**1. SQL Exercise - Advanced concepts**

**Exercise 1: Ranking and Window Functions**

|  |
| --- |
| WITH RankedProducts AS (  SELECT  ProductID,  ProductName,  Category,  Price,  ROW\_NUMBER() OVER (  PARTITION BY Category  ORDER BY Price DESC  ) AS RowNum,  RANK() OVER (  PARTITION BY Category  ORDER BY Price DESC  ) AS RankNum,  DENSE\_RANK() OVER (  PARTITION BY Category  ORDER BY Price DESC  ) AS DenseRankNum  FROM Products  )  SELECT \*  FROM RankedProducts  WHERE RowNum <= 3 OR RankNum <= 3 OR DenseRankNum <= 3 |

A table with numbers and text

AI-generated content may be incorrect.

**Exercise 2: Aggregation with GROUPING SETS, CUBE, and ROLLUP**

**GROUPING SETS**

|  |
| --- |
| SELECT  P.Category,  C.Region,  SUM(OD.QUANTITY \* P.PRICE) AS TOTALSALES  FROM Products AS P  JOIN OrderDETAILS AS OD ON P.ProductID = OD.ProductID  JOIN ORDERS AS O ON OD.OrderID = O.OrderID  JOIN Customers AS C ON O.CustomerID = C.CustomerID  GROUP BY GROUPING SETS(  (CATEGORY),  (REGION),  ()  )  ORDER BY  GROUPING(P.CATEGORY),  GROUPING(C.REGION); |

A screenshot of a computer

AI-generated content may be incorrect.

**ROLLUP**

|  |
| --- |
| SELECT  P.Category,  C.Region,  SUM(OD.QUANTITY \* P.PRICE) AS TOTALSALES  FROM Products AS P  JOIN OrderDETAILS AS OD ON P.ProductID = OD.ProductID  JOIN ORDERS AS O ON OD.OrderID = O.OrderID  JOIN Customers AS C ON O.CustomerID = C.CustomerID  GROUP BY ROLLUP((CATEGORY),(REGION))  ORDER BY  GROUPING(P.CATEGORY),  GROUPING(C.REGION); |

A screenshot of a computer

AI-generated content may be incorrect.

**CUBE**

|  |
| --- |
| SELECT  P.Category,  C.Region,  SUM(OD.QUANTITY \* P.PRICE) AS TOTALSALES  FROM Products AS P  JOIN OrderDETAILS AS OD ON P.ProductID = OD.ProductID  JOIN ORDERS AS O ON OD.OrderID = O.OrderID  JOIN Customers AS C ON O.CustomerID = C.CustomerID  GROUP BY CUBE((CATEGORY),(REGION))  ORDER BY  GROUPING(P.CATEGORY),  GROUPING(C.REGION); |

A screenshot of a computer screen

AI-generated content may be incorrect.

**4. SQL Exercise - Stored procedure**

**Exercise 1: Create a Stored Procedure**

|  |
| --- |
| CREATE PROCEDURE sp\_GetEmployeesByDepartment  @DeptID INT  AS  BEGIN  SELECT  E.EmployeeID,  E.FirstName,  E.LastName,  D.DepartmentName,  FROM Employees AS E  INNER JOIN Departments AS D  ON E.DepartmentID = D.DepartmentID  WHERE E.DepartmentID = @DeptID;  END; |

A screenshot of a computer

AI-generated content may be incorrect.

**Exercise 2: Modify a Stored Procedure**

|  |
| --- |
| ALTER PROCEDURE sp\_GetEmployeesByDepartment  @DeptID INT  AS  BEGIN  SELECT  E.EmployeeID,  E.FirstName,  E.LastName,  E.SALARY,  D.DepartmentName  FROM Employees AS E  INNER JOIN Departments AS D  ON E.DepartmentID = D.DepartmentID  WHERE E.DepartmentID = @DeptID;  END; |

A screenshot of a computer

AI-generated content may be incorrect.

**Exercise 3: Delete a Stored Procedure**

|  |
| --- |
| DROP PROCEDURE sp\_InsertEmployee; |

A screenshot of a computer

AI-generated content may be incorrect.

**Exercise 4: Execute a Stored Procedure**

|  |
| --- |
| EXEC sp\_GetEmployeesByDepartment @DepartmentID = 2; |

A screenshot of a computer

AI-generated content may be incorrect.

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

|  |
| --- |
| CREATE PROCEDURE sp\_CountEmployeesByDepartment  @DepartmentID INT  AS  BEGIN  SELECT  COUNT(\*) AS TotalEmployees  FROM Employees  WHERE DepartmentID = @DepartmentID;  END;  EXEC sp\_CountEmployeesByDepartment @DepartmentID = 3; |

A screenshot of a computer

AI-generated content may be incorrect.