**1. Stored Procedure**

* **Definition**: A stored procedure is a **precompiled block of SQL code** stored in the database, similar to a function.
* **Purpose**: To execute specific tasks (insert, update, delete, select…) in a reusable way.
* **Execution**: It is executed **manually** when the developer or application explicitly calls it.
* **Advantages**:
  + Code reusability (avoid writing the same SQL multiple times).
  + Better performance (compiled and cached).
  + Supports programming logic (IF, WHILE, LOOP).

**2. Trigger**

* **Definition**: A trigger is a **special kind of stored procedure** that automatically executes when a specific event occurs on a table.
* **Purpose**: To enforce rules, maintain data integrity, or log/audit changes.
* **Execution**: It is executed **automatically** (not called manually) after/before events like INSERT, UPDATE, or DELETE.
* **Advantages**:
  + Ensures data integrity automatically.
  + Can log or audit changes.
  + Reduces human error since execution is automatic.

| **Aspect** | **Stored Procedure** | **Trigger** |
| --- | --- | --- |
| **Execution** | Called manually (EXEC ProcedureName) | Fires automatically on events (INSERT, UPDATE, DELETE) |
| **Purpose** | Reusable logic for tasks | Enforce rules, integrity, or audit automatically |
| **Control** | Full control when to run | No control – runs automatically on event |
| **Performance** | Optimized for repeated manual execution | May affect performance if too many triggers fire frequently |

**1. Stored Procedure**

* **Definition**: A precompiled collection of SQL statements stored in the database.
* **Execution**: Called using EXEC or EXECUTE.
* **Return type**: May return **multiple result sets**, or nothing at all (but cannot be used directly in SELECT).
* **Parameters**: Can accept **input and output parameters**.
* **Use cases**:
  + Complex business logic.
  + Multiple queries (insert/update/delete/select).
  + Batch operations.

**2. Function**

* **Definition**: A routine that always returns a **single value** (scalar function) or a **table** (table-valued function).
* **Execution**: Can be called inside a SELECT, WHERE, or other SQL statements.
* **Return type**: Must return **a value (scalar or table)**.
* **Parameters**: Accepts **only input parameters** (no output parameters).
* **Use cases**:
  + Reusable calculations.
  + Transforming values.
  + Returning a small dataset that can be queried like a table.

| **Aspect** | **Stored Procedure** | **Function** |
| --- | --- | --- |
| **Return Value** | Can return 0, 1, or many result sets (not mandatory). | Must return a value (scalar or table). |
| **Use in Queries** | Cannot be used inside a SELECT. | Can be used inside a SELECT, WHERE, etc. |
| **Parameters** | Supports **input & output** parameters. | Supports only **input** parameters. |
| **Transactions** | Can use BEGIN TRANSACTION, ROLLBACK, COMMIT. | Cannot manage transactions. |
| **Purpose** | Complex logic, multiple queries, batch operations. | Reusable computations or table-returning utilities. |

| **Feature** | **DELETE** | **DROP** |
| --- | --- | --- |
| **Definition** | Used to remove rows (records) from a table. | Used to remove an entire database object (table, view, database, etc.). |
| **Scope** | Removes data **inside** the table, but the table structure remains. | Removes the **whole object** (e.g., the table itself and its data). |
| **Rollback (Undo)** | Can be rolled back if used inside a transaction. | Cannot be rolled back (once dropped, the object is gone unless restored from backup). |
| **Usage** | DELETE FROM table\_name WHERE condition; | DROP TABLE table\_name; |
| **Effect on Structure** | Keeps the table structure (you can still insert new data). | Deletes the structure completely (table no longer exists). |
| **Performance** | Slower (especially for large data) since it removes rows one by one. | Faster because it removes the entire object directly. |

| **Feature** | | **SELECT** | | **SELECT INTO** | |
| --- | --- | --- | --- | --- | --- |
| **Definition** | | Used to retrieve (read) data from one or more tables. | | Used to create a **new table** and copy data into it from another table/query. | |
| **Table Creation** | | Does **not** create a new table, only shows data. | | Automatically creates a **new table** with the selected data. | |
| **Existing Table** | | Works on existing tables. | | Creates a new table, so it cannot be used if the table already exists. | |
| **Usage** | | SELECT column1, column2 FROM table\_name WHERE condition; | | SELECT column1, column2 INTO new\_table FROM existing\_table WHERE condition; | |
| **Modification** | | Does not affect schema. | | Creates schema + copies data. | |
| **Purpose** | | For **reading data**. | | For **backing up data** or creating a copy/subset of data. | |
| **Category** | **Full Form** | | **Purpose** | | **Example Commands** |
| **DDL** | Data Definition Language | | Defines database structure | | CREATE, ALTER, DROP |
| **DML** | Data Manipulation Language | | Manipulates data | | INSERT, UPDATE, DELETE |
| **DCL** | Data Control Language | | Controls access/permissions | | GRANT, REVOKE |
| **DQL** | Data Query Language | | Fetches data | | SELECT |

| **Feature** | **Inline Table-Valued Function (TVF)** | **Multi-Statement Table-Valued Function (MSTVF)** |
| --- | --- | --- |
| **Definition** | Returns a table from **one query** | Returns a table using **multiple statements** |
| **Complexity** | Simple | Complex (allows logic, loops, conditions) |
| **Performance** | Faster (optimized like a view) | Slower (uses table variable internally) |
| **Use Case** | When one query is enough | When you need multiple steps to build data |

| **Feature** | **VARCHAR(50)** | **VARCHAR(MAX)** |
| --- | --- | --- |
| **Max Length** | 50 characters | ~2 billion characters |
| **Performance** | Faster (optimized) | Slower for large data |
| **Storage** | Stored in-row | Stored in-row if ≤ 8KB, else out-of-row |
| **Use Case** | Short/medium text (name, email) | Large text (documents, logs, comments) |

| **Feature** | **SQL Authentication** | **Windows Authentication** |
| --- | --- | --- |
| **Credentials** | SQL Server username & password | Windows (AD) account credentials |
| **Security** | Less secure (passwords managed in SQL) | More secure (Kerberos, centralized control) |
| **Management** | Managed inside SQL Server | Managed by Windows/AD |
| **Use Case** | Non-domain users, apps needing SQL logins | Enterprise, domain users, higher security |

| **Feature** | **Inline Function (TVF)** | **View** |
| --- | --- | --- |
| **Definition** | Returns a table from **one query** (like param view) | Virtual table from a query |
| **Parameters** | Supports parameters | No parameters |
| **Flexibility** | More flexible (dynamic filtering) | Fixed, less flexible |
| **Performance** | Optimized like a parameterized query | Optimized but static |
| **Use Case** | Reusable query **with parameters** | Reusable query **without params** |

| **Feature** | **IDENTITY** | **UNIQUE Constraint** |
| --- | --- | --- |
| **Definition** | Auto-generates sequential numbers for a column (e.g., 1, 2, 3…). | Ensures all values in a column (or combination) are **unique**. |
| **Purpose** | Mainly used for **primary keys** (auto-increment IDs). | Used to **prevent duplicate values** in a column/columns. |
| **Automatic?** | Yes, SQL auto-generates values. | No, user must insert values manually. |
| **Nulls** | Not allowed (identity column cannot be NULL). | Allows **one NULL** (per column with UNIQUE constraint). |
| **Scope** | Only one IDENTITY per table. | Can define multiple UNIQUE constraints in one table. |
| **Example** | ID INT IDENTITY(1,1) → generates 1,2,3… automatically. | CONSTRAINT UQ\_Email UNIQUE (Email) → no duplicate emails allowed. |