# Lab 8 – Part 2

## 1. Trigger Vs Stored Procedure

- **Trigger:** A trigger is a special kind of stored procedure that automatically executes when certain events occur in a database (e.g., INSERT, UPDATE, DELETE).

- **Stored Procedure:** A stored procedure is a precompiled collection of SQL statements and optional control-of-flow logic that can be executed explicitly by a user or application.

### Execution Mechanism:

- **Trigger**: Automatically invoked by specific database events like INSERT, UPDATE, or DELETE.
- **Stored Procedure**: Manually invoked by the user or application using EXEC.

### Invocation:

- **Trigger**: Cannot be manually invoked, it fires automatically based on an event.
- **Stored Procedure**: Must be explicitly called by the user or application.

### Parameters:

- **Trigger**: Does not accept parameters.
- **Stored Procedure**: Can accept input parameters, allowing for dynamic execution.

### Error Handling:

- **Trigger**: Limited error handling, may cause the whole transaction to fail.
- **Stored Procedure**: Supports comprehensive error handling with TRY...CATCH blocks.

### Return Values:

- **Trigger**: Does not return values.
- **Stored Procedure**: Can return values, result sets, or output parameters.

### Use Case:

- **Trigger**: Primarily used for automatic actions like enforcing data integrity or auditing changes.
- **Stored Procedure**: Used for complex queries, data manipulation, and operations requiring user input.

### Transaction Handling:

- **Trigger**: Executes within the transaction of the operation that triggered it.
- **Stored Procedure**: Can manage its own transactions (e.g., COMMIT, ROLLBACK).

## 2. stored procedure and functions

### Execution Mechanism:

- **Stored Procedure**: Can be executed manually using EXEC.
- **Function**: Typically called within SQL expressions, such as SELECT, UPDATE, or INSERT, and returns a value.

### Invocation:

- **Stored Procedure**: Manually invoked with EXEC.
- **Function**: Called like a regular expression in SQL, often used in SELECT statements.

### Return Values:

- **Stored Procedure**: Does not always return a value. It can return result sets, output parameters, or status codes.
- **Function:** Always returns a value (scalar or table) to the calling expression.

### Parameters:

- **Stored Procedure**: Can accept input and output parameters.
- **Function:** Can accept input parameters, but cannot have output parameters.

### Side Effects:

- **Stored Procedure**: Can perform actions that modify the database (e.g., INSERT, UPDATE, DELETE).
- **Function**: Generally does not modify the database; used for calculations or returning values.

### Error Handling:

- **Stored Procedure**: Supports error handling with TRY…CATCH.
- **Function**: Does not support error handling (i.e., cannot use TRY…CATCH).

### Use Case:

- **Stored Procedure**: Used for tasks like data manipulation, creating reports, or managing business logic.
- **Function**: Used for returning values, calculations, or when you need reusable expressions in SQL queries.

## 3. DROP and DELETE statements

### Functionality:

- **DROP**: Removes a database object (such as a table, view, or index) permanently from the database.
- **DELETE**: Removes data (rows) from a table but keeps the table structure intact.

### Scope:

- **DROP**: Affects the entire database object and its schema.
- **DELETE**: Affects only the data within the table, not the table itself.

### Data Recovery:

- **DROP**: Once executed, the database object and its data cannot be recovered unless you have a backup.
- **DELETE**: Data can be recovered (if there are mechanisms like transaction logs or backups), unless it is followed by a COMMIT in a transaction.

### Transaction Behavior:

- **DROP**: Cannot be rolled back if executed outside a transaction.
- **DELETE**: Can be rolled back if part of a transaction (using BEGIN TRANSACTION, ROLLBACK).

### Performance:

- **DROP**: Faster than DELETE since it removes the entire object, including all associated data.

- **DELETE**: Slower for large tables since it removes rows one by one and logs each change.

**Use Case**:

- **DROP**: Used when you no longer need a database object or its data.
- **DELETE**: Used when you need to remove specific data from a table while preserving the structure.

## 4. SELECT and SELECT INTO statements

**Functionality**:

- **SELECT**: Retrieves data from one or more tables and displays the results in the result set.
- **SELECT INTO**: Copies data from one table and inserts it into a new table. It creates a new table with the same structure as the source table and populates it with data.

**Output**:

- **SELECT**: Displays data in the result set without modifying the database.
- **SELECT INTO**: Creates a new table (if it doesn't exist) and inserts the selected data into that new table.

**Table Creation**:

- **SELECT**: Does not create or modify tables.
- **SELECT INTO**: Creates a new table based on the result of the query and populates it with data.

**Use Case**:

- **SELECT**: Used for querying and viewing data from existing tables.
- **SELECT INTO**: Used to create a backup or duplicate of data in a new table.

**Performance**:

- **SELECT**: Generally faster as it simply retrieves and displays data.
- **SELECT INTO**: Slower because it creates a new table and inserts the data into it.

## 5. DDL, DML, DCL AND DQL

### DDL (Data Definition Language):

- **Purpose**: Defines and manages database objects like tables, indexes, and schemas.
- **Commands**: CREATE, ALTER, DROP, TRUNCATE, RENAME.
- **Example**: CREATE TABLE Students (ID INT, Name VARCHAR(50));.

### DML (Data Manipulation Language):

- **Purpose**: Manages and manipulates data within tables.
- **Commands**: INSERT, UPDATE, DELETE, MERGE.
- **Example**: INSERT INTO Students (ID, Name) VALUES (1, 'John');.

### DCL (Data Control Language):

- **Purpose**: Controls access and permissions in the database.
- **Commands**: GRANT, REVOKE.
- **Example**: GRANT SELECT ON Students TO User1;.

### DQL (Data Query Language):

- **Purpose**: Retrieves data from the database.
- **Commands**: SELECT.
- **Example**: SELECT * FROM Students;.

## Key Differences:

### Scope:

- DDL deals with structure; DML manipulates data; DCL manages permissions; DQL queries data.

### Impact:

- DDL changes are permanent; DML changes can be rolled back; DCL sets security rules; DQL just reads data.

## 6. Table Valued VS multi statement function

### **Definition**:

- **Table-Valued Function (Inline)**: A simple function that returns a table and contains a single SELECT statement.
- **Multi-Statement Table-Valued Function**: A more complex function that returns a table and allows multiple SQL statements to populate the table.

### **Structure**:

- **Table-Valued Function**: The return table is directly defined within the RETURN clause of the function.
- **Multi-Statement Function**: Uses a table variable to define the structure and populate it with data using multiple statements.

### **Performance**:

- **Table-Valued Function**: Faster due to its simplicity and direct execution.
- **Multi-Statement Function**: Slower as it involves multiple statements and can have more overhead.

### **Flexibility**:

- **Table-Valued Function**: Limited to a single query.
- **Multi-Statement Function**: Flexible, as you can perform complex logic using multiple queries.

### **Use Case**:

- **Table-Valued Function**: Best for simple, straightforward queries returning a table.
- **Multi-Statement Function**: Suitable for complex calculations or logic requiring multiple steps.

## 7. VARCHAR(50) AND VARCHAR(MAX)

### Storage Capacity:

- **VARCHAR(50)**: Can store up to 50 characters.
- **VARCHAR(MAX)**: Can store up to 2^31-1 (2,147,483,647) characters.

### Use Case:

- **VARCHAR(50)**: Used for fields with a predictable and limited length (e.g., names, short descriptions).
- **VARCHAR(MAX)**: Used for fields that can store large or variable-length text (e.g., comments, logs, or documents).

### Performance:

- **VARCHAR(50)**: More efficient in terms of storage and query performance because of its defined size.
- **VARCHAR(MAX)**: May be slower for storage and retrieval due to its potential size and storage in off-row locations for large data.

### Indexing:

- **VARCHAR(50)**: Fully supported in indexes.
- **VARCHAR(MAX)**: Indexing is supported, but only the first 900 bytes are used.

### Memory Usage:

- **VARCHAR(50)**: Occupies space based on the defined maximum length (50 bytes at most).
- **VARCHAR(MAX)**: Dynamically adjusts and may occupy more memory when storing large data

## 8. SQL and Windows Authentication

### Authentication Method:

- **SQL Authentication**: Relies on a username and password stored in SQL Server.
- **Windows Authentication**: Uses the credentials of the logged-in Windows user through Active Directory.

### Security:

- **SQL Authentication**: Passwords are stored in SQL Server, making it more vulnerable if the server is compromised.
- **Windows Authentication**: More secure, as it uses Windows security features and encrypted credentials.

### Management:

- **SQL Authentication**: Requires managing user credentials within SQL Server separately.
- **Windows Authentication**: Credentials are managed centrally in Active Directory.

### Connection:

- **SQL Authentication**: Works regardless of whether the client machine is part of the Windows domain.
- **Windows Authentication**: Requires the client to be part of the same domain or a trusted domain.

### Use Case:

- **SQL Authentication**: Preferred for non-Windows clients or scenarios where database access needs to be shared across non-domain systems.
- **Windows Authentication**: Ideal for Windows environments with Active Directory for integrated security.

### Configuration:

- **SQL Authentication**: Requires enabling mixed mode on the server to use.
- **Windows Authentication**: Enabled by default on SQL Server.

## 9. Inline function and View

### Definition:

- **Inline Function**: A user-defined function that returns a table and allows parameters to be passed for dynamic results.
- **View**: A virtual table representing the result of a predefined SQL query, without accepting parameters.

### Parameters:

- **Inline Function**: Can accept parameters to customize the output.
- **View**: Does not accept parameters; always returns the same result set.

### Execution:

- **Inline Function**: Called like a table in a query and can include additional filtering or joins in the calling query.
- **View**: Queried directly and treated as a virtual table.

### Flexibility:

- **Inline Function**: More dynamic due to the ability to accept parameters.
- **View**: Static and suitable for consistent, reusable queries.

### Performance:

- **Inline Function**: May have slight overhead due to parameter handling.
- **View**: Generally faster for straightforward, predefined queries.

### Use Case:

- **Inline Function**: Used when dynamic, parameterized results are needed.
- **View**: Used to simplify complex queries and improve query reusability.

## 10. Identity and Unique Constraint

### **Purpose**:

- **Identity**: Automatically generates unique values for a column, usually for primary keys.
- **Unique Constraint**: Ensures that all values in a column (or combination of columns) are distinct.

### **Auto-Incrementation**:

- **Identity**: Automatically increments numeric values based on a seed and increment value.
- **Unique Constraint**: Does not generate values; it only enforces uniqueness.

### **Applicability**:

- **Identity**: Used only on numeric columns.
- **Unique Constraint**: Can be applied to any data type (e.g., text, numbers).

### **Use Case**:

- **Identity**: Ideal for auto-generating primary key values.
- **Unique Constraint**: Useful for ensuring uniqueness in non-primary key columns like email or username.

### **Management**:

- **Identity**: Values are system-generated and cannot be directly inserted or updated.
- **Unique Constraint**: Values must be provided manually and adhere to the uniqueness rule.