# **Advanced SQL Server**

**1. SQL Exercise - Advanced concepts**

**Exercise 1: Ranking and Window Functions**

SELECT

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 Rank,

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

FROM Products;

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

-- Assume Orders o, OrderDetails od, Customers c, Products p

SELECT

c.Region,

p.Category,

SUM(od.Quantity) AS TotalQuantity

FROM Orders o

JOIN OrderDetails od ON o.OrderID = od.OrderID

JOIN Customers c ON o.CustomerID = c.CustomerID

JOIN Products p ON od.ProductID = p.ProductID

GROUP BY GROUPING SETS (

(c.Region, p.Category),

(c.Region),

(p.Category),

()

);

-- ROLLUP

GROUP BY ROLLUP (s.Region, p.Category);

-- CUBE

GROUP BY CUBE (s.Region, p.Category);

**Exercise 3: CTEs and MERGE**

#### **a) Recursive CTE for January 2025**

WITH Calendar AS (

SELECT CAST('2025-01-01' AS DATE) AS CalendarDate

UNION ALL

SELECT DATEADD(DAY, 1, CalendarDate)

FROM Calendar

WHERE CalendarDate < '2025-01-31'

)

SELECT \* FROM Calendar;

**b) MERGE for Product Price Update**

MERGE INTO Products AS target

USING StagingProducts AS source

ON target.ProductID = source.ProductID

WHEN MATCHED THEN

UPDATE SET

target.Price = source.Price,

target.LastUpdated = GETDATE()

WHEN NOT MATCHED THEN

INSERT (ProductID, ProductName, Category, Price, LastUpdated)

VALUES (source.ProductID, source.ProductName, source.Category, source.Price, GETDATE());

**Exercise 4: PIVOT and UNPIVOT**

-- PIVOT

SELECT \*

FROM (

SELECT ProductID, MONTH(SaleDate) AS SaleMonth, SalesAmount

FROM Sales

) AS SourceTable

PIVOT (

SUM(SalesAmount)

FOR SaleMonth IN (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)

) AS PivotTable;

-- UNPIVOT

SELECT ProductID, SaleMonth, SalesAmount

FROM PivotTable

UNPIVOT (

SalesAmount

FOR SaleMonth IN (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)

) AS UnpivotTable;

**Exercise 5: CTE to Simplify Customer Order Query**

WITH OrderCount AS (

SELECT CustomerID, COUNT(\*) AS OrderCount

FROM Orders

GROUP BY CustomerID

)

SELECT CustomerID, OrderCount

FROM OrderCount

WHERE OrderCount > 3;

## **2. SQL Exercise - Indexing**

-- Exercise 1: Non-Clustered Index on ProductName

-- Step 1: Query before index creation

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

-- Step 2: Create non-clustered index

CREATE NONCLUSTERED INDEX idx\_ProductName ON Products(ProductName);

-- Step 3: Query after index creation

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

-- Exercise 2: Clustered Index on OrderDate

-- DROP INDEX IF EXISTS idx\_OrderID ON Orders;

-- Step 1: Query before index creation

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

-- Step 2: Create clustered index on OrderDate

CREATE CLUSTERED INDEX idx\_OrderDate ON Orders(OrderDate);

-- Step 3: Query after index creation

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

-- Exercise 3: Composite Index on CustomerID and OrderDate

-- Step 1: Query before index creation

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

-- Step 2: Create composite (multi-column) index

CREATE NONCLUSTERED INDEX idx\_Customer\_OrderDate ON Orders(CustomerID, OrderDate);

-- Step 3: Query after index creation

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

## **3. Views**

-- Employee Management System: SQL Exercises

-- View 1: Basic Employee Info

CREATE VIEW vw\_EmployeeBasicInfo AS

SELECT

e.EmployeeID,

e.FirstName,

e.LastName,

d.DepartmentName

FROM Employees e

JOIN Departments d ON e.DepartmentID = d.DepartmentID;

-- View 2: Employee Full Name

CREATE VIEW vw\_EmployeeFullName AS

SELECT

EmployeeID,

FirstName + ' ' + LastName AS FullName

FROM Employees;

-- View 3: Employee Annual Salary

CREATE VIEW vw\_EmployeeAnnualSalary AS

SELECT

EmployeeID,

FirstName,

LastName,

Salary \* 12 AS AnnualSalary

FROM Employees;

-- View 4: Employee Report with Bonus

CREATE VIEW vw\_EmployeeReport AS

SELECT

e.EmployeeID,

e.FirstName + ' ' + e.LastName AS FullName,

d.DepartmentName,

e.Salary \* 12 AS AnnualSalary,

(e.Salary \* 12) \* 0.10 AS Bonus

FROM Employees e

JOIN Departments d ON e.DepartmentID = d.DepartmentID;

## **4. Stored Procedures**

-- Exercise 1: Create a Stored Procedure to Insert Employee

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;

-- Exercise 2: Modify a Stored Procedure to Include Salary

ALTER PROCEDURE sp\_GetEmployeesByDepartment

@DeptID INT

AS

BEGIN

SELECT EmployeeID, FirstName, LastName, Salary

FROM Employees

WHERE DepartmentID = @DeptID;

END;

-- Exercise 3: Delete a Stored Procedure

DROP PROCEDURE IF EXISTS sp\_InsertEmployee;

-- Exercise 4: Execute a Stored Procedure (for DepartmentID = 1)

EXEC sp\_GetEmployeesByDepartment 1;

-- Exercise 5: Return Employee Count by Department

CREATE PROCEDURE sp\_CountEmployeesByDept

@DeptID INT

AS

BEGIN

SELECT COUNT(\*) AS TotalEmployees

FROM Employees

WHERE DepartmentID = @DeptID;

END;

-- Exercise 6: Output Parameter for Total Salary

CREATE PROCEDURE sp\_TotalSalaryByDept

@DeptID INT,

@TotalSalary DECIMAL(10,2) OUTPUT

AS

BEGIN

SELECT @TotalSalary = SUM(Salary)

FROM Employees

WHERE DepartmentID = @DeptID;

END;

-- Exercise 7: Update Salary Using Multiple Parameters

CREATE PROCEDURE sp\_UpdateEmployeeSalary

@EmpID INT,

@NewSalary DECIMAL(10,2)

AS

BEGIN

UPDATE Employees

SET Salary = @NewSalary

WHERE EmployeeID = @EmpID;

END;

-- Execute Salary Update

EXEC sp\_UpdateEmployeeSalary 1, 5500.00;

-- Exercise 8: Conditional Bonus by Department

CREATE PROCEDURE sp\_GiveBonus

@DeptID INT,

@BonusAmount DECIMAL(10,2)

AS

BEGIN

UPDATE Employees

SET Salary = Salary + @BonusAmount

WHERE DepartmentID = @DeptID;

END;

-- Execute Bonus Procedure

EXEC sp\_GiveBonus 1, 500.00;

-- Exercise 9: Use Transactions in Salary Update

CREATE PROCEDURE sp\_TransactionalSalaryUpdate

@EmpID INT,

@NewSalary DECIMAL(10,2)

AS

BEGIN

BEGIN TRANSACTION;

BEGIN TRY

UPDATE Employees

SET Salary = @NewSalary

WHERE EmployeeID = @EmpID;

COMMIT;

END TRY

BEGIN CATCH

ROLLBACK;

PRINT 'Error: ' + ERROR\_MESSAGE();

END CATCH;

END;

-- Exercise 10: Dynamic SQL Filter

CREATE PROCEDURE sp\_FilterEmployees

@ColumnName NVARCHAR(50),

@Value NVARCHAR(100)

AS

BEGIN

DECLARE @SQL NVARCHAR(MAX);

SET @SQL = 'SELECT \* FROM Employees WHERE ' + QUOTENAME(@ColumnName) + ' = @ValueParam';

EXEC sp\_executesql @SQL, N'@ValueParam NVARCHAR(100)', @Value;

END;

-- Exercise 11: Error Handling in Stored Procedure

CREATE PROCEDURE sp\_SafeSalaryUpdate

@EmpID INT,

@NewSalary DECIMAL(10,2)

AS

BEGIN

BEGIN TRY

IF @NewSalary < 0

THROW 50000, 'Salary must be non-negative.', 1;

UPDATE Employees

SET Salary = @NewSalary

WHERE EmployeeID = @EmpID;

END TRY

BEGIN CATCH

PRINT 'Custom Error: ' + ERROR\_MESSAGE();

END CATCH;

END;

## **5. Functions**

-- Exercise 1: Scalar Function – Calculate Annual Salary

CREATE FUNCTION fn\_CalculateAnnualSalary (@Salary DECIMAL(10,2))

RETURNS DECIMAL(10,2)

AS

BEGIN

RETURN @Salary \* 12;

END;

SELECT

EmployeeID,

FirstName,

LastName,

Salary,

dbo.fn\_CalculateAnnualSalary(Salary) AS AnnualSalary

FROM Employees;

-- Exercise 2: Table-Valued Function – Employees by Department

CREATE FUNCTION fn\_GetEmployeesByDepartment (@DeptID INT)

RETURNS TABLE

AS

RETURN (

SELECT \*

FROM Employees

WHERE DepartmentID = @DeptID

);

SELECT \* FROM dbo.fn\_GetEmployeesByDepartment(2);

-- Exercise 3: Scalar Function – Calculate Bonus

CREATE FUNCTION fn\_CalculateBonus (@Salary DECIMAL(10,2))

RETURNS DECIMAL(10,2)

AS

BEGIN

RETURN @Salary \* 0.10;

END;

SELECT

EmployeeID,

FirstName,

Salary,

dbo.fn\_CalculateBonus(Salary) AS Bonus

FROM Employees;

-- Exercise 4: Modify Bonus Function to 15%

ALTER FUNCTION fn\_CalculateBonus (@Salary DECIMAL(10,2))

RETURNS DECIMAL(10,2)

AS

BEGIN

RETURN @Salary \* 0.15;

END;

SELECT

EmployeeID,

FirstName,

Salary,

dbo.fn\_CalculateBonus(Salary) AS Bonus

FROM Employees;

-- Exercise 5: Drop Bonus Function

DROP FUNCTION IF EXISTS fn\_CalculateBonus;

-- Exercise 6: Execute fn\_CalculateAnnualSalary

SELECT

EmployeeID,

FirstName,

Salary,

dbo.fn\_CalculateAnnualSalary(Salary) AS AnnualSalary

FROM Employees;

-- Exercise 7: Annual Salary of EmployeeID = 1

SELECT

EmployeeID,

FirstName,

Salary,

dbo.fn\_CalculateAnnualSalary(Salary) AS AnnualSalary

FROM Employees

WHERE EmployeeID = 1;

-- Exercise 8: Table-Valued Function for Finance Department

SELECT \* FROM dbo.fn\_GetEmployeesByDepartment(3);

-- Exercise 9: Nested Function – Total Compensation

CREATE FUNCTION fn\_CalculateBonus (@Salary DECIMAL(10,2))

RETURNS DECIMAL(10,2)

AS

BEGIN

RETURN @Salary \* 0.15;

END;

CREATE FUNCTION fn\_CalculateTotalCompensation (@Salary DECIMAL(10,2))

RETURNS DECIMAL(10,2)

AS

BEGIN

RETURN dbo.fn\_CalculateAnnualSalary(@Salary) + dbo.fn\_CalculateBonus(@Salary);

END;

SELECT

EmployeeID,

FirstName,

Salary,

dbo.fn\_CalculateTotalCompensation(Salary) AS TotalCompensation

FROM Employees;

-- Exercise 10: Modify Total Compensation Function

ALTER FUNCTION fn\_CalculateTotalCompensation (@Salary DECIMAL(10,2))

RETURNS DECIMAL(10,2)

AS

BEGIN

RETURN dbo.fn\_CalculateAnnualSalary(@Salary) + dbo.fn\_CalculateBonus(@Salary);

END;

SELECT

EmployeeID,

FirstName,

Salary,

dbo.fn\_CalculateTotalCompensation(Salary) AS TotalCompensation

FROM Employees;

## **6. Triggers**

-- Sample Data Insertion

INSERT INTO Departments (DepartmentID, DepartmentName) VALUES

(1, 'HR'),

(2, 'Finance'),

(3, 'IT'),

(4, 'Marketing');

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

(1, 'Amit', 'Verma', 1, 5000.00, '2022-01-15'),

(2, 'Priya', 'Sharma', 2, 6000.00, '2021-03-22'),

(3, 'Rahul', 'Patel', 3, 7000.00, '2020-07-30'),

(4, 'Sneha', 'Reddy', 4, 5500.00, '2019-11-05');

-- Exercise 1: AFTER Trigger - Log Salary Changes

CREATE TABLE EmployeeChanges (

ChangeID INT IDENTITY(1,1) PRIMARY KEY,

EmployeeID INT,

OldSalary DECIMAL(10,2),

NewSalary DECIMAL(10,2),

ChangeDate DATETIME DEFAULT GETDATE()

);

CREATE TRIGGER trg\_LogSalaryChange

ON Employees

AFTER UPDATE

AS

BEGIN

INSERT INTO EmployeeChanges (EmployeeID, OldSalary, NewSalary)

SELECT

i.EmployeeID,

d.Salary AS OldSalary,

i.Salary AS NewSalary

FROM inserted i

JOIN deleted d ON i.EmployeeID = d.EmployeeID;

END;

-- Exercise 2: INSTEAD OF DELETE Trigger

CREATE TRIGGER trg\_NoDelete

ON Employees

INSTEAD OF DELETE

AS

BEGIN

RAISERROR('Deletion not allowed for Employees table.', 16, 1);

END;

-- Exercise 3: SERVER-LEVEL TRIGGER (LOGON BLOCK)

-- (This requires elevated permissions)

CREATE TRIGGER trg\_BlockLogon

ON ALL SERVER

FOR LOGON

AS

BEGIN

IF (DATEPART(HOUR, GETDATE()) BETWEEN 2 AND 3)

BEGIN

RAISERROR('Maintenance Window: Logins are blocked between 2 AM and 3 AM.', 16, 1);

END

END;

-- Exercise 4: Modify Trigger (done through SSMS typically, so no SQL needed here)

-- Exercise 5: DROP Trigger

DROP TRIGGER trg\_LogSalaryChange;

-- Exercise 6: Computed Column and Trigger for AnnualSalary

ALTER TABLE Employees

ADD AnnualSalary AS Salary \* 12;

CREATE TRIGGER trg\_UpdateAnnualSalary

ON Employees

AFTER UPDATE

AS

BEGIN

UPDATE Employees

SET AnnualSalary = i.Salary \* 12

FROM inserted i

WHERE Employees.EmployeeID = i.EmployeeID;

END;

## **7. Cursors**

-- Sample Data (Indian Names)

INSERT INTO Departments (DepartmentID, DepartmentName) VALUES

(1, 'HR'),

(2, 'IT'),

(3, 'Finance');

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

(1, 'Ravi', 'Kumar', 1, 5000.00, '2020-01-15'),

(2, 'Anjali', 'Mehta', 2, 6000.00, '2019-03-22'),

(3, 'Vikram', 'Shah', 3, 5500.00, '2021-07-30');

-- Exercise 1: Create a Cursor to Print Employee Details

DECLARE @EmpID INT, @FirstName VARCHAR(50), @LastName VARCHAR(50), @DeptID INT, @Salary DECIMAL(10,2), @JoinDate DATE;

DECLARE emp\_cursor CURSOR FOR

SELECT EmployeeID, FirstName, LastName, DepartmentID, Salary, JoinDate FROM Employees;

OPEN emp\_cursor;

FETCH NEXT FROM emp\_cursor INTO @EmpID, @FirstName, @LastName, @DeptID, @Salary, @JoinDate;

WHILE @@FETCH\_STATUS = 0

BEGIN

PRINT 'ID: ' + CAST(@EmpID AS VARCHAR) + ', Name: ' + @FirstName + ' ' + @LastName +

', DeptID: ' + CAST(@DeptID AS VARCHAR) +

', Salary: ' + CAST(@Salary AS VARCHAR) +

', JoinDate: ' + CAST(@JoinDate AS VARCHAR);

FETCH NEXT FROM emp\_cursor INTO @EmpID, @FirstName, @LastName, @DeptID, @Salary, @JoinDate;

END

CLOSE emp\_cursor;

DEALLOCATE emp\_cursor;

-- Exercise 2: Cursor Types

-- Static Cursor

DECLARE static\_cursor CURSOR STATIC FOR

SELECT EmployeeID, FirstName FROM Employees;

OPEN static\_cursor;

FETCH NEXT FROM static\_cursor;

-- Snapshot; changes in table not visible

-- Dynamic Cursor

DECLARE dynamic\_cursor CURSOR DYNAMIC FOR

SELECT EmployeeID, FirstName FROM Employees;

OPEN dynamic\_cursor;

FETCH NEXT FROM dynamic\_cursor;

-- Reflects real-time table changes

-- Forward-Only Cursor

DECLARE forward\_cursor CURSOR FORWARD\_ONLY FOR

SELECT EmployeeID, FirstName FROM Employees;

OPEN forward\_cursor;

FETCH NEXT FROM forward\_cursor;

-- Can only move forward

-- Keyset-Driven Cursor

DECLARE keyset\_cursor CURSOR KEYSET FOR

SELECT EmployeeID, FirstName FROM Employees;

OPEN keyset\_cursor;

FETCH NEXT FROM keyset\_cursor;

## **8. Exception Handling**

-- Sample Data

INSERT INTO Departments (DepartmentID, DepartmentName) VALUES

(1, 'HR'), (2, 'IT'), (3, 'Finance');

-- Question 1: TRY...CATCH with Error Logging

CREATE PROCEDURE AddEmployee

@FirstName VARCHAR(50),

@LastName VARCHAR(50),

@Email VARCHAR(100),

@Salary DECIMAL(10, 2),

@DepartmentID INT

AS

BEGIN

BEGIN TRY

IF @Salary <= 0

RAISERROR('Salary must be greater than zero.', 16, 1);

INSERT INTO Employees (EmployeeID, FirstName, LastName, Email, Salary, DepartmentID)

VALUES ((SELECT ISNULL(MAX(EmployeeID), 0) + 1 FROM Employees),

@FirstName, @LastName, @Email, @Salary, @DepartmentID);

END TRY

BEGIN CATCH

INSERT INTO AuditLog (Action, ErrorMessage)

VALUES ('AddEmployee', ERROR\_MESSAGE());

END CATCH

END;

-- Question 2: Re-raise Errors using THROW

ALTER PROCEDURE AddEmployee

@FirstName VARCHAR(50),

@LastName VARCHAR(50),

@Email VARCHAR(100),

@Salary DECIMAL(10, 2),

@DepartmentID INT

AS

BEGIN

BEGIN TRY

IF @Salary <= 0

RAISERROR('Salary must be greater than zero.', 16, 1);

INSERT INTO Employees (EmployeeID, FirstName, LastName, Email, Salary, DepartmentID)

VALUES ((SELECT ISNULL(MAX(EmployeeID), 0) + 1 FROM Employees),

@FirstName, @LastName, @Email, @Salary, @DepartmentID);

END TRY

BEGIN CATCH

INSERT INTO AuditLog (Action, ErrorMessage)

VALUES ('AddEmployee', ERROR\_MESSAGE());

THROW;

END CATCH

END;

-- Question 3: Custom Error with RAISERROR already handled above.

-- Question 4: Nested TRY...CATCH with RAISERROR

CREATE PROCEDURE TransferEmployee

@EmployeeID INT,

@NewDepartmentID INT

AS

BEGIN

BEGIN TRY

BEGIN TRANSACTION;

IF NOT EXISTS (SELECT 1 FROM Departments WHERE DepartmentID = @NewDepartmentID)

BEGIN

BEGIN TRY

RAISERROR('Invalid DepartmentID. Department does not exist.', 16, 1);

END TRY

BEGIN CATCH

INSERT INTO AuditLog (Action, ErrorMessage)

VALUES ('TransferEmployee', ERROR\_MESSAGE());

THROW;

END CATCH

END

UPDATE Employees

SET DepartmentID = @NewDepartmentID

WHERE EmployeeID = @EmployeeID;

COMMIT;

END TRY

BEGIN CATCH

ROLLBACK;

THROW;

END CATCH

END;

-- Question 5: Logging All Errors in a Transaction

CREATE PROCEDURE BatchInsertEmployees

AS

BEGIN

BEGIN TRY

BEGIN TRANSACTION;

INSERT INTO Employees (EmployeeID, FirstName, LastName, Email, Salary, DepartmentID)

VALUES ((SELECT ISNULL(MAX(EmployeeID), 0) + 1 FROM Employees), 'Rohit', 'Sharma', 'rohit@example.com', 7000.00, 1);

INSERT INTO Employees (EmployeeID, FirstName, LastName, Email, Salary, DepartmentID)

VALUES ((SELECT ISNULL(MAX(EmployeeID), 0) + 1 FROM Employees), 'Neha', 'Patel', 'neha@example.com', -1000.00, 2); -- Invalid salary

INSERT INTO Employees (EmployeeID, FirstName, LastName, Email, Salary, DepartmentID)

VALUES ((SELECT ISNULL(MAX(EmployeeID), 0) + 1 FROM Employees), 'Anil', 'Reddy', 'anil@example.com', 4000.00, 3);

COMMIT;

END TRY

BEGIN CATCH

ROLLBACK;

INSERT INTO AuditLog (Action, ErrorMessage)

VALUES ('BatchInsertEmployees', ERROR\_MESSAGE());

END CATCH

END;

-- Question 6: Dynamic RAISERROR with Severity and State

ALTER PROCEDURE AddEmployee

@FirstName VARCHAR(50),

@LastName VARCHAR(50),

@Email VARCHAR(100),

@Salary DECIMAL(10, 2),

@DepartmentID INT

AS

BEGIN

BEGIN TRY

IF @Salary < 0

RAISERROR('Salary cannot be negative.', 16, 1);

ELSE IF @Salary < 1000

RAISERROR('Salary is too low.', 10, 1);

INSERT INTO Employees (EmployeeID, FirstName, LastName, Email, Salary, DepartmentID)

VALUES ((SELECT ISNULL(MAX(EmployeeID), 0) + 1 FROM Employees),

@FirstName, @LastName, @Email, @Salary, @DepartmentID);

END TRY

BEGIN CATCH

INSERT INTO AuditLog (Action, ErrorMessage)

VALUES ('AddEmployee', ERROR\_MESSAGE());

THROW;

END CATCH

END;