# 1. What Are SQL Security Levels?

Server-Level Login:

* A server-level login is an identity created at the SQL Server instance level. It is used to authenticate users before they can access any database on the server. Example: CREATE LOGIN hr\_login WITH PASSWORD = 'HrStrongP@ss1';

Database-Level User:

* A database user is created inside a specific database and is mapped to a server login. The user can only access resources within that database.

Schema-Level Permissions:

* A schema is a container for database objects like tables, views, and stored procedures. Schema-level permissions allow access to all objects within the schema.

Object-Level Permissions (Brief):

* These are fine-grained permissions granted directly to individual objects like a specific table.

# 2. Benefits of Applying Security Levels

* Data Protection - Restricts access to sensitive data such as salaries or medical records.
* 🔒 Access Control - Limits user capabilities (e.g., read-only, write access).
* Operational Safety - Prevents accidental modifications or deletions of data.
* 📋 Audit & Compliance - Helps meet regulatory and internal security policies.
* 👥 Role-Based Access - Supports assigning permissions by role (e.g., HR, Sales, Developers).

# 3. Real-World Risks Without Proper Security

* 🔓 Full Access to All Users - Anyone can see or delete any data, even sensitive HR or finance records.
* 🧪 Developers Modify Production - Mistakenly delete or overwrite data in live systems.
* Interns Access Private Info - HR, payroll, or confidential data can leak unintentionally.

# 4. Summary of Practical Task

Objective: Restrict each department to view and work only with their own schema.

1. Steps Completed:

* Created logins: hr\_login, sales\_login
* Mapped logins to DB users: hr\_user, sales\_user
* Created schemas: HR, Sales
* Created tables: HR.Employees, Sales.Customers
* Granted each user access only to their schema
* Denied cross-schema access using DENY

# 5. Security Incident “The Overpowered Developer”

* Accidental Data Deletion - Adil mistakenly ran DELETE FROM Employees on production with no backup.
* Salary Data Leak - Adil exported salary data and sent it externally by mistake.
* Unauthorized Role Creation - Adil created a SQL login for a junior dev without DBA approval.
* Schema Confusion - HR-related tables were created in dbo schema instead of HR schema.

# 6. Root Causes

* No Dev/Prod Separation - Adil worked directly in production.
* Full Permissions - Developers had unrestricted access.
* No Schema Boundaries - Tables were not isolated per department.
* No Role-Based Security - No differentiation between admin, dev, or limited access roles.

# 7. Suggested Solutions

* Schema-Level Permissions - Ensure users only access their department’s data.
* Role-Based Access Control - Use roles like ReadOnly\_Dev, HR\_Admin, etc.
* Use Views - Hide sensitive fields from lower-permission users.
* Environment Isolation - Separate development, testing, and production environments.
* Restrict Login Creation - Only DBAs should manage server-level logins.
* Enable Audit Logs - Monitor actions and identify breaches or policy violations.

# 8. Lessons Learned

* Developers should never have unrestricted access to production.
* Apply least privilege: only give users what they need to do their job.
* Always separate roles (e.g., dev, DBA, HR).
* Never assume trust—enforce security with roles and permissions.