Database Administration Security

Cybersecurity Department

Theoretical Lecture 4: SQL Server Security Architecture

Halal Abdulrahman Ahmed

Agenda

- SQL Server Security Architecture - Five Layers Overview
- Layer 1: Windows Domain & Authentication
- Layer 2: SQL Server Instance & Logins
- Layer 3: Database Users & Roles
- Layer 4: Schema-Level Security
- Layer 5: Object-Level Permissions
- GRANT, DENY & REVOKE

Learning Outcomes

By the end of the lecture, students should be able to:

- **Understand** core database security concepts
- **Analyze** security architectures
- **Differentiate** between authentication and authorization
- **Apply** security design principles

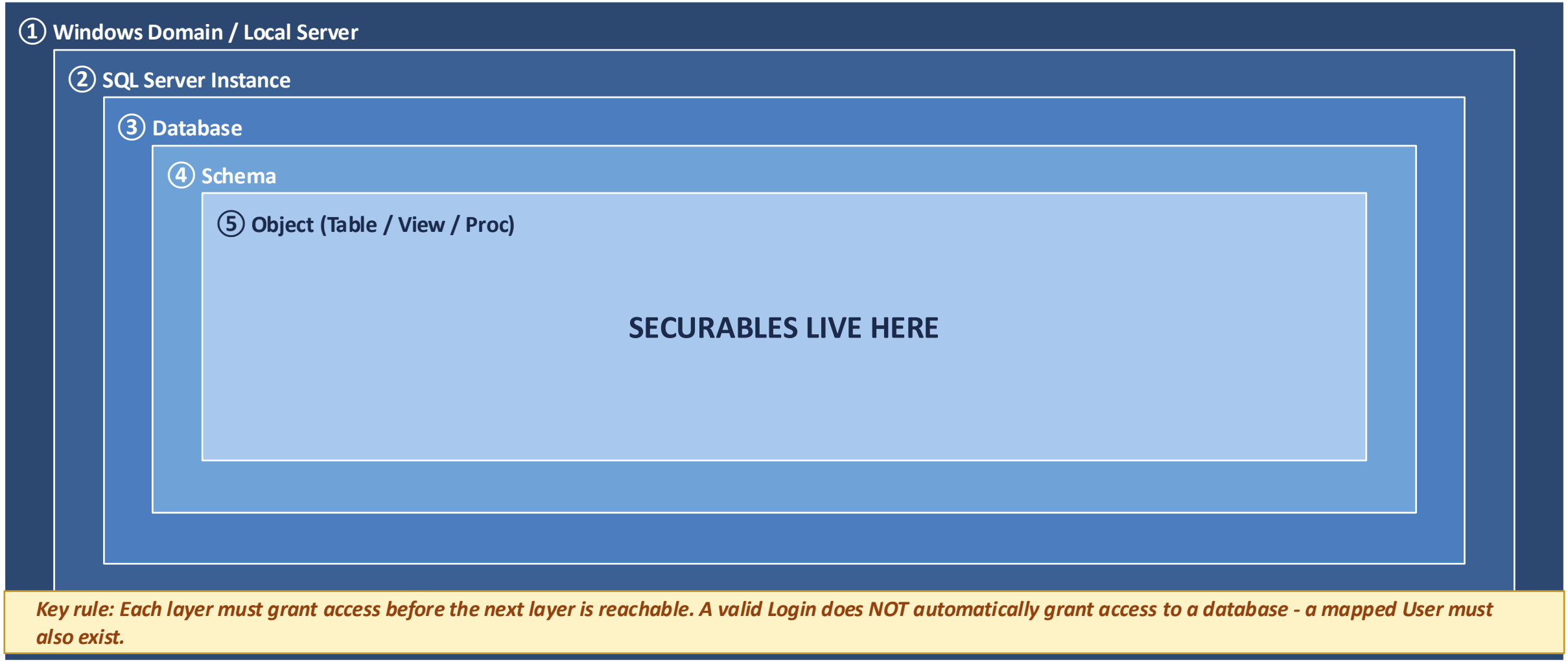
SQL Server Security Architecture - Overview

- SQL Server security is organized as five nested layers, like Russian dolls.

- The five layers from **outside** to **inside** are: Windows Domain, SQL Server Instance, Databases, Schemas, and Objects. Inside each layer there are two key concepts; **Principals** which are who is trying to get in, and **Securables** which are what they are trying to access.

SQL Server Security Architecture - Five Layers

SQL Server security is organised as nested layers. A principal must pass through every outer layer before reaching the inner ones. This design enforces Defence-in-Depth.



Key rule: Each layer must grant access before the next layer is reachable. A valid Login does NOT automatically grant access to a database - a mapped User must also exist.

Architecture Layer 1: Windows Domain

- The outermost layer is the operating system itself. Before SQL Server does anything, Windows checks whether the person connecting has a valid Windows account.
- This layer handles Windows accounts, individual users like DOMAIN\Sara and groups like DOMAIN\DBAdmins. These accounts are managed in Active Directory which is Microsoft's system for managing user accounts across an organization.
- On a standalone computer that is not connected to a domain, local Windows accounts are used instead. The important point is that SQL Server can completely trust Windows to handle authentication at this level, it does not need to verify the password itself.

Architecture Layer 1: Windows Domain (cont.)

(1) Windows Domain / Local Server Layer

Example: A company employee (COMPANY\UserA) → UserA logs into Windows. If SQL Server allows **Windows Authentication**, UserA can access SQL Server without entering another password. → This is called "**Windows Authentication Mode**"

Advantages:

- Centralized identity management
- Stronger security
- No password stored inside SQL Server

Architecture Layer 1: Windows Domain (cont.)

- **Principal: An entity that can request access to SQL Server resources.**
  - Examples: individual user, Windows group, SQL login, database user, role
- Principals are authenticated and authorized.
- **Three Scopes of Principals**
  - Windows-level (local/domain login or group)
  - SQL Server-level (SQL login or fixed server role)
  - Database-level (user or fixed database role)
- **Securable: A resource to which access is controlled.**
  - Examples: server, database, table, column, schema, endpoint.
- Securables have a hierarchy: securing a parent often grants access to children (but not always).

SQL Server Security Architecture – Summary

Principals vs Securables

Principals (Who)

Examples:

- Logins
- Users
- Roles
- Application accounts

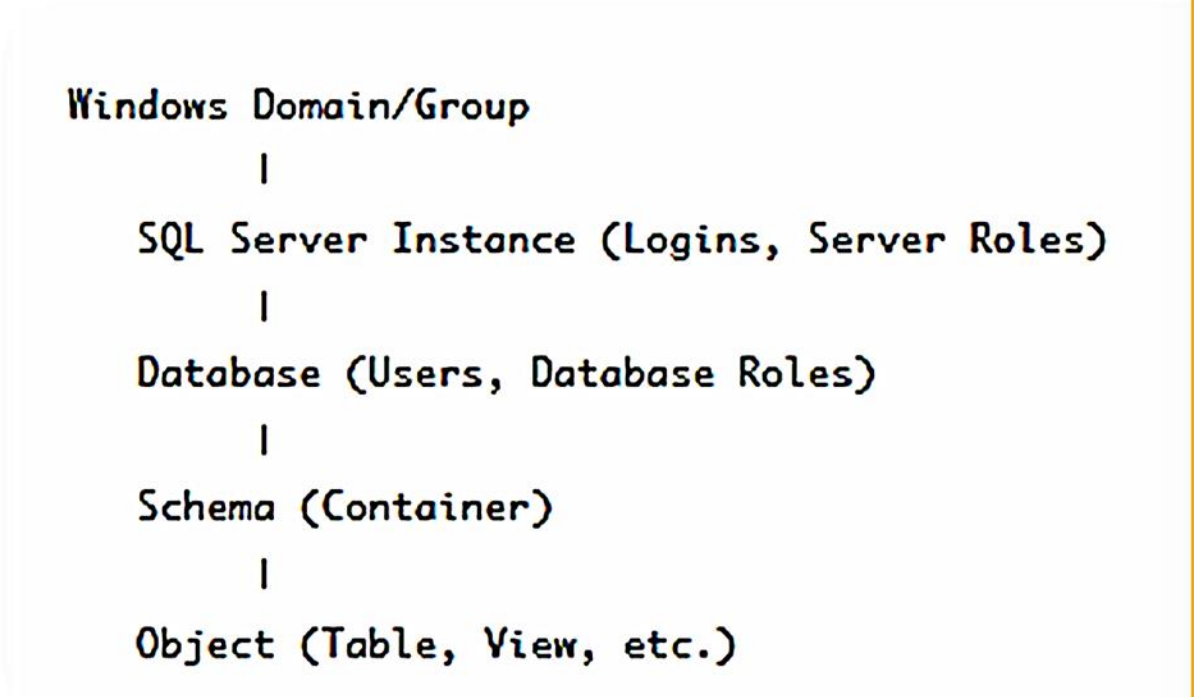
- Ex: MinaUser, db_admin, sales_manager

Securables (What)

Things being protected:

- Server
- Database
- Schema
- Table
- View
- Procedure
- Ex: SalesDB, sales.Customers, sales.Orders

Visualization of SQL Server Security Layers



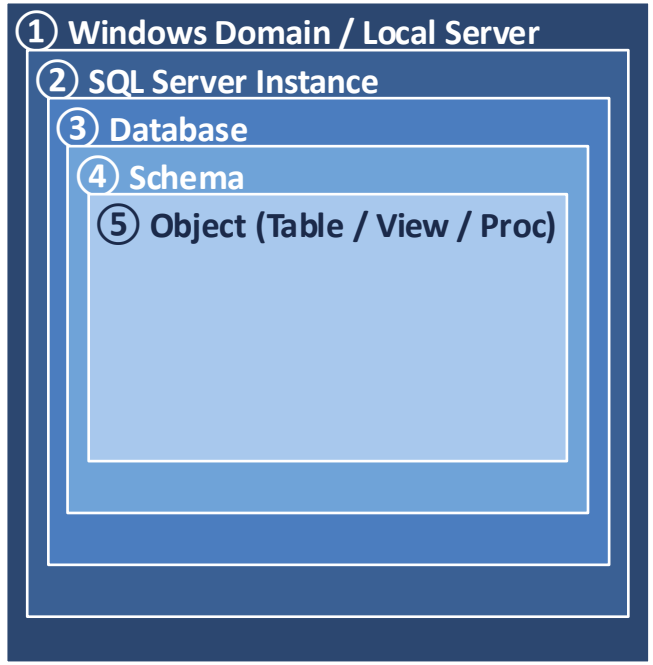
Each layer checks **identity and permissions.**

SQL Server Security Concepts

Authentication vs. Authorization: *The Hotel Lobby Analogy*

How Authentication Works for Logins

- **How Authentication Works for Logins:** Two methods (Book p. 378):
  - Windows Authentication: Windows validates identity → SQL Server trusts it
  - SQL Server Authentication: SQL Server validates username + password itself
- **Login vs User**
  - Login = server-level principal (door to the building)
  - User = database-level principal (key to a specific room)
  - One login can map to many users (one per database)



SQL Server Security Concepts

Authentication vs. Authorization: The Hotel Lobby Analogy

The Hotel Lobby: Represents Login – only allows you to enter the hotel's main door.

Room Key: Represents the User – which opens only your private room (the database).

Hotel Rules: Represent Permissions – you're allowed to stay in the room, but not to paint the walls or change the furniture!

SQL Server Security Concepts

Two Authentication Modes in SQL Server:

- **Windows Authentication Mode:** Only Windows-authenticated logins allowed.
- **Mixed Mode:** Windows authentication + SQL Server authentication.

Recommendation: Use Windows authentication whenever possible (more secure, integrated with domain policies).

SQL Server Security Concepts

- **Permission:** An action that a principal can perform on a securable.
  - **Examples:** SELECT, INSERT, UPDATE, DELETE, EXECUTE, CONTROL.
- **Permissions** can be in one of three states:
  - GRANT means the permission is given.
  - DENY means the permission is explicitly blocked even if another role grants it.
  - REVOKE means a previously given or blocked permission is removed.

These three states are the commands you will use in the lab.

SQL Server Security Concepts

Principals at the Server Level - Logins

- **sa:** System administrator (built-in SQL login).
- **BUILTIN\Administrators:** Windows Administrators group.
- **NT AUTHORITY\SYSTEM:** Windows system account.
- **Custom SQL logins:** e.g., AppUser, BackupAdmin.

SQL Server Security Concepts

Principals at the Server Level – Logins

- **Querying Server-Level Principals**
  - Use system catalog view **sys.server_principals.**

```
SELECT name, principal_id, type_desc
FROM sys.server_principals
WHERE type_desc IN ('SQL_LOGIN', 'WINDOWS_LOGIN', 'WINDOWS_GROUP');
```

SQL Server Security Architecture – Overview

(2) SQL Server Instance Layer

- This layer represents the **entire SQL Server installation**.
- Example instance: SERVER01\SQL2019
- At this level we define **Logins**.
  - Login examples:

```
CREATE LOGIN UserA_login
WITH PASSWORD = 'StrongPassword123';
```

  - Securables: At the instance level we protect (DatabasesServer, configurations, Server-level permissions)

```
GRANT VIEW SERVER STATE TO UserA_login;
```

SQL Server Security Architecture – Overview

(2) SQL Server Instance Layer

- **Example Scenario:** A developer needs to connect to SQL Server.
- Steps:
  - SQL Server checks **Login**
  - If the login is valid → access to instance is granted
- But this **does NOT mean the user can access databases yet**.

SQL Server Security Architecture – Overview

(3) Database Layer

- Inside an instance there are multiple **databases (e.g.,** SalesDB, HRDB, FinanceDB). Each database has **its own users and permissions**.
- Database users
  - CREATE USER UserA FOR LOGIN UserA_login;
    - Login → Instance access
    - User → Database access
- Database-level securables include: tables, views, procedures, and schemas
- Grant Helal access to a database:
  - USE SalesDB
    CREATE USER UserA FOR LOGIN UserA_login
  - (UserA can enter **SalesDB,** But he still cannot access tables unless permissions are given)

SQL Server Security Architecture – Overview

(3) Database Layer

- **Principals at the Database Level – Users**
  - User: A database-level principal that maps to a login (or is contained).
  - A login can be mapped to one user per database.
  - Users own schemas and are granted permissions within the database.
- **Default Database Users:** When a database is created, it includes:
  - **dbo:** Database owner, maps to the login that created the database or any sysadmin.
  - **guest:** Allows logins without a user mapping to connect (disabled by default).
  - **INFORMATION_SCHEMA, sys:** For system metadata views.

SQL Server Security Architecture – Overview

(3) Database Layer

- **Querying Database-Level Principals**
  - Use **sys.database_principals** in the context of the target database.

```
USE YourDatabase;
SELECT name, principal_id, type_desc, default_schema_name
FROM sys.database_principals;
```

SQL Server Security Architecture – Overview

(3) Database Layer

- Principals at the Database Level – Roles
  - **Role**: A group of principals that share a common set of permissions.
  - **Fixed server roles**: Predefined at instance level (e.g., sysadmin, securityadmin).
  - **Fixed database roles**: Predefined at database level (e.g., db_owner, db_datareader).
  - **User-defined roles**: Custom roles created by administrators.

SQL Server Security Architecture – Overview

(3) Database Layer

- Principals at the Database Level – **Fixed Server Roles (Overview)**

Role	Description
sysadmin	Can perform any activity in the instance
securityadmin	Manage logins and server permissions
serveradmin	Configure server-wide settings
setupadmin	Manage linked servers
processadmin	Manage processes

SQL Server Security Architecture – Overview

(3) Database Layer

- Principals at the Database Level – **Fixed Server Roles (Overview)**

Role	Description
diskadmin	Manage backup devices
dbcreator	Create and alter databases
bulkadmin	Perform BULK INSERT operations
public	All logins belong to public

SQL Server Security Architecture – Overview

(3) Database Layer

- Principals at the Database Level – **Fixed Database Roles (Overview)**

Role	Description
db_owner	Full control over the database
db_accessadmin	Manage user access
db_securityadmin	Manage permissions and roles
db_ddladmin	Execute DDL statements
db_backupoperator	Backup the database

SQL Server Security Architecture – Overview

(3) Database Layer

- Principals at the Database Level – **Fixed Database Roles (Overview)**

Role	Description
db_datareader	Read all data
db_datawriter	Modify all data
db_denydatareader	Cannot read data
db_denydatawriter	Cannot modify data

SQL Server Security Architecture – Overview

(3) Database Layer

- Principals at the Database Level – **Custom Database Roles**
  - Created using CREATE ROLE.
  - Grant specific permissions to the role, then add users.
  - Simplifies permission management for groups with similar needs.

```
USE UniversityDB;
CREATE ROLE InstructorRole;
GRANT SELECT, UPDATE ON dbo.Courses TO InstructorRole;
EXEC sp_addrolemember 'InstructorRole', 'domain\instructor1';
```

SQL Server Security Architecture – Overview

(4) Schema Layer

- Schema – A Container for Objects
- Schema: A namespace that groups database objects.
- Objects within a schema have unique names.
- Schemas are owned by principals (users or roles).
- Separates object ownership from the object itself.

SQL Server Security Architecture – Overview

(4) Schema Layer

- **Benefits of Schemas**
  - Easier management of permissions (grant on schema vs individual objects).
  - Ownership transfer without renaming objects.
  - Multiple schemas allow logical separation (e.g., Sales, HR).
- **Default Schemas**
  - Each user has a default schema, **Common default: dbo.**
  - If no schema is specified when creating an object, it is placed in the user's default schema

SQL Server Security Architecture – Overview

- **Permissions – GRANT, DENY, REVOKE**
  - **GRANT:** Gives a permission to a principal.
  - **DENY:** Explicitly denies a permission; overrides **GRANT**.
  - **REVOKE:** Removes a previously granted or denied permission.

```
-- Grant SELECT on a table
GRANT SELECT ON dbo.Students TO StudentUser;

-- Deny INSERT on a table
DENY INSERT ON dbo.Students TO StudentUser;

-- Revoke a previously granted SELECT
REVOKE SELECT ON dbo.Students FROM StudentUser;
```

SQL Server Security Architecture - Overview

- **Permission Hierarchy**
  - Some permissions imply others
    - (e.g., CONTROL on a database implies all permissions on that database).
  - Permissions can be granted at different levels
    - (e.g., server, database, schema, object).
  - Effective permissions are the union of all granted permissions minus any denials..

SQL Server Security Architecture - Overview

- **Exploring Permissions with System Functions**
- **sys.fn_builtin_permissions(DEFAULT)** lists all possible permissions.

```
SELECT * FROM sys.fn_builtin_permissions(DEFAULT);
```

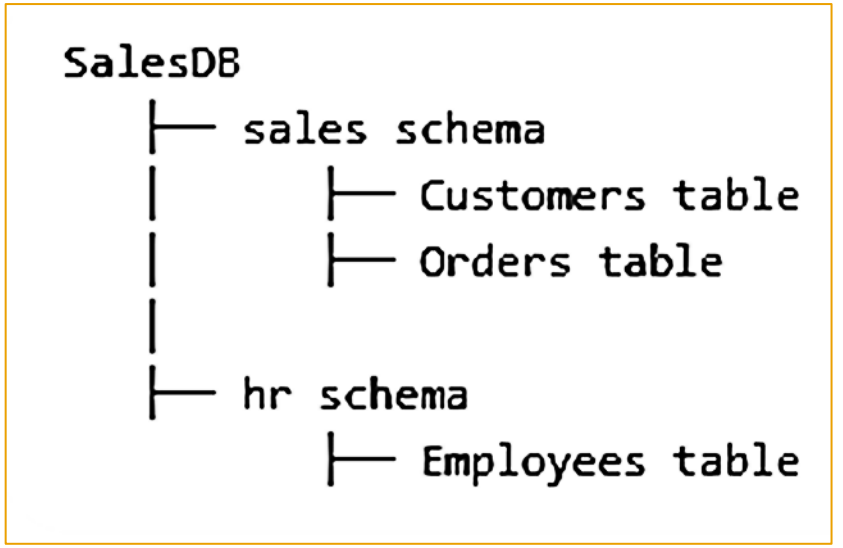
- **Permission Classes**
  - **Permission classes** group permissions by securable type.
  - **Example:** SERVER, DATABASE, SCHEMA, OBJECT, LOGIN, USER, etc.

SQL Server Security Architecture - Overview

(4) Schema Layer

- **A schema (e.g.,** dbo, sales, hr, security) **is a logical container that organizes database objects (e.g.,** object name: sales.Customers, hr.Employees)
- **Why schemas matter for security?**
  - Instead of giving permissions **table by table**, you can secure **entire schemas**.

- Grant access to everything in schema **sales**:
  - GRANT SELECT ON SCHEMA::sales TO UserA;
  - (UserA can read **all tables inside only the sales schema**)



SQL Server Security Architecture - Overview

(5) Object Layer

- This is the **lowest level**. Objects include: table, views, stored procedures, and functions
- Examples: sales.Customers, sales.Orders, hr.Employees
- Granting permission on a single object:
  - GRANT SELECT ON sales.Customers TO UserA;
  - (UserA can only read **Customers table**, not Orders)

SQL Server Security – Securables Hierarchy

- **Server:** endpoints, logins, databases
- **Database:** schemas, users, roles, assemblies
- **Schema:** tables, views, procedures, functions
- **Object:** columns (for granular permissions)

SQL Server Security Architecture – Full Security Flow

Let's walk through a **real scenario**.

User: **Mina**

- Step 1: Windows authentication
  Mina logs into Windows as:

```
COMPANY\Mina
```

- Step 2: SQL Server login

```
CREATE LOGIN [COMPANY\Mina] FROM WINDOWS;
```

- Step 3: Database user

```
USE SalesDB
CREATE USER MinaUser FOR LOGIN [COMPANY\Mina];
```

- Step 4: Schema permission

```
GRANT SELECT ON SCHEMA::sales TO MinaUser;
```

```
SalesDB
    ├── sales schema
    │       ├── Customers table
    │       ├── Orders table
    │
    ├── hr schema
            ├── Employees table
```

Now Mina can read: sales.Customers, sales.Orders but can't accsess hr.Employees or FinanceDB

SQL Server Security Architecture – Full Security Flow

Why This Architecture Is Important for Security?

- The layered model provides:
  - Defense in depth
  - Granular access control
  - Separation of responsibilities
  - Reduced attack surface

For example:

- HR employees should **not see financial tables**
- Developers should **not access production data**

Next Lecture Overview

Server-Level Security – Logins and Authentication

- Creating SQL Server and Windows logins
- Password policies
- Exploring logins with system views
- Using EXECUTE AS to test permissions

Research Tasks

- Why does a new database contain the guest user even though it is disabled?
- What is the purpose of the INFORMATION_SCHEMA and sys users?
- If you create a new SQL Server login, will it automatically have access to UniversityDB? Why or why not?
- How could the existence of the BUILTIN\Administrators login affect security in a university environment?
- What is Single Sign-On?
- What does Defence-in-Depth mean?

References

- Turley, P., & Durkin, J. (2006). *Beginning SQL Server 2005 Express database applications*. Apress. (Chapter 8: Security concepts)
- Carter, J. (2019). *Pro SQL Server 2019 administration*. Apress. (Chapter 10: Security model)
- GeeksforGeeks. (2024). *Database security*. https://www.geeksforgeeks.org/dbms/database-security/


Any
Question