

Configure database authentication and authorization

1. Introduction

<https://learn.microsoft.com/en-us/training/modules/configure-database-authentication-authorization/1-introduction>

Introduction

Completed

Azure SQL offerings provide several authentication and authorization options that differ from those in SQL Server. This is because Azure SQL Database and Azure SQL Managed Instance rely on Microsoft Entra ID instead of Windows Server Active Directory.

You'll explore the best practices for granting permissions and the various permissions available within a database. It also delves into the concept of *least privilege*. While built-in roles in SQL Server and other database engines offer broad security privileges, many applications require more granular security on database objects.

Learning objectives

In this module you'll learn about:

- Explain authentication options for Azure SQL offerings
- Describe security principals
- Explain object permissions
- Identify authentication and authorization failures

2. Describe authentication and identities

Describe authentication and identities

Completed

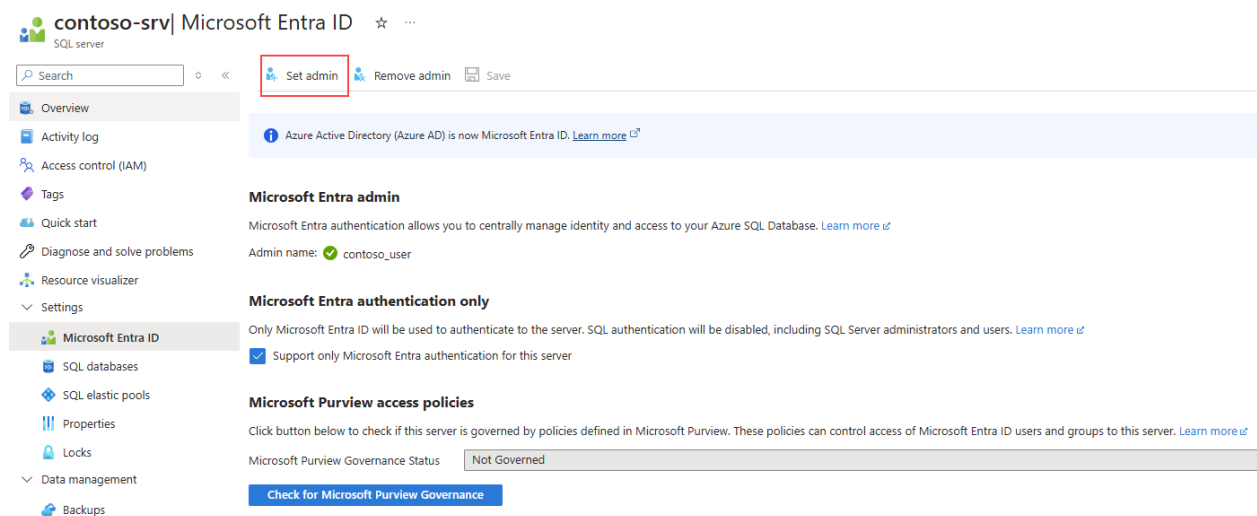
Both on-premises SQL Server and SQL Server in Azure VMs support SQL Server authentication and Windows Authentication. SQL Server authentication stores login details within SQL Server, while Windows Authentication uses Microsoft Entra ID (formerly Active Directory) accounts.

Microsoft Entra ID authentication is more secure and simplifies user management. If a user leaves, only the Microsoft Entra ID account needs to be locked.

Azure SQL Database also supports SQL Server authentication and Microsoft Entra authentication. Microsoft Entra authentication uses the same credentials for other resources like the Azure portal or Microsoft 365.

Microsoft Entra ID can sync with on-premises Active Directory, providing consistent credentials for both environments. It also supports multifactor authentication (MFA) for added security. MFA options include push notifications via the Microsoft Authenticator app, text messages, or access codes. Users with MFA must use the Universal Authentication with MFA option in SQL Server Management Studio.

You can set SQL admin permissions on an Azure SQL Database using the Azure portal.



It's a best practice to make this account a Microsoft Entra group, so access isn't dependent on a single login. The Microsoft Entra admin account grants special permissions and allows the account or group that holds that permission to have `sysadmin` like access to the server and all of the databases within the server. The admin account is only set using Azure Resource Manager and not at the database level. In order to change the account or group, you have to use the Azure portal, PowerShell, or Azure CLI.

Role-based access control (RBAC)

All Azure types of operations for Azure SQL Database are controlled through role-based access control (RBAC). RBAC is currently decoupled from Azure SQL security, but you can think of it as security rights outside of your database in SQL Database, with a scope that includes subscription, resource group, and resource. The rights apply to operations in the Azure portal, the Azure CLI, and Azure PowerShell. RBAC allows for separation of duties between deployment, management, and usage.

Built-in roles are available to reduce the need for higher-level RBAC roles, such as **Owner** or **Contributor**. Effectively, you can use these roles to have certain individuals deploy Azure SQL resources (or manage security policies) but grant other users actual access to use or manage the database. For example, a **SQL Server Contributor** could deploy a server and assign a user to be the admin of the server and databases. The built-in roles for Azure SQL Database include:

- **SQL DB Contributor**: Can create and manage databases but can't access the database (for example, can't connect and read data)
- **SQL Security Manager**: Can manage security policies for databases and instances (such as auditing) but can't access them
- **SQL Server Contributor**: Can manage servers and databases but can't access them.

3. Describe Security Principals

<https://learn.microsoft.com/en-us/training/modules/configure-database-authentication-authorization/4-describe-security-principals>

Describe Security Principals

Completed

Security principals are entities that can request SQL Server resources and to which you can (usually) grant permissions. There are several sets of security principals in SQL Server. Security principals exist at either the server level or the database level and can be either individuals or collections. Some sets have a membership controlled by the SQL Server administrators, and some have a fixed membership.

Note

New logins can be added by administrators on Azure SQL Database, but new server roles can't be created.

Schemas and securables

Before we look at the details of security principals, we need to understand the concepts of securables and schemas. SQL Server and Azure SQL Database have three scopes for securables. Securables are the resources within the database to which the authorization system manages access. For example, a table is a securable. To simplify access control, SQL Server contains securables in nested hierarchies called scopes. The three securable scopes are the server, the database, and the schema. A schema is a collection of objects within your database, which allows objects to be grouped into separate namespaces.

Every user has a default schema. If a user tries to access an object without specifying a schema name, as in: `SELECT name FROM customers`, it's assumed the object is in the user's default schema. If there's no such object in the default schema, SQL Server checks to see if the object is in the predefined `dbo` schema. If there's no object of the specified name in either the user's default schema, or in the `dbo` schema, the user receives an error message. It's a best practice to specify the schema name when accessing objects, so the previous select would be something like: `SELECT name FROM SalesSchema.customers`. If a user hasn't been given a default schema, their default schema is set to `dbo`.

By default, if no schema is specified when a user creates an object, SQL Server attempts to create it in the user's default schema. If the user hasn't been granted permission to create objects in their default schema, the object can't be created.

Logins and users

No matter the mode of authentication that is used, a login name used to access your SQL database is set up as a login within the instance. Those logins are set up at the instance level of SQL Server and stored in the master database. However, you can configure contained users, which are added at the database level. These users can be configured as SQL Server Authentication users and either Windows Authentication users or Microsoft Entra users (depending on which platform you're using). In order to create these users, the database must be configured for partial containment, which is configured by default in Azure SQL Database, and optionally configurable in SQL Server.

These users only have access to the database that the user is set up with. For the purposes of Azure SQL Database, it's a best practice to create users at the scope of the user database, and not in the master database.

```
CREATE USER [dba@contoso.com] FROM EXTERNAL PROVIDER;  
GO
```

The `CREATE USER` statement is executed in the context of the user database. In the example, the user is a Microsoft Entra user as indicated with the `FROM EXTERNAL PROVIDER` syntax.

If logins are created at the instance level in SQL Server, a user should then be created within the database, which maps the user to the server-based login as shown in the following example.

```
USE [master]
GO

CREATE LOGIN demo WITH PASSWORD = 'Pa55.w.rd'
GO

USE [WideWorldImporters]
GO

CREATE USER demo FROM LOGIN demo
GO
```

The login is first created in the master database, and then in the WideWorldImporters database a user is created to map to that login. Logins are used to access the SQL Server or the Azure SQL Database, but to do any work within a database, the login must be mapped to a username. The username is used for all authorization.

Logins and usernames are the most important security principals you need to be aware of, but the next sections describe some of the other concepts and terms when dealing with authorization.

Database roles

As you can imagine, database security can get complicated for applications with many users. In order to make it easier for both administrators and auditors, most database applications use role-based security. Roles are effectively security groups that share a common set of permissions. Combining permissions into a role allows a set of roles to be created for a given application. Some examples of roles would be administrators who had full access to all of the databases and servers, reporting users who only read the database, and an application account that had access to write data into the database. The roles can be defined when the application is designed, and then users can be assigned to those roles as they need access to the database. Role-based access control is a common architecture across computer systems and is how authorization is managed in Azure Resource Manager.

SQL Server and Azure SQL Database both include built-in roles that are defined by Microsoft, and also provide the option to create custom roles. Custom roles can be created at the server or database level. However, server roles can't be granted access to objects within a database directly. Server roles are only available in SQL Server and Azure SQL Managed Instance, not in Azure SQL Database.

Within a database, permissions can be granted to the users that exist within the database. If multiple users all need the same permissions, you can create a database role within the database and grant the needed permissions to this role. Users can be added as members of the database role. The member of the database role inherits the permissions of the database role.

```
CREATE USER [DP300User1] WITH PASSWORD = 'Pa55.w.rd'
GO

CREATE USER [DP300User2] WITH PASSWORD = 'Pa55.w.rd'
GO
```

```

CREATE ROLE [SalesReader]
GO

ALTER ROLE [SalesReader] ADD MEMBER [DP300User1]
GO

ALTER ROLE [SalesReader] ADD MEMBER [DP300User2]
GO

GRANT SELECT, EXECUTE ON SCHEMA::Sales TO [SalesReader]
GO

```

In the above example, you can see that two users are created, and then a role called SalesReader is created. The two new users are added to the newly created role, and then finally the role is granted `SELECT` and `EXECUTE` permissions on the *Sales* schema. Any user who is in that role can select from any object in the *Sales* schema, and execute any stored procedure in the schema.

Application roles

Application roles can also be created within a SQL Server database or Azure SQL Database. Unlike database roles, users aren't made members of an application role. An application role is activated by the user, by supplying the preconfigured password for the application role. Once the role is activated the permissions that are applied to the application role are applied to the user until that role is deactivated.

Built-in database roles

Microsoft SQL Server contains several fixed database roles within each database for which the permissions are predefined. Users can be added as members of one or more roles. These roles give their members a predefined set of permissions. These roles work the same within Azure SQL Database and SQL Server.

Database role	Definition
db_accessadmin	Allows users to create other users within the database. This role doesn't grant access to the schema of any of the tables, nor does it grant access to the data within the database.
db_backupoperator	Allows users to back up a database in a SQL Server or SQL Managed Instance. The role <i>db_backupoperator</i> doesn't confer any permissions in an Azure SQL Database.
db_datareader	Allows users to read from every table and view within the database.
db_datawriter	Allows users to <code>INSERT</code> , <code>UPDATE</code> , and <code>DELETE</code> data from every table and view within the database.
db_ddladmin	Allows users to create or modify objects within the database. Members of this role can change the definition of any object, of any type, but members of this role aren't

Database role	Definition
	granted access to read or write any data within the databases.
db_denydatareader	Users who need to be prevented from reading data from any object in the database, when those users have been granted rights through other roles or directly.
db_denydatawriter	Users who need to be prevented from writing data to any object in the database, when those users have been granted rights through other roles or directly.
db_securityadmin	Users who need to be able to grant access to other users within the database. Members of this role aren't granted access to the data within the database; however members of this role can grant themselves access to the tables within the database. Membership in this database role should be limited to only trusted users.
db_owner	Users who need administrative access to the database. Members of this role can perform any action within the database by default. However, unlike the actual database owner, who has the user name <i>dbo</i> , users in the db_owner role can be blocked from accessing data by placing them in other database roles, such as db_denydatareader , or by denying them access to objects. Membership in this database role should be limited to only trusted users.

All users within a database are automatically members of the **public** role. By default, this role has no permissions granted to it. Permissions can be granted to the public role, but you should consider carefully whether that is really something you want to do. Granting permissions to the public role would grant these permissions to any user, including the guest account, if the guest account was enabled.

The built-in database roles do meet the needs of many applications; however with applications that require more granular security (for example, when you only want to grant access to a specific subset of tables) a custom role is often a better choice.

Note

By default, users in roles like *db_owner* can always see all of the data in the database. Applications can take advantage of encryption options like **Always Encrypted** to protect sensitive data from privileged users.

For Azure SQL Database, there are a few nuances. You can have logins that exist in the virtual **master** database, database users, and even contained database users for Microsoft Entra accounts (recommended). Although the server admin for Azure SQL Database essentially has **sysadmin** rights, you can create more limited admins by using server or database level roles. Two database level roles are available for SQL Database that only exist in the virtual **master** database:

Database Role	Definition
loginmanager	Allows members to create logins for the database server. This role is the equivalent of the dbcreator fixed server role in an on-premises Microsoft SQL Server.

Database Role	Definition
dbmanager	Allows members to create and delete databases for the database server. This role is the equivalent of the securityadmin fixed server role in an on-premises Microsoft SQL Server.

Fixed server-level roles

In addition to database roles, SQL Server and Azure SQL Managed Instance both provide several fixed server-level roles. These roles assign permissions at the scope of the entire server. Server-level principals, which include SQL Server logins, Windows accounts, and Windows group can be added into fixed server-level roles. The permissions for fixed server-level roles are predefined, and no new server roles can be added.

Fixed server-level role	Definition
sysadmin	Allows its members to perform any activity on the server.
serveradmin	Allows its members to change server-wide configuration settings (for example Max Server Memory) and can shut down the server.
securityadmin	Allows its members to manage logins and their properties (for example, changing the password of a login). The members can also grant and revoke server and database level permissions. This role should be treated as being equivalent to the sysadmin role.
processadmin	Allows its members to kill processes running inside of SQL Server.
setupadmin	Allows its members to add and remove linked servers using T-SQL.
bulkadmin	Allows its members to run the BULK INSERT T-SQL statement.
diskadmin	Allows its members to have the ability to manage backup devices in SQL Server.
dbcreator	Allows its members to create, restore, alter, and drop any database.
public	Every SQL Server login belongs to the public user role. Unlike the other fixed server roles, permissions can be granted, denied, or revoked from the public role.

Here's a list of server-level roles in Azure SQL Database:

- **##MS_DatabaseConnector##**
- **##MS_DatabaseManager##**
- **##MS_DefinitionReader##**
- **##MS_LoginManager##**
- **##MS_SecurityDefinitionReader##**
- **##MS_ServerStateReader##**
- **##MS_ServerStateManager##**

4. Describe database and object permissions

<https://learn.microsoft.com/en-us/training/modules/configure-database-authentication-authorization/5-describe-database-object-permissions>

Describe database and object permissions

Completed

All Relational Database Management platforms have four basic permissions, which control data manipulation language (DML) operations. These permissions are `SELECT`, `INSERT`, `UPDATE`, and `DELETE`, and they apply to all SQL Server platforms. All of these permissions can be granted, revoked, or denied on tables and views. If a permission is granted using the `GRANT` statement, then the permission is given to the user or role referenced in the `GRANT` statement. Users can also be denied permissions using the `DENY` command. If a user is granted a permission and denied the same permission, the `DENY` always supersedes the grant, and the user will be denied access to the specific object.

```
GRANT SELECT ON dbo.Company to Demo
GO
DENY SELECT ON dbo.Company to Demo
GO
EXECUTE AS USER = 'Demo'
SELECT Name, Address FROM dbo.Company
```

Msg 229, Level 14, State 5, Line 17
The SELECT permission was denied on the object 'Company', database 'WideWorldImporters', schema 'dbo'.
Completion time: 2020-05-13T14:42:28.8361616-07:00

In the example, the user Demo is granted `SELECT` and then denied `SELECT` permissions on the `dbo.Company` table. When the user tries to execute a query that selects from the `dbo.Company` table, the user receives an error that `SELECT` permission was denied.

Table and view permissions

Tables and views represent the objects on which permissions can be granted within a database. Within those tables and views, you can additionally restrict the columns that are accessible to a given security principal (user or login). SQL Server and Azure SQL Database also include row-level security, which can be used to further restrict access.

Permission	Definition
SELECT	Allows the user to view the data within the object (table or view). When denied, the user is prevented from viewing the data within the object.
INSERT	Allows the user to insert data into the object. When denied, the user is prevented from inserting data into the object.
UPDATE	Allows the user the update data within the object. When denied, the user is prevented from updating data in the object.
DELETE	Allows the user to delete data within the object. When denied, the user is prevented from deleting data from the object.

Azure SQL Database and Microsoft SQL Server have other permissions, which can be granted, revoked, or denied as needed.

Permission	Definition
CONTROL	Grants all rights to the objects. It allows the user who has this permission to perform any action they wish against the object, including deleting the object.
REFERENCES	Grants the user the ability to view the foreign keys on the object.
TAKE OWNERSHIP	Allows the user the ability to take ownership of the object.
VIEW CHANGE TRACKING	Allows the user to view the change tracking setting for the object.
VIEW DEFINITION	Allows the user to view the definition of the object.

Function and stored procedure permissions

Like tables and views, functions and stored procedures have several permissions, which can be granted or denied.

Permission	Definition
ALTER	Grants the user the ability to change the definition of the object.
CONTROL	Grants the user all rights to the object.
EXECUTE	Grants the user the ability to execute the object.
VIEW CHANGE TRACKING	Allows the user to view the change tracking setting for the object.
VIEW DEFINITION	Allows the user to view the definition of the object.

EXECUTE AS

The `EXECUTE AS [user name]`, or `EXECUTE AS [login name]` (only available in SQL Server and Azure SQL Managed Instance) commands allow for the user context to be changed. As subsequent commands and statements are executed using the new context with the permissions granted to that context.

If a user has a permission and the user no longer needs to have that permission, permissions can be removed (either grants or denies) using the `REVOKE` command. The revoke command removes any `GRANT` or `DENY` permissions for the right specified to the user specified.

Ownership Chains

A concept called chaining applies to permissions, which allow users to inherit permissions from other objects. The most common example of chaining is a function or stored procedure that accesses a table during its execution. If the procedure has the same owner as the table, the stored procedure is able to be executed and access the table, even though the user doesn't have rights to access the table directly. This access is available because the user inherits the rights to access the table from the stored procedure, but only during the execution of the stored procedure, and only within the context of the stored procedures execution.

In the example, run as a database owner or server administrator, a new user is created and added as a member of a new *SalesReader* role, which is then granted permission to select from any object and execute any procedure in the Sales schema. A stored procedure is then created in the Sales schema that accesses a table in the Production schema.

The example then changes content to be the new user and an attempt is made to select directly from the table in the Production schema.

```
USE AdventureWorks2016;
GO

CREATE USER [DP300User1] WITH PASSWORD = 'Pa55.w.rd';
GO

CREATE ROLE [SalesReader];
GO

ALTER ROLE [SalesReader] ADD MEMBER [DP300User1];
GO

GRANT SELECT, EXECUTE ON SCHEMA::Sales TO [SalesReader];
GO

CREATE OR ALTER PROCEDURE Sales.DemoProc
AS
SELECT P.Name,
       SUM(SOD.LineTotal) AS TotalSales,
       SOH.OrderDate
FROM Production.Product P
     INNER JOIN Sales.SalesOrderDetail SOD ON (SOD.ProductID = P.ProductID)
     INNER JOIN Sales.SalesOrderHeader SOH ON (SOH.SalesOrderID = SOD.SalesOrderID)
```

```

GROUP BY P.Name,
        SOH.OrderDate
ORDER BY TotalSales DESC;

GO

EXECUTE AS USER = 'DP300User1';

SELECT P.Name,
        SUM(SOD.LineTotal) AS TotalSales,
        SOH.OrderDate
FROM Production.Product P
        INNER JOIN Sales.SalesOrderDetail SOD ON (SOD.ProductID = P.ProductID)
        INNER JOIN Sales.SalesOrderHeader SOH ON (SOH.SalesOrderID = SOD.SalesOrderID)
GROUP BY P.Name,
        SOH.OrderDate
ORDER BY TotalSales DESC;

```

The query results in an error that the user *DP300User1* doesn't have `SELECT` permission, because the role that the user belongs to doesn't have any privileges in the `Production` schema. Now we can try to execute the stored procedure:

```

EXECUTE AS USER = 'DP300User1';

EXECUTE Sales.DemoProc;

```

The *DP300User1* user has `EXECUTE` permission on the stored procedure in the `Sales` schema, because the user's role has `EXECUTE` permission on the `Sales` schema. Because the table has the same owner as the procedure, we have an unbroken ownership chain, and the execution will succeed and results are returned.

Permission changes don't apply when dynamic SQL is being used within stored procedures. The reason that dynamic SQL breaks the permission chain is because the dynamic SQL is executed outside of the context of the calling stored procedure. You can see this behavior by changing the stored procedure to execute using dynamic SQL as follows.

```

CREATE OR ALTER PROCEDURE Sales.DemoProc
AS
DECLARE @sqlstring NVARCHAR(MAX)

SET @sqlstring = '
SELECT P.Name,
        SUM(SOD.LineTotal) AS TotalSales,
        SOH.OrderDate
FROM Production.Product P
        INNER JOIN Sales.SalesOrderDetail SOD ON (SOD.ProductID = P.ProductID)
        INNER JOIN Sales.SalesOrderHeader SOH ON (SOH.SalesOrderID = SOD.SalesOrderID)
GROUP BY P.Name, SOH.OrderDate'

EXECUTE sp_executesql @sqlstring
GO

```

```
--  
  
EXECUTE AS USER = 'DP300User1'  
  
EXECUTE Sales.DemoProc
```

The *DP300User1* user receives an error that the user doesn't have `SELECT` permission on the *Production.Product* table, just like the user tried to execute the query directly. Permission chains don't apply and the user account that is executing the dynamic SQL must have rights to the tables and views that are being used by the code within the dynamic SQL.

Principle of least privilege

The principle of least privilege is fairly simple. The basic idea behind the concept is that users and applications should only be given the permissions needed in order for them to complete the task. Applications should only have permissions that they need to do in order to complete the task at hand.

As an example, if an application accesses all data through stored procedures, then the application should only have the permission to execute the stored procedures, with no access to the tables.

Dynamic SQL

Dynamic SQL is a concept where a query is built programmatically. Dynamic SQL allows T-SQL statements to be generated within a stored procedure or a query itself.

```
SELECT 'BACKUP DATABASE ' + name + ' TO DISK = ''\\backup\sql1\' + name + '.bak'''  
FROM sys.databases
```

The statement generates a list of T-SQL statements to back up all of the database on the server. Typically, this generated T-SQL is executed using `sp_executesql` or passed to another program to execute.

5. Identify authentication and authorization failures

<https://learn.microsoft.com/en-us/training/modules/configure-database-authentication-authorization/6-identify-authentication-and-authorization>

Identify authentication and authorization failures

Completed

A connection failure can result from reconfiguration, firewall settings, connection timeouts, or incorrect login information. Furthermore, if some Azure SQL Database or SQL Managed Instance resources are over capacity, you won't be able to connect.

Transient fault

When heavy workloads increase in the SQL Database service, the Azure infrastructure is able to dynamically reconfigure servers, and the client application may lose connection to the database during this operation.

Transient faults occur during database reconfiguration of a planned event or an unplanned event. These events are brief and shouldn't take longer than 60 seconds to complete.

Here's a list of a few transient errors that applications may receive when connecting to Azure SQL Database:

- Can't open database "%. *ls" requested by the login. The login failed.
- Can't process request. Not enough resources to process request.
- Can't process request. Too many operations in progress for subscription "%ld".

Note

For a complete list of transient errors, see [Troubleshooting connectivity issues and other errors with Azure SQL Database and Azure SQL Managed Instance](#).

How to monitor transient connectivity errors

Error	Action
Login failures	Look for any outages during the time when the application reported the errors at Microsoft Azure Service Dashboard.
Database reaches resource limits	Monitor your database's compute and storage resources carefully, and take action when it reaches its resource limits to prevent transient failures.
Extended authentication failures	File an Azure support request through the Azure portal if your application encounters connectivity error for longer than 60 seconds or if it occurs more than once in a given day.

Retry logic

Application developers should anticipate periodic transient failures when integrating with cloud services, like Azure SQL Database, and implement a retry logic instead of displaying application errors to users. It's important to set a maximum number of retries before the program terminates.

We recommend waiting for 5 seconds at a minimum on your first retry. Each subsequential retry should increase the delay exponentially, up to a maximum of 60 seconds.

Note

If a `SELECT` statement fails with a transient error in SQL Database or SQL Managed Instance, avoid directly retrying it. Instead, retry the `SELECT` statement using a new connection.

Unable to log in to the server

When the error **Login failed for user '< User name >'** happens, the service administrator can follow the following steps:

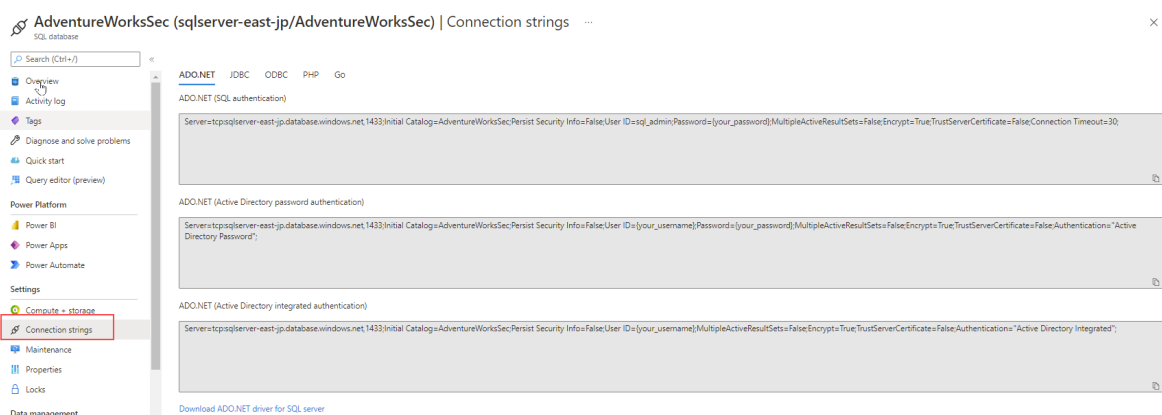
1. Check if the login is disabled by using the `sys.sql_logins` catalog view.
2. If the login is disabled, run `ALTER LOGIN <User name> ENABLE;` to enable it.
3. If the login doesn't exist, create it using the `CREATE LOGIN` statement.
4. Connect to the database you want to grant the user access to, and run the `CREATE USER` statement.
5. Either assign the user a role using the `ALTER ROLE` command, or grant the user access to one or more database objects using the `GRANT` command.

Connection string

When you receive connectivity errors, it's a good practice to make sure your connection string is working properly. This is mostly important when provisioning a new database, or after making infrastructure changes to a database service.

The Azure portal allows you to retrieve the connection string you need to interact with Azure SQL Database.

1. From the Azure portal, select **All services**, and then **SQL databases**. Filter and select your database.
2. On the blade for your database, select **Connection strings**.



3. Copy and edit the connection string by including your password, or replacing the server name as needed.
4. Reference the connection string updated in the client application.

To learn more about connectivity errors for Azure SQL Database and Azure SQL Managed Instance, see [Troubleshooting connectivity issues and other errors with Azure SQL Database and Azure SQL Managed Instance](#).

6. Exercise: Authorize Access to Azure SQL Database with Microsoft Entra ID

<https://learn.microsoft.com/en-us/training/modules/configure-database-authentication-authorization/7-exercise-authorize-access-sql-database-azure-active-directory>

Exercise: Authorize Access to Azure SQL Database with Microsoft Entra ID

Completed

Now it's your chance to configure database authentication and authorization.

In this exercise, you'll learn how to authorize access to Azure SQL Database with Microsoft Entra.

Note

To complete this exercise, you'll need an [Azure subscription](#).

Launch the exercise and follow the instructions.

[Launch Exercise](#)

7. Module assessment

<https://learn.microsoft.com/en-us/training/modules/configure-database-authentication-authorization/8-knowledge-check>

Module assessment

Completed

8. Summary

<https://learn.microsoft.com/en-us/training/modules/configure-database-authentication-authorization/9-summary>

Summary

Completed

Azure SQL offerings provide several authentication and authorization options that differ from those in SQL Server. This is because Azure SQL Database and Azure SQL Managed Instance rely on Microsoft Entra ID instead of Windows Server Active Directory.

This module explores best practices for granting permissions and the various permissions available within a database. It also delves into the concept of *least privilege*. While built-in roles in SQL Server and other database engines offer broad security privileges, many applications require more granular security on database objects.

Now that you've reviewed this module, you should be able to:

- Explain authentication options for Azure SQL offerings
- Describe security principals
- Explain object permissions
- Identify authentication and authorization failures