**1. What are SQL Security Levels?**

* **Server-level login**: Grants access to the SQL Server instance.
* **Database-level user**: Tied to the login, grants access to a specific database.
* **Schema-level permissions**: Controls access to all objects within a schema.
* **Object-level permissions** *(brief)*: Permissions applied on specific tables/views.

**2. Benefits of Applying Security Levels**

* 🛡 Restrict sensitive info like **salaries** and **client data**.
* Prevent **unauthorized edits** or data loss.
* 🤖 Reduce **accidental actions** from junior staff/interns.
* 📋 Satisfy **audit & compliance** (e.g., GDPR, SOX).

**3. Real-World Risks Without Security**

* **Full access = disaster**: Developers might accidentally delete or expose data.
* **Production tampering**: Devs may push test code or changes in live databases.
* **Interns/unauthorized access**: Non-HR staff seeing salary data is a **compliance violation**.

**4. Task Summary**

* Created two logins: hr\_login, sales\_login
* Created users, schemas, and tables
* Granted schema-level permissions — each role **only accesses its own schema**
* **Real-world alignment**:
  + Similar to how HR shouldn't see Sales data and vice versa
  + Promotes **data isolation & clean access control**

**🔍 Security Risk Report: “The Overpowered Developer”**

**1. Summary of the Problems**

* **Data Deletion**: Dev deleted production data mistakenly
* 📤 **Salary Leak**: Exported sensitive salary data and sent externally
* 🔐 **Unauthorized Access**: Created unauthorized login
* 🏗 **Wrong Schema Use**: New tables in incorrect schema

**2. Root Causes**

* No environment separation (dev vs prod)
* Full access to developers (no least privilege)
* No schema-level restrictions
* No role-based access control
* No audit/log controls on login creation

**3. Suggested Solutions**

|  |  |
| --- | --- |
| **Problem** | **Solution** |
| Full Access | Grant only **minimum privileges** |
| Schema Confusion | Use **schema-level permissions** |
| Unauthorized Role Creation | Restrict login creation to DBAs |
| Salary Leak | Use **views** to mask sensitive data |
| Data Loss | Enforce **daily backups** and limit DELETE rights |
| Prod Access | Separate **dev/test/prod environments** strictly |

**4. Lessons Learned**

* Developers need **read-only** access, not full DDL/DML
* DBAs manage **login/user/schema permissions**
* Always follow the **Principle of Least Privilege (PoLP)**
* Prevent mistakes by **automating backups**, **auditing**, and **restricting roles**