In SQL Server, backups are essential for protecting data and recovering from failures. There are three primary types of backups, each serving different purposes:

**1. Full Backup**

* **What it does**: Backs up the **entire database** — all data, objects, system tables, and transaction logs at the time of the backup.
* **Use case**: Base backup for any restore strategy.
* **Size**: Largest among all types.
* **Restore**: You need at least the full backup (plus later differential/log backups, if any) to restore the database.

Think of it as a complete snapshot of your database at a point in time.

**2. Differential Backup**

* **What it does**: Backs up **only the data that has changed** since the **last full backup**.
* **Use case**: Speeds up backup time and saves space between full backups.
* **Size**: Grows over time until the next full backup is taken.
* **Restore**: Requires the **last full backup** + the **latest differential backup**.

Think of it like a "delta" — changes since the last full backup.

**3. Transaction Log Backup**

* **What it does**: Backs up the **transaction log** — a record of all changes made to the database since the last transaction log backup.
* **Use case**: Supports **point-in-time recovery**, minimizes data loss.
* **Size**: Usually small and frequent.
* **Restore**: Requires the **last full backup**, **optional differential backup**, and **all transaction log backups** up to the point in time.

Think of it as a running log of everything that has happened in the database.

P**ermissions** control what users and roles can do within the database environment. Understanding permissions and the differences between **GRANT**, **DENY**, and how they work at different **levels** is key to proper security management.

**What is a Permission?**

A **permission** allows or restricts a user or role to perform certain actions on database objects.

Examples:

* SELECT on a table → allows reading data.
* INSERT on a table → allows adding data.
* EXECUTE on a stored procedure → allows running the procedure.

**GRANT vs DENY (and REVOKE)**

| **Command** | **Effect** | **Overrides?** | **Use case** |
| --- | --- | --- | --- |
| **GRANT** | Gives permission to perform an action. | Can be overridden by DENY | To allow access to a resource. |
| **DENY** | Explicitly prevents the action, even if granted elsewhere. | Overrides GRANT | To block access regardless of other permissions. |
| **REVOKE** | Removes a previously GRANTED or DENIED permission. | N/A | To reset permission (removes explicit permission). |

**Permission Levels**

Permissions can be applied at **different levels** of SQL Server’s hierarchy:

**1. Server Level**

* Applies to **logins** and **server-wide objects**.
* Examples:
  + GRANT ALTER ANY LOGIN TO User1
  + GRANT VIEW SERVER STATE TO User1

**2. Database Level**

* Applies to a specific **database** and its objects.
* Examples:
  + GRANT CONNECT TO User1 (connect to the database)
  + GRANT CREATE TABLE TO User1

**3. Schema Level**

* Applies to a **schema**, which is a container for objects.
* Example:
  + GRANT SELECT ON SCHEMA::Sales TO User1

**4. Object Level**

* Applies to **tables, views, stored procedures, functions**, etc.
* Examples:
  + GRANT SELECT ON dbo.Employees TO User1
  + DENY DELETE ON dbo.Customers TO User1

**5. Column Level (for SELECT, INSERT, UPDATE)**

* Grants or denies permission on specific columns in a table.
* Example:
  + GRANT SELECT (FirstName, LastName) ON dbo.Employees TO User1

**Permission Resolution Rules**

* **DENY overrides GRANT**.
* If a user is part of multiple roles, and one role has DENY, it blocks the permission even if another role has GRANT.
* **Permissions are inherited** from parent objects unless explicitly overridden.

**What is SQL Server Profiler?**

**SQL Server Profiler** is a **graphical tool** provided by Microsoft that allows you to **monitor, capture, and analyze SQL Server events in real time**. It is used to track what's happening inside SQL Server — like queries being executed, performance issues, or user activity.

It is part of **SQL Server Management Studio (SSMS)**.

**What You Can Do With SQL Profiler**

* **Capture and analyze T-SQL statements**
* **Monitor slow-running queries**
* **Audit user activity and security events**
* **Trace deadlocks, locks, and blocking**
* **Diagnose performance bottlenecks**
* **Debug stored procedures and triggers**
* **Capture login/logout events**

**When to Use SQL Profiler**

| **Use Case** | **Why Use It** |
| --- | --- |
| Slow performance | Identify which queries are taking too long |
| Frequent blocking/deadlocks | See which sessions are causing locks |
| Debugging | Watch exactly what SQL is being sent/executed |
| Tuning workload | Export trace data to the Database Engine Tuning Advisor |
| Security auditing | See who accessed what and when |
| Testing stored procedures/triggers | Monitor internal calls during test runs |

**Important Notes**

* **Heavy resource usage**: Profiler can slow down your server if run on a production system.
* **Not recommended** for long-term monitoring on production environments.
* Better alternatives for ongoing performance monitoring:
  + **Extended Events** (lighter and more efficient)
  + **SQL Server Audit**
  + **Performance Monitor**
  + **DMVs (Dynamic Management Views)**

**What is a Trigger?**

A **trigger** is:

* A predefined, automatic action
* Bound to an event like INSERT, UPDATE, or DELETE (on a table or view), or LOGON, DDL operations (at the server or database level)

**Why Use a Trigger?**

Triggers are used for:

* **Enforcing complex business rules**
  + For example: Automatically calculate tax when a new invoice is inserted
* **Auditing or logging changes**
  + Track who made changes and when
* **Validating data before allowing changes**
  + Prevent unwanted updates or deletions
* **Synchronizing tables**
  + Automatically update a related table
* **Restricting or blocking certain actions**
  + Prevent deletion of certain rows

**Levels of Triggers**

| **Level** | **Trigger Type** | **Fires On** |
| --- | --- | --- |
| **Table/View** | DML Triggers (AFTER / INSTEAD OF) | INSERT, UPDATE, DELETE operations |
| **Database** | DDL Triggers | CREATE, ALTER, DROP, GRANT, etc. |
| **Server** | Logon Triggers | LOGON event |

**Types of Triggers**

1. **AFTER Trigger**
   * Executes *after* the DML operation completes.
   * Commonly used to enforce referential integrity or for auditing.
2. **INSTEAD OF Trigger**
   * Replaces the DML action (does not actually perform the insert/update/delete unless written inside the trigger).
3. **DDL Trigger**
   * Responds to changes in schema like creating or altering tables.
4. **Logon Trigger**
   * Fires when a user logs in to the SQL Server instance.

**How Triggers Differ from Stored Procedures**

| **Aspect** | **Trigger** | **Stored Procedure** |
| --- | --- | --- |
| **Execution** | Fires automatically on specific events | Must be called manually |
| **Event-driven** | Yes | No |
| **Bound to** | Tables, views, server or database events | Not bound to any event |
| **Used for** | Auditing, enforcing rules, blocking actions | Business logic, data processing, reporting |
| **Direct user execution** | No | Yes |
| **Returns values** | No (not directly) | Yes (can use output parameters or result sets) |

**When Not to Use Triggers**

* When performance is critical — triggers can add overhead
* When the logic can be handled with constraints or application logic
* When they make the system hard to debug (can execute without visible control)