**Advanced SQL**

**Exercise1: Exercise 1: Ranking and Window Functions**

**Goal: Use RANK(), DENSE\_RANK(), OVER(), and PARTITION BY.**

**Scenario:**

**Find the top 3 most expensive products in each category using different ranking functions.**

**Steps:**

**1. Use ROW NUMBER() to assign a unique rank within each category.**

**2. Use RANK() and DENSE\_RANK() to compare how ties are handled.**

**3. Use PARTITION BY Category and ORDER BY Price DESC.**

**Codes:  
1. Using ROW\_NUMBER() to assign a unique rank**

**WITH RowNumbered AS (**

**SELECT**

**ProductID,**

**ProductName,**

**Category,**

**Price,**

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

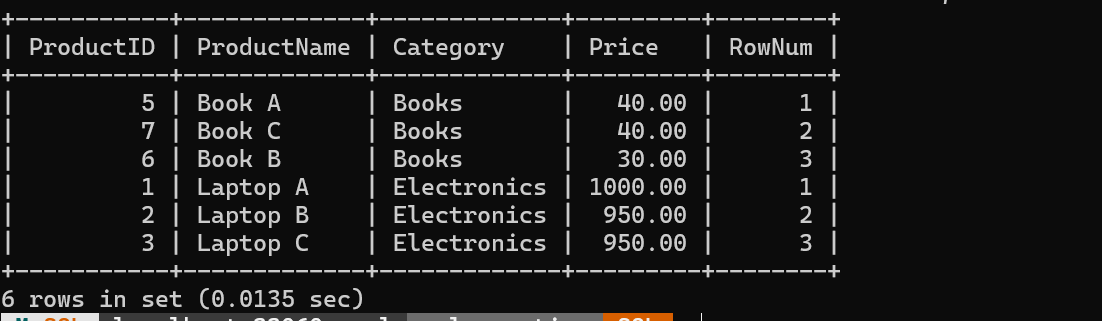
**FROM Products**

**)**

**SELECT \***

**FROM RowNumbered**

**WHERE RowNum <= 3;**

**Output: **

**2.** **Use RANK() to Handle Ties (Skips Numbers)**

**WITH Ranked AS (**

**SELECT**

**ProductID,**

**ProductName,**

**Category,**

**Price,**

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

**FROM Products**

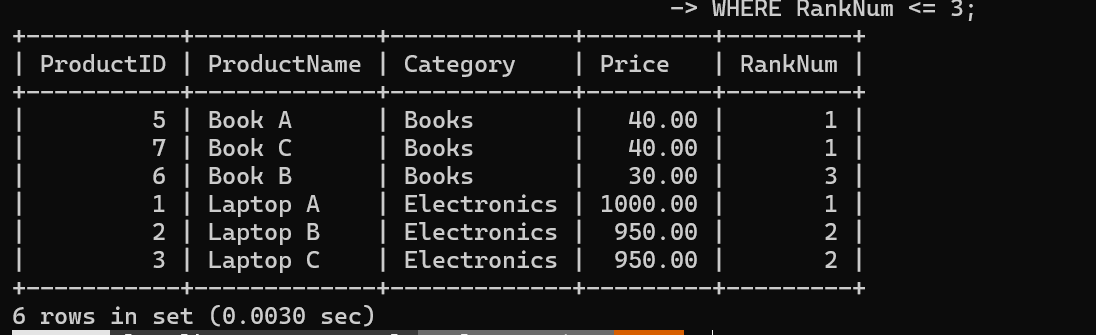
**)**

**SELECT \***

**FROM Ranked**

**WHERE RankNum <= 3;**

**Output:**

****

**3.** **Use DENSE\_RANK() to Handle Ties (No Gaps)**

**WITH DenseRanked AS (**

**SELECT**

**ProductID,**

**ProductName,**

**Category,**

**Price,**

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

**FROM Products**

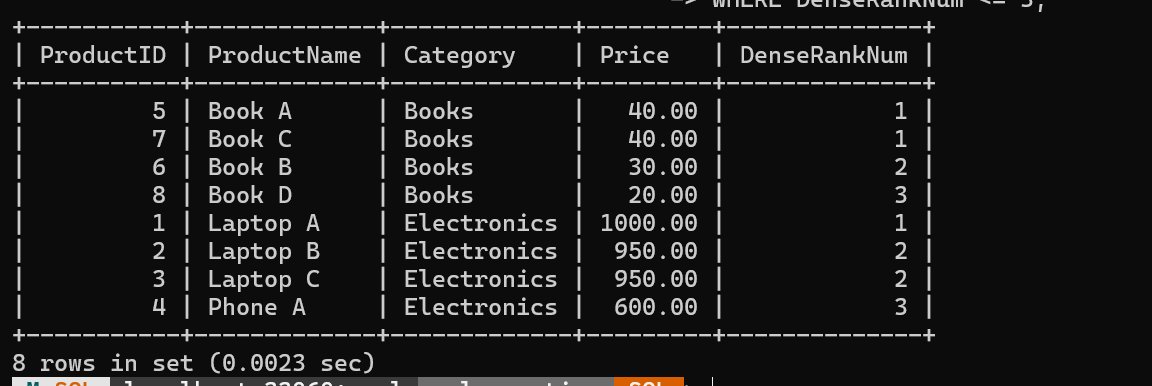
**)**

**SELECT \***

**FROM DenseRanked**

**WHERE DenseRankNum <= 3;**

**Output:**

****

**Exercise 2: SQL Exercises – Index**

**-- Database Schema**

**CREATE TABLE Customers (**

**CustomerID INT PRIMARY KEY,**

**Name VARCHAR(100),**

**Region VARCHAR(50)**

**);**

**CREATE TABLE Products (**

**ProductID INT PRIMARY KEY,**

**ProductName VARCHAR(100),**

**Category VARCHAR(50),**

**Price DECIMAL(10, 2)**

**);**

**CREATE TABLE Orders (**

**OrderID INT PRIMARY KEY,**

**CustomerID INT,**

**OrderDate DATE,**

**FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)**

**);**

**CREATE TABLE OrderDetails (**

**OrderDetailID INT PRIMARY KEY,**

**OrderID INT,**

**ProductID INT,**

**Quantity INT,**

**FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),**

**FOREIGN KEY (ProductID) REFERENCES Products(ProductID)**

**);**

**-- Sample Data**

**INSERT INTO Customers (CustomerID, Name, Region) VALUES**

**(1, 'Alice', 'North'),**

**(2, 'Bob', 'South'),**

**(3, 'Charlie', 'East'),**

**(4, 'David', 'West');**

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

**(1, 'Laptop', 'Electronics', 1200.00),**

**(2, 'Smartphone', 'Electronics', 800.00),**

**(3, 'Tablet', 'Electronics', 600.00),**

**(4, 'Headphones', 'Accessories', 150.00);**

**INSERT INTO Orders (OrderID, CustomerID, OrderDate) VALUES**

**(1, 1, '2023-01-15'),**

**(2, 2, '2023-02-20'),**

**(3, 3, '2023-03-25'),**

**(4, 4, '2023-04-30');**

**INSERT INTO OrderDetails (OrderDetailID, OrderID, ProductID, Quantity) VALUES**

**(1, 1, 1, 1),**

**(2, 2, 2, 2),**

**(3, 3, 3, 1),**

**(4, 4, 4, 3);**

**-- Exercise 1: Creating a Non-Clustered Index**

**-- Goal: Create a non-clustered index on the ProductName column in the Products table and compare query execution time before and after index creation.**

**-- Step 1: Query to fetch product details before index creation**

**SELECT \* FROM Products WHERE ProductName = 'Laptop';**

**-- Step 2: Create a non-clustered index on ProductName**

**CREATE INDEX idx\_product\_name ON Products(ProductName);**

**-- Step 3: Query to fetch product details after index creation**

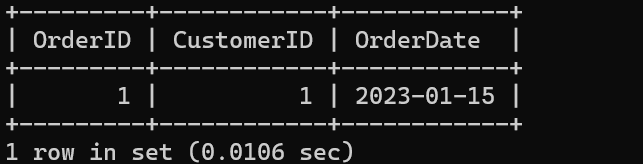
**SELECT \* FROM Products WHERE ProductName = 'Laptop';**

**-- Exercise 2: Creating a Clustered Index**

**-- Goal: Create a clustered index on the OrderDate column in the Orders table and compare query execution time before and after index creation.**

**-- Step 1: Query to fetch orders before index creation**

**SELECT \* FROM Orders WHERE OrderDate = '2023-01-15';**

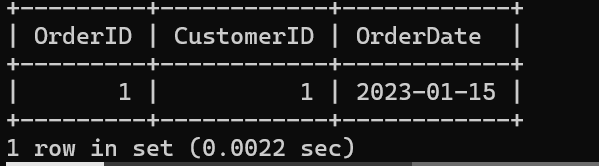
****

**-- Step 2: Create a clustered index on OrderDate (MySQL doesn't support multiple clustered indexes, so use normal index)**

**CREATE INDEX idx\_order\_date ON Orders(OrderDate);**

**-- Step 3: Query to fetch orders after index creation**

**SELECT \* FROM Orders WHERE OrderDate = '2023-01-15';**

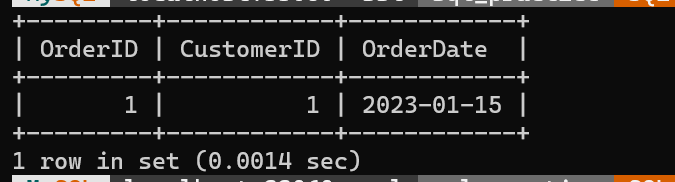
****

**-- Exercise 3: Creating a Composite Index**

**-- Goal: Create a composite index on the CustomerID and OrderDate columns in the Orders table and compare query execution time before and after index creation.**

**-- Step 1: Query to fetch orders before index creation**

**SELECT \* FROM Orders WHERE CustomerID = 1 AND OrderDate = '2023-01-15';**

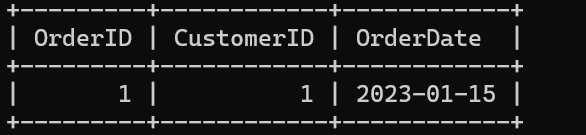
****

**-- Step 2: Create a composite index on CustomerID and OrderDate**

**CREATE INDEX idx\_customer\_orderdate ON Orders(CustomerID, OrderDate);**

**-- Step 3: Query to fetch orders after index creation**

**SELECT \* FROM Orders WHERE CustomerID = 1 AND OrderDate = '2023-01-15';**

****

**Employee Management System System SQL**

**Exercise 1: Create a Stored Procedure  
DELIMITER //**

**CREATE PROCEDURE GetEmployeesByDepartment(IN dept\_id INT)**

**BEGIN**

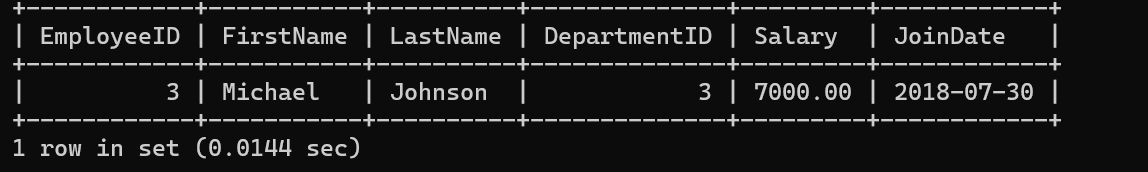
**SELECT \* FROM Employees**

**WHERE DepartmentID = dept\_id;**

**END //**

**DELIMITER ;**

**CALL GetEmployeesByDepartment(3);**

****

**Exercise 4: Execute a Stored Procedure  
USE sql\_practice;**

**DELIMITER //**

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

**BEGIN**

**SELECT**

**EmployeeID,**

**FirstName,**

**LastName,**

**DepartmentID,**

**Salary,**

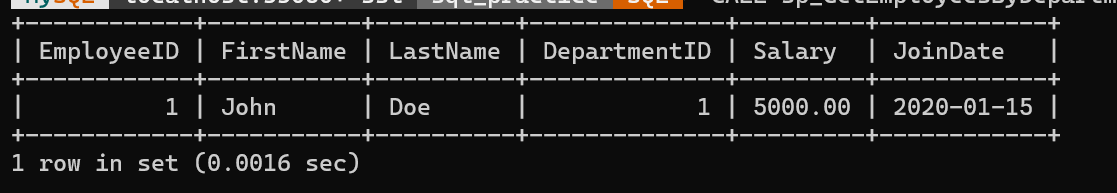
**JoinDate**

**FROM Employees**

**WHERE DepartmentID = dept\_id;**

**END //**

**DELIMITER ;**

**CALL sp\_GetEmployeesByDepartment(1);  
**

**Exercise 5: Return Data from a Stored Procedure**

**Goal: Create a stored procedure that returns the total number of employees in a**

**department.**

**USE sql\_practice;**

**DELIMITER //**

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

**BEGIN**

**SELECT COUNT(\*) AS TotalEmployees**

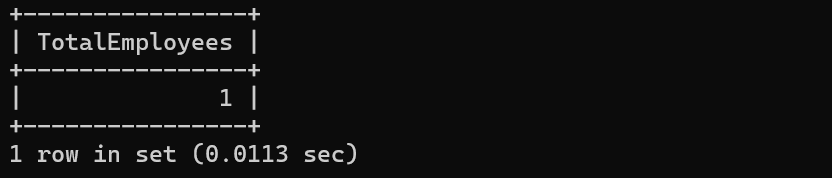
**FROM Employees**

**WHERE DepartmentID = dept\_id;**

**END //**

**DELIMITER ;**

**CALL sp\_CountEmployeesByDepartment(3);**

****

**Exercise 7: Return Data from a Scalar Function**

**Goal: Return the annual salary for a specific employee using Sfn\_CalculateAnnualSalaryS.**

**DELIMITER //**

**CREATE FUNCTION fn\_CalculateAnnualSalary(emp\_id INT)**

**RETURNS DECIMAL(10,2)**

**DETERMINISTIC**

**READS SQL DATA**

**BEGIN**

**DECLARE annual\_salary DECIMAL(10,2);**

**SELECT Salary \* 12 INTO annual\_salary**

**FROM Employees**

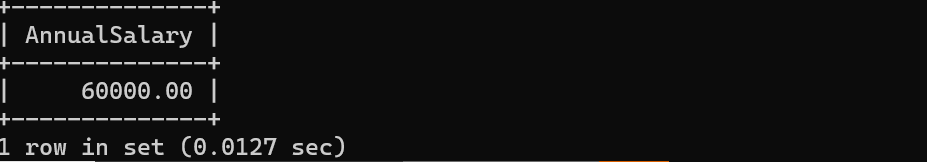
**WHERE EmployeeID = emp\_id;**

**RETURN annual\_salary;**

**END //**

**DELIMITER ;**

**SELECT fn\_CalculateAnnualSalary(1) AS AnnualSalary;**

****