**EXPERIMENT 1**

**Create a table with constraints**

1. **NOT NULL Constraint:** NOT NULL ensures that no NULL values are allowed in the columns.
2. **UNIQUE Constraint:** UNIQUE ensures all values in columns are different (no duplicates allowed).
3. **PRIMARY KEY Constraint:** PRIMARY KEY uniquely identifies each row and implies NOT NULL + UNIQUE.
4. **FOREIGN KEY Constraint:** A **FOREIGN KEY** is a column (or set of columns) in one table that references the primary key in another table to enforce referential integrity.
5. **CHECK Constraint:** CHECK enforces logical rules (e.g., quantity must be between 1 and 999).
6. **DEFAULT Constraint:** DEFAULT automatically sets a value if none is given (e.g., today's date).
7. **CREATE INDEX Constraint:** INDEX improves query performance on large tables, especially for searching/sorting.

**Source code:**

CREATE TABLE departments (

dept\_id INT PRIMARY KEY,

dept\_name VARCHAR(100) UNIQUE NOT NULL);

**Source code:**

CREATE TABLE employees (

emp\_id INT PRIMARY KEY,

first\_name VARCHAR(50) NOT NULL,

last\_name VARCHAR(50) NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

hire\_date DATE DEFAULT CURRENT\_DATE,

salary DECIMAL(10,2) CHECK (salary >= 30000),

dept\_id INT, CONSTRAINT fk\_department FOREIGN KEY (dept\_id)

REFERENCES departments(dept\_id));

**Source code:**

CREATE INDEX idx\_lastname ON employees(last\_name);

**EXPERIMENT 2**

**Implementation of SQL Commands**

1. **Insert values with a single entry**: Adds one row of data into a table using the INSERT INTO statement.
2. **Insert values with multiple entries**: Adds multiple rows at once using a single INSERT INTO statement with multiple value sets.
3. **ALTER Table Structure**: Modifies the structure of an existing table, such as adding or removing columns.
4. **VIEW Table structure**: Displays the schema of a table using commands like DESCRIBE or SHOW COLUMNS.
5. **UPDATE table**: Changes existing data in one or more rows using the UPDATE statement with a WHERE clause.
6. **DELETE Rows in table**: Removes specific rows from a table using the DELETE FROM statement with a WHERE clause.
7. **DROP table**: Permanently deletes an entire table and all its data from the database.

|  |
| --- |
|  |

**Source code:**

CREATE TABLE products (

product\_id INT PRIMARY KEY,

product\_name VARCHAR(100) NOT NULL,

price DECIMAL(10, 2) NOT NULL,

quantity INT NOT NULL);

INSERT INTO products (product\_id, product\_name, price, quantity)

VALUES (101, 'Wireless Mouse', 25.99, 100);

INSERT INTO products (product\_id, product\_name, price, quantity) VALUES (102, 'USB Keyboard', 19.99, 150);

INSERT INTO products (product\_id, product\_name, price, quantity) VALUES (103, 'HDMI Cable', 9.99, 200);

INSERT INTO products (product\_id, product\_name, price, quantity) VALUES (104, 'Laptop Stand', 34.50, 75);

INSERT ALL

INTO products (product\_id, product\_name, price, quantity) VALUES (201, 'Gaming Mouse', 45.99, 80)

INTO products (product\_id, product\_name, price, quantity) VALUES (202, 'Mechanical Keyboard', 89.99, 60)

INTO products (product\_id, product\_name, price, quantity) VALUES (203, 'Webcam HD', 59.49, 120)

SELECT \* FROM dual;

**Source code:**

ALTER TABLE products

ADD category VARCHAR2(50);

**Source code:**

ALTER TABLE products

REBNAME COLUMN quantity TO stock\_available;

**Source code:**

ALTER TABLE products

DROP COLUMN CATEGORY;

**Source code:**

CREATE VIEW product\_summary AS

SELECT products\_id,product\_name,price

FROM products

WHERE price > 50;

**Source code:**

UPDATE products

SET price=100

WHERE product\_id=101;

**Source code:**

DELETE FROM products

WHERE products\_id=103;

**Source code:**

DROP TABLE products;

**EXPERIMENT 3**

**Aggregate Function**

1. **MIN()**: Returns the smallest value in a column.
2. **MAX()**: Returns the largest value in a column.
3. **COUNT()**: Returns the number of rows that match a specified condition.
4. **SUM()**: Calculates the total sum of a numeric column.
5. **AVG()**: Computes the average value of a numeric column.

**SourceCode:**

CREATE TABLE fruits (

product\_id INT PRIMARY KEY,

product\_name VARCHAR2(100) NOT NULL,

price NUMBER(10,2) NOT NULL,

quantity INT NOT NULL);

INSERT INTO fruits (product\_id, product\_name, price, quantity) VALUES (101, 'Apple', 0.99, 100);

INSERT INTO fruits (product\_id, product\_name, price, quantity) VALUES (102, 'Banana', 0.59, 150);

INSERT INTO fruits (product\_id, product\_name, price, quantity) VALUES (103, 'Orange', 1.25, 200);

INSERT INTO fruits (product\_id, product\_name, price, quantity) VALUES (104, 'Mango', 2.50, 75);

INSERT INTO fruits (product\_id, product\_name, price, quantity) VALUES (201, 'Grapes', 3.00, 80);

INSERT INTO fruits (product\_id, product\_name, price, quantity) VALUES (202, 'Pineapple', 2.99, 60);

INSERT INTO fruits (product\_id, product\_name, price, quantity) VALUES (203, 'Watermelon', 5.49, 120);

SELECT COUNT(\*) AS total\_fruits

FROM fruits;

**SourceCode:**

SELECT SUM(quantity) AS total\_quantity

FROM fruits;

**SourceCode:**

SELECT AVG(price) AS avg\_price

FROM fruits;

**SourceCode:**

SELECT MAX(price) AS max\_price,

MIN(price) AS min\_price

FROM fruits;

**EXPERIMENT 4**

1. **GROUP BY**: Organizes rows into groups based on one or more columns, often used with aggregate functions.
2. **ORDER BY**: Sorts the result set of a query by one or more columns in ascending (ASC) or descending (DESC) order.

**Source code:**

CREATE TABLE sales (

sale\_id INT PRIMARY KEY,

product\_name VARCHAR2(100) NOT NULL,

category VARCHAR2(50),

quantity\_sold INT NOT NULL,

sale\_amount NUMBER(10,2) NOT NULL);

INSERT INTO sales (sale\_id, product\_name, category, quantity\_sold, sale\_amount) VALUES (1, 'Apple', 'Fruit', 50, 49.50);

INSERT INTO sales (sale\_id, product\_name, category, quantity\_sold, sale\_amount) VALUES (2, 'Banana', 'Fruit', 30, 17.70);

INSERT INTO sales (sale\_id, product\_name, category, quantity\_sold, sale\_amount) VALUES (3, 'Orange', 'Fruit', 40, 50.00);

INSERT INTO sales (sale\_id, product\_name, category, quantity\_sold, sale\_amount) VALUES (4, 'Mango', 'Fruit', 20, 50.00);

INSERT INTO sales (sale\_id, product\_name, category, quantity\_sold, sale\_amount) VALUES (5, 'Soap', 'Grocery', 15, 45.00);

INSERT INTO sales (sale\_id, product\_name, category, quantity\_sold, sale\_amount) VALUES (6, 'Shampoo', 'Grocery', 10, 60.00);

INSERT INTO sales (sale\_id, product\_name, category, quantity\_sold, sale\_amount) VALUES (7, 'Notebook', 'Stationery', 25, 75.00);

INSERT INTO sales (sale\_id, product\_name, category, quantity\_sold, sale\_amount) VALUES (8, 'Pen', 'Stationery', 50, 25.00);

**Source code:**

SELECT category, SUM(quantity\_sold) AS total\_quantity, SUM(sale\_amount) AS total\_sales

FROM salesGROUP BY category;

**Source code:**

SELECT product\_name, category, quantity\_sold, sale\_amount

FROM sales

ORDER BY sale\_amount DESC;

**Source code:**

SELECT product\_name, category, quantity\_sold, sale\_amount

FROM sales

ORDER BY category ASC, sale\_amount DESC;

**Source code:**

SELECT category, SUM(quantity\_sold) AS total\_quantity, SUM(sale\_amount) AS total\_sales

FROM sales

GROUP BY category

ORDER BY total\_sales DESC;

**EXPERIMENT 5**

1. **Ascending: ASC** sorts query results from lowest to highest.
2. **Descending: DESC** sorts from highest to lowest based on the specified column.

**Source code:**

CREATE TABLE planets (

planet\_id INT PRIMARY KEY,

planet\_name VARCHAR2(50) NOT NULL,

distance\_from\_sun NUMBER(10,2),

diameter NUMBER(10,2));

INSERT INTO planets (planet\_id, planet\_name, distance\_from\_sun, diameter) VALUES (1, 'Mercury', 57.9, 4879);

INSERT INTO planets (planet\_id, planet\_name, distance\_from\_sun, diameter) VALUES (2, 'Venus', 108.2, 12104);

INSERT INTO planets (planet\_id, planet\_name, distance\_from\_sun, diameter) VALUES (3, 'Earth', 149.6, 12756);

INSERT INTO planets (planet\_id, planet\_name, distance\_from\_sun, diameter) VALUES (4, 'Mars', 227.9, 6792);

INSERT INTO planets (planet\_id, planet\_name, distance\_from\_sun, diameter) VALUES (5, 'Jupiter', 778.3, 142984);

INSERT INTO planets (planet\_id, planet\_name, distance\_from\_sun, diameter) VALUES (6, 'Saturn', 1427.0, 120536);

INSERT INTO planets (planet\_id, planet\_name, distance\_from\_sun, diameter) VALUES (7, 'Uranus', 2871.0, 51118);

INSERT INTO planets (planet\_id, planet\_name, distance\_from\_sun, diameter) VALUES (8, 'Neptune', 4497.1, 49528);

**Source code:**

Select \* from planets;

SELECT planet\_name, distance\_from\_sun, diameter

FROM planets

ORDER BY distance\_from\_sun ASC;

**Source code:**

SELECT planet\_name, distance\_from\_sun, diameter

FROM planets

ORDER BY diameter DESC;

**EXPERIMENT 6**

**SQL Operators**

1. **LIKE**: Filters results based on pattern matching using % (any characters) and \_ (single character).
2. **BETWEEN**: Checks if a value lies within a specified inclusive range.
3. **OR**: Returns results if **any** of the given conditions are true.

**Source code:**

CREATE TABLE Suppliers(SupplierID INT PRIMARY KEY,SupplierName VARCHAR(100) NOT NULL,City VARCHAR(50));

CREATE SEQUENCE Suppliers\_seq START WITH 1 INCREMENT BY 1;

CREATE OR REPLACE TRIGGER Suppliers\_on\_insert

BEFORE INSERT ON Suppliers

FOR EACH ROW

BEGIN

SELECT Suppliers\_seq.nextval

INTO :new.SupplierID

FROM dual;

END;

CREATE TABLE Products (ProductID INT PRIMARY KEY,ProductName VARCHAR(100) NOT NULL,Category VARCHAR(50),Price NUMBER(10, 2),StockQuantity INT,SupplierID INT,FOREIGN KEY(SupplierID) REFERENCES Suppliers(SupplierID));

CREATE SEQUENCE Products\_seq

START WITH 1

INCREMENT BY 1;

CREATE OR REPLACE TRIGGER Products\_on\_insert

BEFORE INSERT ON Products

FOR EACH ROW

BEGIN

SELECT Products\_seq.nextval

INTO :new.ProductID

FROM dual;

END;

INSERT INTO Suppliers (SupplierName, City) VALUES ('ToolMaster Pro', 'New York');

INSERT INTO Suppliers (SupplierName, City) VALUES ('Eastern Lumber Co.', 'Boston');

INSERT INTO Suppliers (SupplierName, City) VALUES ('Quick Fasteners Ltd.', 'Miami');

INSERT INTO Suppliers (SupplierName, City) VALUES ('PowerHouse Electric', 'Chicago');

INSERT INTO Suppliers (SupplierName, City) VALUES ('The Metal Works', 'Seattle');

INSERT INTO Suppliers (SupplierName, City) VALUES ('Apex Safety Gear', 'Dallas');

INSERT INTO Suppliers (SupplierName, City) VALUES ('Prime Plumbing Supply', 'Phoenix');

INSERT INTO Suppliers (SupplierName, City) VALUES ('Global Adhesives', 'Denver');

INSERT INTO Suppliers (SupplierName, City) VALUES ('Brick & Mortar Co.', 'Atlanta');

INSERT INTO Suppliers (SupplierName, City) VALUES ('Precision Measuring', 'Houston');

INSERT INTO Suppliers (SupplierName, City) VALUES ('Super Wrench Group', 'New York');

INSERT INTO Suppliers (SupplierName, City) VALUES ('Cedar Creek Wood', 'Boston');

INSERT INTO Suppliers (SupplierName, City) VALUES ('Volta Electrical', 'Miami');

INSERT INTO Suppliers (SupplierName, City) VALUES ('Ironclad Hardware', 'Chicago');

INSERT INTO Suppliers (SupplierName, City) VALUES ('AquaFlow Plumbing', 'Seattle');

INSERT INTO Suppliers (SupplierName, City) VALUES ('SureGrip Fasteners', 'Dallas');

INSERT INTO Suppliers (SupplierName, City) VALUES ('Bright Light Solutions', 'Phoenix');

INSERT INTO Suppliers (SupplierName, City) VALUES ('Durable Paint Co.', 'Denver');

INSERT INTO Suppliers (SupplierName, City) VALUES ('Contractor Essentials', 'Atlanta');

INSERT INTO Suppliers (SupplierName, City) VALUES ('Home Fix Depot', 'Houston');

INSERT INTO Products (ProductName, Category, Price, StockQuantity, SupplierID) VALUES ('Hammer Pro Titanium', 'Tool', 39.99, 150, 1);

INSERT INTO Products (ProductName, Category, Price, StockQuantity, SupplierID) VALUES ('1/4 inch Hex Bolts', 'Fastener', 5.25, 500, 3);

INSERT INTO Products (ProductName, Category, Price, StockQuantity, SupplierID) VALUES ('LED Flood Light', 'Electrical', 48.50, 120, 4);

INSERT INTO Products (ProductName, Category, Price, StockQuantity, SupplierID) VALUES ('Red Cedar 4x4 Post', 'Wood', 18.75, 80, 12);

INSERT INTO Products (ProductName, Category, Price, StockQuantity, SupplierID) VALUES ('Adjustable Wrench 12"', 'Tool', 22.00, 120, 11);

INSERT INTO Products (ProductName, Category, Price, StockQuantity, SupplierID) VALUES ('Hex Head Screws Zinc', 'Fastener', 6.00, 450, 16);

INSERT INTO Products (ProductName, Category, Price, StockQuantity, SupplierID) VALUES ('Copper Wire Spool 12g', 'Electrical', 75.99, 100, 13);

INSERT INTO Products (ProductName, Category, Price, StockQuantity, SupplierID) VALUES ('Oak Plywood 3/4"', 'Wood', 65.00, 50, 12);

INSERT INTO Products (ProductName, Category, Price, StockQuantity, SupplierID) VALUES ('Drill Bit Set (50pcs)', 'Tool', 59.99, 80, 1);

INSERT INTO Products (ProductName, Category, Price, StockQuantity, SupplierID) VALUES ('Masonry Nails (Bulk)', 'Fastener', 12.50, 600, 3);

INSERT INTO Products (ProductName, Category, Price, StockQuantity, SupplierID) VALUES ('White Outlet Cover', 'Electrical', 2.99, 750, 13);

INSERT INTO Products (ProductName, Category, Price, StockQuantity, SupplierID) VALUES ('Pressure Treated Pine', 'Wood', 6.80, 200, 5);

INSERT INTO Products (ProductName, Category, Price, StockQuantity, SupplierID) VALUES ('Safety Goggles Anti-Fog', 'Safety', 10.99, 250, 6);

INSERT INTO Products (ProductName, Category, Price, StockQuantity, SupplierID) VALUES ('Aluminum Sheet Metal', 'Metal', 88.00, 40, 5);

INSERT INTO Products (ProductName, Category, Price, StockQuantity, SupplierID) VALUES ('Magnetic Screwdriver Set', 'Tool', 14.50, 180, 11);

INSERT INTO Products (ProductName, Category, Price, StockQuantity, SupplierID) VALUES ('Teflon Tape 1/2"', 'Plumbing', 1.99, 900, 7);

INSERT INTO Products (ProductName, Category, Price, StockQuantity, SupplierID) VALUES ('PVC Pipe Connector 1"', 'Plumbing', 4.50, 320, 15);

INSERT INTO Products (ProductName, Category, Price, StockQuantity, SupplierID) VALUES ('Heavy Duty Glue Stick', 'Adhesive', 3.75, 400, 8);

INSERT INTO Products (ProductName, Category, Price, StockQuantity, SupplierID) VALUES ('Laser Measure Pro 60m', 'Measuring', 120.00, 30, 10);

INSERT INTO Products (ProductName, Category, Price, StockQuantity, SupplierID) VALUES ('Galvanized Steel Pipe', 'Plumbing', 55.00, 60, 15);

SELECT ProductName, Price, Category FROM Products WHERE ProductName LIKE '%Pro%';

SELECT ProductName, Price FROM Products WHERE Price BETWEEN 10.00 AND 50.00;

SELECT SupplierName, City FROM Suppliers WHERE City = 'New York' OR City = 'Chicago';

**EXPERIMENT 7**

**SQL Joins**

1. **INNER JOIN**: Returns rows with matching values in both tables.
2. **LEFT JOIN**: Returns all rows from the left table and matched rows from the right.
3. **RIGHT JOIN**: Returns all rows from the right table and matched rows from the left.
4. **OUTER JOIN**: Returns all rows when there is a match in one of the tables.
5. **LEFT JOIN excluding INNER JOIN**: Returns unmatched rows from the left table only.
6. **RIGHT JOIN excluding INNER JOIN**: Returns unmatched rows from the right table only.
7. **OUTER JOIN excluding INNER JOIN**: Returns unmatched rows from both tables.

**Source code:**

SELECT P.ProductName, S.SupplierName FROM Products P INNER JOIN Suppliers S ON P.SupplierID = S.SupplierID;

**Source code:**

SELECT P.ProductName, S.SupplierName FROM Products P LEFT JOIN Suppliers S ON P.SupplierID = S.SupplierID;

**Source code:**

SELECT P.ProductName, S.SupplierName, S.City FROM Products P RIGHT JOIN Suppliers S ON P.SupplierID = S.SupplierID;

**Source code:**

SELECT P.ProductName, S.SupplierName, S.City FROM Products P LEFT JOIN Suppliers S ON P.SupplierID = S.SupplierID UNION ALL SELECT P.ProductName, S.SupplierName, S.City FROM Products P RIGHT JOIN Suppliers S ON P.SupplierID = S.SupplierID WHERE P.SupplierID IS NULL;

**Source code:**

SELECT P.ProductName, S.SupplierName FROM Products P LEFT JOIN Suppliers S ON P.SupplierID = S.SupplierID WHERE S.SupplierID IS NULL;

SELECT S.SupplierName, S.City, P.ProductName FROM Products P RIGHT JOIN Suppliers S ON P.SupplierID = S.SupplierID WHERE P.SupplierID IS NULL;

**Source code:**

SELECT P.ProductName, S.SupplierName, S.City FROM Products P LEFT JOIN Suppliers S ON P.SupplierID = S.SupplierID WHERE S.SupplierID IS NULL UNION ALL SELECT P.ProductName, S.SupplierName, S.City FROM Products P RIGHT JOIN Suppliers S ON P.SupplierID = S.SupplierID WHERE P.SupplierID IS NULL;

**EXPERIMENT 8**

**Normal Forms**

1. **1NF (First Normal Form)**: Eliminates repeating groups; ensures atomic values in each column.
2. **2NF (Second Normal Form)**: Removes partial dependencies; every non-key attribute fully depends on the primary key.
3. **3NF:** Non-key columns depend only on the whole primary key — not on other non-key columns.

**Source code:1NF**

CREATE TABLE student\_1nf (

student\_id INT,

student\_name VARCHAR(50),

subject VARCHAR(50));

INSERT INTO student\_1nf VALUES (1, 'Alice', 'Math');

INSERT INTO student\_1nf VALUES (1, 'Alice', 'Science');

INSERT INTO student\_1nf VALUES (2, 'Bob', 'English');

INSERT INTO student\_1nf VALUES (2, 'Bob', 'History');

SELECT \* FROM student\_1nf;

**Source code:2NF:**

CREATE TABLE Students2 (

student\_id INT PRIMARY KEY,

student\_name VARCHAR2(50));

CREATE TABLE Courses2 (

course\_id INT PRIMARY KEY,

course\_name VARCHAR2(50),

instructor\_name VARCHAR2(50));

CREATE TABLE Enrollments2 (

student\_id INT,

course\_id INT,

PRIMARY KEY (student\_id, course\_id),

FOREIGN KEY (student\_id) REFERENCES Students2(student\_id),

FOREIGN KEY (course\_id) REFERENCES Courses2(course\_id));

INSERT INTO Students2 VALUES (1, 'Alice');

INSERT INTO Students2 VALUES (2, 'Bob');

INSERT INTO Students2 VALUES (3, 'Charlie');

INSERT INTO Courses2 VALUES (101, 'Database Systems', 'Dr. Smith');

INSERT INTO Courses2 VALUES (102, 'Operating Systems', 'Prof. Brown');

INSERT INTO Courses2 VALUES (103, 'Networks', 'Dr. Green');

INSERT INTO Enrollments2 VALUES (1, 101);

INSERT INTO Enrollments2 VALUES (1, 102);

INSERT INTO Enrollments2 VALUES (2, 103);

INSERT INTO Enrollments2 VALUES (3, 101);

COMMIT;

SELECT

s.student\_id,

s.student\_name,

c.course\_id,

c.course\_name,

c.instructor\_name

FROM Enrollments2 e

JOIN Students2 s ON e.student\_id = s.student\_id

JOIN Courses2 c ON e.course\_id = c.course\_id

ORDER BY s.student\_id, c.course\_id;

**Source code: 3NF**

CREATE TABLE Students3 (

student\_id INT PRIMARY KEY,

student\_name VARCHAR2(50));

CREATE TABLE Instructors3 (

instructor\_id INT PRIMARY KEY,

instructor\_name VARCHAR2(50));

CREATE TABLE Courses3 (

course\_id INT PRIMARY KEY,

course\_name VARCHAR2(50),

instructor\_id INT,

FOREIGN KEY (instructor\_id) REFERENCES Instructors3(instructor\_id));

CREATE TABLE Enrollments3 (

student\_id INT,

course\_id INT,

PRIMARY KEY (student\_id, course\_id),

FOREIGN KEY (student\_id) REFERENCES Students3(student\_id),

FOREIGN KEY (course\_id) REFERENCES Courses3(course\_id));

INSERT INTO Students3 VALUES (1, 'Alice');

INSERT INTO Students3 VALUES (2, 'Bob');

INSERT INTO Students3 VALUES (3, 'Charlie');

INSERT INTO Instructors3 VALUES (201, 'Dr. Smith');

INSERT INTO Instructors3 VALUES (202, 'Prof. Brown');

INSERT INTO Instructors3 VALUES (203, 'Dr. Green');

INSERT INTO Courses3 VALUES (101, 'Database Systems', 201);

INSERT INTO Courses3 VALUES (102, 'Operating Systems', 202);

INSERT INTO Courses3 VALUES (103, 'Networks', 203);

INSERT INTO Enrollments3 VALUES (1, 101);

INSERT INTO Enrollments3 VALUES (1, 102);

INSERT INTO Enrollments3 VALUES (2, 103);

INSERT INTO Enrollments3 VALUES (3, 101);

COMMIT;

SELECT

s.student\_id,

s.student\_name,

c.course\_id,

c.course\_name,

i.instructor\_name

FROM Enrollments3 e

JOIN Students3 s ON e.student\_id = s.student\_id

JOIN Courses3 c ON e.course\_id = c.course\_id

JOIN Instructors3 i ON c.instructor\_id = i.instructor\_id

ORDER BY s.student\_id, c.course\_id;

**EXPERIMENT 9**

**Nested Queries**: A query within another SQL query, used to perform intermediate filtering or calculations.

**Source code:**

CREATE TABLE department ( dept\_id INT PRIMARY KEY, dept\_name VARCHAR(50));

CREATE SEQUENCE dept\_seq START WITH 1 INCREMENT BY 1;

CREATE OR REPLACE TRIGGER dept\_on\_insert

BEFORE INSERT ON department

FOR EACH ROW

BEGIN

SELECT dept\_seq.nextval INTO :new.dept\_id FROM dual;

END;

CREATE TABLE employee (emp\_id INT PRIMARY KEY,emp\_name VARCHAR(50),salary NUMBER(10,2), dept\_id INT, FOREIGN KEY (dept\_id) REFERENCES department(dept\_id));

CREATE SEQUENCE emp\_seq START WITH 1 INCREMENT BY 1;

CREATE OR REPLACE TRIGGER emp\_on\_insert

BEFORE INSERT ON employee

FOR EACH ROW

BEGIN

SELECT emp\_seq.nextval INTO :new.emp\_id FROM dual;

END;

INSERT INTO employee (emp\_name, salary, dept\_id) VALUES ('Alice', 50000, (SELECT dept\_id FROM department WHERE dept\_name = 'HR'));

INSERT INTO employee (emp\_name, salary, dept\_id) VALUES ('Bob', 60000, (SELECT dept\_id FROM department WHERE dept\_name = 'HR'));

INSERT INTO employee (emp\_name, salary, dept\_id) VALUES ('Charlie', 55000, (SELECT dept\_id FROM department WHERE dept\_name = 'HR'));

INSERT INTO employee (emp\_name, salary, dept\_id) VALUES ('David', 80000, (SELECT dept\_id FROM department WHERE dept\_name = 'IT'));

INSERT INTO employee (emp\_name, salary, dept\_id) VALUES ('Eve', 90000, (SELECT dept\_id FROM department WHERE dept\_name = 'IT'));

INSERT INTO employee (emp\_name, salary, dept\_id) VALUES ('Frank', 85000, (SELECT dept\_id FROM department WHERE dept\_name = 'IT'));

INSERT INTO employee (emp\_name, salary, dept\_id) VALUES ('Grace', 70000, (SELECT dept\_id FROM department WHERE dept\_name = 'Finance'));

INSERT INTO employee (emp\_name, salary, dept\_id) VALUES ('Hannah', 75000, (SELECT dept\_id FROM department WHERE dept\_name = 'Finance'));

INSERT INTO employee (emp\_name, salary, dept\_id) VALUES ('Irene', 45000, (SELECT dept\_id FROM department WHERE dept\_name = 'HR'));

INSERT INTO employee (emp\_name, salary, dept\_id) VALUES ('Jack', 100000, (SELECT dept\_id FROM department WHERE dept\_name = 'IT'));

INSERT INTO employee (emp\_name, salary, dept\_id) VALUES ('Kelly', 68000, (SELECT dept\_id FROM department WHERE dept\_name = 'Finance'));

INSERT INTO employee (emp\_name, salary, dept\_id) VALUES ('Liam', 62000, (SELECT dept\_id FROM department WHERE dept\_name = 'HR'));

INSERT INTO employee (emp\_name, salary, dept\_id) VALUES ('Mia', 95000, (SELECT dept\_id FROM department WHERE dept\_name = 'IT'));

COMMIT;

SELECT emp\_name, salary, dept\_id

FROM employee

WHERE salary > ALL (

SELECT salary

FROM employee e\_hr

WHERE e\_hr.dept\_id = (

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

SELECT emp\_name, salary

FROM employee

WHERE dept\_id NOT IN (

SELECT dept\_id

FROM department

WHERE dept\_name = 'IT');

SELECT e.emp\_name, e.salary, d\_avg.Finance\_Avg\_Salary

FROM employee e

CROSS JOIN (

SELECT AVG(salary) AS Finance\_Avg\_Salary

FROM employee

WHERE dept\_id = (SELECT dept\_id FROM department WHERE dept\_name = 'Finance')) d\_avg;

**EXPERIMENT 10**

**SQL WILD CARD CHARACTERS**

1. **%**: Represents **zero or more characters** in a string.
2. **\_**: Represents **exactly one character**.

**Source code:**

CREATE TABLE Employees\_WC (

emp\_id INT PRIMARY KEY,

emp\_name VARCHAR2(50),

department VARCHAR2(50));

INSERT INTO Employees\_WC VALUES (1, 'Alice Johnson', 'HR');

INSERT INTO Employees\_WC VALUES (2, 'Bob Smith', 'Finance');

INSERT INTO Employees\_WC VALUES (3, 'Charlie Brown', 'IT');

INSERT INTO Employees\_WC VALUES (4, 'Alicia Keys', 'HR');

INSERT INTO Employees\_WC VALUES (5, 'Albert King', 'Sales');

INSERT INTO Employees\_WC VALUES (6, 'Bobby Ray', 'Finance');

COMMIT;

SELECT \* FROM Employees\_WC

WHERE emp\_name LIKE 'Al%';

SELECT \* FROM Employees\_WC

WHERE emp\_name LIKE '%son';

SELECT \* FROM Employees\_WC

WHERE emp\_name LIKE '%ob%';

SELECT \* FROM Employees\_WC

WHERE emp\_name LIKE '\_l%';

SELECT \* FROM Employees\_WC

WHERE department LIKE 'F%';

**EXPERIMENT 11**

**SELECT with Comparison Operator**: Retrieves rows that meet specific conditions using operators like =, >, <, >=, <=, <>.

**Source code:**

SELECT SupplierName, City FROM Suppliers WHERE SupplierID = 5;

SELECT SupplierName, SupplierID FROM Suppliers WHERE SupplierID > 15;

SELECT SupplierName, SupplierID FROM Suppliers WHERE SupplierID < 4;

SELECT SupplierName, SupplierID FROM Suppliers WHERE SupplierID >= 10;

SELECT SupplierName, SupplierID FROM Suppliers WHERE SupplierID <= 6;

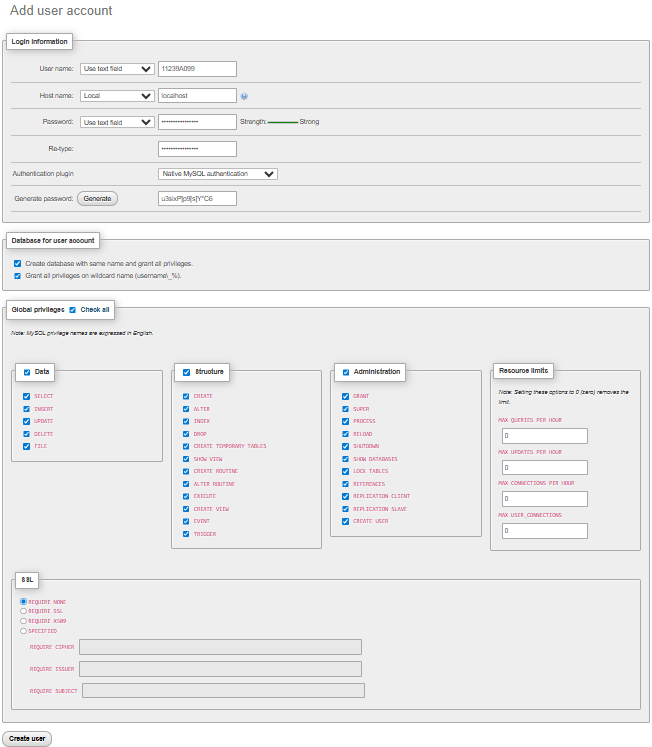
SELECT SupplierName, City FROM Suppliers WHERE City <> 'New York';

**EXPERIMENT 12**

**Working on Local Host XAMPP Server**

1. **Server Variables in XAMPP**: Provide environment and request details via PHP’s $\_SERVER array.

**OUTPUT:**

****

1. **Hierarchical User Access in XAMPP**: Assign different privileges to MySQL users to control database access levels.

**OUTPUT:**

****