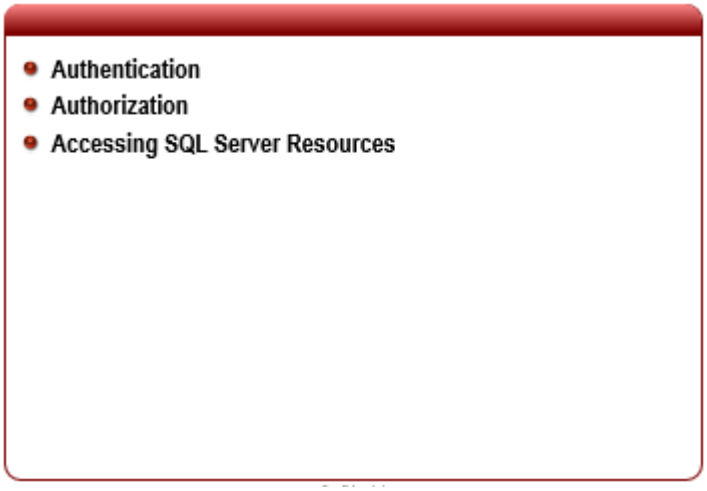
Module 3:
Authentication and Authorization

Contents

Module Overview



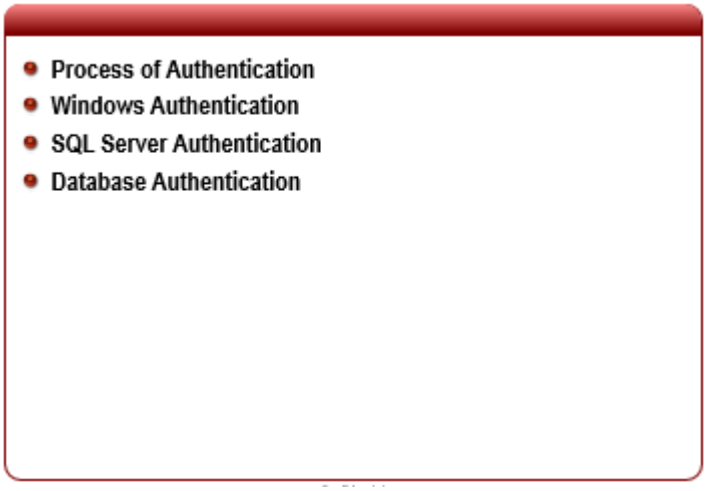
In this module, we will go into more detail about the process of authentication, authorization and how to gain access to concrete SQL Server assets (also known as securables). We will introduce a new feature of authentication (database authentication that comes with SQL Server 2012). Finally, you will understand the relation between logins and users, and how to connect them.

Objectives

After completing this module, you will be able to:

- Understand how to use the authentication process
- Understand and use the authorization process
- Access SQL Server resources

Lesson 1: Authentication



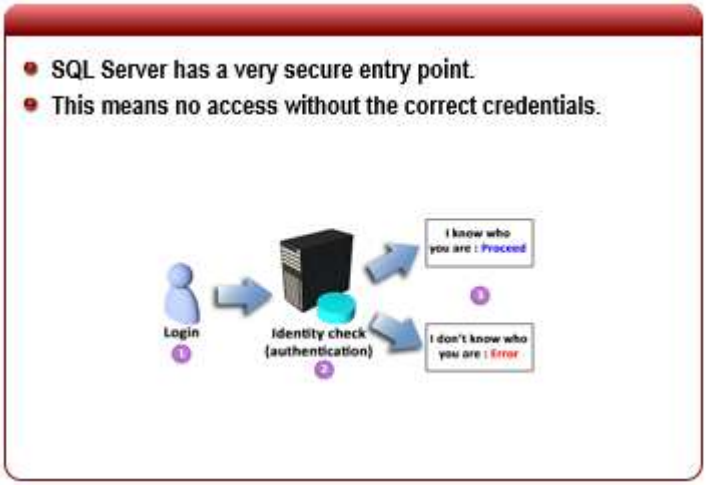
SQL Server has a very powerful mechanism for checking user identity. Based on that process, we can configure all other aspects of security from the higher (server-) to the lower (database-) level. This lesson will go into detail on three aspects of user authentication.

Objectives

After completing this lesson, you will be able to:

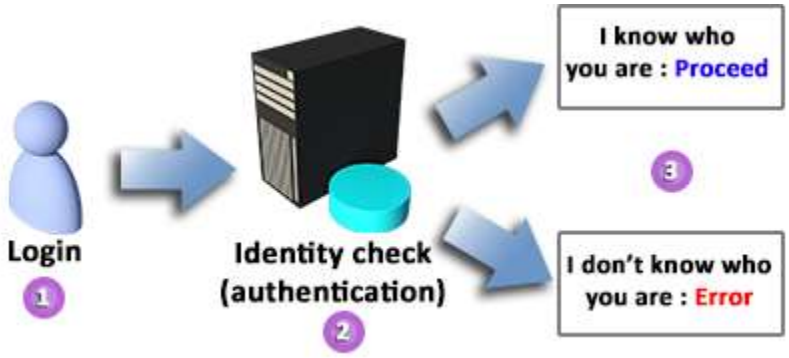
- Understand the process of authentication
- Use Windows Authentication
- Use SQL Server Authentication
- Use Database Authentication

Process of Authentication



During setup, you must select an authentication mode for the Database Engine. There are two possible modes: Windows Authentication Mode and Mixed Mode. Windows Authentication Mode enables Windows Authentication and disables SQL Server Authentication. Mixed Mode enables both Windows Authentication and SQL Server Authentication. Windows Authentication is always available and cannot be disabled.

SQL Server has a very secure entry point. This means no access without the correct credentials. Every information system has some way of checking a user's identity, but SQL Server has three different ways of verifying identity and the ability to select the most appropriate method based on individual or business needs.



Based on the SQL Server reply (I know who you are, or I don't know who you are), the Database Engine will check the access policy. The 2012 version of SQL Server works with three types of authentication:

- Windows Authentication
- SQL Authentication
- Database Authentication

Windows Authentication

As mentioned earlier, there are two possible modes: Windows Authentication Mode and Mixed Mode. Windows Authentication Mode enables Windows Authentication and disables SQL Server Authentication. Windows Authentication is always available and cannot be disabled.

When a user connects through a Windows user account, SQL Server will validate the account name and password using the Windows principal token in the operating system. This means that the user identity is confirmed by Windows. SQL Server does not ask for the password, and does not perform the identity validation (user name and password are NOT transmitted through the network).

Windows Authentication uses Kerberos security protocol, provides password policy enforcement with regards to complexity validation for strong passwords, provides support for account lockout, and supports password expiration. A connection made using Windows Authentication is sometimes called a "trusted connection" because SQL Server trusts the credentials provided by Windows.

A DBSERVER can be a domain server name or the name of a local Windows instance (depending on your server architecture).

```
USE master
GO
  CREATE LOGIN [DBSERVER\DBA]
  FROM WINDOWS
  DEFAULT_DATABASE=[AdventureWorks2012]
GO
```

Windows Authentication is the default authentication mode, and is much more secure than SQL Server Authentication.

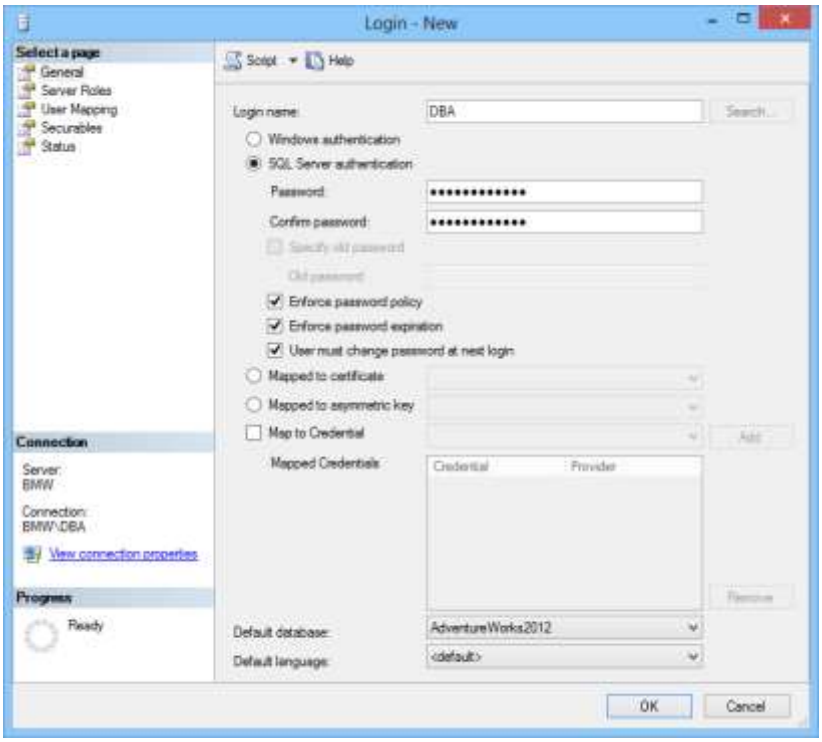
SQL Server Authentication

When using SQL Server Authentication, logins are created in SQL Server that are not based on Windows user accounts. Both the user name and the password are created by using SQL Server and stored in SQL Server. Users connecting through SQL Server Authentication must provide their credentials every time that they connect (user name and password ARE transmitted through the network).

When using SQL Server Authentication, it is highly recommended to set strong passwords for all SQL Server accounts.

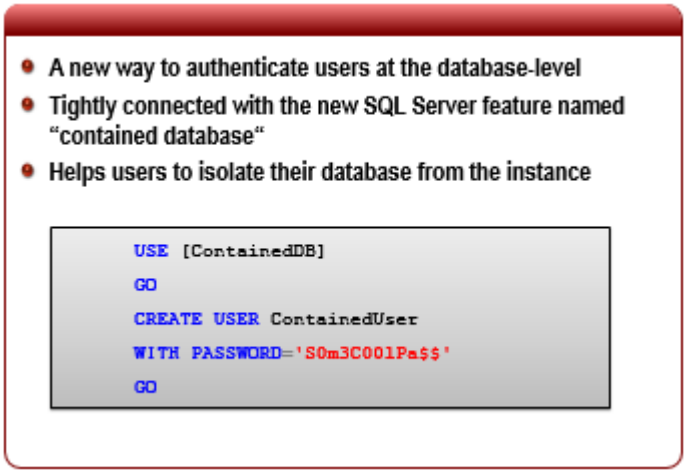
A new SQL Server login can be created that uses SQL Server Authentication. DBA logins have a default database and extra security elements such as password policy that were discussed in Module 02.

```
USE master
GO
  CREATE LOGIN [DBA]
  WITH PASSWORD='S0m3C00lPa$$'
  MUST_CHANGE,
  DEFAULT_DATABASE=[AdventureWorks2012],
  CHECK_EXPIRATION=ON,
  CHECK_POLICY=ON
GO
```



Use SQL Server logins if necessary or in situations where users are connecting from non-Windows environments. The main reason for this is to securely transmit user name and password in clear text through unsecure network channels.

Database Authentication



SQL Server 2012 introduced a new way to authenticate users at the database-level. Database Authentication is connected with the new SQL Server feature called "contained database". Contained database is a developer feature because the database is not dependent on the server configurations. Furthermore, a contained database is a database that is isolated from other databases and from the instance of SQL Server that hosts the database. SQL Server 2012 helps users to isolate database from the instance.

In other words, you can attach/detach a contained database through different SQL Server environments (i.e. development, testing, and production) without worrying about the different security settings between them.

There are two types of users for contained databases:

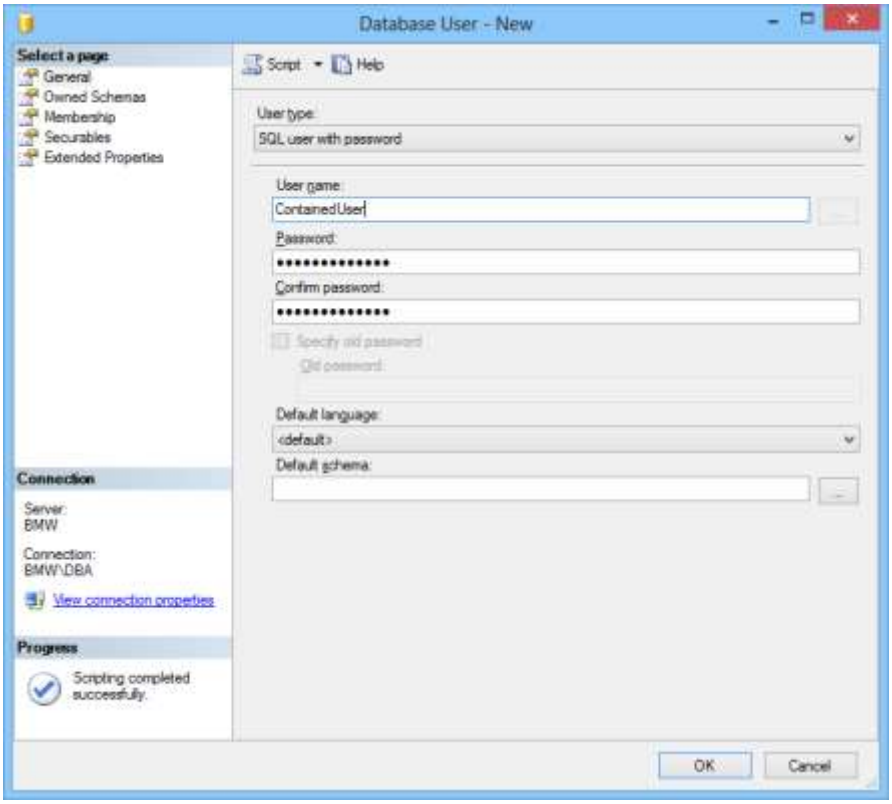
- Contained Database User with Password - authenticated by the database; and
- Windows Principals - authorized Windows users and members of authorized Windows groups.

Users based on logins in the master database can be granted access to a contained database, but such access would create a dependency on the SQL Server instance.

T-SQL code can create a ContainedUser in a ContainedDB without a server-side login.

```
USE [ContainedDB]
GO
CREATE USER ContainedUser
WITH PASSWORD='S0m3C00lPa$$'
GO
```

We can create ContainedUser through GUI.



Contained databases have some unique threats that should be understood and mitigated by SQL Server Database Engine administrators. Most of the threats are related to the USER WITH PASSWORD authentication process, which moves the authentication boundary from the Database Engine level to the database-level.

Practice: Authenticating Users

In this practice, you will create new SQL Server logins.

This practice requires SQL Server 2012 Enterprise Edition in Workgroup, not domain environment.

Exercise 1: Creating new Windows login (local)

In this exercise, you will create a new Windows login through T-SQL code.

1. Open **Control Panel**, click **Administrative Tools**, and then click **Computer Management**.
2. Expand **Local Users and Groups** and expand **Users**.
3. Right-click **New User**.
4. Under **User name**, type **WindowsLogin**.
5. In password field, type **S0m3extr4pa$$**, and click **Create**.
6. Open **SQL Server Management Studio**, expand **Security** node in **Object Explorer**, and expand **Logins**.
7. Right-click **New Login**.
8. Select **Windows authentication**, and in the **Login name**, type **WindowsLogin**.

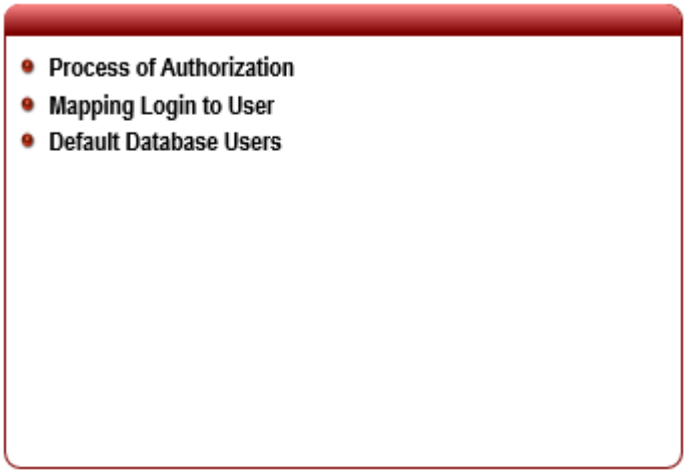
   (Note: you can also use **Search** and **Check Names**).
9. Click **OK**.
10. Now you should have **WindowsLogin** under **Security** node in **Object Explorer**.
11. Click **Logins**.

Exercise 2: Creating a new contained database

In this exercise, you will create a new contained database in SSMS.

1. First, we need to enable this feature under **SQL Server instance**.
2. Open **SQL Server Management Studio** and right-click **Instance Name** in the **Object Explorer**.
3. Under **Properties**, click **Advanced** and then **Containment**.
4. Enable **Contained Databases**, select **True**, and then click **OK**.
5. Right-click **Database** node in **Object Explorer** and then click **New Database**.
6. Under **Database Name**, enter **ContainedDB**, click **Options**, and then under **Containment Type**, select **Partial**.
7. Click **OK**.

Lesson 2: Authorization



After authenticating a user, SQL Server will then determine whether the user has permission to view and/or update data, view metadata, or perform administrative tasks (server-side level, database-side level, or both). If the user, or a group to which the user is a member, has some type of permission within the instance and/or specific databases, SQL Server will let the user connect.

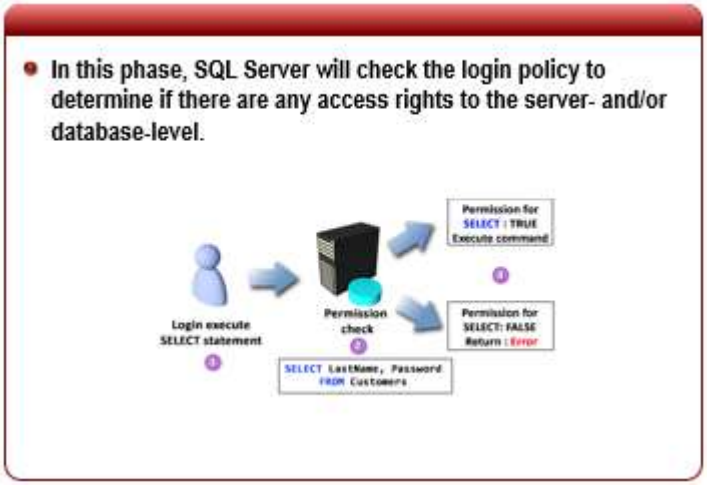
In a nutshell, authorization is the process of checking user access rights to specific securables.

Objectives

After completing this lesson, you will be able to:

- Understand the process of authorization
- Map a login to a user
- Understand default database users

Process of Authorization

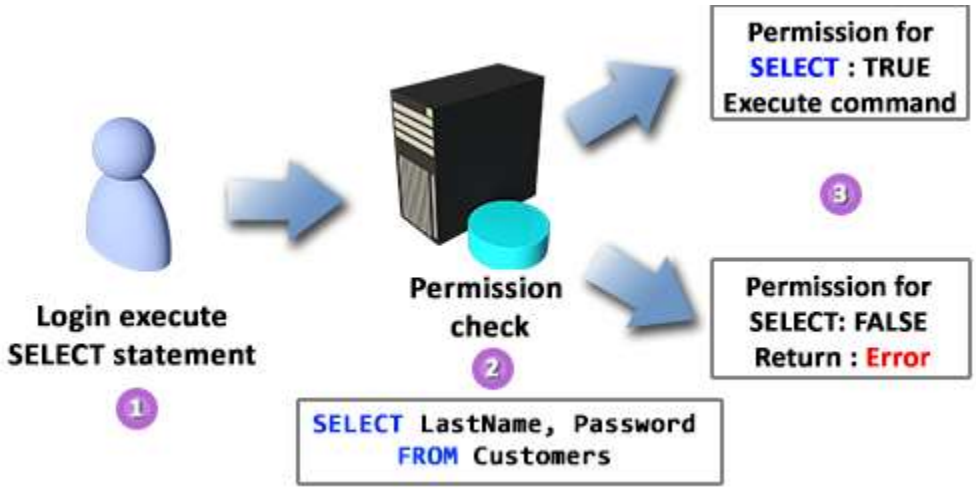


In this phase, SQL Server will check the login policy to determine if there are any access rights to the server- and/or database-level.

Login can have successful authentication, but not any access to the securables. This means that authentication is just one step before login can proceed with any action on SQL Server.

SQL Server will check the authorization process on every T-SQL statement. In other words, if a user has SELECT permissions on some database, SQL Server will not check once and then "forget" until the next authentication/authorization process. Every statement will be verified by the policy to determine if there are any changes.

Permissions are the rules that govern the level of access that principals have to securables. Permissions in a SQL Server system can be granted, revoked, or denied. Each of the SQL Server securables has associated permissions that can be granted to each principal.



Permission Examples

The only way a principal can access a resource in a SQL Server system is if it is granted permission to do so, either directly or indirectly, through membership of a secondary principal such as a role or a group. You can manage permissions by using Object Explorer in SQL Server Management Studio or by selecting the GRANT, REVOKE, or DENY T-SQL statements options.

Inherited Permissions

Certain permissions in SQL Server can be inherited through a permission granted at a higher level in the securable scope hierarchy. For example:

- A principal that has been granted SELECT permission on a schema automatically inherits SELECT permission on all objects in the schema.
- A principal granted CONTROL permission on a database object will automatically inherit CONTROL permission on all securables contained in that database and all securables contained in the schemas within that database.

Effective Permissions

The effective permissions for a principal are evaluated in the same way as in previous versions of Microsoft SQL Server. A principal can perform a particular action if:

- Permission has been granted explicitly to the principal or to a collection that the principal is a member of; and
- Permission has not been explicitly denied to the principal or to a collection that the principal is a member of.

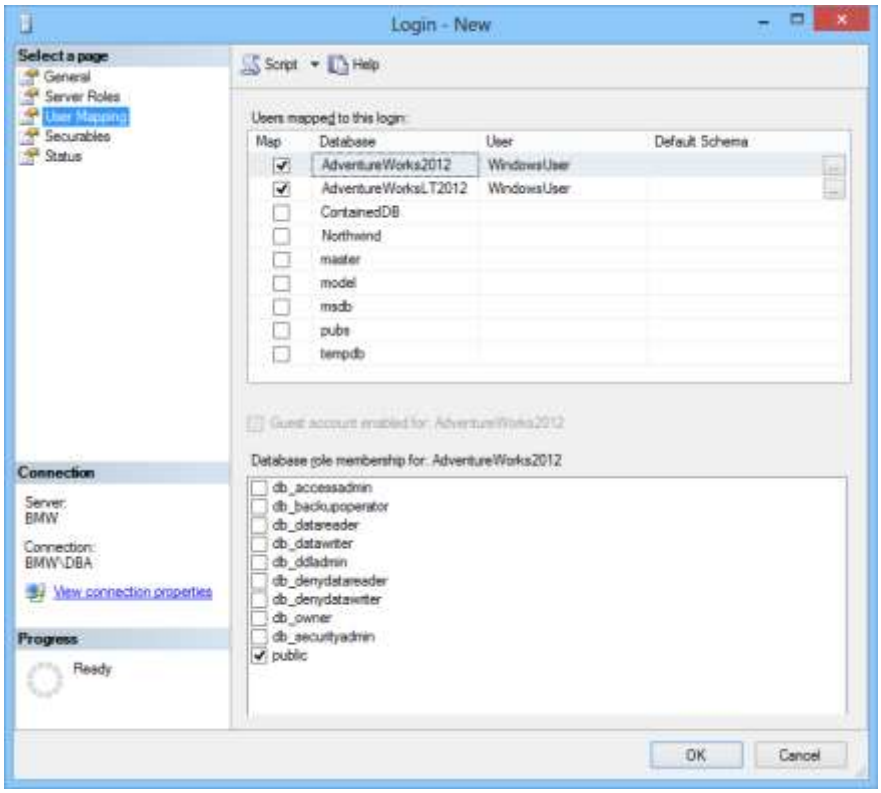
An explicit DENY statement always overrides a GRANT statement. For example, if a user has been explicitly granted UPDATE permission on a particular table, but is a member of a role that has been explicitly denied UPDATE permission on that table, the user will not be able to execute a SELECT statement against the table.

Mapping Login to User

At this point, it is important to note that authentication and authorization are two different processes, but they work in conjunction with one another. Furthermore, the terms "login" and "user" are to be used very carefully, as they are not a same.

- Login is the authentication part
- User is the authorization part

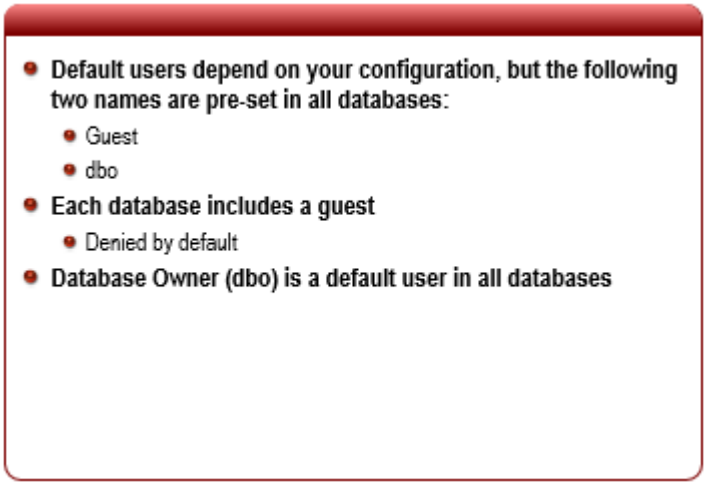
Prior to accessing any database on SQL Server, the login needs to be mapped as a user.



Each login can have one or many user instances in different databases. For example, one login can have READ permission in AdventureWorks2012 and WRITE permission in AdventureWorksLT2012. This type of granular security is one of the biggest SQL Server security features. A user can have the same or different name as the login name.

```
CREATE USER DataReader
 FOR LOGIN [SomeDomain\Imran]
```

Default Database Users



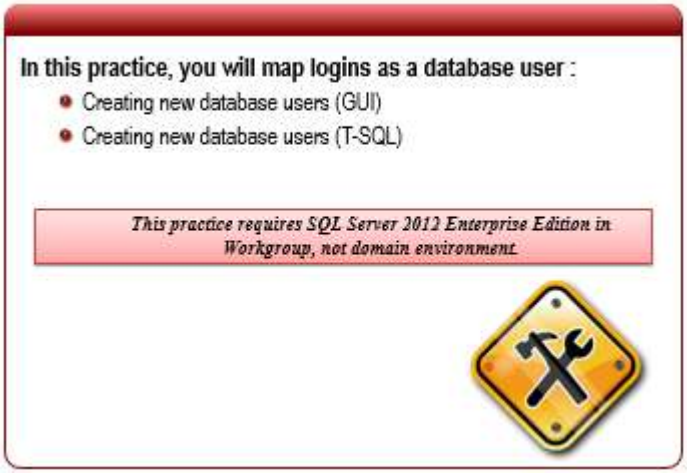
If we look at the security node of a database, we will notice some default names which are default database users. Some of the default names depend on your configuration, but the following two names are pre-set in all databases:

- Guest user; and
- Database Owner (DBO) user.

Each database includes a guest. Permissions granted to the guest user are inherited by users who have access to the database, but who do not have a user account in the database. The guest user cannot be dropped, but it can be disabled by revoking its CONNECT permission. The CONNECT permission can be revoked by executing REVOKE CONNECT FROM GUEST within any database other than master or tempdb.

DBO is a default user in all databases. SQL Server logins, with permission to create a database, are automatically mapped as DBO in the created database because a database needs to have an owner. Also, **sa** SQL Server Login and members of sysadmin fixed server role are mapped as DBO in every user database.

Practice: Authorizing Users



In this practice, you will map logins as a database user.

This practice requires SQL Server 2012 Enterprise Edition in Workgroup, not domain environment.

Exercise 1: Creating new database users

In this exercise, you will create a new database user login through GUI (we will use login created in Practice: Authenticating Users Exercise 1).

1. Open **SQL Server Management Studio**, expand **AdventureWorks2012** database node, **Security** node, and **Users** node.
2. Right-click and select **New User**.
3. Under **User**, select **Windows User**.
4. Under **User name**, type **DBUser**.
5. Under **Login name**, use the **Browse** button to locate the login from **Practice: Authenticating Users Exercise 1**.
6. Click **OK**.
7. Check if there are new database users in **AdventureWorks2012** sample database.

   If none visible, right-click the **Users** node and select the **Refresh** option.

Exercise 2: Creating new database users

In this exercise, you will create a new database user login through T-SQL code using the login created in Practice: Authenticating Users Exercise 1.

1. Open **SQL Server Management Studio** and open **New Query Windows**.

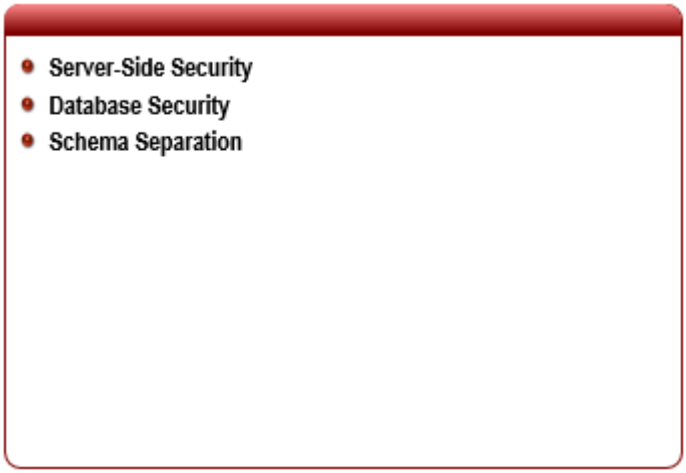
2. Type the following code and press **F5**:

```
USE AdventureWorks2012
GO
CREATE USER DBUser
 FOR LOGIN [YourMashineName\WindowsLogin]
```

3. Check if there are new database users in **AdventureWorks2012** sample database.

   If none visible, right-click the **Users** node and select the **Refresh** option.

Lesson 3: Accessing SQL Server Resources



Now that we understand the authentication/authorization process, we can create more detailed user access policies on the server- and/or database-level. Also, we will go in more detail about fixed server-side and database roles.

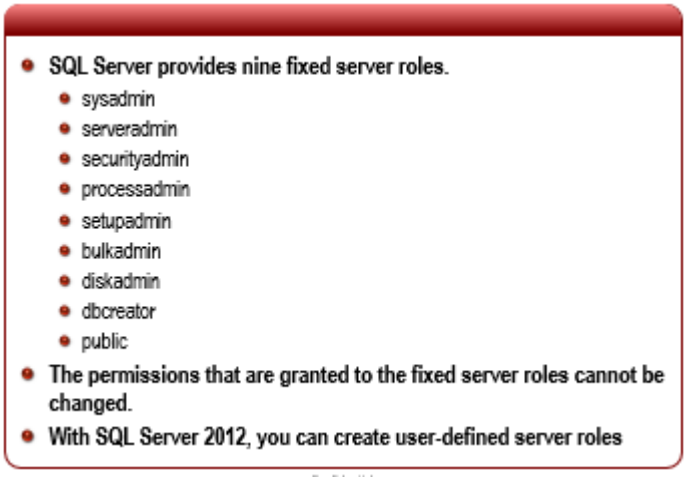
Finally, you will learn how to create a login/user access policy with different elements.

Objectives

After completing this lesson, you will be able to:

- Understand and use server-side security
- Understand and use database security
- Understand and use schema separation

Server-Side Security



Fixed server roles are provided for convenience and backward compatibility. SQL Server provides nine fixed server roles. The permissions that are granted to the fixed server roles cannot be changed. Beginning with SQL Server 2012, you can create user-defined server roles and add server-level permissions to those roles.

Fixed Server-Level Role	Description
sysadmin	Members of the sysadmin fixed server role can perform any activity in the server.
serveradmin	Members of the serveradmin fixed server role can change server-wide configuration options and shut down the server.
securityadmin	Members of the securityadmin fixed server role manage logins and their properties. They can GRANT, DENY, and REVOKE server-level permissions. They can also GRANT, DENY, and REVOKE database-level permissions if they have access to a database. Additionally, they can reset passwords for SQL Server logins. The ability to grant access to the Database Engine and to configure user permissions allows the securityadmin to assign most server permissions. The securityadmin role should be treated as equivalent to the sysadmin role.
processadmin	Members of the processadmin fixed server role can end processes that are running in an instance of SQL Server.

setupadmin	Members of the setupadmin fixed server role can add and remove linked servers.
bulkadmin	Members of the bulkadmin fixed server role can run the BULK INSERT statement.
diskadmin	The diskadmin fixed server role is used for managing disk files.
dbcreator	Members of the dbcreator fixed server role can create, alter, drop, and restore any database.
public	Every SQL Server login belongs to the public server role. When a server principal has not been granted or denied specific permissions on a securable object, the user inherits the permissions granted to public on that object. Only assign public permissions on any object when you want the object to be available to all users. You cannot change membership in public.

Only server-level permissions can be added to user-defined server roles. We can list the server-level permissions by querying sys.fn_builtin_permissions.

```
SELECT * FROM sys.fn_builtin_permissions('SERVER')
ORDER BY permission_name;
```

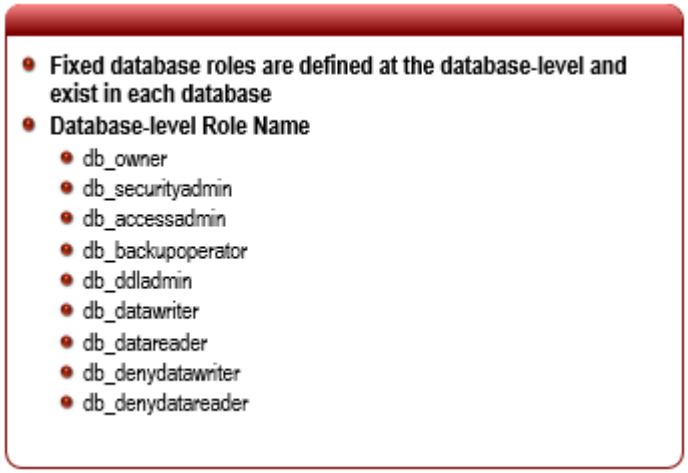
We can create a new SQL Server login and assign a permission to create new databases on a server with the CREATE LOGIN statement.

```
USE [master]
GO

CREATE LOGIN [DatabaseCreator]
WITH PASSWORD=N'S0m3C00lP4ss',
DEFAULT_DATABASE=[master],
CHECK_EXPIRATION=ON,
CHECK_POLICY=ON
GO

ALTER SERVER ROLE [dbcreator] ADD MEMBER [DatabaseCreator]
GO
```

Database Security



Fixed database roles are defined at the database-level and exist in each database. Members of the db_owner and db_securityadmin database roles can manage fixed database role memberships.

However, only members of the db_owner database role can add members to the db_owner fixed database role.

The following table shows all of the fixed database-level roles:

Database-Level Role Name	Description
db_owner	Members of the db_owner fixed database role can perform all configuration and maintenance activities on the database, and can also drop the database.
db_securityadmin	Members of the db_securityadmin fixed database role can modify role membership and manage permissions. Adding principals to this role could enable unintended privilege escalation.
db_accessadmin	Members of the db_accessadmin fixed database role can add or remove access to the database for Windows logins, Windows groups, and SQL Server logins.
db_backupoperator	Members of the db_backupoperator fixed database role can backup the database.
db_ddladmin	Members of the db_ddladmin fixed database role can run any Data Definition Language (DDL) command in a database.
db_datawriter	Members of the db_datawriter fixed database role can add, delete, or change data in all user tables.

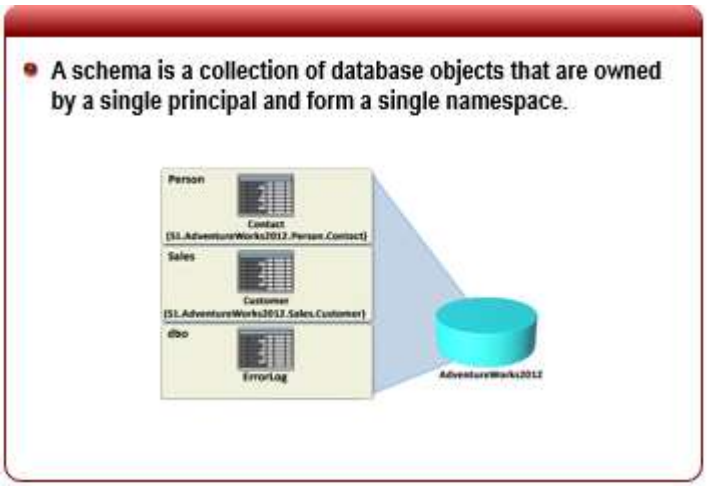
db_datareader	Members of the db_datareader fixed database role can read all data from all user tables.
db_denydatawriter	Members of the db_denydatawriter fixed database role cannot add, modify, or delete any data in the user tables within a database.
db_denydatareader	Members of the db_denydatareader fixed database role cannot read any data in the user tables within a database.

Every database user belongs to the public database role. When a user has not been granted or denied specific permissions on a securable object, the user inherits the permissions granted to the public on that object.

This T-SQL code assign two fixed database roles to the user "DatabaseCreator", but only in the AdventureWorks 2012 database. One of the fixed database roles is the explicit read role and the other is the explicit deny role on data change.

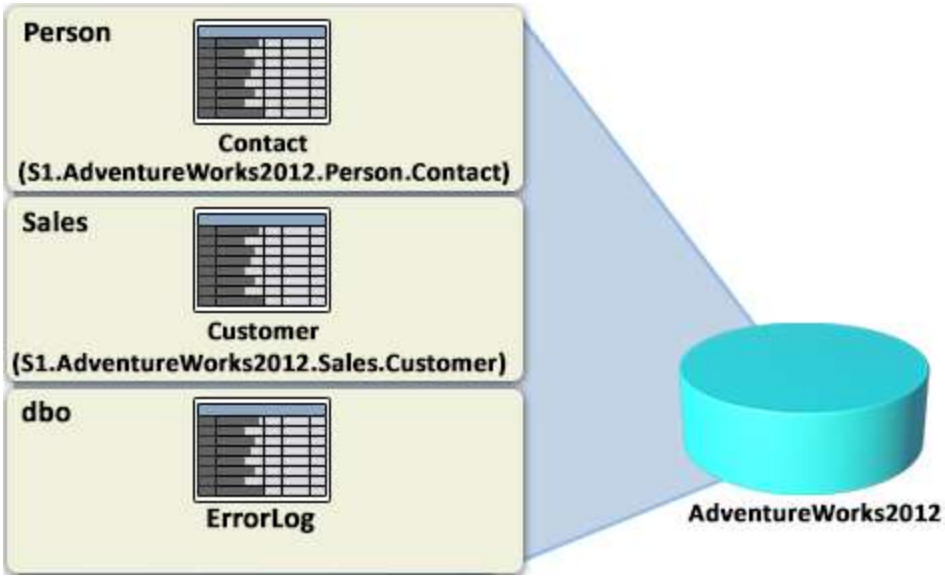
```
USE [AdventureWorks2012]
GO
ALTER ROLE [db_datareader] ADD MEMBER [DatabaseCreator]
GO
USE [AdventureWorks2012]
GO
ALTER ROLE [db_denydatawriter] ADD MEMBER [DatabaseCreator]
GO
```

Schema Separation



In Microsoft SQL Server 2012, a schema is a collection of database objects that are owned by a single principal and form a single namespace.

All objects within a schema must be uniquely named and a schema itself must be uniquely named in the database catalog. SQL Server (since version 2005) breaks the link between users and schemas. In other words, users do not own objects; schemas own objects and principals own schemas.



A schema can be owned by either a primary or secondary principal, with the term “principal” meaning any SQL Server entity that can access securable objects.

The following types of principals can own schemas:

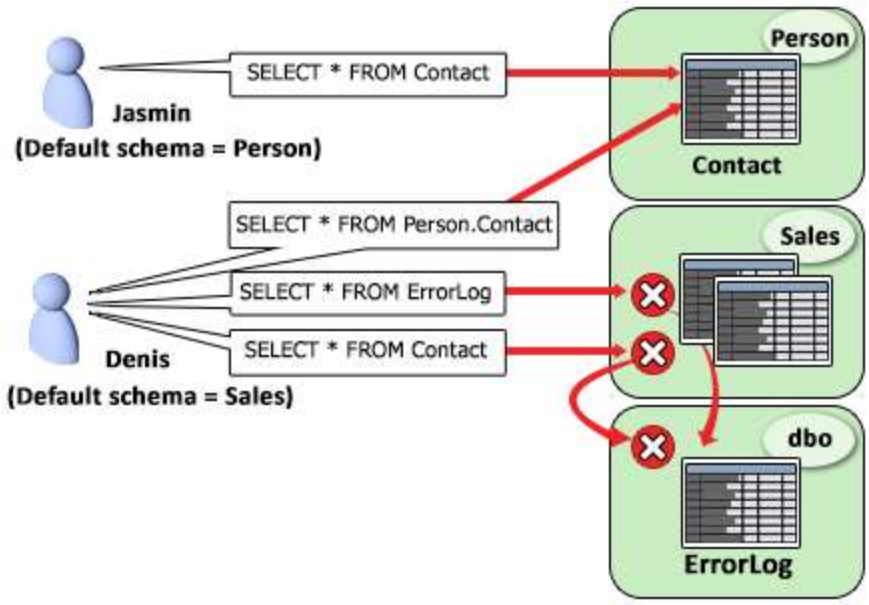
Primary

- SQL Server Login
- Database User
- Windows Login

Secondary

- SQL Server Roles
- Windows Groups
- Default Schemas

Users can now have a default schema assigned using the DEFAULT_SCHEMA option from the CREATE USER and ALTER USER commands. If a default schema is not supplied for a user, then the dbo will be used as the default schema.



If a user from a different default schema needs to access objects in another schema, then the user will need to type a full name. For example, Denis needs to query the Contact tables in the Person schema, but he is in Sales. To resolve this, he would type: SELECT * FROM Person.Contact.

Keep in mind that the default schema is dbo. When database objects are created and not explicitly put in schemas, SQL Server will assign them in dbo default database schema. Therefore, there is no need to type “dbo” because it is a default schema.

```
USE AdventureWorks2012;
GO
CREATE SCHEMA Secret AUTHORIZATION dbo
GO
CREATE TABLE Audit (AuditID int, Who nvarchar(100),
            What nvarchar(100), Time nvarchar(100))
GO
```

Practice: Applying Security Policy

In this practice, you will apply a user access policy.

This practice requires SQL Server 2012 Enterprise Edition in Workgroup, not domain environment.

Exercise 1: Applying server-side policy

In this exercise, you will create a new server-side policy through a GUI using the login created in Practice: Authenticating Users Exercise 1. WindowsUser login needs a permission for creating new databases on the instance and define server-wide security policy.

1. Open **SQL Server Management Studio** and expand the **Security** node.
2. Right-click **WindowsUser** and select **Properties**.
3. Select **Server Role**.
4. Click **dbcreator** and then **securityadmin**.
5. Click **OK**.
6. You can practice the same task through the following T-SQL code:

```
USE master
GO
ALTER SERVER ROLE [dbcreator] ADD MEMBER
[YourMashineName\WindowsUser]
GO
ALTER SERVER ROLE [securityadmin] ADD MEMBER
[YourMashineName\WindowsUser]
GO
```

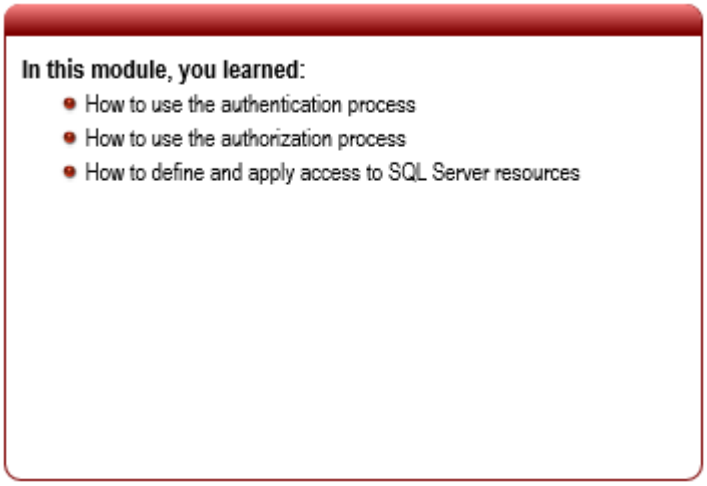
Exercise 2: Applying database access policy

In this exercise, you will create a new database access policy through a GUI using a user created in Practice: Authorizing Users, Exercise 1. The DBUser needs to have a policy to SELECT, INSERT, UPDATE and DELETE operations.

1. Open **SQL Server Management Studio**, expand the **AdventureWorks2012** database node, expand the **Security** node, and expand **Users**.
2. Right-click **DBUser**, click **Properties**, then click **Membership**.
3. Select **db_datareader** and **db_datawriter**.
4. Click **OK**.
5. You can practice the same task through the following T-SQL code:

```
USE [AdventureWorks2012]
GO
ALTER ROLE [db_datareader] ADD MEMBER [DBUser]
GO
USE [AdventureWorks2012]
GO
ALTER ROLE [db_datawriter] ADD MEMBER [DBUser]
GO
```

Summary



In summary, understanding the basics of security and being aware of security issues contribute to an effective authentication/authorization policy. Furthermore, the precision of your permission setting will yield a better security and permission policy. It is important to remember that users' access should be based on their needs in order to accomplish their jobs. In other words, a user's access rights should be restricted before allowing the user to access a database. It is bad practice to grant all access to a user, and then later restrict the access rights. By granting all access, you are weakening your security policy and promoting damage to the database.

In this lesson, you have learned how to deal with server- and database-side security and how to define and apply a user access policy.

Objectives

After completing this module, you learned:

- How to use the authentication process
- How to use the authorization process
- How to define and apply access to SQL Server resources