Name:SIDDARTH S

Week 1 Assignment

Aggregate and Atomicity

Question 1: What is an aggregate function in SQL? Give an example.
Answer 1: An aggregate function in SQL performs a calculation on a set of values and returns a single value. Examples include SUM, AVG, COUNT, MAX, and MIN.
Example: `SELECT AVG(Salary) FROM Employees;`

Question 2: How can you use the GROUP BY clause in combination with aggregate functions?
Answer 2: The GROUP BY clause groups rows that have the same values in specified columns into summary rows, often used with aggregate functions like COUNT, SUM, AVG, MAX, or MIN to perform operations on each group.
Example: `SELECT Department, AVG(Salary) FROM Employees GROUP BY Department;`

Question 3: Describe a scenario where atomicity is crucial for database operations.
Answer 3: In a banking application, atomicity is crucial when processing transactions such as transfers. If a customer transfers money from one account to another, both the debit from one account and the credit to another must occur together, or not at all, to ensure data integrity and prevent discrepancies in account balances.

OLAP and OLTP

Question 4: Mention any 2 of the differences between OLAP and OLTP?
Answer 4: 1. OLAP (Online Analytical Processing) is designed for complex queries and data analysis, often used for decision support. OLTP (Online Transaction Processing) is designed for managing transaction-oriented applications, focusing on fast query processing and maintaining data integrity in multi-access environments.
2. OLAP databases are typically read-heavy and optimized for query performance, while OLTP databases are write-heavy and optimized for transactional speed and data consistency.

Question 5: How do you optimize an OLTP database for better performance? Hint: index
Answer 5: To optimize an OLTP database for better performance, you can create indexes on frequently queried columns to speed up data retrieval. Additionally, normalization can help reduce data redundancy and improve data integrity, while proper indexing strategies can balance read and write performance.

Data Encryption and Storage

Question 6: What are the different types of data encryption available in MSSQL?
Answer 6: In MSSQL, the different types of data encryption include Transparent Data Encryption (TDE), which encrypts the entire database, column-level encryption, which encrypts specific columns, and Always Encrypted, which ensures sensitive data is encrypted both in transit and at rest. Additionally, backup encryption can secure database backups.

SQL, NoSQL, Applications, Embedded

Question 7: What is the main difference between SQL and NoSQL databases?
Answer 7: The main difference between SQL and NoSQL databases is that SQL databases are relational and use structured query language (SQL) for defining and manipulating data, with a predefined schema. NoSQL databases are non-relational, can store unstructured data, and provide flexible schemas, often using JSON, BSON, or key-value pairs for data representation.

DDL

Question 8: How do you create a new schema in MSSQL?
Answer 8: To create a new schema in MSSQL, you use the CREATE SCHEMA statement.
Example: `CREATE SCHEMA Sales;`

Question 9: Describe the process of altering an existing table.
Answer 9: To alter an existing table, you use the ALTER TABLE statement followed by the specific alteration you need, such as adding, dropping, or modifying columns.
Example: `ALTER TABLE Employees ADD Email VARCHAR(255);`

Question 10: What is the difference between a VIEW and a TABLE in MSSQL?
Answer 10: A TABLE is a database object that stores data in rows and columns. A VIEW is a virtual table that provides a specific representation of

data from one or more tables through a SELECT query. Views do not store data themselves but display data stored in tables.

Question 11: Explain how to create and manage indexes in a table.
Answer 11: To create an index, you use the CREATE INDEX statement, specifying the index name and the columns to be indexed. Managing indexes involves creating, altering, and dropping indexes to optimize query performance.
Example: `CREATE INDEX idx_email ON Users (Email);`

DML

Question 12: What are the most commonly used DML commands?
Answer 12: The most commonly used Data Manipulation Language (DML) commands are SELECT (to retrieve data), INSERT (to add new data), UPDATE (to modify existing data), and DELETE (to remove data).

Question 13: How do you retrieve data from multiple tables using a JOIN?
Answer 13: To retrieve data from multiple tables using a JOIN, you use the SELECT statement with a JOIN clause, specifying the type of join (INNER, LEFT, RIGHT, FULL) and the conditions for joining the tables.
Example: `SELECT Orders.OrderID, Customers.CustomerName FROM Orders INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;`

Question 14: Explain how relational algebra is used in SQL queries.
Answer 14: Relational algebra provides a theoretical foundation for SQL queries, using operations like selection, projection, union, set difference,

Cartesian product, and rename to manipulate relations (tables). SQL queries implement these operations to retrieve and manipulate data.

Question 15: What are the implications of using complex queries in terms of performance?
Answer 15: Complex queries can impact performance by increasing the processing time and resource usage. They may require more memory and CPU, and involve multiple joins, subqueries, or aggregations, leading to longer execution times. Optimizing queries, indexing, and proper query planning can help mitigate these performance issues.

Aggregate Functions

Question 16: How does the HAVING clause differ from the WHERE clause when using aggregate functions?
Answer 16: The HAVING clause is used to filter groups of rows after the GROUP BY clause has been applied, while the WHERE clause filters rows before any grouping is done. HAVING is used with aggregate functions, whereas WHERE cannot be used with aggregate functions.

Filters

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

Filters in SQL restrict the data returned by a query based on specific conditions, implemented using clauses like WHERE, HAVING, LIMIT, and OFFSET.

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

The WHERE clause is used to specify conditions that must be met for rows to be selected.

SELECT * FROM employees WHERE department_id = 5;

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

SELECT * FROM employees WHERE department_id = 5 AND salary > 50000;
This query selects employees from department 5 with a salary greater than 50,000.

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

The CASE statement allows for conditional logic within SQL queries, enabling different outputs based on specified conditions.

SELECT name, CASE WHEN salary < 50000 THEN 'Low' WHEN salary BETWEEN 50000 AND 100000 THEN 'Medium' ELSE 'High' END as salary_range FROM employees;

Operators

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

MSSQL supports various types of operators:

- **Arithmetic Operators**: +, -, *, /, %
- **Comparison Operators**: =, !=, <>, >, <, >=, <=
- **Logical Operators**: AND, OR, NOT

- **Bitwise Operators**: &, |, ^, ~, <<, >>
- **String Operators**: + (concatenation)

22. How do arithmetic operators work in SQL?

Arithmetic operators perform mathematical calculations on numeric data types.

SELECT salary, salary + 5000 AS new_salary FROM employees;

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

The LIKE operator is used for pattern matching in strings. Wildcards % (matches zero or more characters) and _ (matches a single character) are used.

SELECT * FROM employees WHERE name LIKE 'J%'; This finds employees whose names start with 'J'.

Multiple Tables: Normalization

24. What is normalization and why is it important?

Normalization is a process to organize database tables to reduce redundancy and improve data integrity. It involves dividing large tables into smaller ones and defining relationships between them.

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

- **1NF (First Normal Form)**: Ensures that each column contains atomic values, and each column contains only one type of data.

- **2NF (Second Normal Form)**: Meets all requirements of 1NF and ensures that all non-key attributes are fully functionally dependent on the primary key.
- **3NF (Third Normal Form)**: Meets all requirements of 2NF and ensures that no transitive dependency exists between non-key attributes.

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

Normalization can lead to increased complexity in queries, which may require joining multiple tables, potentially slowing down read performance.

Indexes & Constraints

27. What are indexes and why are they used?

Indexes are database objects that improve data retrieval speed by providing a quick way to look up rows in a table based on the values of one or more columns.

28. How do you create a unique constraint on a table column?

A unique constraint ensures that all values in a column are unique.

ALTER TABLE employees ADD CONSTRAINT unique_email UNIQUE (email);

29. Explain the difference between clustered and non-clustered indexes.

- **Clustered Index**: Determines the physical order of data in a table and can only have one per table.
- **Non-Clustered Index**: Contains a sorted list of pointers to the table rows and does not affect the physical order. A table can have multiple non-clustered indexes.

30. How would you optimize index usage in a highly transactional database?

Regularly update statistics, use indexing strategies like covering indexes, avoid excessive indexing, and periodically rebuild or reorganize fragmented indexes.

Joins

31. What are the different types of joins available in MSSQL?

- INNER JOIN
- LEFT JOIN (LEFT OUTER JOIN)
- RIGHT JOIN (RIGHT OUTER JOIN)
- FULL JOIN (FULL OUTER JOIN)
- CROSS JOIN
- SELF JOIN

32. Provide an example of a LEFT JOIN query.

SELECT e.name, d.name FROM employees e LEFT JOIN departments d ON e.department_id = d.id;

33. Explain the concept of a self-join and when it might be used.

A self-join is a regular join but the table is joined with itself. It is used to find related rows in the same table.

SELECT e1.EmployeeID, e1.Name AS EmployeeName, e2.Name AS ManagerName FROM Employees e1 JOIN Employees e2 ON e1.ManagerID = e2.EmployeeID;

34. How do you perform a full outer join and what is its significance?

A full outer join returns all rows from both tables, with matching rows from both sides where available. If there is no match, NULLs are returned for non-matching rows.

SELECT a.column1, b.column2 FROM table1 a FULL OUTER JOIN table2 b ON a.common_column = b.common_column;

Alias

35. What is an alias in SQL and how is it used?

An alias is a temporary name for a table or column used within a query for ease of reference.

SELECT e.name AS employee_name FROM employees e;

Here, e is an alias for the employees table, and employee_name is an alias for the name column.

36. Give an example of using table aliases in a query.

SELECT e.name, d.name FROM employees e JOIN departments d ON e.department_id = d.id;

In this query, e and d are aliases for the employees and departments tables, respectively.

37. How do you use column aliases in conjunction with aggregate functions?

Column aliases can rename the result of an aggregate function for clarity.

SELECT department_id, COUNT(*) AS total_employees FROM employees GROUP BY department_id;

total_employees is an alias for the count of employees in each department.

38. Explain the benefits of using aliases in complex queries.

Aliases simplify complex queries by making them more readable and easier to manage, especially when dealing with long table or column names, or when performing self-joins.

Joins vs Sub Queries

39. What is the difference between joins and subqueries?

- **Joins**: Combine columns from multiple tables into a single result set based on related columns.
- **Subqueries**: Nested queries used within another query to perform operations on data, often returning values to be used in the main query.

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

Subqueries are preferable when you need to isolate specific logic, especially if you need to use the result of the subquery as a condition or to return a single value.

41. Explain how correlated subqueries work with an example.

A correlated subquery depends on the outer query for its values and is evaluated once for each row processed by the outer query.

SELECT e1.name FROM employees e1 WHERE e1.salary > (SELECT AVG(e2.salary) FROM employees e2 WHERE e2.department_id = e1.department_id);

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

Joins are generally more efficient and faster because they are often optimized by SQL engines, especially for large datasets. Subqueries can be less efficient, particularly correlated subqueries, which may require the inner query to run multiple times.

Types

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

Common data types include:

- **Numeric**: INT, FLOAT, DECIMAL
- **String**: VARCHAR, CHAR, TEXT
- **Date and Time**: DATE, TIME, DATETIME
- **Binary**: BINARY, VARBINARY
- **Miscellaneous**: BIT, UNIQUEIDENTIFIER

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

Consider the nature of the data, its size, and performance requirements. For example, use INT for integers, VARCHAR for variable-length strings, and DATETIME for date and time data.

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

Data type conversions can be handled using the CAST or CONVERT functions.

SELECT CAST(salary AS VARCHAR(10)) FROM employees;

Correlation and Non-Correlation

46. What is a correlated subquery?

A correlated subquery is a subquery that references columns from the outer query, making it dependent on the outer query for its values.

47. What is a non-correlated subquery? Hint: Outer query checks the data from inner sub query

A non-correlated subquery is independent of the outer query, meaning it can be run on its own. The outer query uses the result of the subquery.

SELECT name FROM employees WHERE department_id = (SELECT department_id FROM departments WHERE name = 'Sales');

48. Explain how correlated subqueries can affect query performance.

Correlated subqueries can significantly affect performance because the subquery is executed for each row processed by the outer query, potentially leading to multiple executions.

Introduction to TSQL, Procedures, Functions, Triggers, Indices

49. What is TSQL and how does it extend standard SQL?

TSQL (Transact-SQL) is an extension of SQL used in Microsoft SQL Server. It includes procedural programming, error handling, and transaction control capabilities, making it more powerful for complex operations.

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

```
CREATE PROCEDURE GetEmployeeDetails (@EmployeeID INT)
 AS
BEGIN
        SELECT * FROM employees WHERE id = @EmployeeID;
END;
```

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

- **Functions**: Return a single value or a table and are used in SQL statements like SELECT.
- **Procedures**: Perform actions, can return multiple results, and support control-of-flow constructs. They are called using the EXECUTE command.

52. Describe the use of triggers and provide an example scenario.
Triggers are special types of stored procedures that automatically execute or fire when certain events occur in a database. They can be used to enforce business rules, validate data, or synchronize tables.

Example Scenario:

```
CREATE TRIGGER UpdateOrderTimestamp ON Orders
AFTER UPDATE
AS
BEGIN
    UPDATE Orders SET last_modified = GETDATE() WHERE OrderID =
    inserted.OrderID;
END;
```

Comparison input on TSQL with PL/SQL

53. What are the main differences between TSQL and PL/SQL?

- **T-SQL (Transact-SQL)** is Microsoft's proprietary extension to SQL used in SQL Server. It includes procedural programming, local variables, and control-of-flow constructs.
- **PL/SQL (Procedural Language/SQL)** is Oracle's procedural extension to SQL. It also supports procedural programming, but with different syntax and capabilities, such as exception handling and cursors.

Aggregate and Atomicity

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

```
SELECT MONTH(sale_date) AS Month, YEAR(sale_date) AS Year,
SUM(amount) AS TotalSales

FROM Sales

GROUP BY MONTH(sale_date), YEAR(sale_date);
```

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

```
BEGIN TRANSACTION;
BEGIN TRY
    -- Perform operations
    UPDATE Account SET balance = balance - 100 WHERE account_id = 1;
    UPDATE Account SET balance = balance + 100 WHERE account_id =
2;
```

```
    COMMIT;
END TRY
BEGIN CATCH
    ROLLBACK;
END CATCH;
```

Security and Accessibility

56. 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?

To ensure that only authorized users have access to sensitive customer data:

- Implement user authentication and authorization controls.
- Use roles and permissions to restrict access to sensitive data.
- Apply encryption to protect data both at rest and in transit.

Example:

```
-- Create a role
CREATE ROLE SensitiveDataAccess;
-- Grant select permission on sensitive data
GRANT SELECT ON Customers TO SensitiveDataAccess;
-- Assign role to a user
EXEC sp_addrolemember 'SensitiveDataAccess', 'username';
```

MSSQL Install and Configure, OLAP and OLTP

57. A large retail company needs a high-performing OLTP system for processing sales and an OLAP system for analyzing sales data. Explain how you would design and optimize these systems.

- **OLTP (Online Transaction Processing) System:** Focus on high-speed transactions and reliability. Design a normalized schema to minimize redundancy, and use indexing to speed up query performance.
- **OLAP (Online Analytical Processing) System:** Optimize for complex queries and data analysis. Use a star schema or snowflake schema to facilitate efficient querying, and implement aggregation tables to speed up reporting and analysis.

Data Encryption and Storage

58. What is authentication vs authorization?

- **Authentication** is the process of verifying the identity of a user or system (e.g., through usernames and passwords).
- **Authorization** determines what an authenticated user is allowed to do (e.g., what resources or operations they can access).

DDL

59. You need to create a new table with columns for EmployeeID, Name, Position, and Salary. Write the SQL statement to create this table. Add unique constraint, primary key accordingly.

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

60. A table needs to be altered to add a new column Email, but only if it doesn't already exist. Write the SQL statement to achieve this.

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

61. What is a Composite Key?

A composite key is a combination of two or more columns in a table that uniquely identifies a row in that table. It is used when a single column is not sufficient to uniquely identify records.

DML

62. Write a query to update the Salary column in the Employees table, increasing all salaries by 10%.

```
UPDATE Employees SET Salary = Salary * 1.10;
```

63. You need to retrieve data from the Orders and Customers tables to find all orders placed by customers from a specific city. Write the query.

```
SELECT Orders.* FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID
WHERE Customers.City = 'SpecificCity';
```

Aggregate Functions

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

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

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

The RANK function is used to rank employees based on their sales performance to identify top performers. For example, if you have a Sales table with columns for EmployeeID and TotalSales, you can use RANK to assign ranks based on the total sales amount.

```
SELECT EmployeeID, SalesAmount,
RANK() OVER (ORDER BY SalesAmount DESC) AS SalesRank
FROM Sales;
```

FILTERS

66. 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;
```

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

The CASE statement can be used to create new categories based on conditions. For example:

```
SELECT ProductID, ProductName,
   CASE
      WHEN Price < 50 THEN 'Low'
      WHEN Price BETWEEN 50 AND 100 THEN 'Medium'
      ELSE 'High'
   END AS PriceCategory
FROM Products;
```

OPERATORS

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

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

Multiple Tables: Normalization

69. A denormalized database is causing performance issues. Describe the steps you would take to normalize it and improve performance till 3NF.

- **First Normal Form (1NF):** Ensure that each column contains only atomic values, and each row is unique.
- **Second Normal Form (2NF):** Remove partial dependencies, ensuring all non-key attributes are fully dependent on the primary key.
- **Third Normal Form (3NF):** Remove transitive dependencies, ensuring that non-key attributes are only dependent on the primary key.

Indexes & Constraints

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

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

Joins

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

```
SELECT Orders.OrderID, Customers.CustomerName FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

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

A full outer join is useful when you need to retrieve all records from both tables, even if there is no match. For example, to find all products and all orders, including those without corresponding data:

SELECT Products.ProductName, Orders.OrderID FROM Products

FULL OUTER JOIN Orders ON Products.ProductID = Orders.ProductID;

ALIAS

73. 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 AS o

INNER JOIN Customers AS c ON o.CustomerID = c.CustomerID

INNER JOIN Products AS p ON o.ProductID = p.ProductID;

74. Explain how column aliases can be used in a query with aggregate functions to make the results more readable.
Column aliases make query results easier to understand by providing meaningful names to the output columns. For example:

SELECT AVG(Salary) AS AverageSalary FROM Employees;

Here, AverageSalary is a column alias for the result of the AVG(Salary) function.

Joins vs Sub Queries

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

Subqueries are useful when you need to filter results based on an aggregated value. For example, to find employees who earn more than the average salary:

```
SELECT EmployeeID, Name, Salary FROM Employees
WHERE Salary > (SELECT AVG(Salary) FROM Employees);
```

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

- **Join:** Often more efficient for retrieving related data from multiple tables in a single query.
- **Subquery:** Can be less efficient, especially if used in the WHERE clause, as it may result in multiple executions of the subquery.

TYPES

77. Describe a scenario where using a DATETIME data type would be essential, and explain how you would store and retrieve this data.
A DATETIME data type is essential for recording timestamps of transactions or events. For example, in a transaction table, you might store the date and time of each transaction:

```
CREATE TABLE Transactions (
   TransactionID INT PRIMARY KEY,
   TransactionDate DATETIME,
   Amount DECIMAL(10, 2)
);
```

To retrieve transactions from a specific date range:

```
SELECT * FROM Transactions
WHERE TransactionDate BETWEEN '2024-01-01' AND '2024-12-31';
```

78. Explain how you would handle data type conversions when importing data from a CSV file with mixed data types.
You would:

- Identify the correct data types for each column.
- Use tools or scripts to convert data types during the import process.
- Validate and clean the data before importing to ensure compatibility with the database schema.

Correlation and Non-Correlation

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

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

80. Explain how non-correlated subqueries can be optimized for better performance compared to correlated subqueries.
Non-correlated subqueries are executed once and the result is used multiple times, which is often more efficient than correlated subqueries that execute for each row in the outer query. Optimizing non-correlated

subqueries involves indexing the columns used in the subquery to speed up execution.

Introduction to TSQL, Procedures, Functions, Triggers, Indices

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

```
CREATE PROCEDURE AddEmployee
    @EmployeeID INT,
    @Name VARCHAR(100),
    @Position VARCHAR(50),
    @Salary DECIMAL(10, 2)
AS
BEGIN
    INSERT INTO Employees (EmployeeID, Name, Position, Salary)
    VALUES (@EmployeeID, @Name, @Position, @Salary);
END;
```

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

```
CREATE TRIGGER CheckSalary
ON Employees
AFTER INSERT, UPDATE
AS
BEGIN
```

```
    IF EXISTS (
        SELECT 1
        FROM Employees e
        JOIN Employees m ON e.ManagerID = m.EmployeeID
        WHERE e.Salary > m.Salary
    )
    BEGIN
        RAISERROR ('Employee salary cannot exceed manager salary.', 16,
1);
        ROLLBACK;
    END
END;
```

Security and Accessibility

83. Mention any 2 common security measures to protect a SQL Server database.

- **Encryption:** Encrypt sensitive data both at rest and in transit.
- **Access Control:** Implement strict user roles and permissions to limit access to sensitive data.

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

```
-- Create a new user
CREATE LOGIN newUser WITH PASSWORD = 'password123';
CREATE USER newUser FOR LOGIN newUser;
-- Assign roles
```

```
EXEC sp_addrolemember 'db_datareader', 'newUser';

EXEC sp_addrolemember 'db_datawriter', 'newUser';
```

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

Encryption for data at rest can be implemented using **Transparent Data Encryption (TDE)**, which encrypts the entire database and log files.

To enable TDE:

```
-- Create a master key

CREATE MASTER KEY ENCRYPTION BY PASSWORD =
'YourStrongPassword';

-- Create a certificate

CREATE CERTIFICATE MyCertificate

WITH SUBJECT = 'My Database Encryption Certificate';

-- Create a database encryption key

CREATE DATABASE ENCRYPTION KEY

WITH ALGORITHM = AES_256

ENCRYPTION BY SERVER CERTIFICATE MyCertificate;

-- Enable encryption

ALTER DATABASE YourDatabase

SET ENCRYPTION ON;
```