

Domain 2: Introduction to Microsoft Azure

Data relational data in Azure

Explore fundamental relational data concepts

1. Introduction

<https://learn.microsoft.com/en-us/training/modules/explore-relational-data-offerings/1-introduction>

Introduction

Completed

In the early years of computing systems, every application stored data in its own unique structure. When developers wanted to build applications to use that data, they had to know a lot about the particular data structure to find the data they needed. These data structures were inefficient, hard to maintain, and hard to optimize for good application performance. The *relational* database model was designed to solve the problem of multiple arbitrary data structures. The relational model provides a standard way of representing and querying data that can be used by any application. One of the key advantages of the relational database model is its use of *tables*, which are an intuitive, efficient, and flexible way to store and access structured information.

The simple yet powerful relational model is used by organizations of all types and sizes for a broad variety of information management needs. Relational databases are used to track inventories, process ecommerce transactions, manage huge amounts of mission-critical customer information, and much more. A relational database is useful for storing any information containing related data elements that must be organized in a rules-based, consistent structure.

In this module, you'll learn about the key characteristics of relational databases, and explore relational data structures.

2. Understand relational data

Understand relational data

Completed

In a relational database, you model collections of entities from the real world as *tables*. An entity can be anything for which you want to record information; typically important objects and events. For example, in a retail system example, you might create tables for customers, products, orders, and line items within an order. A table contains rows, and each row represents a single instance of an entity. In the retail scenario, each row in the customer table contains the data for a single customer, each row in the product table defines a single product, each row in the order table represents an order made by a customer, and each row in the line item table represents a product that was included in an order.

Customer						
ID	FirstName	MiddleName	LastName	Email	Address	City
1	Joe	David	Jones	joe@litware.com	1 Main St.	Seattle
2	Samir		Nadoy	samir@northwind.com	123 Elm Pl.	New York

Product		
ID	Name	Price
123	Hammer	2.99
162	Screwdriver	3.49
201	Wrench	4.25

Order		
OrderNo	OrderDate	Customer
1000	1/1/2022	1
1001	1/1/2022	2

LineItem			
OrderNo	ItemNo	ProductID	Quantity
1000	1	123	1
1000	2	201	2
1001	1	123	2

Relational tables are a format for structured data, and each row in a table has the same columns; though in some cases, not all columns need to have a value – for example, a customer table might include a **MiddleName** column; which can be empty (or *NULL*) for rows that represent customers with no middle name or whose middle name is unknown.

Each column stores data of a specific datatype. For example, an **Email** column in a **Customer** table would likely be defined to store character-based (text) data (which might be fixed or variable in length), a **Price** column in a **Product** table might be defined to store decimal numeric data, while a **Quantity** column in an **Order** table might be constrained to integer numeric values; and an **OrderDate** column in the same **Order** table would be defined to store date/time values. The available datatypes that you can use when defining a table depend on the database system you are using; though there are standard datatypes defined by the American National Standards Institute (ANSI) that are supported by most database systems.

3. Understand normalization

<https://learn.microsoft.com/en-us/training/modules/explore-relational-data-offerings/3-normalization>

Understand normalization

Completed

Normalization is a term used by database professionals for a schema design process that minimizes data duplication and enforces data integrity.

While there are many complex rules that define the process of refactoring data into various levels (or *forms*) of normalization, a simple definition for practical purposes is:

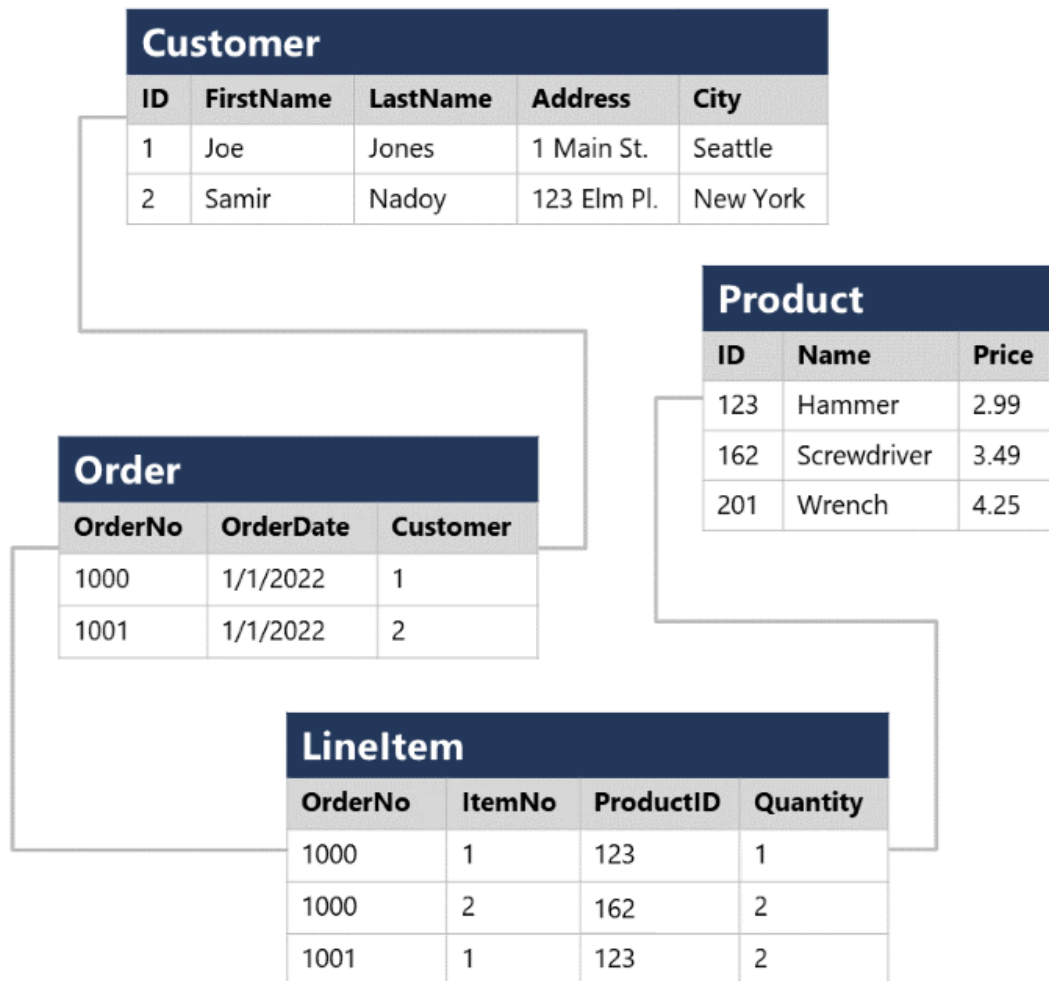
1. Separate each *entity* into its own table.
2. Separate each discrete *attribute* into its own column.
3. Uniquely identify each entity instance (row) using a *primary key*.
4. Use *foreign key* columns to link related entities.

To understand the core principles of normalization, suppose the following table represents a spreadsheet that a company uses to track its sales.

Sales Data				
OrderNo	OrderDate	Customer	Product	Quantity
1000	1/1/2022	Joe Jones, 1 Main St, Seattle	Hammer (\$2.99)	1
1000	1/1/2022	Joe Jones- 1 Main St, Seattle	Screwdriver (\$3.49)	2
1001	1/1/2022	Samir Nadoy, 123 Elm Pl, New York	Hammer (\$2.99)	2
...	...	...	...	...

Notice that the customer and product details are duplicated for each individual item sold; and that the customer name and postal address, and the product name and price are combined in the same spreadsheet cells.

Now let's look at how normalization changes the way the data is stored.



Each entity that is represented in the data (customer, product, sales order, and line item) is stored in its own table, and each discrete attribute of those entities is in its own column.

Recording each instance of an entity as a row in an entity-specific table removes duplication of data. For example, to change a customer's address, you need only modify the value in a single row.

The decomposition of attributes into individual columns ensures that each value is constrained to an appropriate data type - for example, product prices must be decimal values, while line item quantities must be integer numbers. Additionally, the creation of individual columns provides a useful level of granularity in the data for querying - for example, you can easily filter customers to those who live in a specific city.

Instances of each entity are uniquely identified by an ID or other key value, known as a *primary key*; and when one entity references another (for example, an order has an associated customer), the

primary key of the related entity is stored as a *foreign key*. You can look up the address of the customer (which is stored only once) for each record in the **Order** table by referencing the corresponding record in the **Customer** table. Typically, a relational database management system (RDBMS) can enforce referential integrity to ensure that a value entered into a foreign key field has an existing corresponding primary key in the related table – for example, preventing orders for non-existent customers.

In some cases, a key (primary or foreign) can be defined as a *composite* key based on a unique combination of multiple columns. For example, the **LineItem** table in the example above uses a unique combination of **OrderNo** and **ItemNo** to identify a line item from an individual order.

4. Explore SQL

<https://learn.microsoft.com/en-us/training/modules/explore-relational-data-offerings/4-query-with-sql>

Explore SQL

Completed

SQL stands for *Structured Query Language*, and is used to communicate with a relational database. It's the standard language for relational database management systems. SQL statements are used to perform tasks such as update data in a database, or retrieve data from a database. Some common relational database management systems that use SQL include Microsoft SQL Server, MySQL, PostgreSQL, MariaDB, and Oracle.

Note

SQL was originally standardized by the American National Standards Institute (ANSI) in 1986, and by the International Organization for Standardization (ISO) in 1987. Since then, the standard has been extended several times as relational database vendors have added new features to their systems. Additionally, most database vendors include their own proprietary extensions that are not part of the standard, which has resulted in a variety of dialects of SQL.

You can use SQL statements such as **SELECT**, **INSERT**, **UPDATE**, **DELETE**, **CREATE**, and **DROP** to accomplish almost everything that you need to do with a database. Although these SQL statements are part of the SQL standard, many database management systems also have their own additional proprietary extensions to handle the specifics of that database management system. These extensions provide functionality not covered by the SQL standard, and include areas such as security management and programmability. For example, Microsoft SQL Server, and Azure database services that are based on the SQL Server database engine, use Transact-SQL. This implementation includes proprietary extensions for writing stored procedures and triggers (application code that can be

stored in the database), and managing user accounts. PostgreSQL and MySQL also have their own versions of these features.

Some popular dialects of SQL include:

- *Transact-SQL (T-SQL)*. This version of SQL is used by Microsoft SQL Server and Azure SQL services.
- *pgSQL*. This is the dialect, with extensions implemented in PostgreSQL.
- *PL/SQL*. This is the dialect used by Oracle. PL/SQL stands for Procedural Language/SQL.

Users who plan to work specifically with a single database system should learn the intricacies of their preferred SQL dialect and platform.

Note

The SQL code examples in this module are based on the Transact-SQL dialect, unless otherwise indicated. The syntax for other dialects is generally similar, but may vary in some details.

SQL statement types

SQL statements are grouped into three main logical groups:

- Data Definition Language (DDL)
- Data Control Language (DCL)
- Data Manipulation Language (DML)

DDL statements

You use DDL statements to create, modify, and remove tables and other objects in a database (table, stored procedures, views, and so on).

The most common DDL statements are:

Statement	Description
CREATE	Create a new object in the database, such as a table or a view.
ALTER	Modify the structure of an object. For instance, altering a table to add a new column.
DROP	Remove an object from the database.
RENAME	Rename an existing object.

Warning

The **DROP** statement is very powerful. When you drop a table, all the rows in that table are lost. Unless you have a backup, you won't be able to retrieve this data.

The following example creates a new database table. The items between the parentheses specify the details of each column, including the name, the data type, whether the column must always contain a value (NOT NULL), and whether the data in the column is used to uniquely identify a row (PRIMARY KEY). Each table should have a primary key, although SQL doesn't enforce this rule.

Note

Columns marked as **NOT NULL** are referred to as *mandatory* columns. If you omit the *NOT NULL* clause, you can create rows that don't contain a value in the column. An empty column in a row is said to have a *NULL* value.

```
CREATE TABLE Product
(
    ID INT PRIMARY KEY,
    Name VARCHAR(20) NOT NULL,
    Price DECIMAL NULL
);
```

The datatypes available for columns in a table will vary between database management systems. However, most database management systems support numeric types such as INT (an integer, or whole number), DECIMAL (a decimal number), and string types such as VARCHAR (VARCHAR stands for variable length character data). For more information, see the documentation for your selected database management system.

DCL statements

Database administrators generally use DCL statements to manage access to objects in a database by granting, denying, or revoking permissions to specific users or groups.

The three main DCL statements are:

Statement	Description
GRANT	Grant permission to perform specific actions
DENY	Deny permission to perform specific actions
REVOKE	Remove a previously granted permission

For example, the following **GRANT** statement permits a user named *user1* to read, insert, and modify data in the **Product** table.

```
GRANT SELECT, INSERT, UPDATE
ON Product
```



```
TO user1;
```

DML statements

You use DML statements to manipulate the rows in tables. These statements enable you to retrieve (query) data, insert new rows, or modify existing rows. You can also delete rows if you don't need them anymore.

The four main DML statements are:

Statement	Description
SELECT	Read rows from a table
INSERT	Insert new rows into a table
UPDATE	Modify data in existing rows
DELETE	Delete existing rows

The basic form of an **INSERT** statement will insert one row at a time. By default, the **SELECT**, **UPDATE**, and **DELETE** statements are applied to every row in a table. You usually apply a **WHERE** clause with these statements to specify criteria; only rows that match these criteria will be selected, updated, or deleted.

Warning

SQL doesn't provide *are you sure?* prompts, so be careful when using DELETE or UPDATE without a WHERE clause because you can lose or modify a lot of data.

The following code is an example of a SQL statement that selects all columns (indicated by *) from the **Customer** table where the **City** column value is "Seattle":

```
SELECT *  
FROM Customer  
WHERE City = 'Seattle';
```

To retrieve only a specific subset of columns from the table, you list them in the **SELECT** clause, like this:

```
SELECT FirstName, LastName, Address, City  
FROM Customer  
WHERE City = 'Seattle';
```

If a query returns many rows, they don't necessarily appear in any specific sequence. If you want to sort the data, you can add an **ORDER BY** clause. The data will be sorted by the specified column:

```
SELECT FirstName, LastName, Address, City
FROM Customer
WHERE City = 'Seattle'
ORDER BY LastName;
```

You can also run **SELECT** statements that retrieve data from multiple tables using a **JOIN** clause. Joins indicate how the rows in one table are connected with rows in the other to determine what data to return. A typical join condition matches a foreign key from one table and its associated primary key in the other table.

The following query shows an example that joins **Customer** and **Order** tables. The query makes use of table *aliases* to abbreviate the table names when specifying which columns to retrieve in the **SELECT** clause and which columns to match in the **JOIN** clause.

```
SELECT o.OrderNo, o.OrderDate, c.Address, c.City
FROM Order AS o
JOIN Customer AS c
ON o.Customer = c.ID
```

The next example shows how to modify an existing row using SQL. It changes the value of the **Address** column in the **Customer** table for rows that have the value 1 in the **ID** column. All other rows are left unchanged:

```
UPDATE Customer
SET Address = '123 High St.'
WHERE ID = 1;
```

Warning

If you omit the **WHERE** clause, an **UPDATE** statement will modify **every** row in the table.

Use the **DELETE** statement to remove rows. You specify the table to delete from, and a **WHERE** clause that identifies the rows to be deleted:

```
DELETE FROM Product
WHERE ID = 162;
```

Warning

If you omit the **WHERE** clause, a **DELETE** statement will remove **every** row from the table.

The **INSERT** statement takes a slightly different form. You specify a table and columns in an **INTO** clause, and a list of values to be stored in these columns. Standard SQL only supports inserting one row at a time, as shown in the following example. Some dialects allow you to specify multiple **VALUES** clauses to add several rows at a time:

```
INSERT INTO Product(ID, Name, Price)
VALUES (99, 'Drill', 4.99);
```

Note

This topic describes some basic SQL statements and syntax in order to help you understand how SQL is used to work with objects in a database. If you want to learn more about querying data with SQL, review the [Get Started Querying with Transact-SQL](https://learn.microsoft.com/en-us/training/modules/explore-relational-data-offerings/5-database-objects) learning path on Microsoft Learn.

5. Describe database objects

<https://learn.microsoft.com/en-us/training/modules/explore-relational-data-offerings/5-database-objects>

Describe database objects

Completed

In addition to tables, a relational database can contain other structures that help to optimize data organization, encapsulate programmatic actions, and improve the speed of access. In this unit, you learn about three of these structures in more detail: *views*, *stored procedures*, and *indexes*.

What is a view?

A view is a virtual table based on the results of a **SELECT** query. You can think of a view as a window on specified rows in one or more underlying tables. For example, you could create a view on the **Order** and **Customer** tables that retrieves order and customer data to provide a single object that makes it easy to determine delivery addresses for orders:

```
CREATE VIEW Deliveries
AS
SELECT o.OrderNo, o.OrderDate,
       c.FirstName, c.LastName, c.Address, c.City
FROM Order AS o JOIN Customer AS c
ON o.Customer = c.ID;
```

You can query the view and filter the data in much the same way as a table. The following query finds details of orders for customers who live in Seattle:

```
SELECT OrderNo, OrderDate, LastName, Address
FROM Deliveries
```

```
WHERE City = 'Seattle';
```

What is a stored procedure?

A stored procedure defines SQL statements that can be run on command. Stored procedures are used to encapsulate programmatic logic in a database for actions that applications need to perform when working with data.

You can define a stored procedure with parameters to create a flexible solution for common actions that might need to be applied to data based on a specific key or criteria. For example, the following stored procedure could be defined to change the name of a product based on the specified product ID.

```
CREATE PROCEDURE RenameProduct
    @ProductID INT,
    @NewName VARCHAR(20)
AS
UPDATE Product
SET Name = @NewName
WHERE ID = @ProductID;
```

When a product must be renamed, you can execute the stored procedure, passing the ID of the product and the new name to be assigned:

```
EXEC RenameProduct 201, 'Spanner';
```

What is an index?

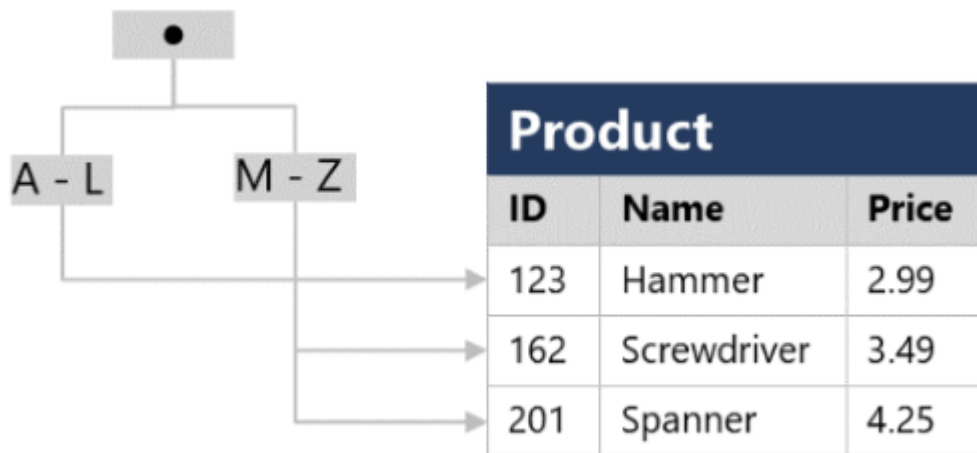
An index helps you search for data in a table. Think of an index over a table like an index at the back of a book. A book index contains a sorted set of references, with the pages on which each reference occurs. When you want to find a reference to an item in the book, you look it up through the index. You can use the page numbers in the index to go directly to the correct pages in the book. Without an index, you might have to read through the entire book to find the references you're looking for.

When you create an index in a database, you specify a column from the table, and the index contains a copy of this data in a sorted order, with pointers to the corresponding rows in the table. When the user runs a query that specifies this column in the **WHERE** clause, the database management system can use this index to fetch the data more quickly than if it had to scan through the entire table row by row.

For example, you could use the following code to create an index on the **Name** column of the **Product** table:

```
CREATE INDEX idx_ProductName
ON Product(Name);
```

The index creates a tree-based structure that the database system's query optimizer can use to quickly find rows in the **Product** table based on a specified **Name**.



For a table containing few rows, using the index is probably not any more efficient than simply reading the entire table and finding the rows requested by the query (in which case the query optimizer will ignore the index). However, when a table has many rows, indexes can dramatically improve the performance of queries.

You can create many indexes on a table. So, if you also wanted to find products based on price, creating another index on the **Price** column in the **Product** table might be useful. However, indexes aren't free. An index consumes storage space, and each time you insert, update, or delete data in a table, the indexes for that table must be maintained. This additional work can slow down insert, update, and delete operations. You must strike a balance between having indexes that speed up your queries versus the cost of performing other operations.

6. Module assessment

<https://learn.microsoft.com/en-us/training/modules/explore-relational-data-offerings/6-knowledge-check>

Module assessment

Completed

7. Summary

Summary

Completed

Relational databases are a common way for transactional applications to store and manage data. They consist of a schema of *tables*, which are linked through common key values. You use SQL to query and manipulate the data in the tables, and can enrich the database by creating objects like views, stored procedures, and indexes.

In this module you learned how to:

- Identify characteristics of relational data
- Define normalization
- Identify types of SQL statement
- Identify common relational database objects

Next steps

Now that you've learned about relational databases, consider learning more about data-related workloads on Microsoft Azure by pursuing a Microsoft certification in [Azure Data Fundamentals](#).

Explore relational database services in Azure

1. Introduction

<https://learn.microsoft.com/en-us/training/modules/explore-provision-deploy-relational-database-offerings-azure/1-introduction>

Introduction

Completed

Azure supports multiple database services, enabling you to run popular relational database management systems, such as SQL Server, PostgreSQL, and MySQL, in the cloud.

Most Azure database services are fully managed, freeing up valuable time you'd otherwise spend managing your database. Enterprise-grade performance with built-in high availability means you can scale quickly and reach global distribution without worrying about costly downtime. Developers can take advantage of industry-leading innovations such as built-in security with automatic monitoring and threat detection, automatic tuning for improved performance. On top of all of these features, is guaranteed availability.

In this module, you explore the options available for relational database services in Azure.

2. Describe Azure SQL services and capabilities

<https://learn.microsoft.com/en-us/training/modules/explore-provision-deploy-relational-database-offerings-azure/2-azure-sql>

Describe Azure SQL services and capabilities

Completed

Azure SQL is a collective term for a family of Microsoft SQL Server based database services in Azure. Specific Azure SQL services include:

- **SQL Server on Azure Virtual Machines (VMs)** - A virtual machine running in Azure with an installation of SQL Server. The use of a VM makes this option an infrastructure-as-a-service (IaaS) solution that virtualizes hardware infrastructure for compute, storage, and networking in Azure; making it a great option for "lift and shift" migration of existing on-premises SQL Server installations to the cloud.
- **Azure SQL Managed Instance** - A platform-as-a-service (PaaS) option that provides near-100% compatibility with on-premises SQL Server instances while abstracting the underlying hardware and operating system. The service includes automated software update management, backups, and other maintenance tasks, reducing the administrative burden of supporting a database server instance.
- **Azure SQL Database** - A fully managed, highly scalable PaaS database service that is designed for the cloud. This service includes the core database-level capabilities of on-premises SQL Server, and is a good option when you need to create a new application in the cloud.

Compare Azure SQL services

--	SQL Server on Azure VMs	Azure SQL Managed Instance	Azure SQL Database
			
Type of cloud service	IaaS	PaaS	PaaS
SQL Server compatibility	Fully compatible with on-premises physical and virtualized installations. Applications and databases can easily be "lift and shift" migrated without change.	Near-100% compatibility with SQL Server. Most on-premises databases can be migrated with minimal code changes by using the Azure Database Migration service	Supports most core database-level capabilities of SQL Server. Some features depended on by an on-premises application may not be available.
Architecture	SQL Server instances are installed in a virtual machine. Each instance can support multiple databases.	Each managed instance can support multiple databases. Additionally, <i>instance pools</i> can be used to share resources efficiently across smaller instances.	You can provision a <i>single database</i> in a dedicated, managed (logical) server; or you can use an <i>elastic pool</i> to share resources across multiple databases and take advantage of on-demand scalability.
Availability	99.99%	99.99%	99.995%
Management	You must manage all aspects of the server, including operating system	Fully automated updates, backups, and recovery.	Fully automated updates, backups, and recovery.

--	SQL Server on Azure VMs	Azure SQL Managed Instance	Azure SQL Database
	and SQL Server updates, configuration, backups, and other maintenance tasks.		
Use cases	Use this option when you need to migrate or extend an on-premises SQL Server solution and retain full control over all aspects of server and database configuration.	Use this option for most cloud migration scenarios, particularly when you need minimal changes to existing applications.	Use this option for new cloud solutions, or to migrate applications that have minimal instance-level dependencies.

SQL Server on Azure Virtual Machines

SQL Server on Virtual Machines enables you to use full versions of SQL Server in the Cloud without having to manage any on-premises hardware. This is an example of the IaaS approach.

SQL Server running on an Azure virtual machine effectively replicates the database running on real on-premises hardware. Migrating from the system running on-premises to an Azure virtual machine is no different than moving the databases from one on-premises server to another.

This approach is suitable for migrations and applications requiring access to operating system features that might be unsupported at the PaaS level. SQL virtual machines are *lift-and-shift* ready for existing applications that require fast migration to the cloud with minimal changes. You can also use SQL Server on Azure VMs to extend existing on-premises applications to the cloud in hybrid deployments.

Note

A *hybrid deployment* is a system where part of the operation runs on-premises, and part in the cloud. Your database might be part of a larger system that runs on-premises, although the database elements might be hosted in the cloud.

You can use SQL Server in a virtual machine to develop and test traditional SQL Server applications. With a virtual machine, you have the full administrative rights over the DBMS and operating system. It's a perfect choice when an organization already has IT resources available to maintain the virtual machines.

These capabilities enable you to:

- Create rapid development and test scenarios when you don't want to buy on-premises non-production SQL Server hardware.

- Become lift-and-shift ready for existing applications that require fast migration to the cloud with minimal changes or no changes.
- Scale up the platform on which SQL Server is running, by allocating more memory, CPU power, and disk space to the virtual machine. You can quickly resize an Azure virtual machine without the requirement that you reinstall the software that is running on it.

Business benefits

Running SQL Server on virtual machines allows you to meet unique and diverse business needs through a combination of on-premises and cloud-hosted deployments, while using the same set of server products, development tools, and expertise across these environments.

It's not always easy for businesses to switch their DBMS to a fully managed service. There may be specific requirements that must be satisfied in order to migrate to a managed service that requires making changes to the database and the applications that use it. For this reason, using virtual machines can offer a solution, but using them doesn't eliminate the need to administer your DBMS as carefully as you would on-premises.

Azure SQL Managed Instance

Azure SQL Managed instance effectively runs a fully controllable instance of SQL Server in the cloud. You can install multiple databases on the same instance. You have complete control over this instance, much as you would for an on-premises server. SQL Managed Instance automates backups, software patching, database monitoring, and other general tasks, but you have full control over security and resource allocation for your databases. You can find detailed information at [What is Azure SQL Managed Instance?](#).

Managed instances depend on other Azure services such as Azure Storage for backups, Azure Event Hubs for telemetry, Microsoft Entra ID for authentication, Azure Key Vault for Transparent Data Encryption (TDE) and a couple of Azure platform services that provide security and supportability features. The managed instances make connections to these services.

All communications are encrypted and signed using certificates. To check the trustworthiness of communicating parties, managed instances constantly verify these certificates through certificate revocation lists. If the certificates are revoked, the managed instance closes the connections to protect the data.

Use cases

Consider Azure SQL Managed Instance if you want to *lift-and-shift* an on-premises SQL Server instance and all its databases to the cloud, without incurring the management overhead of running SQL Server on a virtual machine.

Azure SQL Managed Instance provides features not available in Azure SQL Database (discussed below). If your system uses features such as linked servers, Service Broker (a message processing system that can be used to distribute work across servers), or Database Mail (which enables your

database to send email messages to users), then you should use managed instance. To check compatibility with an existing on-premises system, you can install [Data Migration Assistant \(DMA\)](#). This tool analyzes your databases on SQL Server and reports any issues that could block migration to a managed instance.

Business benefits

Azure SQL Managed Instance enables a system administrator to spend less time on administrative tasks because the service either performs them for you or greatly simplifies those tasks. Automated tasks include operating system and database management system software installation and patching, dynamic instance resizing and configuration, backups, database replication (including system databases), high availability configuration, and configuration of health and performance monitoring data streams.

Azure SQL Managed Instance has near 100% compatibility with SQL Server Enterprise Edition, running on-premises.

Azure SQL Managed Instance supports SQL Server Database engine logins and logins integrated with Microsoft Entra ID. SQL Server Database engine logins include a username and a password. You must enter your credentials each time you connect to the server. Microsoft Entra logins use the credentials associated with your current computer sign-in, and you don't need to provide them each time you connect to the server.

Azure SQL Database

Azure SQL Database is a PaaS offering from Microsoft. You create a managed database server in the cloud, and then deploy your databases on this server.

Note

A SQL Database server is a logical construct that acts as a central administrative point for multiple single or pooled databases, logins, firewall rules, auditing rules, threat detection policies, and failover groups.

Azure SQL Database is available as a *Single Database* or an *Elastic Pool*.

Single Database

This option enables you to quickly set up and run a single SQL Server database. You create and run a database server in the cloud, and you access your database through this server. Microsoft manages the server, so all you have to do is configure the database, create your tables, and populate them with your data. You can scale the database if you need more storage space, memory, or processing power. By default, resources are preallocated, and you're charged per hour for the resources you've requested. You can also specify a *serverless* configuration. In this configuration, Microsoft creates its own server, which might be shared by databases belonging to other Azure subscribers. Microsoft

ensures the privacy of your database. Your database automatically scales and resources are allocated or deallocated as required.

Elastic Pool

This option is similar to *Single Database*, except that by default multiple databases can share the same resources, such as memory, data storage space, and processing power through multiple-tenancy. The resources are referred to as a *pool*. You create the pool, and only your databases can use the pool. This model is useful if you have databases with resource requirements that vary over time, and can help you to reduce costs. For example, your payroll database might require plenty of CPU power at the end of each month as you handle payroll processing, but at other times the database might become much less active. You might have another database that is used for running reports. This database might become active for several days in the middle of the month as management reports are generated, but with a lighter load at other times. Elastic Pool enables you to use the resources available in the pool, and then release the resources once processing has completed.

Use cases

Azure SQL Database gives you the best option for low cost with minimal administration. It isn't fully compatible with on-premises SQL Server installations. It's often used in new cloud projects where the application design can accommodate any required changes to your applications.

Note

You can use the Data Migration Assistant to detect compatibility issues with your databases that can impact database functionality in Azure SQL Database. For more information, see [Overview of Data Migration Assistant](#).

Azure SQL Database is often used for:

- Modern cloud applications that need to use the latest stable SQL Server features.
- Applications that require high availability.
- Systems with a variable load that need the database server to scale up and down quickly.

Business benefits

Azure SQL Database automatically updates and patches the SQL Server software to ensure that you're always running the latest and most secure version of the service.

The scalability features of Azure SQL Database ensure that you can increase the resources available to store and process data without having to perform a costly manual upgrade.

The service provides high availability guarantees, to ensure that your databases are available at least 99.995% of the time. Azure SQL Database supports point-in-time restore, enabling you to recover a

database to the state it was in at any point in the past. Databases can be replicated to different regions to provide more resiliency and disaster recovery.

Advanced threat protection provides advanced security capabilities, such as vulnerability assessments, to help detect and remediate potential security problems with your databases. Threat protection also detects anomalous activities that indicate unusual and potentially harmful attempts to access or exploit your database. It continuously monitors your database for suspicious activities, and provides immediate security alerts on potential vulnerabilities, SQL injection attacks, and anomalous database access patterns. Threat detection alerts provide details of the suspicious activity, and recommend action on how to investigate and mitigate the threat.

Auditing tracks database events and writes them to an audit log in your Azure storage account. Auditing can help you maintain regulatory compliance, understand database activity, and gain insight into discrepancies and anomalies that might indicate business concerns or suspected security violations.

SQL Database helps secure your data by providing encryption that protects data that is stored in the database (*at rest*) and while it's being transferred across the network (*in motion*).

3. Describe Azure services for open-source databases

<https://learn.microsoft.com/en-us/training/modules/explore-provision-deploy-relational-database-offerings-azure/3-azure-database-open-source>

Describe Azure services for open-source databases

Completed

In addition to Azure SQL services, Azure data services are available for other popular relational database systems, including MySQL and PostgreSQL. The primary reason for these services is to enable organizations that use them in on-premises apps to move to Azure quickly, without making significant changes to their applications.

What are MySQL and PostgreSQL?

MySQL and PostgreSQL are relational database management systems that are tailored for different specializations.

MySQL started life as a simple-to-use open-source database management system. It's the leading open source relational database for *Linux, Apache, MySQL, and PHP* (LAMP) stack apps. It's available

in several editions; Community, Standard, and Enterprise. The Community edition is available free-of-charge, and has historically been popular as a database management system for web applications, running under Linux. Versions are also available for Windows. Standard edition offers higher performance, and uses a different technology for storing data. Enterprise edition provides a comprehensive set of tools and features, including enhanced security, availability, and scalability. The Standard and Enterprise editions are the versions most frequently used by commercial organizations, although these versions of the software aren't free.

PostgreSQL is a hybrid relational-object database. You can store data in relational tables, but a PostgreSQL database also enables you to store custom data types, with their own non-relational properties. The database management system is extensible; you can add code modules to the database, which can be run by queries. Another key feature is the ability to store and manipulate geometric data, such as lines, circles, and polygons.

PostgreSQL has its own query language called *pgsql*. This language is a variant of the standard relational query language, SQL, with features that enable you to write stored procedures that run inside the database.

Azure Database for MySQL



Azure Database for MySQL is a PaaS implementation of MySQL in the Azure cloud, based on the MySQL Community Edition.

The Azure Database for MySQL service includes high availability at no additional cost, and scalability as required. You only pay for what you use. Automatic backups are provided, with point-in-time restore.

The server provides connection security to enforce firewall rules and, optionally, require SSL connections. Many server parameters enable you to configure server settings such as lock modes, maximum number of connections, and timeouts.

Azure Database for MySQL provides a global database system that scales up to large databases without the need to manage hardware, network components, virtual servers, software patches, and other underlying components.

Certain operations aren't available with Azure Database for MySQL. These functions are primarily concerned with security and administration. Azure manages these aspects of the database server itself.

Benefits of Azure Database for MySQL

You get the following features with Azure Database for MySQL:

- High availability features built-in.
- Predictable performance.
- Easy scaling that responds quickly to demand.
- Secure data, both at rest and in motion.
- Automatic backups and point-in-time restore for the last 35 days.
- Enterprise-level security and compliance with legislation.

The system uses pay-as-you-go pricing so you only pay for what you use.

Azure Database for MySQL servers provides monitoring functionality to add alerts, and to view metrics and logs.

Azure Database for MySQL Flexible Server

The flexible-server deployment option is a fully managed database service designed to provide more granular control and flexibility over database management functions and configuration settings. It provides cost optimization controls and is the recommended deployment option for new workloads.

Azure Database for PostgreSQL



If you prefer PostgreSQL, you can choose Azure Database for PostgreSQL to run a PaaS implementation of PostgreSQL in the Azure Cloud. This service provides the same availability, performance, scaling, security, and administrative benefits as the MySQL service.

Some features of on-premises PostgreSQL databases aren't available in Azure Database for PostgreSQL. These features are mostly concerned with the extensions that users can add to a database to perform specialized tasks, such as writing stored procedures in various programming languages (other than psql, which is available), and interacting directly with the operating system. A core set of the most frequently used extensions is supported, and the list of available extensions is under continuous review.

Azure Database for PostgreSQL Flexible Server

The flexible-server deployment option for PostgreSQL is a fully managed database service. It provides a high level of control and server configuration customizations, and provides cost optimization controls.

Benefits of Azure Database for PostgreSQL

Azure Database for PostgreSQL is a highly available service. It contains built-in failure detection and failover mechanisms.

Users of PostgreSQL are familiar with the **pgAdmin** tool, which you can use to manage and monitor a PostgreSQL database. You can continue to use this tool to connect to Azure Database for PostgreSQL. However, some server-focused functionality, such as performing server backup and restore, aren't available because the server is managed and maintained by Microsoft.

Azure Database for PostgreSQL records information about queries run against databases on the server, and saves them in a database named *azure_sys*. You query the *query_store.qs_view* view to see this information, and use it to monitor the queries that users are running. This information can prove invaluable if you need to fine-tune the queries performed by your applications.

4. Exercise: Explore Azure relational database services

<https://learn.microsoft.com/en-us/training/modules/explore-provision-deploy-relational-database-offerings-azure/4-exercise-provision-relational-azure-data-services>

Exercise: Explore Azure relational database services

Completed

Now it's your opportunity to explore relational database services in Azure.

Note

To complete this lab, you will need an [Azure subscription](#) in which you have administrative access.

Launch the exercise and follow the instructions to explore Azure SQL Database.

[Launch Exercise](#)

Launch the exercise and follow the instructions to explore Azure Database for PostgreSQL.

[Launch Exercise](#)

Launch the exercise and follow the instructions to explore Azure Database for MySQL.

[Launch Exercise](#)

5. Module assessment

<https://learn.microsoft.com/en-us/training/modules/explore-provision-deploy-relational-database-offerings-azure/5-knowledge-check>

Module assessment

Completed

6. Summary

<https://learn.microsoft.com/en-us/training/modules/explore-provision-deploy-relational-database-offerings-azure/6-summary>

Summary

Completed

Azure supports a range of database services that you can use to support new cloud applications or migrate existing applications to the cloud.

In this module, you learned how to:

- Identify options for Azure SQL services
- Identify options for open-source databases in Azure
- Provision a database service on Azure

Next steps

Now that you've learned about Azure relational database services, consider learning more about data-related workloads on Azure by pursuing a Microsoft certification in [Azure Data Fundamentals](#).