



# IT-234 – database concepts

UNIT 9 – ADVANCED SQL - SECURITY AND TRANSACTION  
CONTROL

In a business setting, there may be sensitive data.



That data needs to be segmented in such a way that only those who have need of the data are allowed to reach it.



For example, only HR personnel should be able to get to employee records and payroll data.

overview

---

Salesmen and buyers should be able to access vendors and suppliers, but other groups do not need to know with whom the company does business.

---

Working with the chief executive officer (CEO) and chief information officer (CIO), the database administrator (DBA) should create these layers of accessibility in the database.



overview

One mechanism for controlling who has access to data is user permissions.

This controls access to specific tables.

You will learn how to use Data Control Language statements to manage user permissions.

overview

# overview



As you are building SQL scripts, the work may become very complex.



There may be a need for all the work to be completed.



The answer is to wrap the statements into a transaction.

# overview

When defined as a transaction, the transaction will pass or fail as a single unit.

When all the statements within a transaction pass, then you can COMMIT the transaction.

If anything goes wrong during execution, then you must ROLLBACK the transaction.

The Transaction Control Language commands enable this functionality.

# overview

A database view is effectively a predefined query.

You create and use views most frequently for the following security-related purposes:

Hiding table columns (for security protection)

Presenting pre-computed columns (in lieu of table columns)

Hiding queries (so that the query outputs are available without running the queries)

# overview

After completing this unit, you should be able to:

- Use Data Control Language (DCL) statements that manage database user permissions.
- Utilize the TCL statements that manage changes made by Data Manipulation Language (DML) statements.
- Generate database views to help maintain data confidentiality.



## Authentication and Authorization components

SQL Server provides three types of components for controlling which users can log onto SQL Server, what data they can access, and which operations they can carry out:

- ▶ **Principals:** Individuals, groups, or processes granted access to the SQL Server instance, either at the server level or database level.
  - ▶ Server-level principals include logins and server roles.
  - ▶ Database-level principals include users and database roles.

# AUTHENTICAT ION AND AUTHORIZATI ON COMPONENT S

- ▶ Securables: Objects that make up the server and database environment. The objects can be broken into three hierarchical levels:
  - ▶ Server-level securables include databases and availability groups.
  - ▶ Database-level securables include schemas and full-text catalogs.
  - ▶ Schema-level securables include tables, views, functions, and stored procedures.

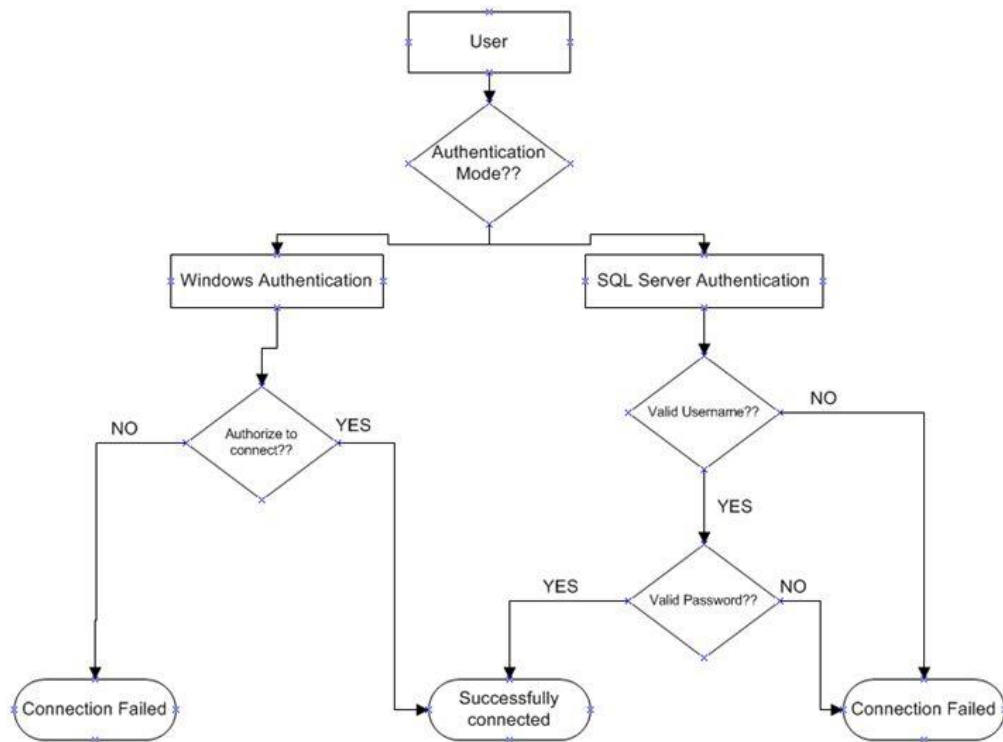
# AUTHENTICATION AND AUTHORIZATION COMPONENTS

- ▶ Permissions: The types of access permitted to principals on specific securables.
  - ▶ You can grant or deny permissions to securables at the server, database, or schema level.
  - ▶ The permissions you grant at a higher level of the hierarchy also apply to children and grandchildren objects, unless you specifically deny those permissions at the lower level.

# SQL SERVER AUTHENTICATION methods

Microsoft SQL Server database connections can be through **Windows Authentication** or **SQL Server Authentication**, which entails a login with a username and password.

**Windows Authentication** doesn't require a username and password because Windows and SQL Server automatically recognize the current operating system user and grants them the permissions that are assigned to that user.



## SQL SERVER AUTHENTICATION methods

## Windows authentication

**Windows Authentication** means the account resides in Active Directory (AD) for the Domain.

SQL Server knows to check AD to see if the account is active, password works, and then checks what level of permissions are granted to the single SQL server instance when using this account.

# Windows authentication



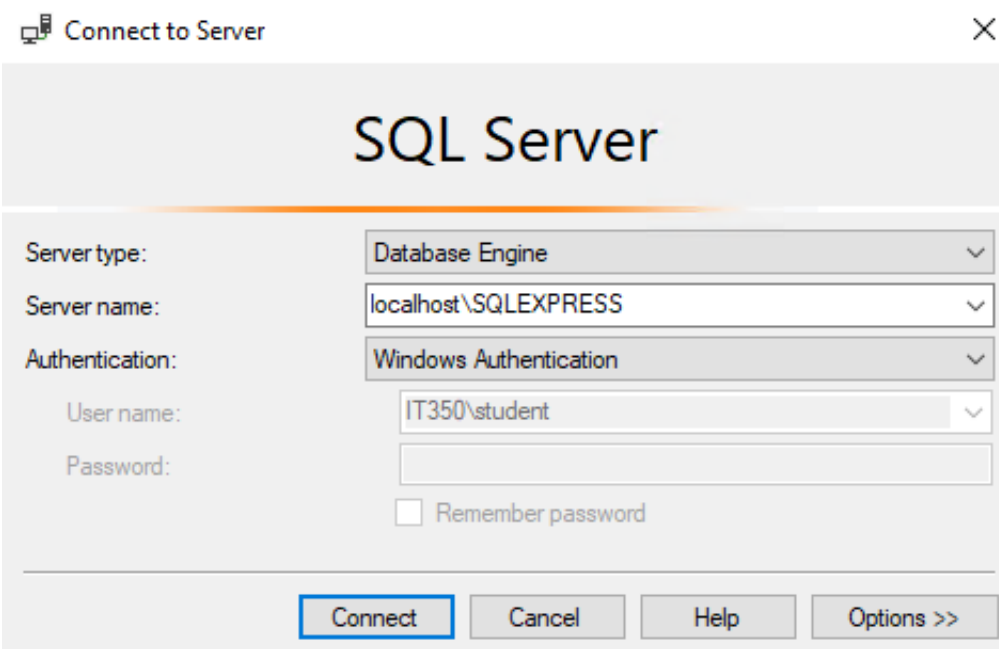
This helps with account management since the account and password only need to be defined once.



You can enforce your company's security policies on the account (Password complexity, password expiration, etc.).

# Windows authentication

- ▶ In **Windows Authentication** mode, you are using SQL Server from the same computer where it is installed
- ▶ SQL Server doesn't ask for username and password as shown below.



Connect to Server

## SQL Server

Server type: Database Engine

Server name: localhost\SQLEXPRESS

Authentication: Windows Authentication

User name: IT350\student

Password:

Remember password

Connect Cancel Help Options >>



## SQL SERVER AUTHENTICATION

**SQL Server Authentication** means the account resides in the SQL server master database but nowhere on the Domain.

The username and password are stored in the master database.

If this account needs to access more than one SQL Server instance, then it has to be created on each instance.

## SQL SERVER AUTHENTICATION

- ▶ An instance of SQL Server can have multiple user accounts with various usernames and passwords.
- ▶ In a shared environment different users have different access on different databases, so **SQL Server Authentication** should be used.

Connect to Server

### SQL Server

Server type: Database Engine

Server name: localhost\SQLEXPRESS

Authentication: SQL Server Authentication

Login: Joe

Password: [masked]

Remember password

Connect Cancel Help Options >>

---

Logins are used at the Instance level and Users are used at the Database level.

---

T-SQL Syntax:

---

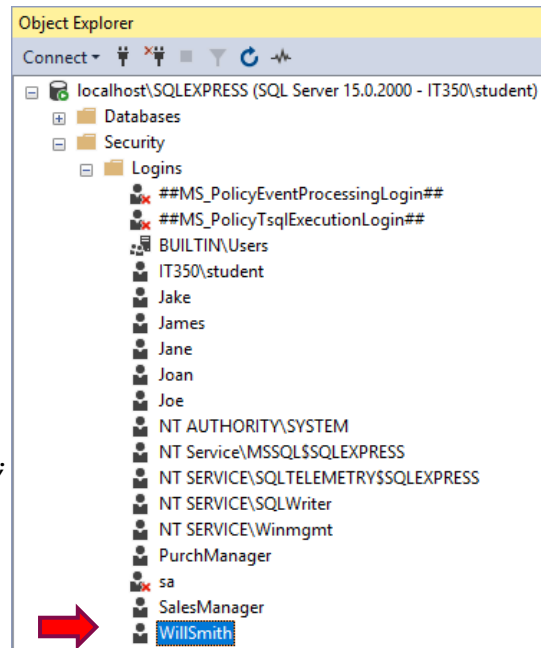
```
CREATE LOGIN <login_name> WITH  
    PASSWORD='<password>',  
    DEFAULT_DATABASE =  
MASTER,      DEFAULT_LANGUAGE  
= US_ENGLISH;
```



SQL SERVER  
AUTHENTICATION  
– Create login

Example:

```
USE MASTER;  
GO  
CREATE LOGIN WillSmith WITH  
PASSWORD='P@$w0rd',  
DEFAULT_DATABASE = MASTER,  
DEFAULT_LANGUAGE =  
US_ENGLISH  
GO
```



SQL  
SERVER  
AUTHENT  
ICATION  
—  
CREATE  
LOGIN

## SQL SERVER AUTHENTICATION – Create USER

The CREATE USER statement creates a database user to log into SQL Server.

A database user is mapped to a Login, which is an identity used to connect to a SQL Server instance.

T-SQL Syntax:

- **CREATE USER <user\_name>**
- **FOR LOGIN <login\_name>;**

Example:

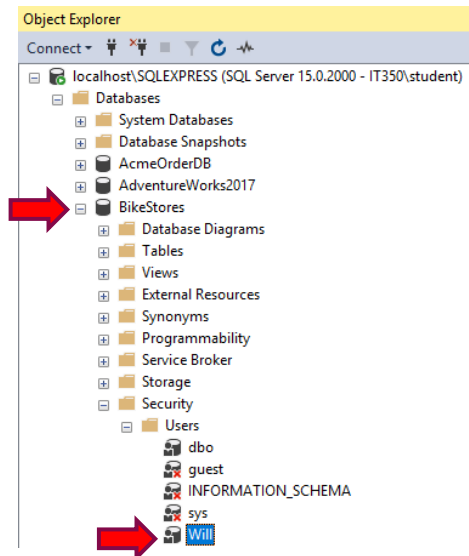
```
USE BikeStores;
```

```
GO
```

```
CREATE USER Will
```

```
FOR LOGIN WillSmith;
```

```
GO
```



SQL  
SERVER  
AUTHENT  
ICATION

—  
CREATE  
USER

# Data control language

The Data Control Language is a subset of the Structured Query Language.



Database administrators use DCL to configure security access to relational databases.



It complements the Data Definition Language, which adds and deletes database objects, and the Data Manipulation Language, which retrieves, inserts, and modifies the contents of a database.

# Data control language

- ▶ Examples of DCL commands:
  - GRANT - gives user's access privileges to the database.
  - REVOKE - withdraw user's access privileges given by using the GRANT command.
  - DENY - prevents a user from receiving a particular permission.



# Data control language

- ▶ The **GRANT** command adds new permissions to a database user.
  - It has a very simple syntax, defined as follows:

```
GRANT [privilege]
ON [object]
TO [user]
[WITH GRANT OPTION]
```

- ▶ **Privilege** - can be either the keyword **ALL** (to grant a wide variety of permissions) or a specific database permission or set of permissions
  - Examples include CREATE DATABASE, SELECT, INSERT, UPDATE, DELETE, EXECUTE and CREATE VIEW.

# Data control language



**Object** - can be any database object.

The valid privilege options vary based on the type of database object you include in this clause.

Typically, the object will be either a database, function, stored procedure, table or view.



**User** - can be any database user.

You can also substitute a role for the user in this clause if you wish to make use of role-based database security.

# Data control language

- ▶ If you include the optional **WITH GRANT OPTION** clause at the end of the **GRANT** command, you not only grant the specified user the permissions defined in the SQL statement but also give the user permission to further grant those same permissions to other database users.
  - For this reason, use this clause with care.

# Data control language

Example:

- Assume you wish to grant the user Joe the ability to retrieve information from the employees table in a database called HR.
- Use the following SQL command:

```
GRANT SELECT  
ON HR.employees  
TO Joe
```

- Joe can retrieve information from employees
- He will not, however, be able to grant other users permission to retrieve information from that table because the DCL script did not include the **WITH GRANT OPTION** clause.

# Data control language

- ▶ The REVOKE command removes database access from a user previously granted such access.
  - The syntax for this command is defined as follows:

```
REVOKE [GRANT OPTION FOR] [permission]
ON [object]
FROM [user]
[CASCADE]
```

- ▶ **Permission** - specifies the database permissions to remove from the identified user.
  - The command revokes both GRANT and DENY assertions previously made for the identified permission.

# Data control language

**Object** - can be any database object.

- The valid privilege options vary based on the type of database object you include in this clause.
- Typically, the object will be either a database, function, stored procedure, table, or view.

**User** - can be any database user.

- You can also substitute a role for the user in this clause if you wish to make use of role-based database security.

# Data control language

The **GRANT OPTION FOR** clause removes the specified user's ability to grant the specified permission to other users.

- If you include the **GRANT OPTION FOR** clause in a **REVOKE** statement, the primary permission is not revoked.
- This clause revokes only the granting ability.

The **CASCADE** option also revokes the specified permission from any users that the specified user granted the permission.

# Data control language

Example:

- The following command revokes the permission granted to Joe in the previous example:

```
REVOKE SELECT  
ON HR.employees  
FROM Joe
```



# Data control language

The **DENY** command explicitly prevents a user from receiving a particular permission.

This feature is helpful when a user is a member of a role or group that is granted a permission, and you want to prevent that individual user from inheriting the permission by creating an exception.

# Data control language

The syntax for the **DENY** command is as follows:

```
DENY [permission]
ON [object]
TO [user]
```

The parameters for the **DENY** command are identical to those used for the **GRANT** command.

# Data control language

Example:

- If you wished to ensure that Matthew would never receive the ability to delete information from the employees table, issue the following command:

```
DENY DELETE  
ON HR.employees  
TO Matthew
```

# DATABASE roles

A database role is a group of users that share a common set of database-level permissions.

SQL Server supports both fixed and user-defined database roles.

To set up a user-defined database role, you must create the role, grant permissions to the role and add members to the role (or add members and then grant permissions).

The permissions assigned to the fixed-database roles cannot be changed

# Fixed DATABASE roles

**db\_owner** – can perform all configuration and maintenance activities on the database

**db\_securityadmin** – can modify role membership for roles that have been created by an administrator or another user (user-defined roles)

**db\_accessadmin** – can add or remove access to the database for Windows logins and groups

**db\_backupoperator** – can back up the database

**db\_ddladmin** – can run any DDL command in a database

**db\_datawriter** – can add, delete or change data in all user-defined tables

**db\_datareader** – can read all data from user-defined tables

**db\_denydatawriter** – users cannot add, modify or delete any data in user-defined tables

**db\_denydatareader** – users cannot read any data in user-defined tables

User-  
defined  
DATABASE  
roles  
creating  
roles

## Role creation syntax:

- **CREATE ROLE**  
**<role\_name>;**

## Examples:

- **CREATE ROLE**  
**GeneralUser;**
- **CREATE ROLE**  
**Salesperson;**

# User-defined DATABASE roles Granting permissions

- ▶ The syntax for granting/revoking role privileges in SQL Server is:

```
GRANT <privileges> ON  
<object> TO <role>;
```

```
REVOKE <privileges> ON  
<object> FROM <role>;
```

- ▶ Examples:

```
GRANT SELECT
```

```
    ON Sales.Customers TO  
GeneralUser;
```

```
GRANT SELECT, INSERT, UPDATE,  
DELETE
```

```
    ON Sales.Customers TO  
Salesperson;
```

# User-defined DATABASE roles ADDING MEMBERS

- ▶ The syntax for assigning roles to users in SQL Server is:

```
ALTER ROLE <role>  
    ADD MEMBER <user>;
```

- ▶ Examples:

```
ALTER ROLE GeneralUser  
    ADD MEMBER Will;  
  
ALTER ROLE Salesperson  
    ADD MEMBER Vanessa;
```



# User-defined DATABASE roles removing members

- ▶ The syntax for assigning roles to users in SQL Server is:

```
ALTER ROLE <role>  
    DROP MEMBER <user>;
```

- ▶ Examples:

```
ALTER ROLE GeneralUser  
    DROP MEMBER Will;  
  
ALTER ROLE Salesperson  
    DROP MEMBER Vanessa;
```

# transaction control language



A transaction is a unit of work that is performed against a database.



Transactions are units or sequences of work accomplished in a logical order, whether in a manual fashion by a user or automatically by some sort of a database program.

# transaction control language

A transaction is the propagation of one or more changes to the database.

For example, if you are creating a record or updating a record or deleting a record from the table, then you are performing a transaction on that table.

# transaction control language

---

It is important to control these transactions to ensure the data integrity and to handle database errors.

---

Practically, database users will combine many SQL queries into a group and will execute all of them together as a part of a transaction.

# transaction control language

- ▶ Transactions have the following four standard properties, usually referred to by the acronym ACID.
  - **Atomicity** – ensures that all operations within the work unit are completed successfully. Otherwise, the transaction is aborted at the point of failure and all the previous operations are rolled back to their former state.

## transaction control language

**Consistency** – ensures that the database properly changes states upon a successfully committed transaction.

**Isolation** – enables transactions to operate independently of and transparent to each other.

**Durability** – ensures that the result or effect of a committed transaction persists in case of a system failure.

# transaction control language

Transaction Control Language (TCL) commands deal with the transaction within the database.

Examples of TCL commands:

**COMMIT** – commits a transaction.

**ROLLBACK** – rolls back a transaction in case of any error occurs.

**SAVEPOINT** – sets a save point within a transaction.

**SET TRANSACTION** – specify characteristics for the transaction.