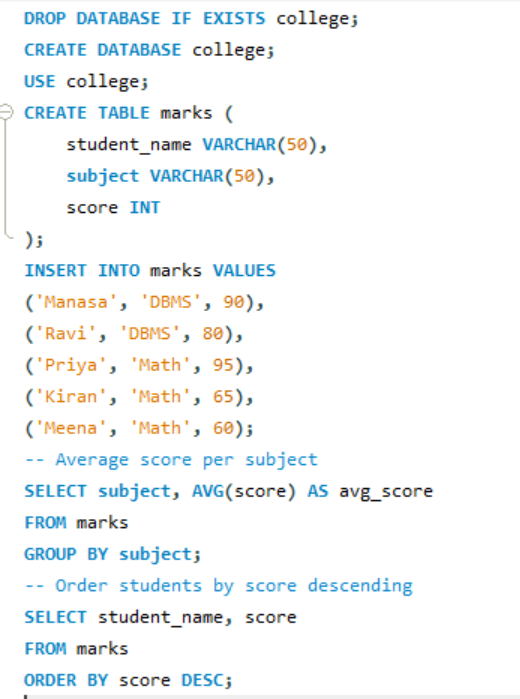
-- Order students by score descending

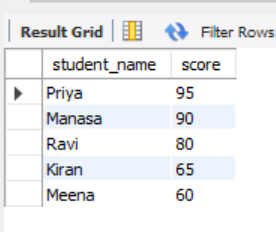
SELECT student\_name, score

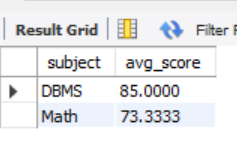
FROM marks

ORDER BY score DESC;

**Program:**



**Output:**



**Experiment 5 : Ascending, Descending**

**Source Code:**

-- Create database

CREATE DATABASE IF NOT EXISTS college;

USE college;

-- Create table

DROP TABLE IF EXISTS student;

CREATE TABLE student (

id INT PRIMARY KEY,

name VARCHAR(30),

age INT,

branch VARCHAR(10)

);

-- Insert sample data

INSERT INTO student VALUES

(1, 'Manasa', 20, 'CSE'),

(2, 'Ravi', 22, 'ECE'),

(3, 'Kiran', 21, 'CSE'),

(4, 'Priya', 19, 'EEE'),

(5, 'Arjun', 23, 'MECH');

-- View all records

SELECT \* FROM student;

-- Ascending order by name

SELECT \* FROM student

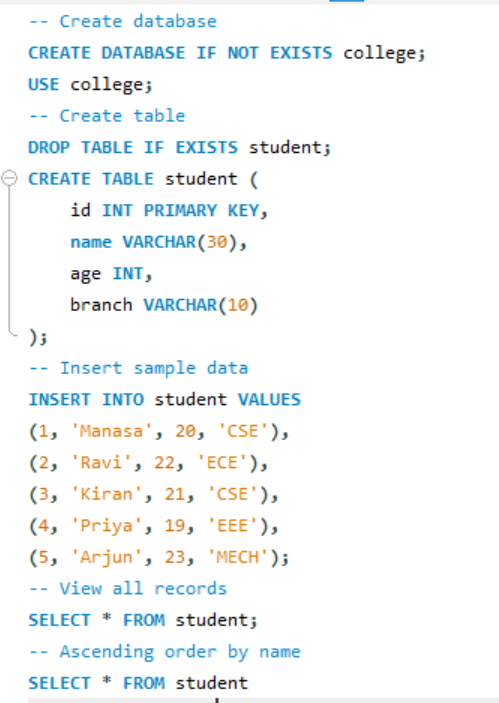
ORDER BY name ASC;

-- Descending order by age

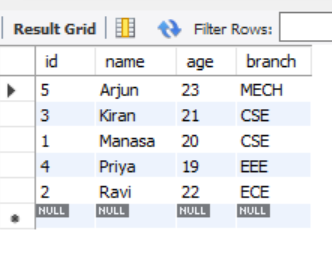
SELECT \* FROM student

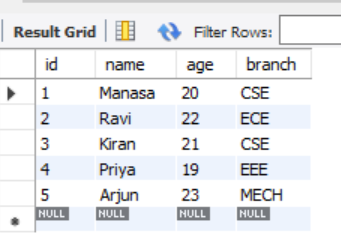
ORDER BY age DESC;

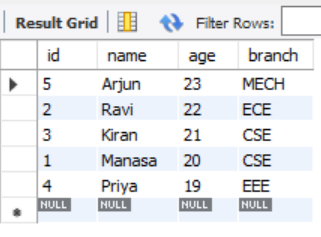
**Program:**

****

****

**Output:**

****



**Experiment 6 :SQL Operators**

**1. Like operator (%,-)**

**2. Between & or**

**Source Code:**

DROP DATABASE IF EXISTS college;

CREATE DATABASE college;

USE college;

CREATE TABLE student (

student\_id INT PRIMARY KEY,

name VARCHAR(50),

age INT,

dept\_id INT

);

INSERT INTO student VALUES

(101, 'Manasa', 20, 1),

(102, 'Ravi', 21, 2),

(103, 'Priya', 19, 1),

(104, 'Kiran', 22, 1),

(105, 'Meena', 20, 3);

-- LIKE operator

SELECT \* FROM student WHERE name LIKE 'M%';

SELECT \* FROM student WHERE name LIKE '%an%';

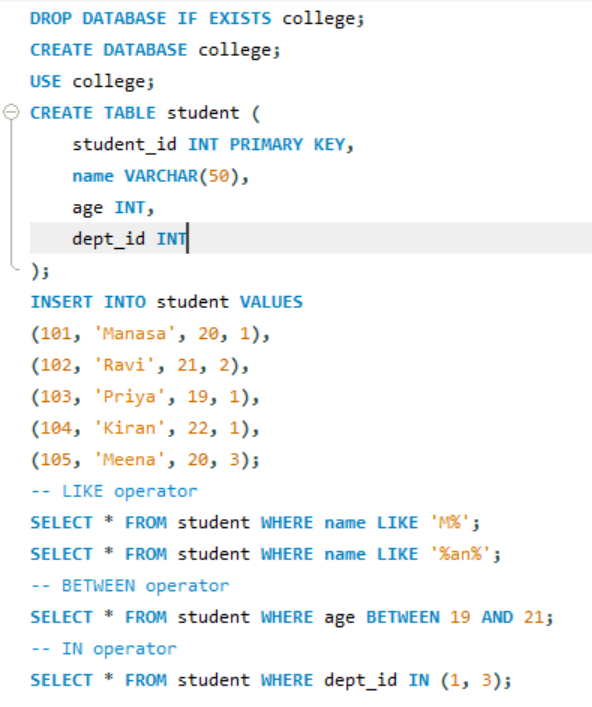
-- BETWEEN operator

SELECT \* FROM student WHERE age BETWEEN 19 AND 21;

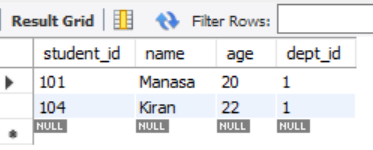
-- IN operator

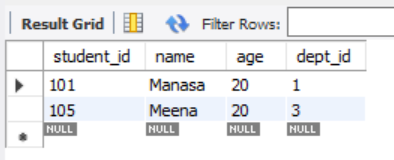
SELECT \* FROM student WHERE dept\_id IN (1, 3);

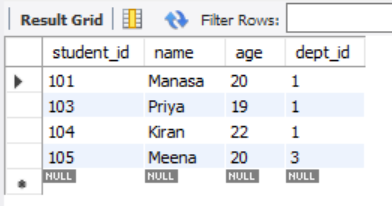
**Program:**

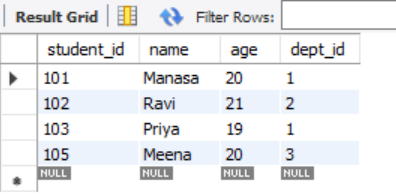
****

**Output:**



****



****

**Experiment 7 : SQL Joins**

**1. Inner Join**

**2. Left Join**

**3. Right Join**

**4. Outer Join**

**5. Left Join exclude Inner Join**

**6. Right Join exclude Inner Join**

**7. Outer Join exclude Inner Jain**

**Source Code:**

-- Create tables

DROP TABLE IF EXISTS Students;

DROP TABLE IF EXISTS Courses;

CREATE TABLE Students (

StudentID INT,

Name VARCHAR(50),

CourseID INT

);

CREATE TABLE Courses (

CourseID INT,

CourseName VARCHAR(50)

);

-- Insert data

INSERT INTO Students VALUES

(1, 'John', 101),

(2, 'Emma', 102),

(3, 'Raj', 103),

(4, 'Sara', NULL);

INSERT INTO Courses VALUES

(101, 'Math'),

(102, 'Science'),

(104, 'History');

-- 1. INNER JOIN

SELECT s.Name, c.CourseName

FROM Students s

INNER JOIN Courses c ON s.CourseID = c.CourseID;

-- 2. LEFT JOIN

SELECT s.Name, c.CourseName

FROM Students s

LEFT JOIN Courses c ON s.CourseID = c.CourseID;

-- 3. RIGHT JOIN

SELECT s.Name, c.CourseName

FROM Students s

RIGHT JOIN Courses c ON s.CourseID = c.CourseID;

-- 4. FULL OUTER JOIN (some databases: use UNION of LEFT + RIGHT)

SELECT s.Name, c.CourseName

FROM Students s

LEFT JOIN Courses c ON s.CourseID = c.CourseID

UNION

SELECT s.Name, c.CourseName

FROM Students s

RIGHT JOIN Courses c ON s.CourseID = c.CourseID;

-- 5. LEFT JOIN EXCLUDING INNER JOIN

SELECT s.Name, c.CourseName

FROM Students s

LEFT JOIN Courses c ON s.CourseID = c.CourseID

WHERE c.CourseID IS NULL;

-- 6. RIGHT JOIN EXCLUDING INNER JOIN

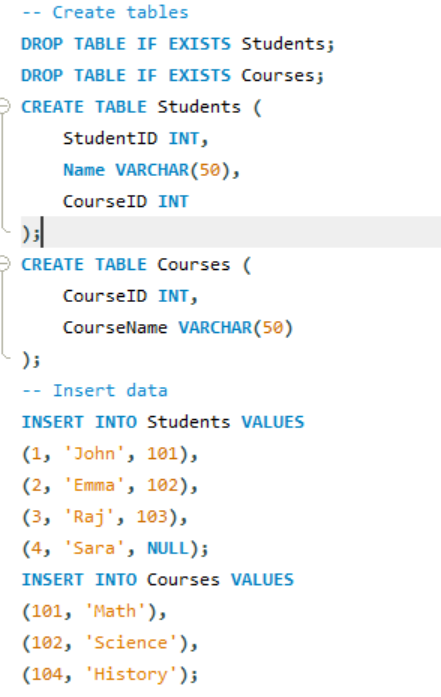
SELECT s.Name, c.CourseName

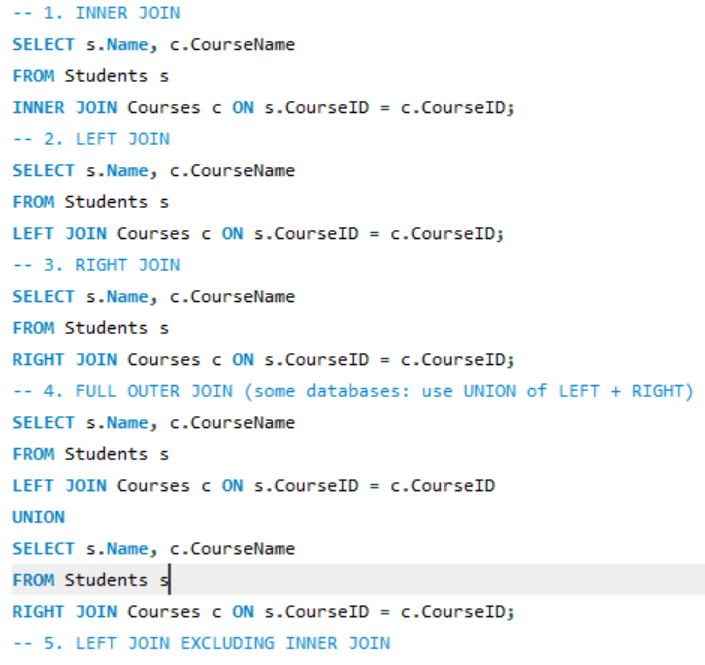
FROM Students s

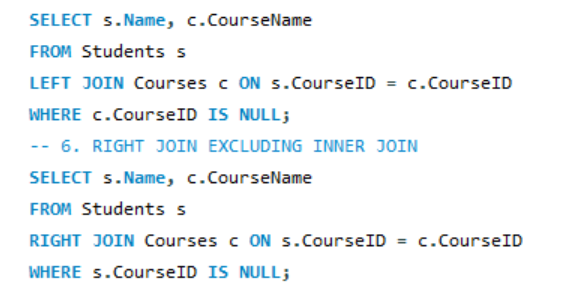
RIGHT JOIN Courses c ON s.CourseID = c.CourseID

WHERE s.CourseID IS NULL;

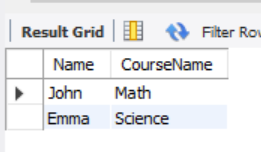
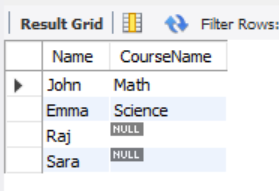
**Program:**

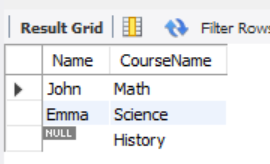
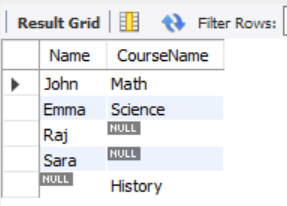


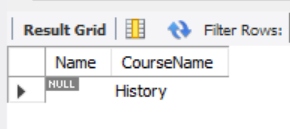


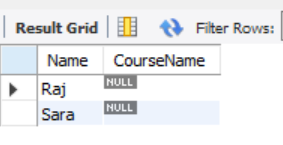


**Output:**

****

****



****

**Experiment 8 : Normal Forms**

**1. INF**

**Source Code:**

DROP DATABASE IF EXISTS normalforms;

CREATE DATABASE normalforms;

USE normalforms;

CREATE TABLE orders\_1nf (

order\_id INT,

customer\_name VARCHAR(30),

product VARCHAR(30),

quantity INT

);

INSERT INTO orders\_1nf VALUES

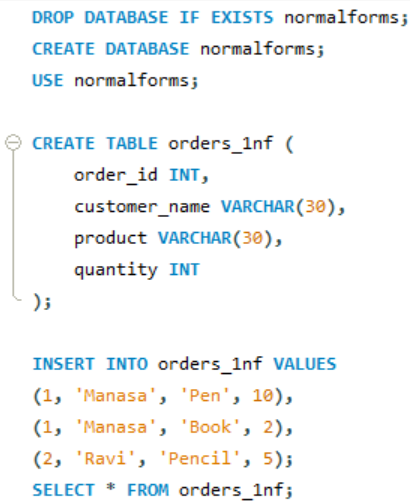
(1, 'Manasa', 'Pen', 10),

(1, 'Manasa', 'Book', 2),

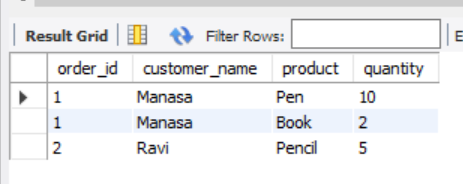
(2, 'Ravi', 'Pencil', 5);

SELECT \* FROM orders\_1nf;

**Program:**



**Output:**



**2. 2NF**

**Source Code:**

-- Clean safely (drop children first)

DROP TABLE IF EXISTS items;

DROP TABLE IF EXISTS orders\_2nf;

DROP TABLE IF EXISTS customer;

-- Recreate tables

CREATE TABLE customer (

cust\_id INT PRIMARY KEY,

cust\_name VARCHAR(30)

);

CREATE TABLE orders\_2nf (

order\_id INT PRIMARY KEY,

cust\_id INT,

FOREIGN KEY (cust\_id) REFERENCES customer(cust\_id)

);

CREATE TABLE items (

item\_id INT PRIMARY KEY,

order\_id INT,

product VARCHAR(30),

quantity INT,

FOREIGN KEY (order\_id) REFERENCES orders\_2nf(order\_id)

);

-- Insert data

INSERT INTO customer VALUES (1, 'Manasa'), (2, 'Ravi');

INSERT INTO orders\_2nf VALUES (101, 1), (102, 2);

INSERT INTO items VALUES

(1, 101, 'Pen', 10),

(2, 101, 'Book', 2),

(3, 102, 'Pencil', 5);

-- Verify

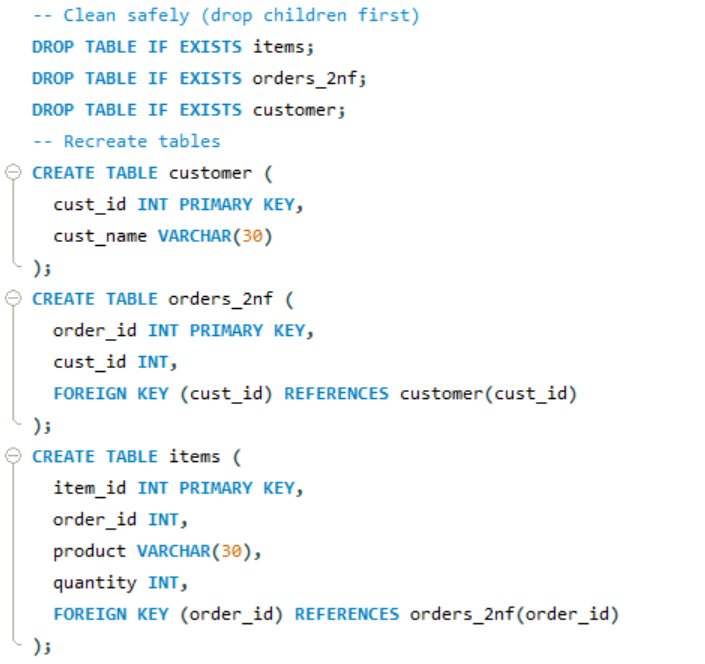
SELECT o.order\_id, c.cust\_name, i.product, i.quantity

FROM orders\_2nf o

JOIN customer c ON o.cust\_id = c.cust\_id

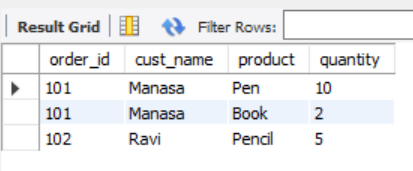
JOIN items i ON o.order\_id = i.order\_id;

**Program:**

****

****

**Output:**

****

**3. 3NF**

**Source Code:**

-- Create database

CREATE DATABASE normalforms3;

USE normalforms3;

-- Tables for 3NF

CREATE TABLE city (

city\_name VARCHAR(30) PRIMARY KEY,

state VARCHAR(30)

);

CREATE TABLE customer (

cust\_id INT PRIMARY KEY,

cust\_name VARCHAR(30),

city\_name VARCHAR(30),

FOREIGN KEY (city\_name) REFERENCES city(city\_name)

);

CREATE TABLE orders (

order\_id INT PRIMARY KEY,

cust\_id INT,

product VARCHAR(30),

quantity INT,

FOREIGN KEY (cust\_id) REFERENCES customer(cust\_id)

);

-- Insert data

INSERT INTO city VALUES

('Vijayawada', 'Andhra Pradesh'),

('Hyderabad', 'Telangana');

INSERT INTO customer VALUES

(1, 'Manasa', 'Vijayawada'),

(2, 'Ravi', 'Hyderabad');

INSERT INTO orders VALUES

(101, 1, 'Pen', 10),

(102, 1, 'Book', 5),

(103, 2, 'Pencil', 6);

-- Display result

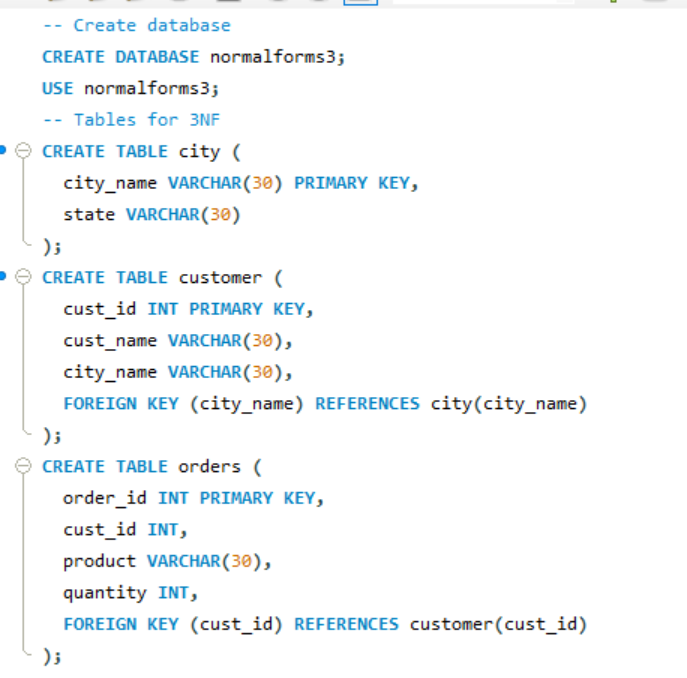
SELECT o.order\_id, c.cust\_name, c.city\_name, ci.state, o.product, o.quantity

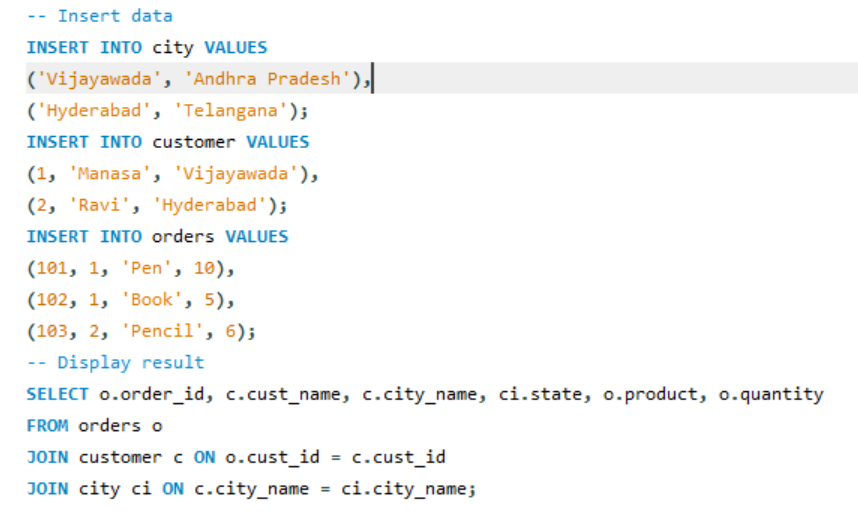
FROM orders o

JOIN customer c ON o.cust\_id = c.cust\_id

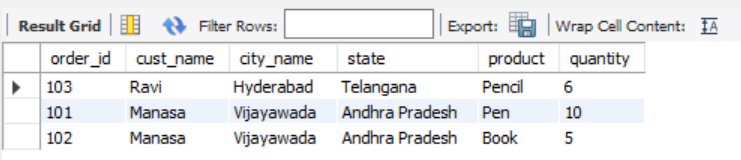
JOIN city ci ON c.city\_name = ci.city\_name;

**Program:**

****

****

**Output:**



**Experiment 9 :Nested Queries Using select across two tables**

**Source Code:**

CREATE TABLE department (

dept\_id INT PRIMARY KEY,

dept\_name VARCHAR(20)

);

CREATE TABLE student (

stu\_id INT PRIMARY KEY,

stu\_name VARCHAR(20),

dept\_id INT,

FOREIGN KEY (dept\_id) REFERENCES department(dept\_id)

);

INSERT INTO department VALUES (1, 'CSE'), (2, 'ECE');

INSERT INTO student VALUES

(101, 'Manasa', 1),

(102, 'Ravi', 2),

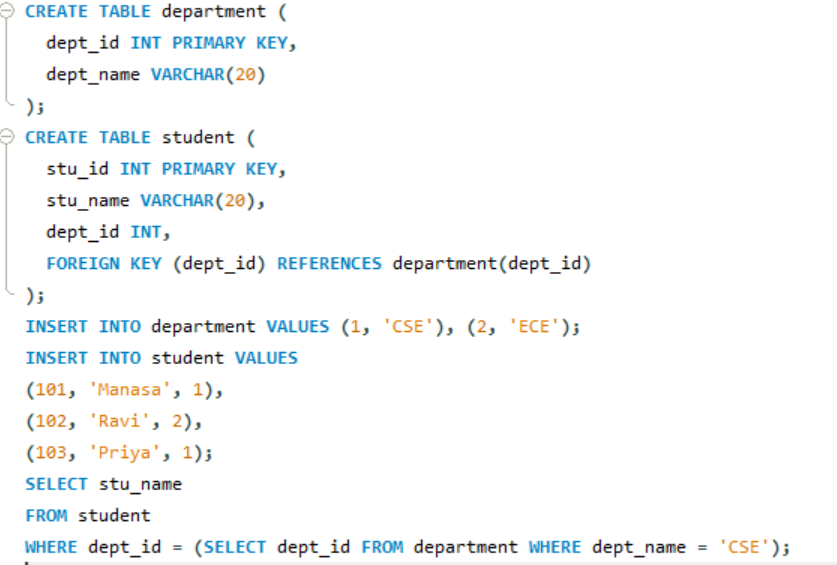
(103, 'Priya', 1);

SELECT stu\_name

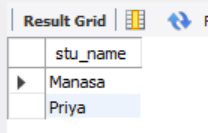
FROM student

WHERE dept\_id = (SELECT dept\_id FROM department WHERE dept\_name = 'CSE');

**Program:**

****

**Output:**

****

**Experiment 10 :SQL wild card characters**

**Source Code:**

DROP TABLE IF EXISTS student;

CREATE TABLE student (

id INT,

name VARCHAR(30)

);

INSERT INTO student (id, name) VALUES

(1, 'Manasa'),

(2, 'Ravi'),

(3, 'Manoj'),

(4, 'Priya'),

(5, 'Kiran');

-- Names starting with 'Ma'

SELECT name

FROM student

WHERE name LIKE 'Ma%';

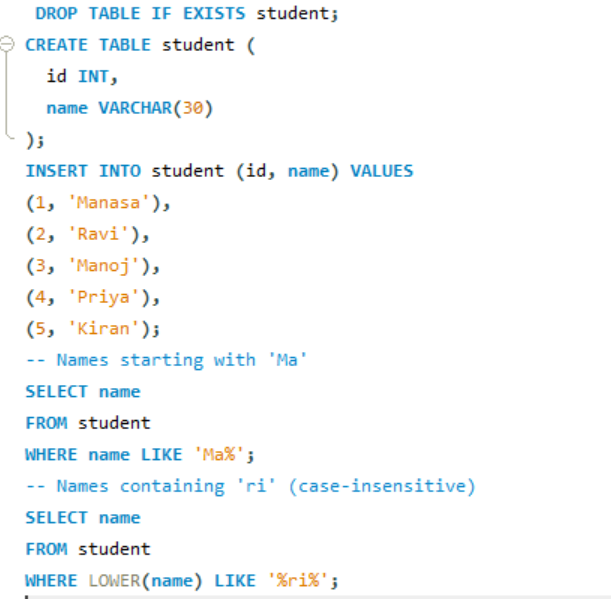
-- Names containing 'ri' (case-insensitive)

SELECT name

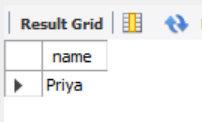
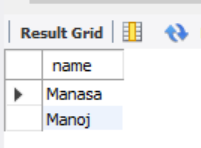
FROM student

WHERE LOWER(name) LIKE '%ri%';

**Program:**

****

**Output:**



**Experiment 11 :Retrieve Database using SELECT with Comparison operator ( = , > . <. >= , <= )**

**Source Code:**

DROP TABLE IF EXISTS student;

CREATE TABLE student (

id INT,

name VARCHAR(30),

marks INT

);

INSERT INTO student (id, name, marks) VALUES

(1, 'Manasa', 85),

(2, 'Ravi', 72),

(3, 'Manoj', 90),

(4, 'Priya', 78),

(5, 'Kiran', 60);

-- Single query showing all comparison operators

SELECT

name,

marks,

CASE

WHEN marks = 78 THEN 'Equal to 78'

WHEN marks > 80 THEN 'Greater than 80'

WHEN marks < 75 THEN 'Less than 75'

WHEN marks >= 78 THEN 'Greater than or equal to 78'

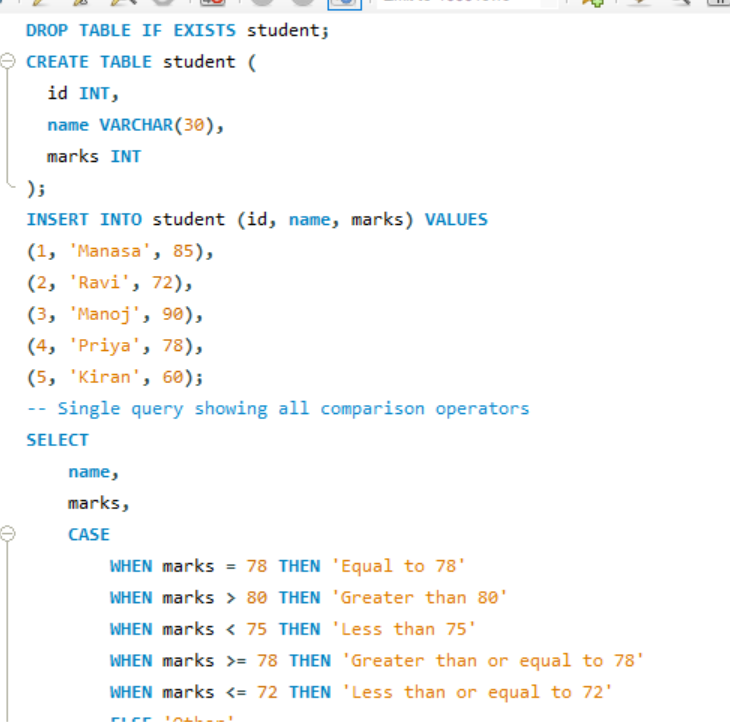
WHEN marks <= 72 THEN 'Less than or equal to 72'

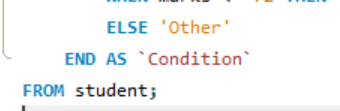
ELSE 'Other'

END AS `Condition`

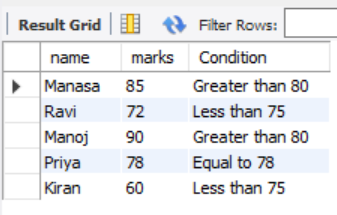
FROM student;

**Program:**

****

****

**Output:**

****

**Experiment 12 : Working on Local Host XAMPP Server**

**1. Exploring Server Variables.( Servername. User, pwd, DBname)**

**Source Code:**

-- Get the current MySQL user

SELECT USER();

-- Get the current selected database

SELECT DATABASE();

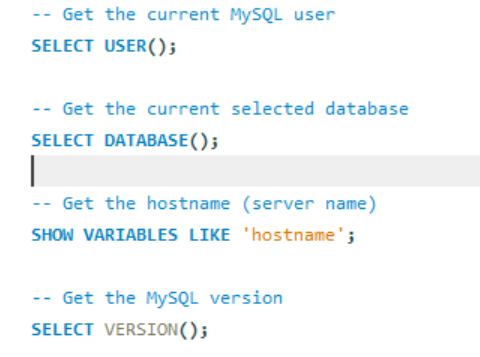
-- Get the hostname (server name)

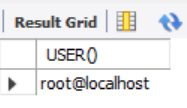
SHOW VARIABLES LIKE 'hostname';

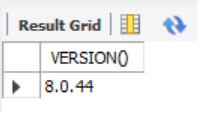
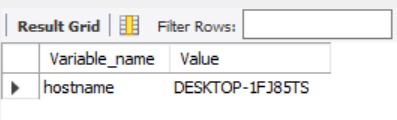
-- Get the MySQL version

SELECT VERSION();

**Program:**



**Output:**



**2. Exploring Create hierarchical user access with privileges**

**Source Code:**

-- Check if the database exists and then drop it

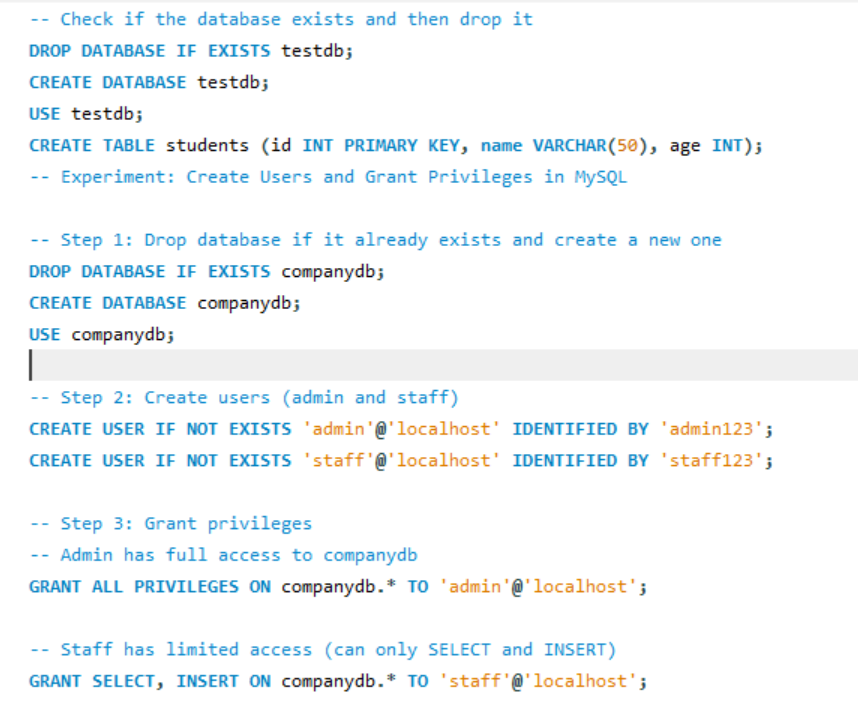
DROP DATABASE IF EXISTS testdb;

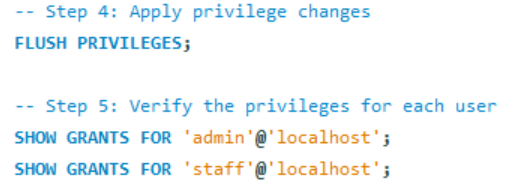
CREATE DATABASE testdb;

USE testdb;

CREATE TABLE students (id INT PRIMARY KEY, name VARCHAR(50), age INT);

**Program:**

****

****

**Output:**