**SQL Exercise: Advance concept**

**Exercise 1: Ranking and Window Functions**

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

So for doing the exercise-1 I have used my own datasets:

| **ProductName** | **Category** | **Price** | **DenseRankNum** |
| --- | --- | --- | --- |
| A1 | Electronics | 1000 | 1 |
| A2 | Electronics | 1000 | 1 |
| A3 | Electronics | 800 | 2 |
| B1 | Fashion | 500 | 1 |
| B2 | Fashion | 500 | 1 |
| B3 | Fashion | 400 | 2 |
| C1 | Toys | 300 | 1 |
| C2 | Toys | 250 | 2 |
| C3 | Toys | 250 | 2 |

1st-Question-Solution:

SELECT \*

FROM (

SELECT \*,

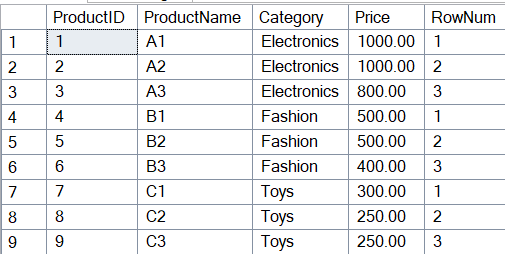
ROW\_NUMBER() OVER (PARTITION BY Category ORDER BY Price DESC) AS RowNum

FROM Products

) AS Ranked

WHERE RowNum <= 3;

Output:



2nd-Question-Solution:

SELECT \*

FROM (

SELECT \*,

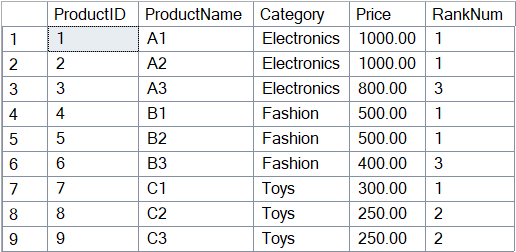
RANK() OVER (PARTITION BY Category ORDER BY Price DESC) AS RankNum

FROM Products

) AS Ranked

WHERE RankNum <= 3;

Output:



3rd-Question-Solution:

SELECT \*

FROM (

SELECT \*,

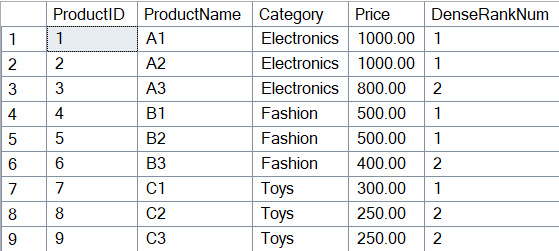
DENSE\_RANK() OVER (PARTITION BY Category ORDER BY Price DESC) AS DenseRankNum

FROM Products

) AS Ranked

WHERE DenseRankNum <= 3;

Output:



**SQL Exercise - Stored procedure**

**Exercise 1: Create a Stored Procedure**

**Goal**: Create a stored procedure to retrieve employee details by department

SQL Query:

GO

CREATE PROCEDURE sp\_InsertEmployee

@FirstName VARCHAR(50),

@LastName VARCHAR(50),

@DepartmentID INT,

@Salary DECIMAL(10,2),

@JoinDate DATE

AS

BEGIN

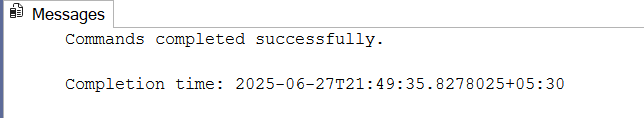
INSERT INTO Employees (FirstName, LastName, DepartmentID, Salary, JoinDate)

VALUES (@FirstName, @LastName, @DepartmentID, @Salary, @JoinDate);

END;

GO

Output:



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

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

SQL Query:

GO

CREATE PROCEDURE sp\_GetEmployeeCountByDept

@DepartmentID INT

AS

BEGIN

SELECT COUNT(\*) AS TotalEmployees

FROM Employees

WHERE DepartmentID = @DepartmentID;

END;

GO

TO Execute it: EXEC sp\_GetEmployeeCountByDept @DepartmentID = 3;

Output:



**Additional important hands-on:**

**SQL Exercise – Index**

Hands-on in this document

SQL Query:

-- 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 with and without the index.

-- Without index

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

-- Create the index

CREATE NONCLUSTERED INDEX IX\_Products\_ProductName

ON Products (ProductName);

-- With index

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

-- Drop index

-- DROP INDEX IX\_Products\_ProductName ON Products;

-- Exercise 2: Composite Index

-- Goal: Create a composite index on (Category, Price) in the Products table to speed up category-price based queries.

-- Without index

SELECT \* FROM Products WHERE Category = 'Electronics' AND Price < 1000;

-- Create composite index

CREATE NONCLUSTERED INDEX IX\_Products\_Category\_Price

ON Products (Category, Price);

-- With index

SELECT \* FROM Products WHERE Category = 'Electronics' AND Price < 1000;

-- Exercise 3: Index on Foreign Keys

-- Index on Orders.CustomerID

CREATE NONCLUSTERED INDEX IX\_Orders\_CustomerID

ON Orders (CustomerID);

-- Index on OrderDetails.OrderID and OrderDetails.ProductID

CREATE NONCLUSTERED INDEX IX\_OrderDetails\_OrderID

ON OrderDetails (OrderID);

CREATE NONCLUSTERED INDEX IX\_OrderDetails\_ProductID

ON OrderDetails (ProductID);

-- Example join query

SELECT o.OrderID, c.Name, p.ProductName

FROM Orders o

JOIN Customers c ON o.CustomerID = c.CustomerID

JOIN OrderDetails od ON o.OrderID = od.OrderID

JOIN Products p ON od.ProductID = p.ProductID

WHERE c.Region = 'North';

-- Exercise 4: Dropping Indexes

-- DROP INDEX IX\_Products\_Category\_Price ON Products;

-- DROP INDEX IX\_Orders\_CustomerID ON Orders;

-- DROP INDEX IX\_OrderDetails\_OrderID ON OrderDetails;

-- DROP INDEX IX\_OrderDetails\_ProductID ON OrderDetails;

-- Exercise 5: Best Practices Comments

-- - Avoid over-indexing: too many indexes slow down INSERT/UPDATE/DELETE.

-- - Use indexes on columns frequently used in WHERE, JOIN, ORDER BY.

-- - Composite indexes should match query column order for efficiency.

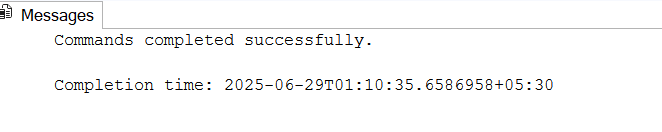
-- - Monitor performance using execution plans and statistics.

-- To check existing indexes (SQL Server)

-- SELECT \* FROM sys.indexes WHERE object\_id = OBJECT\_ID('Products');

Output:

I have filled the blanked gaps and the output is:



**SQL Exercise – Functions**

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

**Goal:** Return the annual salary for a specific employee using ‘fn\_CalculateAnnualSalary’.

SQL Query:

GO

CREATE FUNCTION fn\_CalculateAnnualSalary

(

@MonthlySalary DECIMAL(10,2)

)

RETURNS DECIMAL(10,2)

AS

BEGIN

RETURN @MonthlySalary \* 12;

END;

GO

SELECT

EmployeeID,

FirstName,

LastName,

Salary AS MonthlySalary,

dbo.fn\_CalculateAnnualSalary(Salary) AS AnnualSalary

FROM Employees

WHERE EmployeeID = 1;

Output:



**SQL Exercise - Stored procedure**

**Exercise 4: Execute a Stored Procedure**

**Goal**: Execute the stored procedure to retrieve employee details for a specific department

SQL Query:

EXEC sp\_InsertEmployee

@FirstName = 'Alice',

@LastName = 'Williams',

@DepartmentID = 3,

@Salary = 7500.00,

@JoinDate = '2023-05-20';

SELECT \* FROM Employees;

Output:

