**Deep Skilling Week 2 Assignment**

**Exercise 1: Ranking and Window Functions**

**Code:**

CREATE DATABASE db;

USE db;

CREATE TABLE Products (

ProductID INT PRIMARY KEY,

ProductName VARCHAR(100),

Category VARCHAR(100),

Price DECIMAL(10, 2)

);

INSERT INTO Products (ProductID, ProductName, Category, Price) VALUES

(1, 'Laptop', 'Electronics', 1000),

(2, 'Smartphone', 'Electronics', 800),

(3, 'Tablet', 'Electronics', 1000),

(4, 'TV', 'Electronics', 700),

(5, 'Jeans', 'Clothing', 500),

(6, 'T-Shirt', 'Clothing', 300),

(7, 'Jacket', 'Clothing', 700),

(8, 'Shoes', 'Clothing', 300),

(9, 'Watch', 'Accessories', 600),

(10, 'Belt', 'Accessories', 400),

(11, 'Hat', 'Accessories', 600),

(12, 'Sunglasses', 'Accessories', 200);

SELECT \*FROM (

SELECT ProductID, ProductName, Category, Price,

ROW\_NUMBER() OVER (PARTITION BY Category ORDER BY Price DESC) AS RowNum

FROM Products

) AS ranked

WHERE RowNum <= 3;

SELECT \*

FROM (

SELECT ProductID, ProductName, Category, Price,

RANK() OVER (PARTITION BY Category ORDER BY Price DESC) AS RankNum

FROM Products

) AS ranked

WHERE RankNum <= 3;

SELECT \*

FROM (

SELECT ProductID, ProductName, Category, Price,

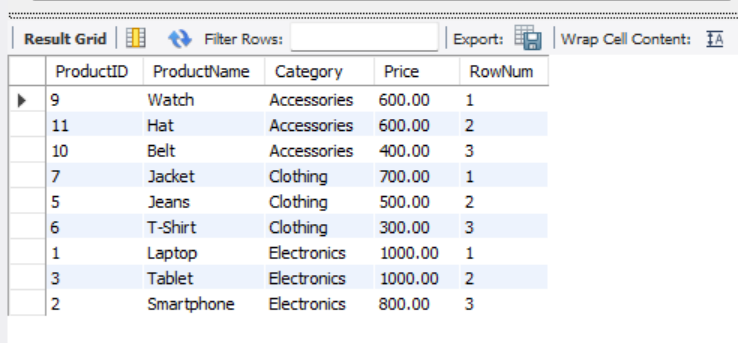
DENSE\_RANK() OVER (PARTITION BY Category ORDER BY Price DESC) AS DenseRankNum

FROM Products

) AS ranked

WHERE DenseRankNum <= 3;

**Output:**



**Exercise 2: Create a Stored Procedure**

**Code:**

CREATE DATABASE db;

USE db;

CREATE TABLE Departments (

DepartmentID INT PRIMARY KEY,

DepartmentName VARCHAR(100)

);

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY AUTO\_INCREMENT,

FirstName VARCHAR(50),

LastName VARCHAR(50),

DepartmentID INT,

Salary DECIMAL(10,2),

JoinDate DATE,

FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)

);

INSERT INTO Departments (DepartmentID, DepartmentName) VALUES

(1, 'HR'),

(2, 'Finance'),

(3, 'IT'),

(4, 'Marketing');

INSERT INTO Employees (FirstName, LastName, DepartmentID, Salary, JoinDate) VALUES

('John', 'Doe', 1, 5000.00, '2020-01-15'),

('Jane', 'Smith', 2, 6000.00, '2019-03-22'),

('Michael', 'Johnson', 3, 7000.00, '2018-07-30'),

('Emily', 'Davis', 4, 5500.00, '2021-11-05');

DELIMITER $$

CREATE PROCEDURE sp\_GetEmployeesByDepartment(IN dept\_id INT)

BEGIN

SELECT

E.EmployeeID,

E.FirstName,

E.LastName,

D.DepartmentName,

E.Salary,

E.JoinDate

FROM

Employees E

JOIN Departments D ON E.DepartmentID = D.DepartmentID

WHERE

E.DepartmentID = dept\_id;

END $$

DELIMITER ;

CALL sp\_GetEmployeesByDepartment(3); -- IT Department

DELIMITER $$

CREATE PROCEDURE sp\_InsertEmployee (

IN fname VARCHAR(50),

IN lname VARCHAR(50),

IN dept\_id INT,

IN sal DECIMAL(10,2),

IN doj DATE

)

BEGIN

INSERT INTO Employees (FirstName, LastName, DepartmentID, Salary, JoinDate)

VALUES (fname, lname, dept\_id, sal, doj);

END $$

DELIMITER ;

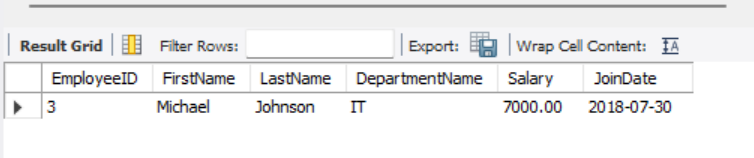
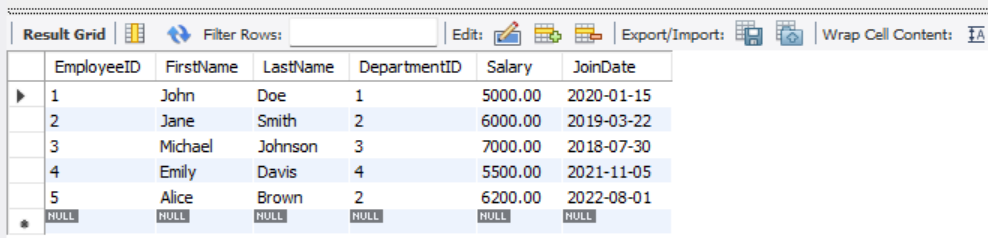
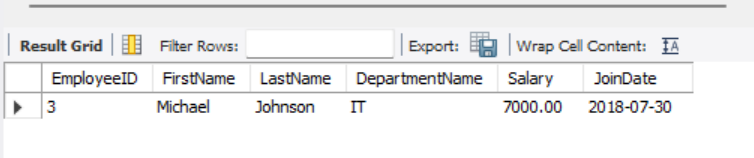
CALL sp\_InsertEmployee('Alice', 'Brown', 2, 6200.00, '2022-08-01');

SELECT \* FROM Employees;

DROP PROCEDURE IF EXISTS sp\_GetEmployeesByDepartment;

DELIMITER $$

**Output:**

****

**Exercise 3: Return Data from a Stored Procedure**

**Code:**

CREATE PROCEDURE sp\_GetEmployeesByDepartment(IN dept\_id INT)

BEGIN

SELECT

E.EmployeeID,

E.FirstName,

E.LastName,

D.DepartmentName,

E.Salary,

E.JoinDate

FROM

Employees E

JOIN Departments D ON E.DepartmentID = D.DepartmentID

WHERE

E.DepartmentID = dept\_id;

END $$

DELIMITER ;

DROP PROCEDURE IF EXISTS sp\_GetEmployeesByDepartment;

CALL sp\_GetEmployeesByDepartment(3); -- Replace 3 with desired DepartmentID

DELIMITER $$

CREATE PROCEDURE sp\_CountEmployeesByDepartment(IN dept\_id INT)

BEGIN

SELECT COUNT(\*) AS TotalEmployees

FROM Employees

WHERE DepartmentID = dept\_id;

END $$

DELIMITER ;

CALL sp\_CountEmployeesByDepartment(2);

DELIMITER $$

CREATE PROCEDURE sp\_TotalSalaryByDepartment(

IN dept\_id INT,

OUT total\_salary DECIMAL(10,2)

)

BEGIN

SELECT SUM(Salary) INTO total\_salary

FROM Employees

WHERE DepartmentID = dept\_id;

END $$

DELIMITER ;

CALL sp\_TotalSalaryByDepartment(2, @total);

SELECT @total;

DELIMITER $$

CREATE PROCEDURE sp\_UpdateEmployeeSalary(

IN emp\_id INT,

IN new\_salary DECIMAL(10,2)

)

BEGIN

UPDATE Employees

SET Salary = new\_salary

WHERE EmployeeID = emp\_id;

END $$

DELIMITER ;

CALL sp\_UpdateEmployeeSalary(1, 5500.00);

DELIMITER $$

CREATE PROCEDURE sp\_GiveBonus(

IN dept\_id INT,

IN bonus DECIMAL(10,2)

)

BEGIN

UPDATE Employees

SET Salary = Salary + bonus

WHERE DepartmentID = dept\_id;

END $$

DELIMITER ;

CALL sp\_GiveBonus(1, 500.00);

DELIMITER $$

CREATE PROCEDURE sp\_TransactionalSalaryUpdate(

IN emp\_id INT,

IN new\_salary DECIMAL(10,2)

)

BEGIN

START TRANSACTION;

UPDATE Employees

SET Salary = new\_salary

WHERE EmployeeID = emp\_id;

COMMIT;

END $$

DELIMITER ;

CALL sp\_FilterEmployees('DepartmentID', '2');

DELIMITER $$

CREATE PROCEDURE sp\_SafeUpdateSalary(

IN emp\_id INT,

IN new\_salary DECIMAL(10,2)

)

BEGIN

DECLARE CONTINUE HANDLER FOR SQLEXCEPTION

BEGIN

SELECT 'Error occurred while updating salary.' AS ErrorMessage;

END;

UPDATE Employees

SET Salary = new\_salary

WHERE EmployeeID = emp\_id;

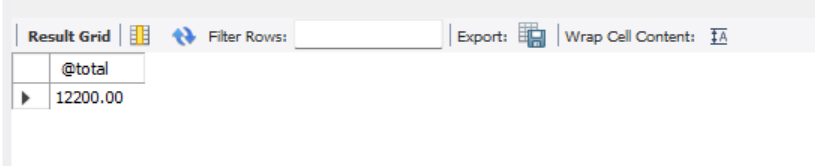
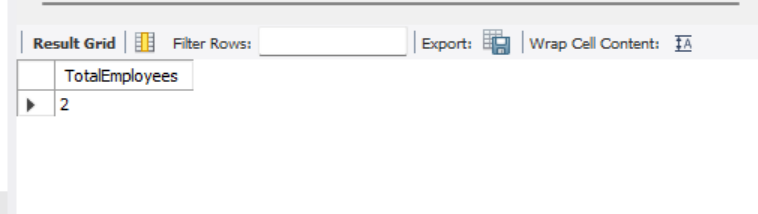
SELECT 'Salary updated successfully.' AS Message;

END $$

DELIMITER ;

CALL sp\_SafeUpdateSalary(1, 5800.00);

**Output:**

****

**NUnit-Handson**

1. What is Unit Testing?

* Unit Testing involves testing the smallest testable part of an application, typically individual functions or methods.
* These tests are isolated from the rest of the system using mocking.

2. Types of Testing

* Unit Testing – Testing individual units/methods.
* Functional Testing – Testing the complete application functionalities.
* Automated Testing – Running tests via tools/scripts without manual intervention.
* Performance Testing – Checking speed, scalability, and stability.

3. Benefits of Automated Testing

* Faster feedback
* Reduces human error
* Encourages cleaner code
* Supports continuous integration/deployment (CI/CD)

4. Loosely Coupled & Testable Design

* Write code not tightly bound to specific implementations.
* Use interfaces or dependency injection.

**Moq-Handson**

#### **1. Understand how Mocking can enhance Test-Driven Development (TDD)**

**Mocking:** A technique used to replace real objects with fake ones that simulate behavior for testing purposes.

**Isolation:** Mocking helps isolate the unit under test by removing dependencies on external systems (like databases, APIs, etc.).

**Test Doubles:** These include:

* **Mocks** – Pre-programmed expectations with verification.
* **Fakes** – Simplified implementations (e.g., in-memory DB).
* **Stubs** – Provide predefined responses.

**Mock vs Fake vs Stub:**

* **Mock** verifies behavior.
* **Stub** provides canned answers.
* **Fake** is a working implementation with no production intent.

**Key Advantages of TDD with Mocking:**

* Ensures code correctness from the beginning.
* Supports frequent regression testing.
* Encourages loosely coupled and testable code.
* Reduces testing time and external dependency failures.

#### **2. Explain the meaning of Mocking in Unit Testing and why use mocks in Unit Testing**

**Mocking in Unit Testing:** Mocking is creating fake implementations of objects that mimic real dependencies (like file systems, mail servers, etc.) so that they can be tested in isolation.

**Why Use Mocks:**

* **To avoid side effects** (e.g., sending real emails, updating a real database).
* **To control behavior** and simulate various scenarios (success, failure).
* **To isolate code** from external systems.
* **To speed up tests** by avoiding slow operations.

**Isolation Using Mocks and Stubs:** Mocks help isolate units by mimicking collaborators, while stubs are used to provide controlled responses to method calls.

#### **3. Understand the basics of DI (Dependency Injection) and how dependency injection helps unit testing in applications**

**Dependency Injection (DI):** A design pattern in which an object’s dependencies are **provided externally** rather than the object creating them itself.

**Types of DI:**

* **Constructor Injection:** Dependencies passed via constructor.
* **Method Injection:** Dependencies passed via method parameters.

**How DI Helps in Unit Testing:**

* Enables mocking dependencies easily.
* Promotes loose coupling and cleaner code.
* Makes classes more testable by abstracting out external components.

#### **4. Demonstrate on how to create a testable code with Moq**

**Testable Code with Moq:**

1. Define an interface for dependencies.
2. Inject that interface into the class via constructor.
3. In test project, use Moq to create a mock object:

This approach allows testing business logic without invoking real infrastructure.

#### **5. Demonstrate on how to create a mock object that accesses database for unit tests**

**Mocking Database Access:**

1. Abstract database logic using an interface.
2. Implement mock using Moq.
3. Use the mock repository in the class under test to simulate DB behavior without hitting a real database.

#### 