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);

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

Table created.
```

Output:

```
SQL> desc student
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ID                                        NOT NULL NUMBER(38)
 NAME                                      NOT NULL VARCHAR2(50)
```

B. UNIQUE

Code:

create table student_uni(roll_no int unique,email varchar2(100) unique);

```
SQL> create table student_uni(roll_no int unique,email varchar2(100) unique);

Table created.
```

Output:

```
SQL> desc student_uni
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ROLL_NO                                            NUMBER(38)
 EMAIL                                              VARCHAR2(100)

SQL> create table student_pri(roll_no int primary key,name varchar2(100));
```

C. PRIMARY KEY

Code:

create table student_pri(roll_no int primary key,name varchar2(100));

```
SQL> create table student_pri(roll_no int primary key,name varchar2(100));

Table created.
```

Output:

```
SQL> desc student_pri
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ROLL_NO                                   NOT NULL NUMBER(38)
 NAME                                               VARCHAR2(100)
```

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));

```
SQL> create table dept(dept_id int primary key,dept_name varchar2(100));

Table created.

SQL> create table student_fore(roll_no int primary key,dept_id int,foreign key (dept_id) references dept(dept_id));

Table created.
```

Output:

```
SQL> desc student_fore
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ROLL_NO                                   NOT NULL NUMBER(38)
 DEPT_ID                                            NUMBER(38)
```

E. CHECK

Code:

create table student_check(id int primary key,age int check(age>=18));

```
SQL> create table student_check(id int primary key,age int check(age>=18));

Table created.
```

Output:

```
SQL> desc student_check
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ID                                        NOT NULL NUMBER(38)
 AGE                                               NUMBER(38)
```

F. DEFAULT

Code:

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

```
SQL> create table student_default(id int primary key,city varchar2(50) default 'kadiri');

Table created.
```

Output:

```
SQL> desc student_default
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ID                                        NOT NULL NUMBER(38)
 CITY                                              VARCHAR2(50)
```

G. CREATE INDEX

Code:

create index idx_name on student_pri(name);

```
SQL> create index idx_name on student_pri(name);

Index created.
```

Output:

```
SQL> select index_name,table_name,column_name
  2  from user_ind_columns
  3  where table_name='STUDENT_PRI';

INDEX_NAME
--------------------------------------------------------------------------------
TABLE_NAME
--------------------------------------------------------------------------------
COLUMN_NAME
--------------------------------------------------------------------------------
SYS_C008359
STUDENT_PRI
ROLL_NO

IDX_NAME
STUDENT_PRI
NAME

INDEX_NAME
--------------------------------------------------------------------------------
TABLE_NAME
--------------------------------------------------------------------------------
COLUMN_NAME
--------------------------------------------------------------------------------
```

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:

```sql
-- 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
-- ========================================

-- 1. 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');

-- 2. 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);
```

```sql
-- 3 Insert into Courses table
INSERT INTO Courses (CourseID, CourseName)
VALUES
(101, 'Database Systems'),
(102, 'Operating Systems'),
(103, 'Computer Networks');
-- 4 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;
```

Output:

Result Grid | Filter Rows: | Edit: | Export/Import:

StudentID	FirstName	LastName	Email	Age	City
1	John	Doe	john.doe@example.com	20	New York
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	Unknown
NULL	NULL	NULL	NULL	NULL	NULL

Result Grid | Filter Rows:

CourseID	CourseName
101	Database Systems
102	Operating Systems
103	Computer Networks
NULL	NULL

Result Grid | Filter Rows:

EnrollmentID	StudentID	CourseID
1	1	101
2	2	102
3	3	103
4	4	101
NULL	NULL	NULL

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:

```
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;
```

Output:

Result Grid | Filter Rows:

student_name	subject	score
Manasa	DBMS	90
Ravi	DBMS	80
Priya	Math	95
Kiran	Math	65
Meena	Math	60

Result Grid | Filter Rows: | Export: | Wra

min_score	max_score	total_students	total_score	avg_score
60	95	5	390	78.0000

Experiment 4 :Group By and Order By

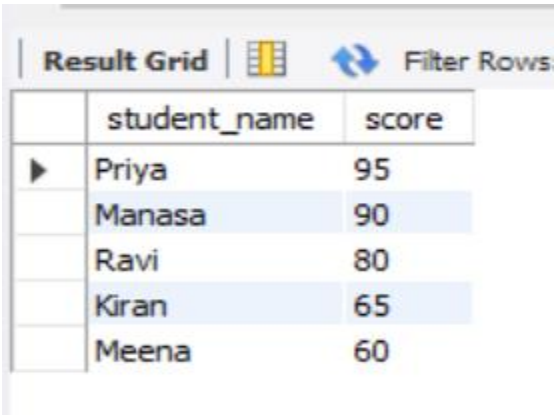
Source Code:

```
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:

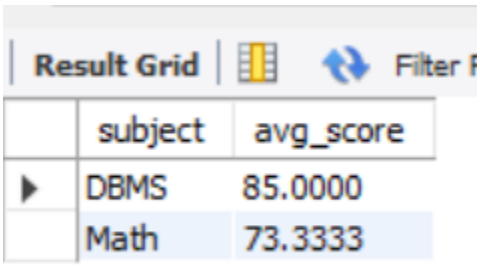
```
DROP DATABASE IF EXISTS college;
CREATE DATABASE college;
USE college;
CREATE TABLE marks (
    student_name VARCHAR(50),
    subject VARCHAR(50),
    score INT
);
INSERT INTO marks VALUES
('Manasa', 'DBMS', 90),
('Ravi', 'DBMS', 80),
('Priya', 'Math', 95),
('Kiran', 'Math', 65),
('Meena', 'Math', 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;
```

Output:



Result Grid | Filter Rows

student_name	score
Priya	95
Manasa	90
Ravi	80
Kiran	65
Meena	60



Result Grid | Filter

subject	avg_score
DBMS	85.0000
Math	73.3333

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:

```sql
-- 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;
```

Output:

Result Grid | Filter Rows:

id	name	age	branch
1	Manasa	20	CSE
2	Ravi	22	ECE
3	Kiran	21	CSE
4	Priya	19	EEE
5	Arjun	23	MECH
NULL	NULL	NULL	NULL

Result Grid | Filter Rows:

id	name	age	branch
5	Arjun	23	MECH
3	Kiran	21	CSE
1	Manasa	20	CSE
4	Priya	19	EEE
2	Ravi	22	ECE
NULL	NULL	NULL	NULL

Result Grid | Filter Rows:

id	name	age	branch
5	Arjun	23	MECH
2	Ravi	22	ECE
3	Kiran	21	CSE
1	Manasa	20	CSE
4	Priya	19	EEE
NULL	NULL	NULL	NULL

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:

```sql
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);
```

Output:

Result Grid | Filter Rows:

student_id	name	age	dept_id
101	Manasa	20	1
105	Meena	20	3
NULL	NULL	NULL	NULL

Result Grid | Filter Rows:

student_id	name	age	dept_id
101	Manasa	20	1
104	Kiran	22	1
NULL	NULL	NULL	NULL

Result Grid | Filter Rows:

student_id	name	age	dept_id
101	Manasa	20	1
102	Ravi	21	2
103	Priya	19	1
105	Meena	20	3
NULL	NULL	NULL	NULL

Result Grid | Filter Rows:

student_id	name	age	dept_id
101	Manasa	20	1
103	Priya	19	1
104	Kiran	22	1
105	Meena	20	3
NULL	NULL	NULL	NULL

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:

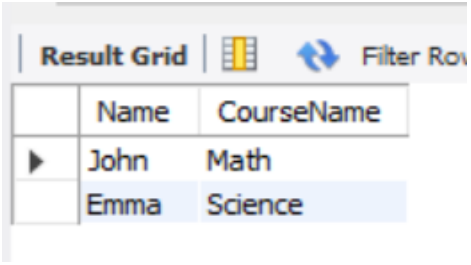
```
-- Create tables
DROP TABLE IF EXISTS Students;
DROP TABLE IF EXISTS Courses;
CREATE TABLE Students (
   StudentID INT,
   Name VARCHAR(50),
   CourseID INT
);

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;
```

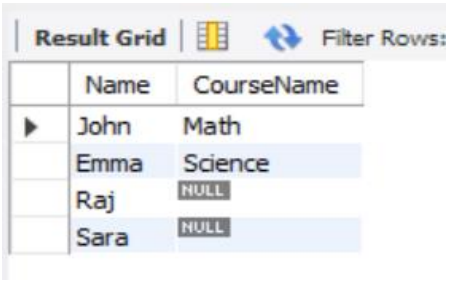
```
-- 2. LEFT JOIN
SELECT s.Name, c.CourseName
FROM Students s
LEFT JOIN Courses c ON s.CourseID = c.CourseID;

-- 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;
```

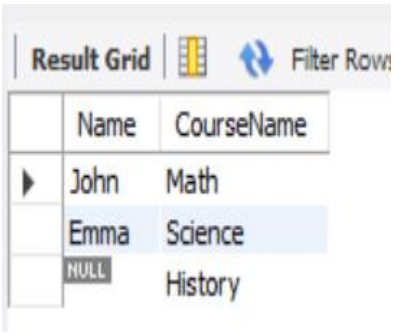
Program:

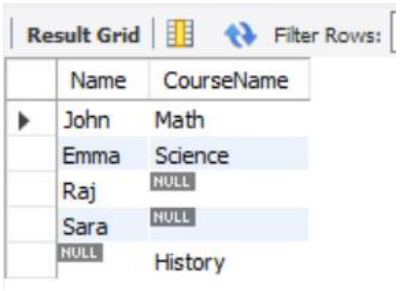
```sql
-- Create tables
DROP TABLE IF EXISTS Students;
DROP TABLE IF EXISTS Courses;
CREATE TABLE Students (
    StudentID INT,
    Name VARCHAR(50),
    CourseID INT
);
CREATE TABLE Courses (
    CourseID INT,
    CourseName VARCHAR(50)
);
-- Insert data
INSERT INTO Students VALUES
(1, 'John', 101),
(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;
```

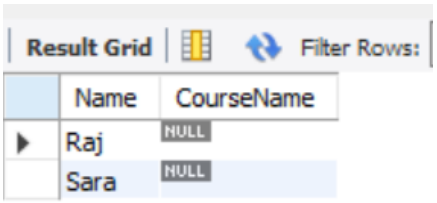
Output:

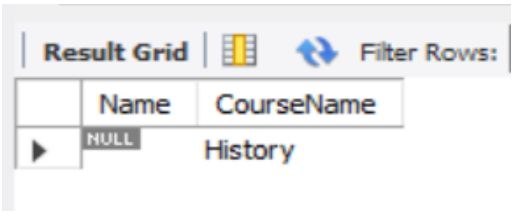

Result Grid
Filter Ro
Name
CourseName
John
Math
Emma
Science


Result Grid
Filter Rows:
Name
CourseName
John
Math
Emma
Science
Raj
NULL
Sara
NULL


Result Grid
Filter Rows
Name
CourseName
John
Math
Emma
Science
NULL
History


Result Grid
Filter Rows:
Name
CourseName
John
Math
Emma
Science
Raj
NULL
Sara
NULL
NULL
History


Result Grid
Filter Rows:
Name
CourseName
Raj
NULL
Sara
NULL


Result Grid
Filter Rows:
Name
CourseName
NULL
History

Experiment 8 : Normal Forms

1. INF

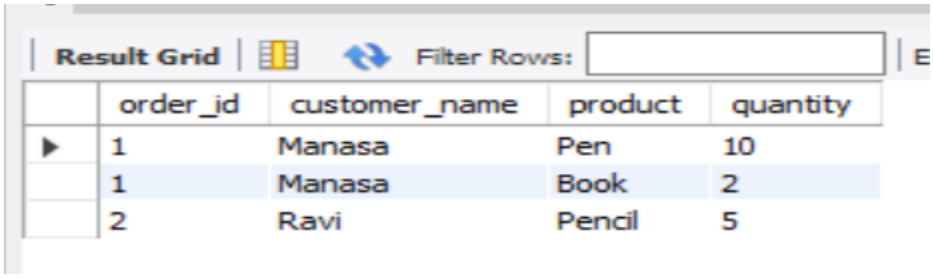
Source Code:

```
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:

```sql
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;
```

Output:



Result Grid | Filter Rows:

order_id	customer_name	product	quantity
1	Manasa	Pen	10
1	Manasa	Book	2
2	Ravi	Pencil	5

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:

```sql
-- 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;
```

Output:

Result Grid | Filter Rows:

order_id	cust_name	product	quantity
101	Manasa	Pen	10
101	Manasa	Book	2
102	Ravi	Pencil	5

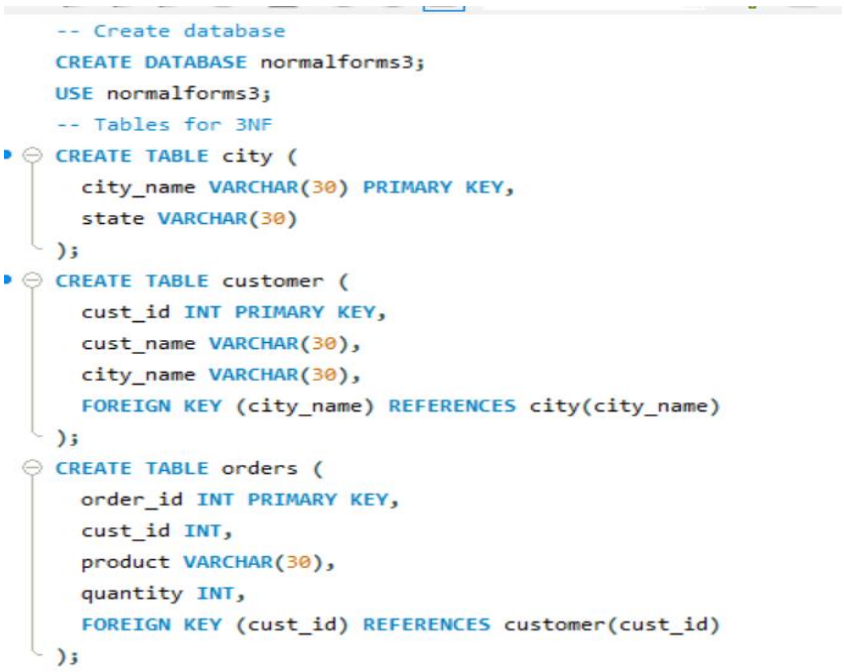
3. 3NF

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;
```

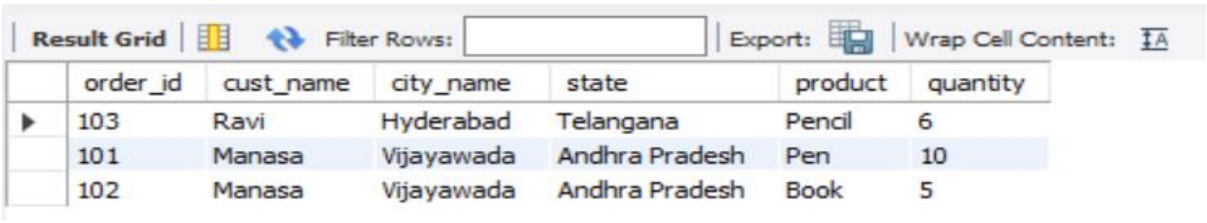
Program:



```
-- 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;
```

Output:



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

order_id	cust_name	city_name	state	product	quantity
103	Ravi	Hyderabad	Telangana	Pencil	6
101	Manasa	Vijayawada	Andhra Pradesh	Pen	10
102	Manasa	Vijayawada	Andhra Pradesh	Book	5

Experiment 9 :Nested Queries Using select across two tables

Source Code:

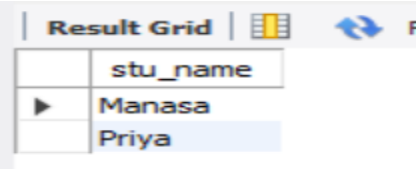
```
CREATE TABLE department (
  dept_id INT PRIMARY KEY,
  dept_name VARCHAR(20)
);
CREATE TABLE student (
  stu_id INT PRIMARY KEY,
  stu_name VARCHAR(20),
  dept_id INT,
  FOREIGN KEY (dept_id) REFERENCES department(dept_id)
);
INSERT INTO department VALUES (1, 'CSE'), (2, 'ECE');
INSERT INTO student 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:

```
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');
```

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

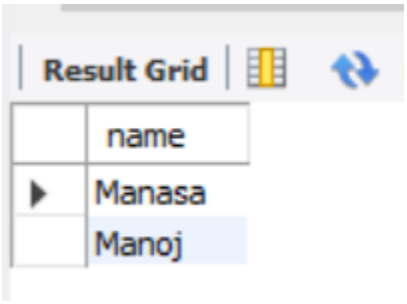
FROM student

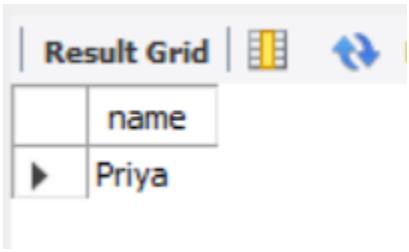
WHERE LOWER(name) LIKE '%ri%';

Program:

```
 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%';
```

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:

```sql
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;
```

Output:

Result Grid | Filter Rows:

	name	marks	Condition
▶	Manasa	85	Greater than 80
	Ravi	72	Less than 75
	Manoj	90	Greater than 80
	Priya	78	Equal to 78
	Kiran	60	Less than 75

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';

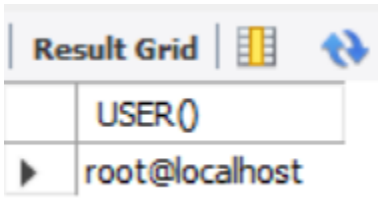
-- Get the MySQL version

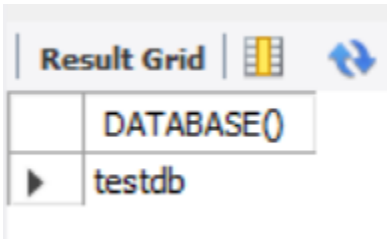
SELECT VERSION();

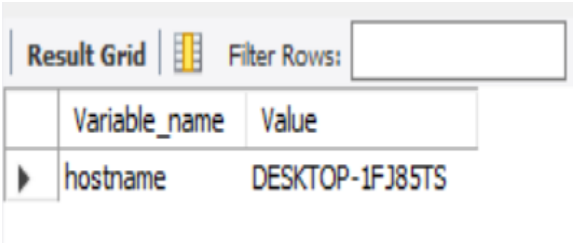
Program:

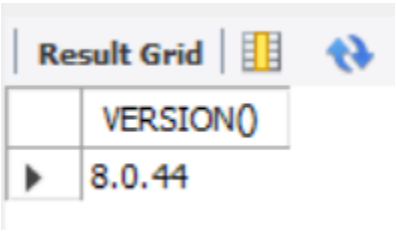
```
-- Get the current MySQL user
SELECT USER();

-- Get the current selected database
SELECT DATABASE();

-- Get the hostname (server name)
SHOW VARIABLES LIKE 'hostname';

-- Get the MySQL version
SELECT VERSION();
```

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;

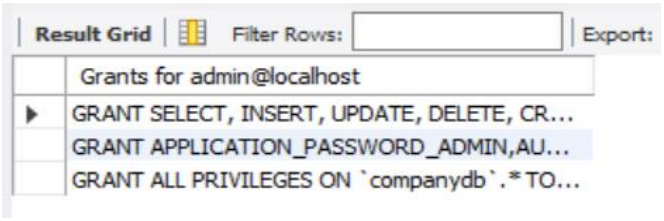
CREATE TABLE students (id INT PRIMARY KEY, name VARCHAR(50), age INT);

Program:

```
-- 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);
-- Experiment: Create Users and Grant Privileges in MySQL

-- Step 1: Drop database if it already exists and create a new one
DROP DATABASE IF EXISTS companydb;
CREATE DATABASE companydb;
USE companydb;

-- Step 2: Create users (admin and staff)
CREATE USER IF NOT EXISTS 'admin'@'localhost' IDENTIFIED BY 'admin123';
CREATE USER IF NOT EXISTS 'staff'@'localhost' IDENTIFIED BY 'staff123';

-- Step 3: Grant privileges
-- Admin has full access to companydb
GRANT ALL PRIVILEGES ON companydb.* TO 'admin'@'localhost';

-- Staff has limited access (can only SELECT and INSERT)
GRANT SELECT, INSERT ON companydb.* TO 'staff'@'localhost';
```

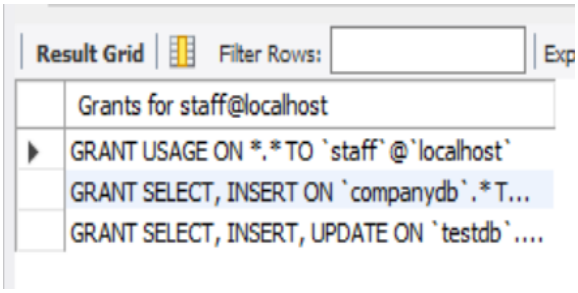
```
-- Step 4: Apply privilege changes
FLUSH PRIVILEGES;

-- Step 5: Verify the privileges for each user
SHOW GRANTS FOR 'admin'@'localhost';
SHOW GRANTS FOR 'staff'@'localhost';
```

Output:



Result Grid | Filter Rows: | Export:

	Grants for admin@localhost
▶	GRANT SELECT, INSERT, UPDATE, DELETE, CR...
	GRANT APPLICATION_PASSWORD_ADMIN,AU...
	GRANT ALL PRIVILEGES ON `companydb`.* TO...



Result Grid | Filter Rows: | Exp

	Grants for staff@localhost
▶	GRANT USAGE ON *.* TO `staff`@`localhost`
	GRANT SELECT, INSERT ON `companydb`.* T...
	GRANT SELECT, INSERT, UPDATE ON `testdb`....