Task: Enforcing Schema-Level Access in a Company Database

📘 Scenario

You are the database administrator of a system that contains two main departments:

- HR (Human Resources)
- Sales

Your job is to restrict access so that each department only views and works with its own data.

Objective

1. Create SQL logins and map them to users inside the database.
2. Create two schemas: HR and Sales.
3. Create a few sample tables inside each schema.
4. Assign **schema-level** permissions so:
   - HR users cannot access Sales data.
   - Sales users cannot access HR data.

Task Output Checklist

1. Take screenshots of:
   - Login creation
   - User creation
   - Schema permissions
   - Query results showing access works only for their assigned schema
2. Try to:
   - Connect as hr_login and **access HR.Employees** (✅ should work)
   - Try to access Sales.Customers (❌ should be denied)

3. Write a short explanation:
    - Why schema-level security is better than table-by-table permissions
    - How this setup supports **data segregation** in real-world companies

📝 Reflection Report Instructions

📄 **Title:** ***Understanding SQL Security Levels and Real-World Risks***

✅ **Your Report Should Include:**

1. What are SQL Security Levels?

Explain:

- Server-level login
- Database-level user
- Schema-level permissions
- Object-level permissions (mention only briefly)

2. Benefits of Applying Security Levels

Examples:

- Restrict sensitive data (e.g., salaries, finance)
- Prevent unauthorized changes
- Reduce human error
- Meet compliance/audit requirements

3. Real-World Risks Without Security

Explain what might happen if:

- Everyone has full access
- Developers modify production data
- Interns access HR data

4. Your Task Summary

Explain:

- How you created logins, users, and schemas

- How schema permission limited access
- How this applies to real companies

🚨 Security Scenario: When Access Goes Wrong

Scenario: "The Overpowered Developer"

You’re part of a company building an internal **Payroll Management System**. During development, a database developer named **Adil** was given **full control** on the production database to “speed up” testing and updates. However, the following problems occurred:

What Went Wrong

1. **Accidental Data Deletion**
    - Adil ran a DELETE FROM Employees command thinking he was connected to the test database.
    - No backup was taken before running the query.
2. **Salary Data Leaked**
    - Adil created a report for testing that included all employee salaries.
    - He shared the exported Excel file with an external UI developer by mistake.
3. **Unauthorized Role Creation**
    - To “help,” Adil created a new SQL login for a junior developer without informing the DB admin.
    - The junior dev used that login to explore the entire database, including sensitive HR data.
4. **Schema Confusion**
    - Adil created new tables inside the wrong schema (dbo instead of HR) which caused permission issues for HR team users.

Trainee Reflection Task: Security Analysis Report

Your Job: Analyze the above scenario and write a Security Risk Report with the following points:

✅ Report Sections

1. Summary of the Problems

List and describe what went wrong (based on the points above).

2. Root Causes

Identify the security flaws:

- No separation between development and production
- Full access given to developers
- No schema-level restrictions
- Lack of role-based permission control

3. Suggested Solutions

Explain how these issues could have been avoided using:

- Schema-level permissions
- Separation of roles (e.g., read-only, data entry)
- Use of views to hide sensitive columns
- Audit logs or restricted role creation
- Environment separation (dev vs prod)

4. Lessons Learned

- What should developers have access to?
- What should be restricted to DBAs or admins?
- Why is "minimum privilege" important?

Bonus Activity (Optional)

simulate:

- Creating a role like ReadOnly_Dev and granting only SELECT on a schema.
- Trying to run an INSERT or DELETE command using that limited role to observe permission denial.