Name : Prathik Balaji N
Date   : 27/07/24 - 28/07/24

Aggregate Function in SQL

1. What is an aggregate function in SQL? Give an example.

An aggregate function in SQL performs a calculation on a set of values and returns a single value. Common aggregate functions include SUM(), AVG(), COUNT(), MAX(), and MIN().

```
SELECT AVG(salary) AS average_salary
FROM employees;
```

2. How can you use the GROUP BY clause in combination with aggregate functions?

The GROUP BY clause groups rows that have the same values in specified columns into summary rows. It is often used with aggregate functions to perform calculations on each group of rows.

```
SELECT department, AVG(salary) AS average_salary
FROM employees
GROUP BY department;
```

3. Describe a scenario where atomicity is crucial for database operations.

```
CREATE OR ALTER PROCEDURE ProcessBalance
AS
BEGIN
        BEGIN TRY
                BEGIN TRANSACTION;
                        UPDATE accounts
                        SET balance = balance - 100
                        WHERE accountid = 'A';

                        UPDATE accounts
                        SET balance = balance + 100
                        WHERE accountid = 'B';

                        IF @@ERROR <> 0
                        BEGIN
                ROLLBACK TRANSACTION;
                PRINT 'Transaction failed and was rolled back';
            END
            ELSE
            BEGIN
                COMMIT TRANSACTION;
                PRINT 'Transaction succeeded';
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
        BEGIN
            ROLLBACK TRANSACTION;
        END

        SELECT ERROR_MESSAGE();
    END CATCH
END
```

OLAP and OLTP

1. Mention any 2 of the differences between OLAP and OLTP

OLAP (Online Analytical Processing):

1. **Purpose:** Designed for complex queries and data analysis (e.g., analyzing sales trends).
2. **Data Volume:** Handles large volumes of historical data (e.g., data warehouses).

OLTP (Online Transaction Processing):

1. **Purpose:** Manages transactional data for day-to-day operations (e.g., processing bank transactions).
2. **Data Volume:** Handles numerous short, atomic transactions (e.g., retail point-of-sale systems).

2. How do you optimize an OLTP database for better performance?

By Creating Indexes.

```
CREATE INDEX idx_customer_name ON customers (customer_name);
```

Data Encryption and Storage

1.What are the different types of data encryption available in MSSQL?

Transparent Data Encryption (TDE):

- Encrypts the entire database at rest.
- Protects data files and backup files.
- Requires no changes to the application.

Cell-Level Encryption:

- Encrypts specific columns or cells within a table.
- Provides granular control over which data is encrypted.

Always Encrypted:

- Ensures data is encrypted both at rest and in transit.
- Encryption keys are managed outside the database.

SQL, NOSQL, Applications, Embedded

1. What is the main difference between SQL and NoSQL databases?

SQL Databases:

- **Structure:** Relational, with a predefined schema.
- **Query Language:** Use SQL.
- **ACID Compliance:** Ensures strong consistency and transactional integrity.
- **Examples:** MySQL, PostgreSQL, SQL Server.

NoSQL Databases:

- **Structure:** Non-relational, with flexible schemas.
- **Data Models:** Key-value, document, column-family, and graph.
- **Scalability:** Designed for horizontal scaling and large volumes of data.
- **BASE Compliance:** Eventually consistent.
- **Examples:** MongoDB, Cassandra, Redis, Neo4j.

DDL (Data Definition Language)

1. How do you create a new schema in MSSQL?

To create a new schema in Microsoft SQL Server, you use the CREATE SCHEMA statement.

Example:

CREATE SCHEMA NewSchema;

2. Describe the process of altering an existing table.

To alter an existing table in Microsoft SQL Server, you use the ALTER TABLE statement. You can add, modify, or drop columns and constraints.

Example:

Add a column:
ALTER TABLE Employees ADD DateOfBirth DATE;

Modify a column:
ALTER TABLE Employees ALTER COLUMN LastName NVARCHAR(100);

Drop a column:
ALTER TABLE Employees DROP COLUMN DateOfBirth;

3. What is the difference between a VIEW and a TABLE in MSSQL?

- **TABLE:**

  A table is a database object that stores data in rows and columns.

- **VIEW:**

  A view is a virtual table based on a SELECT query.

4. Explain how to create and manage indexes in a table.

Creating an Index:

Example:
CREATE INDEX idx_lastname ON Employees (LastName);

Managing Indexes:

Rebuilding an Index:
ALTER INDEX idx_lastname ON Employees REBUILD;

Dropping an Index:
DROP INDEX idx_lastname ON Employees;

DML (Data Manipulation Language)

1. What are the most commonly used DML commands?

The most commonly used DML commands are:

- **SELECT:** Retrieves data from the database.
- **INSERT:** Adds new records to a table.
- **UPDATE:** Modifies existing records in a table.
- **DELETE:** Removes records from a table.

2. How do you retrieve data from multiple tables using a JOIN?

You can retrieve data from multiple tables using different types of joins: INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN.

Example:

SELECT Employees.EmployeeID, Employees.LastName, Orders.OrderID FROM Employees INNER JOIN Orders ON Employees.EmployeeID = Orders.EmployeeID;

3. Explain how relational algebra is used in SQL queries.

Relational algebra provides a theoretical foundation for SQL operations. SQL queries often use relational algebra concepts such as:

- **Selection (σ):** Filtering rows (using WHERE).
- **Projection (π):** Selecting specific columns.
- **Join (⋈):** Combining rows from two or more tables based on a related column.

- **Union (∪), Intersection (∩), Difference (−):** Combining or comparing result sets.

4. What are the implications of using complex queries in terms of performance?

- **Increased Load:** Complex queries can increase the load on the database server.
- **Longer Execution Time:** May result in longer execution times and slower performance.
- **Resource Intensive:** Can consume more CPU, memory, and I/O resources.
- **Optimization Needed:** Often require query optimization techniques such as indexing, query rewriting, and proper database design.

Aggregate Functions

1. How does the HAVING clause differ from the WHERE clause when using aggregate functions? Show whether you could use HAVING before GROUP BY in the SELECT statement.

- **WHERE Clause:** Filters rows before any groupings are made.
- **HAVING Clause:** Filters groups after the GROUP BY operation has been performed.

Example:

SELECT Department, COUNT(EmployeeID) AS EmployeeCountFROM Employees GROUP BY Department HAVING COUNT(EmployeeID) > 5;

Filters

1. What are filters in SQL and how are they used in queries?

Filters in SQL are used to restrict the data returned by a query based on specific conditions. They help in narrowing down the result set to include only those rows that meet the criteria specified.

Usage:

- Filters are typically applied using the WHERE clause.

Example:
SELECT * FROM Employees WHERE Department = 'Sales';

2. How do you use the WHERE clause to filter data in MSSQL?

The WHERE clause is used to specify the condition(s) that rows must meet to be included in the result set.

Example:

SELECT * FROM Employees WHERE Age > 30 AND Department = 'HR';

3. How can you combine multiple filter conditions using logical operators?

You can combine multiple filter conditions using logical operators such as AND, OR, and NOT.

Examples:

AND: All conditions must be true.

SELECT * FROM Employees WHERE Age > 30 AND Department = 'HR';

OR: At least one condition must be true.

SELECT * FROM Employees WHERE Age > 30 OR Department = 'HR';

NOT: Negates the condition.
SELECT * FROM Employees WHERE NOT Department = 'HR';

4. Explain the use of CASE statements for filtering data in a query.

The CASE statement allows you to apply conditional logic within a query, enabling different outcomes based on conditions.

Example:

SELECT EmployeeID, Name,

CASE

WHEN Age < 30 THEN 'Young'

WHEN Age BETWEEN 30 AND 50 THEN 'Middle-aged'

ELSE 'Senior' END AS AgeGroupFROM Employees;

Operators

1. What are the different types of operators available in MSSQL?

SQL Server supports various types of operators:

- **Arithmetic Operators:** +, -, *, /, %
- **Comparison Operators:** =, !=, <, >, <=, >=
- **Logical Operators:** AND, OR, NOT
- **String Operators:** + (concatenation)
- **Bitwise Operators:** &, |, ^, ~

2. How do arithmetic operators work in SQL?

Arithmetic operators perform mathematical operations on numeric data types.

Examples:

- **Addition:** SELECT 10 + 5; returns 15.
- **Subtraction:** SELECT 10 - 5; returns 5.
- **Multiplication:** SELECT 10 * 5; returns 50.
- **Division:** SELECT 10 / 5; returns 2.
- **Modulo:** SELECT 10 % 3; returns 1.

3. Explain the use of LIKE operator with wildcards for pattern matching.

The LIKE operator is used for pattern matching with string data.

Wildcards:

% : Matches zero or more characters.

SELECT * FROM EmployeesWHERE Name LIKE 'J%';

_ : Matches exactly one character.

SELECT * FROM EmployeesWHERE Name LIKE 'J_n';

Multiple Tables: Normalization

1. What is normalization and why is it important?

Normalization is the process of organizing data in a database to reduce redundancy and improve data integrity. It involves dividing a database into tables and establishing relationships between them.

Importance:

- Minimizes duplication of data.
- Ensures data dependencies make sense.
- Helps in efficient data retrieval and update.

2. Describe the basic normal forms. Hint: 1NF 2NF 3NF

- **First Normal Form (1NF):**

  Ensures that each column contains atomic (indivisible) values and each record is unique.

  Example: A table with columns for StudentID, Name, and Course.

- **Second Normal Form (2NF):**

  Builds on 1NF. Ensures that all non-key attributes are fully functionally dependent on the primary key.

  Example: In a table with StudentID, Course, and Instructor, separate Instructor into its own table.

- **Third Normal Form (3NF):**

  Builds on 2NF. Ensures that all attributes are only dependent on the primary key and not on other non-key attributes.

Example: Separate Course details into a different table if Instructor is only dependent on Course.

3. Mention any one impact of normalization on database performance.

Impact:

- **Increased Complexity:** Normalization can lead to more complex queries and joins, which may affect query performance.

Joins vs Subqueries

1. What is the difference between joins and subqueries?

- **Joins:**

  Combine rows from two or more tables based on a related column.

  Typically used to retrieve data from multiple tables in a single query.

  More efficient for fetching related data across tables.

Example:

```
SELECT Employees.Name, Departments.DepartmentName FROM Employees
INNER JOIN Departments ON Employees.DepartmentID =
Departments.DepartmentID;
```

- **Subqueries:**

  A query nested inside another query.

  Can be used in the SELECT, WHERE, or FROM clause.

  Useful for performing operations that require intermediate results.

Example:

SELECT Name FROM Employees WHERE DepartmentID = (SELECT DepartmentID FROM Departments WHERE DepartmentName = 'Sales');

2. When would you prefer a subquery over a join?

- **Subqueries:**

  When you need to filter or aggregate data based on a result from another query.

  Useful for situations where the result of the subquery is needed as a condition in the outer query.

  Ideal for comparing a value against a list or a single value returned by another query.

Example:

SELECT Name FROM Employees WHERE Salary > (SELECT AVG(Salary) FROM Employees);

3. Explain how correlated subqueries work with an example.

A **correlated subquery** is a subquery that references columns from the outer query. It executes once for each row processed by the outer query.

Example:

SELECT e1.Name FROM Employees e1 WHERE e1.Salary > (SELECT AVG(e2.Salary) FROM Employees e2 WHERE e1.DepartmentID = e2.DepartmentID);

- The subquery depends on the current row of the outer query, using e1.DepartmentID.

4. Discuss the performance implications of using joins vs subqueries.

- **Joins:**

  Typically more efficient for combining and retrieving related data.

Databases are optimized for join operations and can use indexes effectively.

- **Subqueries:**

  Can be less efficient, especially if the subquery is executed multiple times.

  Correlated subqueries may lead to performance issues if they are run for each row of the outer query.

  Performance can vary depending on the complexity of the subquery and the database's execution plan.

Types

1. What are the different data types available in MSSQL?

Data Types in MSSQL:

- **Numeric Types:** INT, BIGINT, SMALLINT, FLOAT, DECIMAL, NUMERIC
- **Character Types:** CHAR, VARCHAR, TEXT, NCHAR, NVARCHAR
- **Date and Time Types:** DATE, TIME, DATETIME, DATETIME2, SMALLDATETIME, DATETIMEOFFSET
- **Binary Types:** BINARY, VARBINARY, IMAGE
- **Other Types:** BIT, UNIQUEIDENTIFIER, XML, JSON (via functions)

2. How do you choose the appropriate data type for a column?

- **Nature of Data:** Select a data type that matches the type of data you will store (e.g., numeric, text, date).
- **Size:** Choose data types that use space efficiently. For example, use INT instead of BIGINT if the range is sufficient.
- **Performance:** Consider performance implications, such as indexing and storage efficiency.
- **Compatibility:** Ensure the data type supports the operations and functions needed for your application.

3. How do you handle data type conversions in queries?

CAST Function: Converts an expression from one data type to another.
SELECT CAST(Price AS DECIMAL(10,2)) FROM Products;

CONVERT Function: Similar to CAST, but with additional style options for date and time.
SELECT CONVERT(VARCHAR, OrderDate, 101) FROM Orders;

- **Implicit Conversion:** SQL Server often automatically converts data types as needed, but explicit conversion is recommended for clarity and control.

Correlation and Non-Correlation

1. What is a correlated subquery?

A **correlated subquery** is a subquery that refers to columns from the outer query. It must be executed for each row processed by the outer query.

Example:

SELECT e.Name FROM Employees e WHERE e.Salary > (SELECT AVG(e2.Salary) FROM Employees e2 WHERE e.DepartmentID = e2.DepartmentID);

2. What is a non-correlated subquery?

A **non-correlated subquery** is independent of the outer query and can be executed independently. It does not reference columns from the outer query.

Example:

SELECT Name FROM Employees WHERE DepartmentID IN (SELECT DepartmentID FROM Departments WHERE DepartmentName = 'Sales');

- The inner query is executed once and does not depend on the outer query.

3. Explain how correlated subqueries can affect query performance.

- **Performance Impact:** Correlated subqueries can be less efficient because the subquery is executed once for each row processed by the outer query.
- **Increased Overhead:** Can lead to high computational overhead and slow performance, especially with large datasets.

- **Optimization:** To mitigate performance issues, consider using joins or indexing to improve query efficiency.

Introduction to T-SQL, Procedures, Functions, Triggers, Indices

1. What is T-SQL and how does it extend standard SQL?

- **T-SQL (Transact-SQL)** extends SQL with procedural programming features such as variables, loops, and error handling. It supports control-of-flow commands like IF...ELSE and WHILE, and allows for complex error handling with TRY...CATCH.

2. How do you create a stored procedure in MSSQL?

Use CREATE PROCEDURE to define a stored procedure.

Example:

```
CREATE PROCEDURE GetEmployeeDetails @EmployeeID INT

AS

BEGIN

   SELECT Name, Department FROM Employees WHERE EmployeeID =
@EmployeeID;

END;
```

3. Explain the difference between functions and procedures in MSSQL.

- **Functions:** Return a single value or table, used in queries.
- **Procedures:** Perform actions, may return multiple result sets or status codes.

4. Describe the use of triggers and provide an example scenario.

- **Triggers** execute automatically in response to changes (e.g., INSERT, UPDATE). Example: Update a log table when employee salaries change.

Comparison of T-SQL with PL/SQL

1. What are the main differences between T-SQL and PL/SQL?

- **T-SQL** (Microsoft SQL Server) supports procedural programming and error handling with TRY...CATCH.
- **PL/SQL** (Oracle) uses DECLARE...BEGIN...EXCEPTION...END blocks and offers more extensive procedural capabilities.

Aggregate and Atomicity

1. Describe a scenario where you would use the SUM aggregate function to calculate total sales for each month.

Example: Calculate monthly sales totals:

SELECT YEAR(SaleDate) AS Year, MONTH(SaleDate) AS Month, SUM(SaleAmount) AS TotalSales FROM Sales GROUP BY YEAR(SaleDate), MONTH(SaleDate);

2. You have a banking application where transactions must be all-or-nothing. Explain how you would implement atomicity to ensure this.

- **Use Transactions:** Ensure all operations succeed or none at all using BEGIN TRANSACTION, COMMIT, and ROLLBACK.

Security and Accessibility

1. Imagine you are setting up a new database for an e-commerce website. How would you ensure that only authorized users have access to sensitive customer data?

- **Use Role-Based Access Control (RBAC)**, create views to limit data exposure, encrypt sensitive data, and regularly audit access logs.

MSSQL Install and Configure, OLAP and OLTP

1. Designing and Optimizing OLTP and OLAP Systems

- **OLTP System:**

  Design: Normalize tables; index primary/foreign keys.

  Optimize: Use indexes, partition tables, and handle concurrency.

- **OLAP System:**

Design: Use star or snowflake schema for data warehousing.

Optimize: Implement data aggregation, use bitmap indexes, and materialized views.

Data Encryption and Storage

1. What is authentication vs authorization?

- **Authentication:** Verifies identity (e.g., logging in with a username and password).
- **Authorization:** Determines access levels and permissions (e.g., access to specific resources).

DDL (Data Definition Language)

1. Create a new table with constraints.

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    Name NVARCHAR(100),
    Position NVARCHAR(50),
    Salary DECIMAL(10, 2),
    CONSTRAINT UQ_EmployeeID UNIQUE (EmployeeID)
);
```

2. Add a new column if it doesn't exist.

```
IF NOT EXISTS (SELECT * FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = 'Employees' AND COLUMN_NAME = 'Email')
BEGIN
    ALTER TABLE Employees ADD Email NVARCHAR(255);
```

END

3. What is a Composite Key?

- **Composite Key:** A primary key made up of two or more columns to uniquely identify a record.

DML (Data Manipulation Language)

1. Update all salaries by 10%.

UPDATE Employees SET Salary = Salary * 1.10;

2. Retrieve orders from customers in a specific city.

SELECT o.OrderID, o.OrderDate, c.CustomerName, c.City FROM Orders o JOIN Customers c ON o.CustomerID = c.CustomerIDWHERE c.City = 'SpecificCity';

Aggregate Functions

1. Write a query to find the average Salary in the Employees table, grouped by Department.

SELECT Department, AVG(Salary) AS AverageSalary FROM Employees GROUP BY Department;

2. Describe a scenario where you would use the RANK function to assign ranks to employees based on their sales performance.

Scenario: A sales manager wants to rank employees based on their total sales to identify top performers for bonus allocation.

Example Query:

SELECT EmployeeID, EmployeeName, TotalSales,RANK() OVER (ORDER BY TotalSales DESC) AS SalesRank FROM SalesPerformance;

Filters

1. Write a query to filter out all products from the Products table that have a Price greater than Rs. 100.

```
SELECT * FROM Products WHERE Price > 100;
```

2. Explain how you would use the CASE statement to filter data based on multiple conditions, such as categorizing products into different price ranges.

Example Query:

```
SELECT ProductID, ProductName, Price,
    CASE
      WHEN Price < 50 THEN 'Cheap'
      WHEN Price BETWEEN 50 AND 100 THEN 'Moderate'
      ELSE 'Expensive'
    END AS PriceCategory
FROM Products;
```

Operators

1. Write a query to find all employees whose Name starts with the letter 'A'.

```
SELECT * FROM Employees WHERE Name LIKE 'A%';
```

Multiple Tables: Normalization

1. Steps to Normalize a Denormalized Database to 3NF

1. **Identify and Remove Redundant Data (1NF):**

   Ensure each table has a primary key.

   Eliminate repeating groups and ensure each column contains atomic values.

2. **Eliminate Partial Dependencies (2NF):**

Ensure that all non-key attributes are fully dependent on the entire primary key.

Create separate tables for attributes that depend on part of a composite primary key.

3. **Eliminate Transitive Dependencies (3NF):**

   Ensure that non-key attributes are not dependent on other non-key attributes.

   Remove transitive dependencies by creating new tables for data that only indirectly relates to the primary key.

Example:

Before Normalization:
Table: Orders

Columns: OrderID, CustomerName, ProductName, OrderDate, CustomerAddress

After Normalization (to 3NF):
Table: Orders

Columns: OrderID, CustomerID, ProductID, OrderDate

Table: Customers

Columns: CustomerID, CustomerName, CustomerAddress

Table: Products

Columns: ProductID, ProductName

Indexes & Constraints

1. Write a SQL statement to create an index on the Email column of the Users table.

```
CREATE INDEX IDX_Email ON Users (Email);
```

Joins

1. Write a query to perform an inner join between the Orders and Customers tables to retrieve all orders along with customer names.

SELECT o.OrderID, o.OrderDate, c.CustomerName FROM Orders oINNER JOIN Customers c ON o.CustomerID = c.CustomerID;

2. Describe a scenario where a full outer join would be necessary, and provide a query example.

Scenario: You want to list all customers and their orders, including customers who have not placed any orders and orders that have no associated customers.

Example Query:

SELECT c.CustomerName, o.OrderID FROM Customers cFULL OUTER JOIN Orders o ON c.CustomerID = o.CustomerID;

Alias

1. Write a query using table aliases to simplify a complex join between three tables: Orders, Customers, and Products.

SELECT o.OrderID, c.CustomerName,p.ProductName FROM Orders o INNER JOIN Customers c ON o.CustomerID = c.CustomerID INNER JOIN Products p ON o.ProductID = p.ProductID;

2. Explain how column aliases can be used in a query with aggregate functions to make the results more readable.

Example Query:

SELECT Department, AVG(Salary) AS AverageSalary, MAX(Salary) AS HighestSalary FROM Employees GROUP BY Department;

Explanation: Column aliases like AverageSalary and HighestSalary make the output clearer and more descriptive.

Joins vs Sub Queries

1. Provide a scenario where a subquery would be more appropriate than a join, and write the corresponding query.

Scenario: Find employees who have made more sales than the average number of sales by all employees.

Example Query:

SELECT EmployeeID, EmployeeName FROM Employees WHERE SalesCount > (SELECT AVG(SalesCount) FROM Employees);

2. Compare the performance implications of using a join versus a subquery for retrieving data from large tables.

- **Joins:**

  Performance: Typically faster for large tables if properly indexed. Joins are optimized by the database engine and can efficiently handle large datasets.

- **Subqueries:**

  Performance: Can be slower for large datasets, especially if the subquery involves aggregations or multiple levels of nesting. Performance can vary based on the complexity of the subquery and indexing.

Types

1. Describe a scenario where using a DATETIME data type would be essential, and explain how you would store and retrieve this data.

Scenario:

- **Use Case:** Tracking employee work hours and scheduling. Accurate timestamps are crucial for recording when employees clock in and out, or for scheduling meetings.

Storing Data:
CREATE TABLE WorkHours (

  EmployeeID INT,

  ClockIn DATETIME,

ClockOut DATETIME);

Retrieving Data:

Query Example:
SELECT EmployeeID, ClockIn, ClockOut FROM WorkHours WHERE ClockIn >= '2024-07-01' AND ClockOut <= '2024-07-31';

Correlation and Non-Correlation

1. Write a query using a correlated subquery to find all employees who have a salary higher than the average salary in their department.

SELECT e.EmployeeID, e.EmployeeName, e.Salary FROM Employees e WHERE e.Salary > (SELECT AVG(e2.Salary) FROM Employees e2 WHERE e2.DepartmentID = e.DepartmentID);

2. Explain how non-correlated subqueries can be optimized for better performance compared to correlated subqueries.

Non-Correlated Subqueries:

- **Definition:** Subqueries that can be executed independently of the outer query.
- **Optimization:** Often faster because the subquery result can be computed once and reused. Indexes can be applied to the subquery results, and the database engine can optimize execution.

Correlated Subqueries:

- **Definition:** Subqueries that depend on the outer query's data.
- **Performance Impact:** Executed once for each row in the outer query, which can be slower for large datasets. Optimization may involve using indexes on correlated columns and rewriting the query to minimize the number of executions.

Introduction to TSQL, Procedures, Functions, Triggers, Indices

1. Write a simple stored procedure to insert a new record into the Employees table.

```
CREATE PROCEDURE AddEmployee

    @EmployeeID INT,

    @Name NVARCHAR(100),

    @Position NVARCHAR(50),

    @Salary DECIMAL(10, 2)

AS

BEGIN

    INSERT INTO Employees (EmployeeID, Name, Position, Salary) VALUES
(@EmployeeID, @Name, @Position, @Salary);

END
```

2. Describe a scenario where you would use a trigger to enforce business rules.

Scenario:

- **Use Case:** Enforcing a rule that prevents employees from having a salary greater than a certain limit unless they are in a specific department.

Example Trigger:

```
CREATE TRIGGER trg_CheckSalary ON Employees

AFTER INSERT, UPDATE

AS

BEGIN

    IF EXISTS (SELECT * FROM inserted WHERE Salary > 100000      AND
DepartmentID != 1)

    BEGIN

        RAISERROR('Salary exceeds limit for certain  departments.', 16, 1);
        ROLLBACK;

    END

END
```

Security and Accessibility

1. Mention any 2 of the common security measures to protect a SQL Server database.

- **Encryption:** Use Transparent Data Encryption (TDE) or cell-level encryption to protect sensitive data.
- **Access Control:** Implement role-based access control (RBAC) to restrict permissions based on user roles.

2. How do you create a user and assign roles in MSSQL?

Creating a User:

```
CREATE USER [username] FOR LOGIN [username];
```

Assigning Roles:

```
ALTER SERVER ROLE [db_owner] ADD MEMBER [username];
```

3. Explain how encryption can be implemented for data at rest in MSSQL.

- **Transparent Data Encryption (TDE):**

  Setup: Encrypt the database files using TDE.

  Steps:

  1. Create a database encryption key.
  2. Create a master key.
  3. Encrypt the database using the encryption key.

```
CREATE TABLE Users(

UserID INT PRIMARY KEY IDENTITY,

EncryptedName VARBINARY(250)

)
```

```
CREATE MASTER KEY ENCRYPTION BY PASSWORD='mydb@123'

GO

CREATE CERTIFICATE MySimpleCertificate WITH Subject='My Simple
Encryption Certificate'

CREATE SYMMETRIC KEY MySimpleSymmetricKey

WITH ALGORITHM = AES_256

ENCRYPTION BY CERTIFICATE MySimpleCertificate;


CREATE PROCEDURE InsertEncryptedName @Name NVARCHAR(100)

AS

BEGIN

OPEN SYMMETRIC KEY MySimpleSymmetricKey

DECRYPTION BY CERTIFICATE MySimpleCertificate;

DECLARE @EncryptedName VARBINARY(MAX);

SET @EncryptedName =
ENCRYPTBYKEY(KEY_GUID('MySimpleSymmetricKey'), @Name)

INSERT INTO Users(EncryptedName) VALUES(@EncryptedName)

CLOSE SYMMETRIC KEY MySimpleSymmetricKey;

END;

EXec InsertEncryptedName @Name= 'John Doe';

select * from users;

Go

CREATE PROCEDURE GetDecryptedName @UserID INT
```

```
AS

BEGIN

OPEN SYMMETRIC KEY MySimpleSymmetricKey

DECRYPTION BY CERTIFICATE MySimpleCertificate;

DECLARE @EncryptedName VARBINARY(MAX);

DECLARE @DecryptedName NVARCHAR(100);


SELECT @EncryptedName = EncryptedName FROM Users where UserID =
@UserID;

SET @DecryptedName =
CONVERT(nvarchar(100),Decryptbykey(@EncryptedName));

CLOSE SYMMETRIC KEY MySimpleSymmetricKey;

Select @DecryptedName as Name;

END;

GO

Declare @userId int = 1;

exec GetDecryptedName @UserID = @UserID
```