**DP 900 Azure Data Fundamentals(Microsoft Learn)**

**1st Chapter:-Explore Core Data Concepts**

**What is data?**

Data is a collection of facts such as numbers, descriptions, and observations used in decision making. You can classify data as structured, semi-structured, or unstructured.

Structured data is typically tabular data that is represented by rows and columns in a database. Databases that hold tables in this form are called *relational databases* (the mathematical term *relation* refers to an organized set of data held as a table). Each row in a table has the same set of columns. The image below illustrates an example showing two tables in an ecommerce database. The first table contains the details of customers for an organization, and the second holds information about products that the organization sells.

*Semi-structured* data is information that doesn't reside in a relational database but still has some structure to it. Examples include documents held in *JavaScript Object Notation* (JSON) format. The example below shows a pair of documents representing customer information. In both cases, each customer document includes child documents containing the name and address, but the fields in these child documents vary between customers.

JSONCopy

## Document 1 ##

{

"customerID": "103248",

"name":

{

"first": "AAA",

"last": "BBB"

},

"address":

{

"street": "Main Street",

"number": "101",

"city": "Acity",

"state": "NY"

},

"ccOnFile": "yes",

"firstOrder": "02/28/2003"

}

## Document 2 ##

{

"customerID": "103249",

"name":

{

"title": "Mr",

"forename": "AAA",

"lastname": "BBB"

},

"address":

{

"street": "Another Street",

"number": "202",

"city": "Bcity",

"county": "Gloucestershire",

"country-region": "UK"

},

"ccOnFile": "yes"

}

There are other types of semi-structured data as well. Examples include *key-value* stores and *graph* databases.

A key-value database stores Associative arrays. In those arrays, a Key serves as a unique identifier to retrieve a specific value. Those values can be anything from a number or a string to a complex object, like a JSON file.

A key-value database stores data as a single collection without structure or relation. That makes them different to a relational database where tables are made up of rows and columns with predefined data types.

You can use a graph database to store and query information about complex relationships. A graph contains nodes (information about objects), and edges (information about the relationships between objects)..

Not all data is structured or even semi-structured. For example, audio and video files, and binary data files might not have a specific structure. They're referred to as *unstructured* data.

**How is data defined, stored, and accessed in cloud computing?**

Depending on the type of data such as structured, semi-structured, or unstructured, data will be stored differently. Structured data is typically stored in a relational database such as SQL Server or Azure SQL Database. Azure SQL Database is a service that runs in the cloud. You can use it to create and access relational tables. The service is managed and run by Azure, you just specify that you want a database server to be created. The act of setting up the database server is called *provisioning*.

You can provision other services as well in Azure. For example, if you want to store unstructured data such as video or audio files, you can use Azure Blob storage (*Blob* is an acronym for Binary Large Object). If you want to store semi-structured data such as documents, you can use a service such as Azure Cosmos DB.

After your service is provisioned, the service needs to be configured so that users can be given access to the data. You can typically define several levels of access.

* *Read-only* access means the users can read data but can't modify any existing data or create new data.
* *Read/write* access gives users the ability to view and modify existing data.
* *Owner* privilege gives full access to the data including managing the security like adding new users and removing access to existing users.

You can also define which users should be allowed to access the data in the first place. If the data is sensitive (or secret), you may want to restrict access to a few select users.

In the example where you're a data analyst for a large consumer organization you have decided to give read-only access to the whole management team. The management team have no need to modify data, but have security clearance to see any data. Read-write access is given to the app that salespeople use to record sales. The individual users won't need to access the system directly, but will make edits via their app. Data analysts and data managers will have owner privileges because they need to manage the access of other users and administer the system.

**Describe data processing solutions**

Data processing solutions often fall into one of two broad categories: analytical systems, and transaction processing systems.

**What is a transactional system?**

A transactional system is often what most people consider the primary function of business computing. A transactional system records *transactions*. A transaction could be financial, such as the movement of money between accounts in a banking system, or it might be part of a retail system, tracking payments for goods and services from customers. Think of a transaction as a small, discrete, unit of work.

Transactional systems are often high-volume, sometimes handling many millions of transactions in a single day. The data being processed has to be accessible very quickly. The work performed by transactional systems is often referred to as Online Transactional Processing (OLTP).

To support fast processing, the data in a transactional system is often divided into small pieces. For example, if you're using a relational system each table involved in a transaction only contains the columns necessary to perform the transactional task. In the bank transfer example, a table holding information about the funds in the account might only contain the account number and the current balance. Other tables not involved in the transfer operation would hold information such as the name and address of the customer, and the account history. Splitting tables out into separate groups of columns like this is called *normalization*. The next unit discusses this process in more detail. Normalization can enable a transactional system to cache much of the information required to perform transactions in memory, and speed throughput.

While normalization enables fast throughput for transactions, it can make querying more complex. Queries involving normalized tables will frequently need to join the data held across several tables back together again. This can make it difficult for business users who might need to examine the data.

**What is an analytical system?**

In contrast to systems designed to support OLTP, an analytical system is designed to support business users who need to query data and gain a *big picture* view of the information held in a database.

Analytical systems are concerned with capturing raw data, and using it to generate insights. An organization can use these insights to make business decisions. For example, detailed insights for a manufacturing company might indicate trends enabling them to determine which product lines to focus on, for profitability.

Most analytical data processing systems need to perform similar tasks: data ingestion, data transformation, data querying, and data visualization. The image below illustrates the components in a typical data processing system.

* **Data Ingestion**: Data ingestion is the process of capturing the raw data. This data could be taken from control devices measuring environmental information such as temperature and pressure, point-of-sale devices recording the items purchased by a customer in a supermarket, financial data recording the movement of money between bank accounts, and weather data from weather stations. Some of this data might come from a separate OLTP system. To process and analyze this data, you must first store the data in a repository of some sort. The repository could be a file store, a document database, or even a relational database.
* **Data Transformation/Data Processing**: The raw data might not be in a format that is suitable for querying. The data might contain anomalies that should be filtered out, or it may require transforming in some way. For example, dates or addresses might need to be converted into a standard format. After data is ingested into a data repository, you may want to do some cleaning operations and remove any questionable or invalid data, or perform some aggregations such as calculating profit, margin, and other Key Performance Indicators (KPIs). KPIs are how businesses are measured for growth and performance.
* **Data Querying**: After data is ingested and transformed, you can query the data to analyze it. You may be looking for trends, or attempting to determine the cause of problems in your systems. Many database management systems provide tools to enable you to perform ad-hoc queries against your data and generate regular reports.
* **Data Visualization**: Data represented in tables such as rows and columns, or as documents, aren't always intuitive. Visualizing the data can often be useful as a tool for examining data. You can generate charts such as bar charts, line charts, plot results on geographical maps, pie charts, or illustrate how data changes over time. Microsoft offers visualization tools like Power BI to provide rich graphical representation of your data.

# Identify types of data and data storage

Completed100 XP

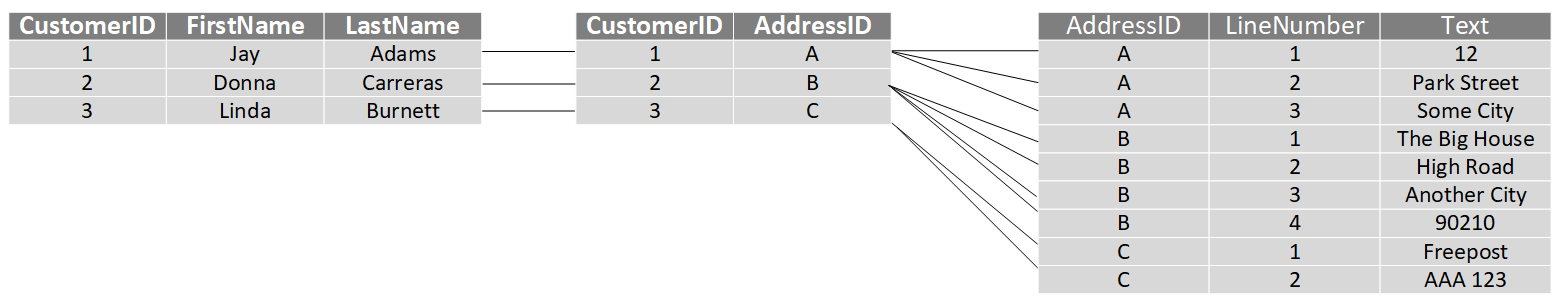
* 9 minutes

You can categorize data in many different ways, depending not only on how it's structured, but also on how the data is used. In this unit, you'll learn about the characteristics of different types of data.

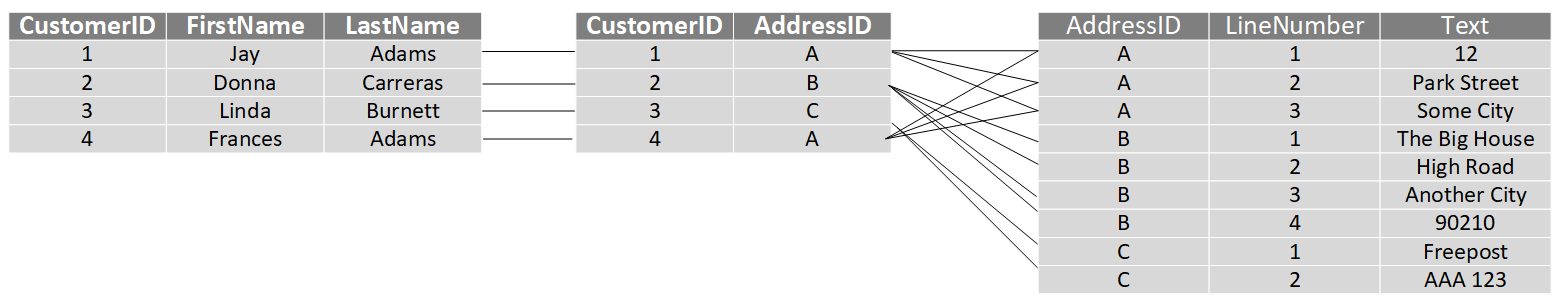
## Describe the characteristics of relational and non-relational data

Relational databases provide probably the most well-understood model for holding data. The simple structure of tables and columns makes them easy to use initially, but the rigid structure can cause some problems. For example, in a database holding customer information, how do you handle customers that have more than one address? Do you add columns to hold the details for each address? If so, how many of these columns should you add? If you allow for three addresses, what happens if a customer has only one address? What do you store in the spare columns? What then happens if you suddenly have a customer with four addresses? Similarly, what information do you store in an address (street name, house number, city, zip code)? What happens if a house has a name rather than a number, or is located somewhere that doesn't use zip codes?

You can solve these problems by using a process called [normalization](https://docs.microsoft.com/en-us/office/troubleshoot/access/database-normalization-description). Typically, the end result of the normalization process is that your data is split into a large number of narrow, well-defined tables (a narrow table is a table with few columns), with references from one table to another, as shown in the image below. However, querying the data often requires reassembling information from multiple tables by joining the data back together at run-time (illustrated by the lines in the diagram). These types of queries can be expensive.



Non-relational databases enable you to store data in a format that more closely matches the original structure. For example, in a document database, you could store the details of each customer in a single document, as shown by the example in the previous unit. Retrieving the details of a customer, including the address, is a matter of reading a single document. There are some disadvantages to using a document database though. If two customers cohabit and have the same address, in a relational database you would only need to store the address information once. In the diagram below, Jay and Frances Adams both share the same address.



In a document database, the address would be duplicated in the documents for Jay and Francis Adams. This duplication not only increases the storage required, but can also make maintenance more complex (if the address changes, you must modify it in two documents).

JSONCopy

## Document for Jay Adams ##

{

"customerID": "1",

"name":

{

"firstname": "Jay",

"lastname": "Adams"

},

"address":

{

"number": "12",

"street": "Park Street",

"city": "Some City",

}

}

## Document for Frances Adams ##

{

"customerID": "4",

"name":

{

"firstname": "Francis",

"lastname": "Adams"

},

"address":

{

"number": "12",

"street": "Park Street",

"city": "Some City",

}

}

## Describe transactional workloads

Relational and non-relational databases are suited to different workloads. A primary use of relational databases is to handle transaction processing.

A transaction is a sequence of operations that are atomic. This means that either all operations in the sequence must be completed successfully, or if something goes wrong, all operations run so far in the sequence must be undone. Bank transfers are a good example; you deduct funds from one account and credit the equivalent funds to another account. If the system fails after deducting the funds, they must be reinstated in the original account (they mustn't be lost). You can then attempt to perform the transfer again. Similarly, you shouldn't be able to credit an account twice with the same funds.

Each database transaction has a defined beginning point, followed by steps to modify the data within the database. At the end, the database either commits the changes to make them permanent, or rolls back the changes to the starting point, when the transaction can be tried again.

A transactional database must adhere to the ACID (Atomicity, Consistency, Isolation, Durability) properties to ensure that the database remains consistent while processing transactions.

* Atomicity guarantees that each transaction is treated as a single unit, which either succeeds completely, or fails completely. If any of the statements constituting a transaction fails to complete, the entire transaction fails and the database is left unchanged. An atomic system must guarantee atomicity in each and every situation, including power failures, errors, and crashes.
* Consistency ensures that a transaction can only take the data in the database from one valid state to another. A consistent database should never lose or create data in a manner that can't be accounted for. In the bank transfer example described earlier, if you add funds to an account, there must be a corresponding deduction of funds somewhere, or a record that describes where the funds have come from if they have been received externally. You can't suddenly create (or lose) money.
* Isolation ensures that concurrent execution of transactions leaves the database in the same state that would have been obtained if the transactions were executed sequentially. A concurrent process can't see the data in an inconsistent state (for example, the funds have been deducted from one account, but not yet credited to another.)
* Durability guarantees that once a transaction has been committed, it will remain committed even if there's a system failure such as a power outage or crash.

Database systems that process transactional workloads are inherently complex. They need to manage concurrent users possibly attempting to access and modify the same data at the same time, processing the transactions in isolation while keeping the database consistent and recoverable. Many systems implement relational consistency and isolation by applying locks to data when it is updated. The lock prevents another process from reading the data until the lock is released. The lock is only released when the transaction commits or rolls back. Extensive locking can lead to poor performance, while applications wait for locks to be released.

Distributed databases are widely used in many organizations. A distributed database is a database in which data is stored across different physical locations. It may be held in multiple computers located in the same physical location (for example, a datacenter), or may be dispersed over a network of interconnected computers. When compared to non-distributed database systems, any data update to a distributed database will take time to apply across multiple locations. If you require transactional consistency in this scenario, locks may be retained for a very long time, especially if there's a network failure between databases at a critical point in time. To counter this problem, many distributed database management systems relax the strict isolation requirements of transactions and implement "eventual consistency." In this form of consistency, as an application writes data, each change is recorded by one server and then propagated to the other servers in the distributed database system asynchronously. While this strategy helps to minimize latency, it can lead to temporary inconsistencies in the data. Eventual consistency is ideal where the application doesn't require any ordering guarantees. Examples include counts of shares, likes, or non-threaded comments in a social media system.

## Describe analytical workloads

Analytical workloads are typically read-only systems that store vast volumes of historical data or business metrics, such as sales performance and inventory levels. Analytical workloads are used for data analysis and decision making. Analytics are generated by aggregating the facts presented by the raw data into summaries, trends, and other kinds of “Business information.”

Analytics can be based on a snapshot of the data at a given point in time, or a series of snapshots. Decision makers usually don't require all the details of every transaction. They want the bigger picture.

An example of analytical information is a report on monthly sales. As the head of sales department, you may not need to see all daily transactions that took place (transactional information), but you definitely would like a monthly sales report to identify trends and to make decisions (analytical information).

Transactional information, however, is an integral part of analytical information. If you don't have good records of daily sales, you can't compile a useful report to identify trends. That’s why efficient handling of transactional information is very important.

# Describe the difference between batch and streaming data

Completed100 XP

* 3 minutes

Data processing is simply the conversion of raw data to meaningful information through a process. Depending on how the data is ingested into your system, you could process each data item as it arrives, or buffer the raw data and process it in groups. Processing data as it arrives is called streaming. Buffering and processing the data in groups is called batch processing.

## Understand batch processing

In batch processing, newly arriving data elements are collected into a group. The whole group is then processed at a future time as a batch. Exactly when each group is processed can be determined in a number of ways. For example, you can process data based on a scheduled time interval (for example, every hour), or it could be triggered when a certain amount of data has arrived, or as the result of some other event.

An example of batch processing is the way that votes are typically counted in elections. The votes are not entered when they are cast, but are all entered together at one time in a batch.

Advantages of batch processing include:

* Large volumes of data can be processed at a convenient time.
* It can be scheduled to run at a time when computers or systems might otherwise be idle, such as overnight, or during off-peak hours.

Disadvantages of batch processing include:

* The time delay between ingesting the data and getting the results.
* All of a batch job's input data must be ready before a batch can be processed. This means data must be carefully checked. Problems with data, errors, and program crashes that occur during batch jobs bring the whole process to a halt. The input data must be carefully checked before the job can be run again. Even minor data errors, such as typographical errors in dates, can prevent a batch job from running.

An example of an effective use of batch processing would be a connection to a mainframe system. Vast amounts of data need to be transferred into a data analysis system and the data is not real-time. An example of ineffective batch-processing would be to transfer small amounts of real-time data, such as a financial stock-ticker.

## Understand streaming and real-time data

In stream processing, each new piece of data is processed when it arrives. For example, data ingestion is inherently a streaming process.

Streaming handles data in real time. Unlike batch processing, there's no waiting until the next batch processing interval, and data is processed as individual pieces rather than being processed a batch at a time. Streaming data processing is beneficial in most scenarios where new, dynamic data is generated on a continual basis.

Examples of streaming data include:

* A financial institution tracks changes in the stock market in real time, computes value-at-risk, and automatically rebalances portfolios based on stock price movements.
* An online gaming company collects real-time data about player-game interactions, and feeds the data into its gaming platform. It then analyzes the data in real time, offers incentives and dynamic experiences to engage its players.
* A real-estate website that tracks a subset of data from consumers’ mobile devices, and makes real-time property recommendations of properties to visit based on their geo-location.

Stream processing is ideal for time-critical operations that require an instant real-time response. For example, a system that monitors a building for smoke and heat needs to trigger alarms and unlock doors to allow residents to escape immediately in the event of a fire.

## Understand differences between batch and streaming data

Apart from the way in which batch processing and streaming processing handle data, there are other differences:

* Data Scope: Batch processing can process all the data in the dataset. Stream processing typically only has access to the most recent data received, or within a rolling time window (the last 30 seconds, for example).
* Data Size: Batch processing is suitable for handling large datasets efficiently. Stream processing is intended for individual records or micro batches consisting of few records.
* Performance: The latency for batch processing is typically a few hours. Stream processing typically occurs immediately, with latency in the order of seconds or milliseconds. Latency is the time taken for the data to be received and processed.
* Analysis: You typically use batch processing for performing complex analytics. Stream processing is used for simple response functions, aggregates, or calculations such as rolling averages.

# Explore job roles in the world of data

Completed100 XP

* 4 minutes

There's a wide variety of roles involved in managing, controlling, and using data. Some roles are business-oriented, some involve more engineering, some focus on research, and some are hybrid roles that combine different aspects of data management. In this unit, you'll explore the most common job roles in the world of data. Your organization may define roles differently, or give them different names, but the roles described in this unit encapsulate the most common division of labor and responsibilities.

## What are the roles in the world of data?

There are three key job roles that deal with data in most organizations:

* **Database Administrators** manage databases, assigning permissions to users, storing backup copies of data and restore data in case of any failures.
* **Data Engineers** are vital in working with data, applying data cleaning routines, identifying business rules, and turning data into useful information.
* **Data Analysts** explore and analyze data to create visualizations and charts to enable organizations to make informed decisions.

## Azure Database Administrator role

 An Azure database administrator is responsible for the design, implementation, maintenance, and operational aspects of on-premises and cloud-based database solutions built on Azure data services and SQL Server. They're responsible for the overall availability and consistent performance and optimizations of the database solutions. They work with stakeholders to implement policies, tools, and processes for backup and recovery plans to recover following a natural disaster or human-made error.

The database administrator is also responsible for managing the security of the data in the database, granting privileges over the data, granting or denying access to users as appropriate.

## Data Engineer role

 A data engineer collaborates with stakeholders to design and implement data-related assets that include data ingestion pipelines, cleansing and transformation activities, and data stores for analytical workloads. They use a wide range of data platform technologies, including relational and nonrelational databases, file stores, and data streams.

They're also responsible for ensuring that the privacy of data is maintained within the cloud and spanning from on-premises to the cloud data stores. They also own the management and monitoring of data stores and data pipelines to ensure that data loads perform as expected.

## Data Analyst role

 A data analyst enables businesses to maximize the value of their data assets. They're responsible for designing and building scalable models, cleaning and transforming data, and enabling advanced analytics capabilities through reports and visualizations.

A data analyst processes raw data into relevant insights based on identified business requirements to deliver relevant insights.

# Review tasks and tools for database administration

Completed100 XP

* 3 minutes

Database Administrators are tasked with managing and organizing databases. A database administrator's primary job is to ensure that data is available, protected from loss, corruption, or theft, and is easily accessible as needed.

## Database Administrator tasks and responsibilities

Some of the most common roles and responsibilities of a database administrator include:

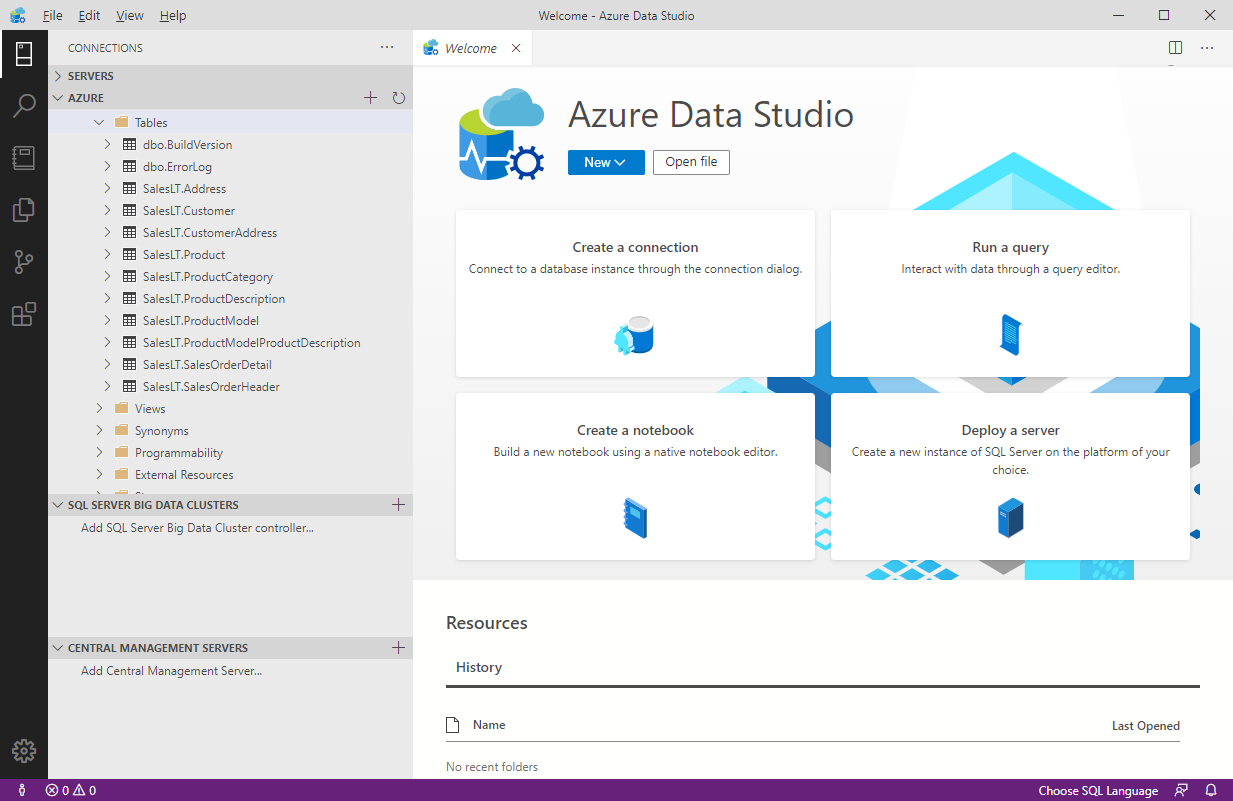
* Installing and upgrading the database server and application tools.
* Allocating system storage and planning storage requirements for the database system.
* Modifying the database structure, as necessary, from information given by application developers.
* Enrolling users and maintaining system security.
* Ensuring compliance with database vendor license agreement.
* Controlling and monitoring user access to the database.
* Monitoring and optimizing the performance of the database.
* Planning for backup and recovery of database information.
* Maintaining archived data.
* Backing up and restoring databases.
* Contacting database vendor for technical support.
* Generating various reports by querying from database as per need.
* Managing and monitoring data replication.

## Common database administrator tools

Most database management systems provide their own set of tools to assist with database administration. For example, SQL Server Database Administrators use SQL Server Management Studio for most of their day-to-day database maintenance activities. Other systems have their own database-specific interfaces, such as pgAdmin for PostgreSQL systems, or MySQL Workbench for MySQL. There are also a number of cross-platform database administration tools available. One example is Azure Data Studio.

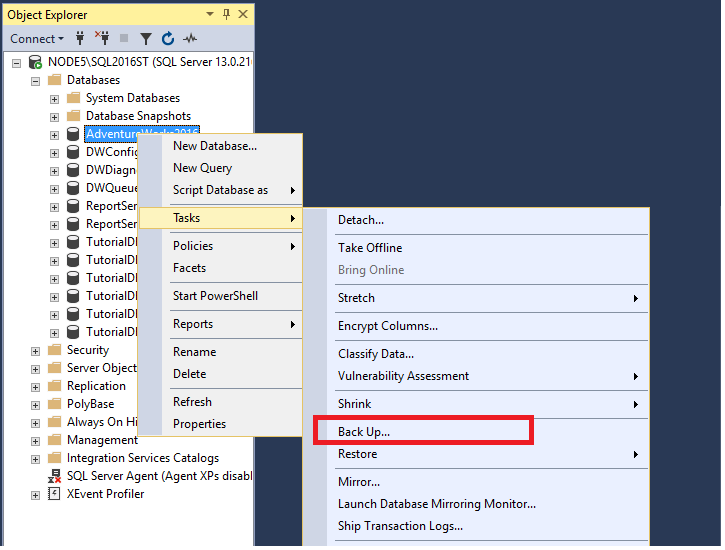
### What is Azure Data Studio?

Azure Data Studio provides a graphical user interface for managing many different database systems. It currently provides connections to on-premises SQL Server databases, Azure SQL Database, PostgreSQL, Azure SQL Data Warehouse, and SQL Server Big Data Clusters, amongst others. It's an extensible tool, and you can download and install extensions from third-party developers that connect to other systems, or provide wizards that help to automate many administrative tasks.

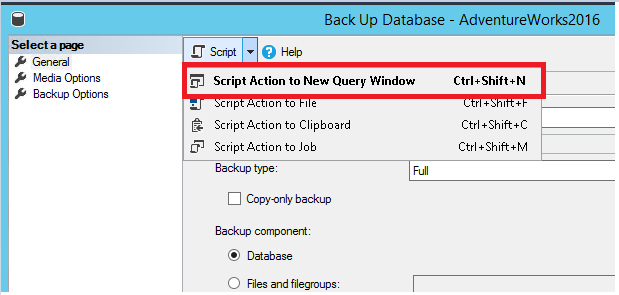


### What is SQL Server Management Studio?

SQL Server Management Studio provides a graphical interface, enabling you to query data, perform general database administration tasks, and generate scripts for automating database maintenance and support operations. The example below shows SQL Server Management Studio being used to back up a database.



A useful feature of SQL Server Management Studio is the ability to generate Transact-SQL scripts for almost all of the functionality that SQL Server Management Studio provides. This gives the DBA the ability to schedule and automate many common tasks.



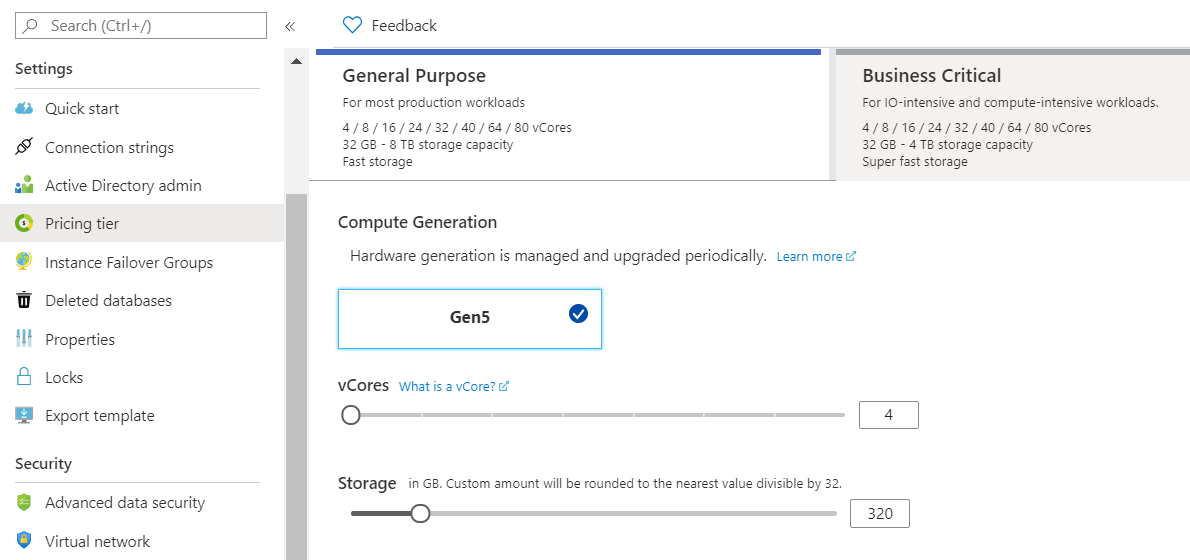
**Note**

Transact-SQL is a set of programming extensions from Microsoft that adds several features to the Structured Query Language (SQL), including transaction control, exception and error handling, row processing, and declared variables.

### Use the Azure portal to manage Azure SQL Database

Azure SQL database provides database services in Azure. It's similar to SQL Server, except that it runs in the cloud. You can manage Azure SQL database using [Azure portal](https://portal.azure.com/#home).

Typical configuration tasks such as increasing the database size, creating a new database, and deleting an existing database are done using the Azure portal.



You can use the Azure portal to dynamically manage and adjust resources such as the data storage size and the number of cores available for the database processing. These tasks would require the support of a system administrator if you were running the database on-premises.

# Review tasks and tools for data engineering

Completed100 XP

* 3 minutes

Data engineers are tasked with managing and organizing data, while also monitoring for trends or inconsistencies that will impact business goals. It’s a highly technical position, requiring experience and skills in areas like programming, mathematics, and computer science. But data engineers also need soft skills to communicate data trends to others in the organization and to help the business make use of the data it collects.

## Data Engineer tasks and responsibilities

Some of the most common roles and responsibilities of a data engineer include:

* Developing, constructing, testing, and maintaining databases and data structures.
* Aligning the data architecture with business requirements.
* Data acquisition.
* Developing processes for creating and retrieving information from data sets.
* Using programming languages and tools to examine the data.
* Identifying ways to improve data reliability, efficiency, and quality.
* Conducting research for industry and business questions.
* Deploying sophisticated analytics programs, machine learning, and statistical methods.
* Preparing data for predictive and prescriptive modeling.
* Using data to discover tasks that can be automated.

## Common data engineering tools

To master data engineering, you'll need to be familiar with a range of tools that enable you to create well-designed databases, optimized for the business processes that will be run. You must have a thorough understanding of the architecture of the database management system, the platform on which the system runs, and the business requirements for the data being stored in the database.

If you're using a relational database management system, you need to be fluent in SQL. You must be able to use SQL to create databases, tables, indexes, views, and the other objects required by the database. Many database management systems provide tools that enable you to create and run SQL scripts. For example, SQL Server Management Studio (described in the previous unit), lets you create and query tables visually, but you can also create your own SQL scripts manually.

In some cases, you may need to interact with a database from the command line. Many database management systems provide a command-line interface that supports these operations. For example, you can use the sqlcmd utility to connect to Microsoft SQL Server and Azure SQL Database, and run ad-hoc queries and commands.

As a SQL Server professional, your primary data manipulation tool might be Transact-SQL. As a data engineer you might use additional technologies, such as [Azure Databricks](https://docs.microsoft.com/en-us/azure/azure-databricks/what-is-azure-databricks), and [Azure HDInsight](https://docs.microsoft.com/en-us/azure/hdinsight/hdinsight-overview) to generate and test predictive models. If you're working in the non-relational field, you might use [Azure Cosmos DB](https://docs.microsoft.com/en-us/azure/cosmos-db/introduction) as your primary data store. To manipulate and query the data, you might use languages such as HiveQL, R, or Python.

# Review tasks and tools for data visualization and reporting

Completed100 XP

* 3 minutes

Data analysts are responsible for understanding what data actually means. A skilled data analyst will explore the data and use it to determine trends, issues, and gain other insights that might be of benefit to the company.

A large part of the data analyst role is concerned with communication and visualization. Data visualization is key to presenting large amounts of information in ways that are universally understandable or easy to interpret and spot patterns, trends, and correlations. These representations include charts, graphs, infographics, and other pictorial diagrams. Data visualization analysts use visualization tools and software to communicate information in these ways, for clients or for their own company. A good data analyst requires experience and skills in reporting tools such as Microsoft Power BI and SQL Server Reporting Services.

## Data Analyst tasks and responsibilities

The primary functions of a data analyst usually include the following:

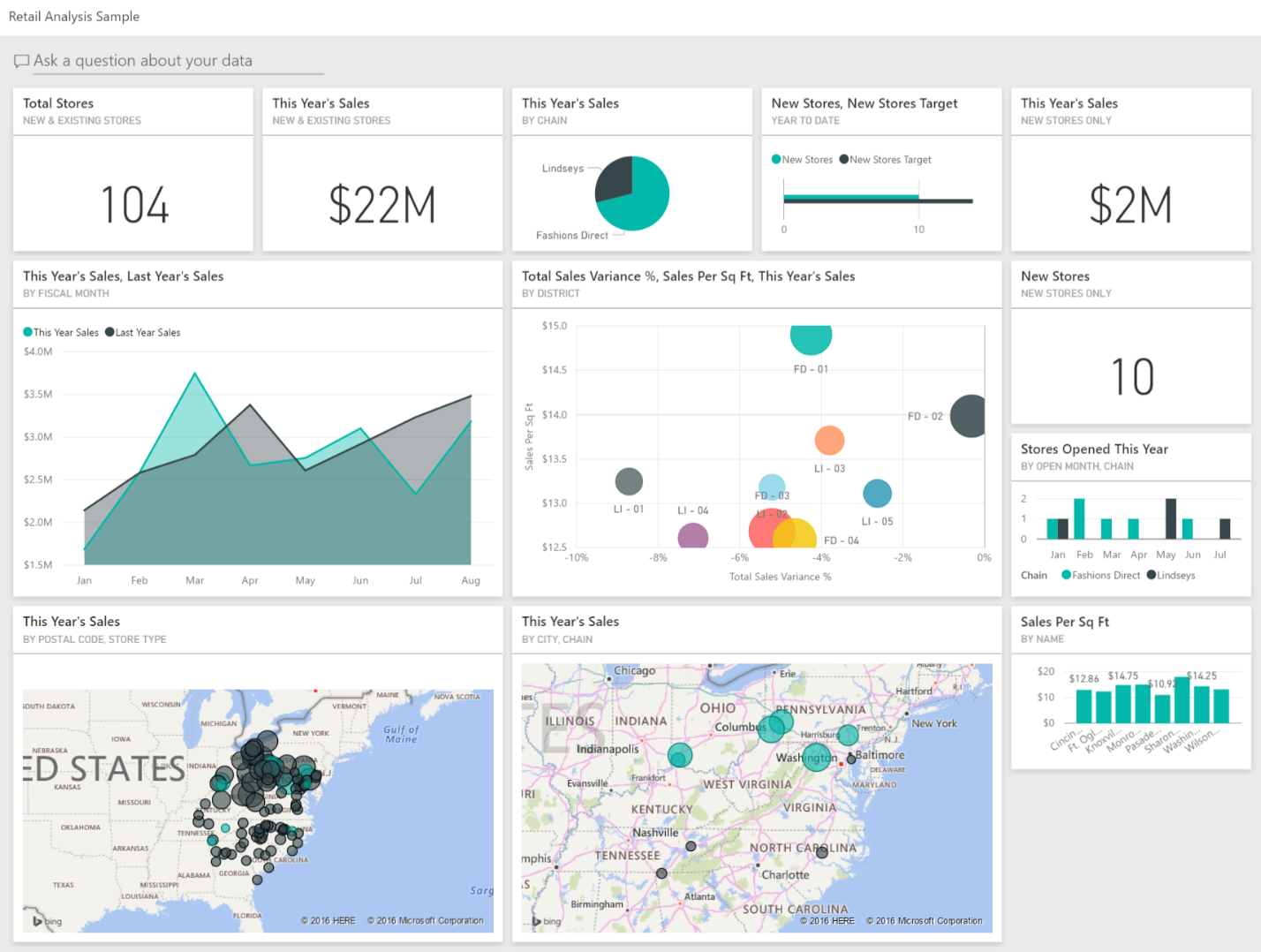
* Making large or complex data more accessible, understandable, and usable.
* Creating charts and graphs, histograms, geographical maps, and other visual models that help to explain the meaning of large volumes of data, and isolate areas of interest.
* Transforming, improving, and integrating data from many sources, depending on the business requirements.
* Combining the data result sets across multiple sources. For example, combining sales data and weather data provides a useful insight into how weather influenced sales of certain products such as ice creams.
* Finding hidden patterns using data.
* Delivering information in a useful and appealing way to users by creating rich graphical dashboards and reports.

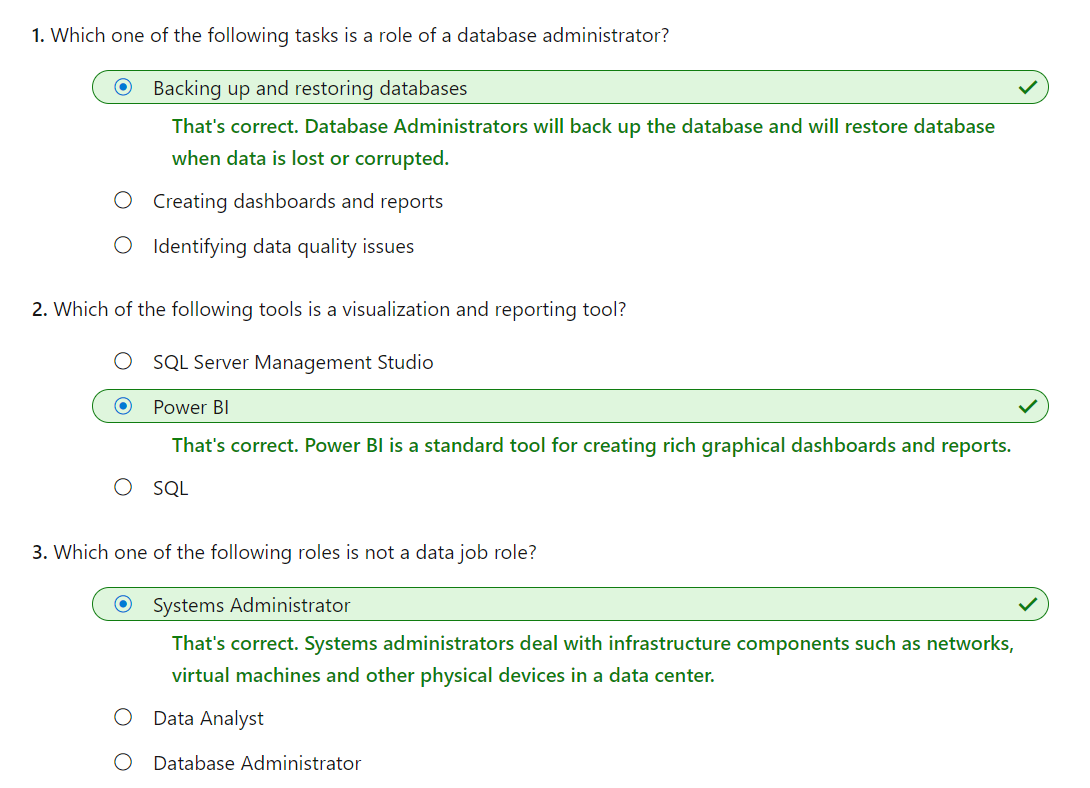
## Common data visualization tools

Traditionally, many data analysts used Microsoft Office Apps such as Microsoft Excel for creating rich visual reports. Many analysts now use Microsoft Power BI, a powerful visualization platform, to create rich, graphical dashboards and reports over data that can vary dynamically.

Power BI is a collection of software services, apps, and connectors that work together to turn your unrelated sources of data into coherent, visually immersive, and interactive insights. Your data might be held somewhere local such as an Excel spreadsheet, or in a collection of cloud-based and on-premises databases, or some other set of data sources. Power BI lets you easily connect to your data sources, discover what's important in that data, and share your findings with others in the organization.

The image below shows an example of a dashboard created using Power BI. In this example, the analyst is using Power BI to examine retail sales data for items sold across multiple stores and districts. The metrics compare this year's performance to last year's for sales, units, gross margin, and variance, as well as new-store analysis.





# Explore the characteristics of relational data

Completed100 XP

* 6 minutes

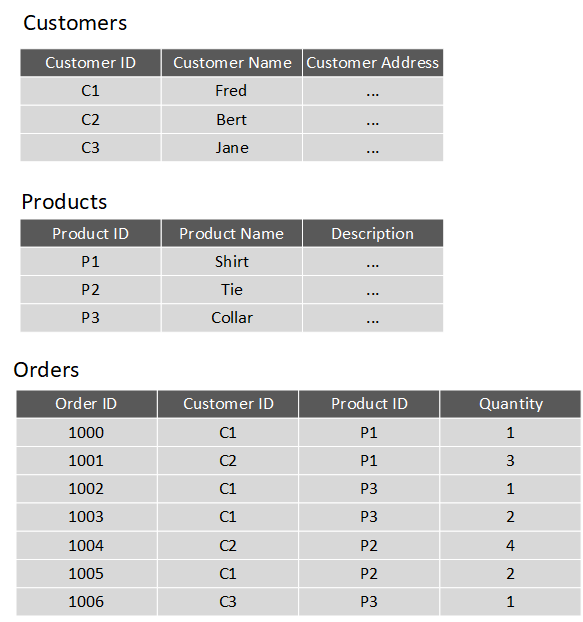
One of the main benefits of computer databases is that they make it easy to store information so it's quick and easy to find. For example, an ecommerce system might use a database to record information about the products an organization sells, and the details of customers and the orders they've placed. A relational database provides a model for storing the data, and a query capability that enables you to retrieve data quickly.

In this unit, you'll learn more about the characteristics of relational data, and how you can store this information and query it in a relational database.

## Understand the characteristics of relational data

In a relational database, you model collections of entities from the real world as tables. An entity is described as a thing about which information needs to be known or held. In the ecommerce example, you might create tables for customers, products, and orders. A table contains rows, and each row represents a single instance of an entity. In the ecommerce scenario, each row in the customers table contains the data for a single customer, each row in the products table defines a single product, and each row in the orders table represents an order made by a customer.

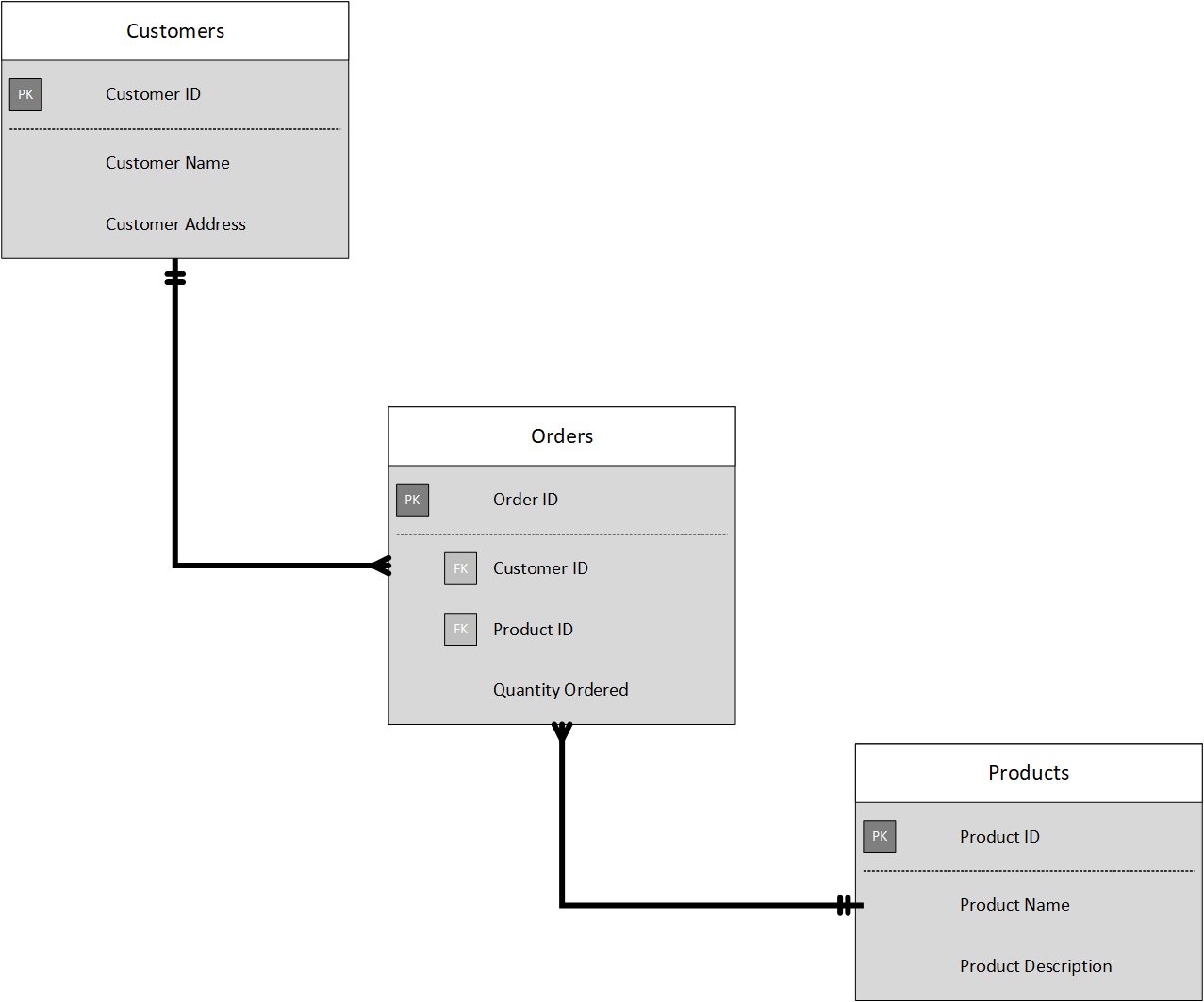
The rows in a table have one or more columns that define the properties of the entity, such as the customer name, or product ID. All rows in the same table have the same columns. Some columns are used to maintain relationships between tables. In the image below, the Orders table contains both a Customer ID and a Product ID. The Customer ID relates to the Customers table to identify the customer that placed the order, and the Product ID relates to the Products table to indicate what product was purchased.



You design a relational database by creating a data model. The model below shows the structure of the entities from the previous example. In this diagram, the columns marked PK are the Primary Key for the table. The primary key indicates the column (or combination of columns) that uniquely identify each row. Every table should have a primary key.

The diagram also shows the relationships between the tables. The lines connecting the tables indicate the type of relationship. In this case, the relationship from customers to orders is 1-to-many (one customer can place many orders, but each order is for a single customer). Similarly, the relationship between orders and products is many-to-1 (several orders might be for the same product).

The columns marked FK are Foreign Key columns. They reference, or link to, the primary key of another table, and are used to maintain the relationships between tables. A foreign key also helps to identify and prevent anomalies, such as orders for customers that don't exist in the Customers table. In the model below, the Customer ID and Product ID columns in the Orders table link to the customer that placed the order and the product that was ordered:



The main characteristics of a relational database are:

* All data is tabular. Entities are modeled as tables, each instance of an entity is a row in the table, and each property is defined as a column.
* All rows in the same table have the same set of columns.
* A table can contain any number of rows.
* A primary key uniquely identifies each row in a table. No two rows can share the same primary key.
* A foreign key references rows in another, related table. For each value in the foreign key column, there should be a row with the same value in the corresponding primary key column in the other table.

**Note**

Creating a relational database model for a large organization is not a trivial task. It can take several iterations to define tables to match the characteristics described above. Sometimes you have to split an entity into more than one table. This process is called [**normalization**](https://docs.microsoft.com/en-us/office/troubleshoot/access/database-normalization-description).

Most relational databases support Structured Query Language (SQL). You use SQL to create tables, insert, update, and delete rows in tables, and to query data. You use the CREATE TABLE command to create a table, the INSERT statement to store data in a table, the UPDATE statement to modify data in a table, and the DELETE statement to remove rows from a table. The SELECT statement retrieves data from a table. The example query below finds the details of every customer from the sample database shown above.

SQLCopy

SELECT CustomerID, CustomerName, CustomerAddress

FROM Customers

Rather than retrieve every row, you can filter data by using a WHERE clause. The next query fetches the order ID and product ID for all orders placed by customer 1.

SQLCopy

SELECT OrderID, ProductID

FROM Orders

WHERE CustomerID = 'C1'

You can combine the data from multiple tables in a query using a join operation. A join operation spans the relationships between tables, enabling you to retrieve the data from more than one table at a time. The following query retrieves the name of every customer, together with the product name and quantity for every order they've placed. Notice that each column is qualified with the table it belongs to:

SQLCopy

SELECT Customers.CustomerName, Orders.QuantityOrdered, Products.ProductName

FROM Customers JOIN Orders

ON Customers.CustomerID = Orders.CustomerID

JOIN Products

ON Orders.ProductID = Products.ProductID

You can find full details about SQL on the Microsoft website, on the [Structured Query Language (SQL)](https://docs.microsoft.com/en-us/sql/odbc/reference/structured-query-language-sql) page.

## Explore relational database use cases

You can use a relational database any time you can easily model your data as a collection of tables with a fixed set of columns. In theory, you could model almost any dataset in this way, but some scenarios lend themselves to the relational model better than others.

For example, if you have a collection of music, video, or other media files, attempting to force this data into the relational model could be difficult. You may be better off using unstructured storage, such as that available in Azure Blob storage. Similarly, social networking sites use databases to store data about millions of users, each of whom can be linked to any number of other users in a highly complex web of relationships. This type of data lends itself more to a graph database structure rather than a collection of relational tables.

Relational databases are commonly used in ecommerce systems, but one of the major use cases for using relational databases is Online Transaction Processing (OLTP). OLTP applications are focused on transaction-oriented tasks that process a very large number of transactions per minute. Relational databases are well suited for OLTP applications because they naturally support insert, update, and delete operations. A relational database can often be tuned to make these operations fast. Also, the nature of SQL makes it easy for users to perform ad-hoc queries over data.

Examples of OLTP applications that use relational databases are:

* Banking solutions
* Online retail applications
* Flight reservation systems
* Many online purchasing applications.

# Explore relational data structures

Completed100 XP

* 4 minutes

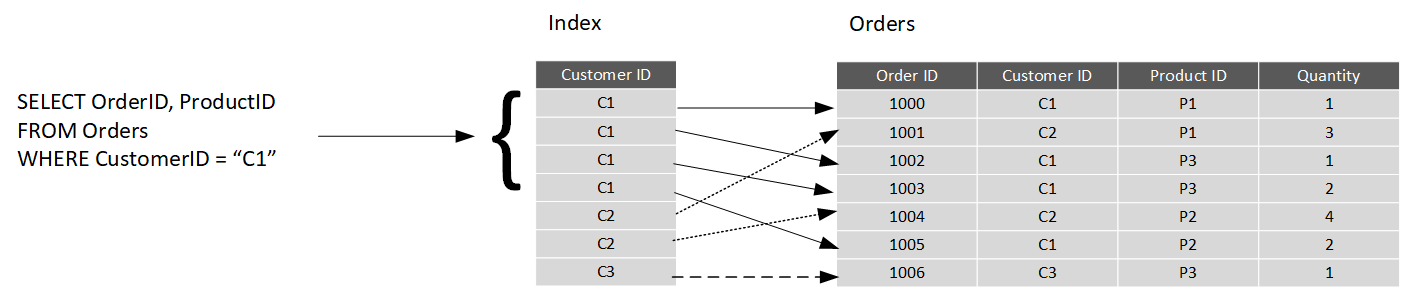
A relational database comprises a set of tables. A table can have zero (if the table is empty) or more rows. Each table has a fixed set of columns. You can define relationships between tables using primary and foreign keys, and you can access the data in tables using SQL.

Apart from tables, a typical relational database contains other structures that help to optimize data organization, and improve the speed of access. In this unit, you'll look at two of these structures in more detail: indexes and views.

## What is an index?

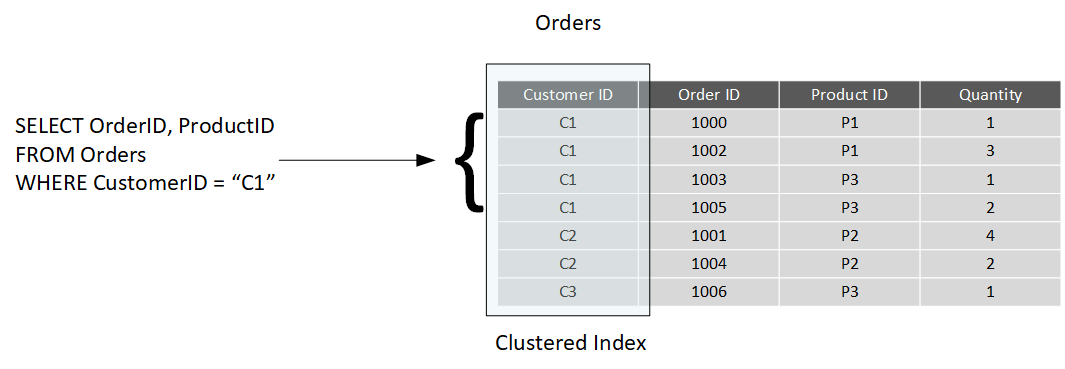
An index helps you search for data in a table. Think of an index over a table like an index at the back of a book. A book index contains a sorted set of references, with the pages on which each reference occurs. When you want to find a reference to an item in the book, you look it up through the index. 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. In the example below, the query retrieves all orders for customer C1. The Orders table has an index on the Customer ID column. The database management system can consult the index to quickly find all matching rows in the Orders table.



You can create many indexes on a table. So, if you also wanted to find all orders for a specific product, then creating another index on the Product ID column in the Orders table, would be useful. However, indexes aren't free. An index might consume additional 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, and incur additional processing charges. Therefore, when deciding which indexes to create, you must strike a balance between having indexes that speed up your queries versus the cost of performing other operations. In a table that is read only, or that contains data that is modified infrequently, more indexes will improve query performance. If a table is queried infrequently, but subject to a large number of inserts, updates, and deletes (such as a table involved in OLTP), then creating indexes on that table can slow your system down.

Some relational database management systems also support clustered indexes. A clustered index physically reorganizes a table by the index key. This arrangement can improve the performance of queries still further, because the relational database management system doesn't have to follow references from the index to find the corresponding data in the underlying table. The image below shows the Orders table with a clustered index on the Customer ID column.



In database management systems that support them, a table can only have a single clustered index.

## What is a view?

A view is a virtual table based on the result set of a query. In the simplest case, you can think of a view as a window on specified rows in an underlying table. For example, you could create a view on the Orders table that lists the orders for a specific product (in this case, product P1) like this:

SQLCopy

CREATE VIEW P1Orders AS

SELECT CustomerID, OrderID, Quantity

FROM Orders

WHERE ProductID = "P1"

You can query the view and filter the data in much the same way as a table. The following query finds the orders for customer C1 using the view. This query will only return orders for product P1 made by the customer:

SQLCopy

SELECT CustomerID, OrderID, Quantity

FROM P1Orders

WHERE CustomerID = "C1"

A view can also join tables together. If you regularly needed to find the details of customers and the products that they've ordered, you could create a view based on the join query shown in the previous unit:

SQLCopy

CREATE VIEW CustomersProducts AS

SELECT Customers.CustomerName, Orders.QuantityOrdered, Products.ProductName

FROM Customers JOIN Orders

ON Customers.CustomerID = Orders.CustomerID

JOIN Products

ON Orders.ProductID = Products.ProductID

The following query finds the customer name and product names of all orders greater than QuantityOrdered 100, using this view:

SQLCopy

SELECT CustomerName, ProductName

FROM CustomersProducts

WHERE QuantityOrdered > 100

# Choose the right platform for a relational workload

Completed100 XP

* 5 minutes

Cloud computing has grown in popularity, promising flexibility for enterprises, opportunities for saving time and money, and improving agility and scalability. On the other hand, on-premises software, installed on a company’s own servers and behind its firewall, still has its appeal. On-premises applications are reliable, secure, and allow enterprises to maintain close control.

Relational database management systems are one example of where the cloud has enabled organizations to take advantage of improved scalability. However, this scalability has to be balanced against the need for close control over the data. Data is arguably one of the most valuable assets that an organization has, and some companies aren't willing or able to hand over responsibility for protecting this data to a third party.

In this unit, you'll look at some of the advantages and disadvantages of running a database management system in the cloud.

## Compare on-premises hosting to the cloud

Whether a company places its relational workload in the cloud or whether it decides to keep it on premises, data security will always be paramount. But for those businesses in highly regulated industries, the decision might already be made for them as to whether to host their applications on-premises. Knowing that your data is located within your in-house servers and IT infrastructure might also provide more peace of mind.

Hosting a relational database on-premises requires that an enterprise not only purchases the database software, but also maintains the necessary hardware on which to run the database. The organization is responsible for maintaining the hardware and software, applying patches, backing up databases, restoring them when necessary, and generally performing all the day-to-day management required to keep the platform operational. Scalability is also a concern. If you need to scale your system, you will need to upgrade or add more servers. You then need to expand your database onto these servers. This can be a formidable task that requires you to take a database offline while the operation is performed. In the cloud, many of these operations can be handled for you by the data center staff, in many cases with no (or minimal) downtime. You're free to focus on the data itself and leave the management concerns to others (this is what you pay your Azure fees for, after all).

A cloud-based approach uses virtual technology to host a company’s applications offsite. There are no capital expenses, data can be backed up regularly, and companies only have to pay for the resources they use. For those organizations that plan aggressive expansion on a global basis, the cloud has even greater appeal because it allows you to connect with customers, partners, and other businesses anywhere with minimal effort. Additionally, cloud computing gives you nearly instant provisioning because everything is already configured. Thus, any new software that is integrated into your environment is ready to use immediately once a company has subscribed. With instant provisioning, any time spent on installation and configuration is eliminated and users can access the application right away.

## Understand IaaS and PaaS

You generally have two options when moving your operations and databases to the cloud. You can select an IaaS approach, or PaaS.

**IaaS** is an acronym for Infrastructure-as-a-Service. Azure enables you to create a virtual infrastructure in the cloud that mirrors the way an on-premises data center might work. You can create a set of virtual machines, connect them together using a virtual network, and add a range of virtual devices. In many ways, this approach is similar to the way in which you run your systems inside an organization, except that you don't have to concern yourself with buying or maintaining the hardware. However, you're still responsible for many of the day-to-day operations, such as installing and configuring the software, patching, taking backups, and restoring data when needed. You can think of IaaS as a transition to fully managed operations in the cloud; you don't have to worry about the hardware, but running and managing the software is still very much your responsibility.

You can run any software for which you have the appropriate licenses using this approach. You're not restricted to any specific database management system.

The IaaS approach is best for migrations and applications requiring operating system-level access. SQL virtual machines are lift-and-shift. That is, you can copy your on-premises solution directly to a virtual machine in the cloud. The system should work more or less exactly as before in its new location, except for some small configuration changes (changes in network addresses, for example) to take account of the change in environment.

**PaaS** stands for Platform-as-a-service. Rather than creating a virtual infrastructure, and installing and managing the database software yourself, a PaaS solution does this for you. You specify the resources that you require (based on how large you think your databases will be, the number of users, and the performance you require), and Azure automatically creates the necessary virtual machines, networks, and other devices for you. You can usually scale up or down (increase or decrease the size and number of resources) quickly, as the volume of data and the amount of work being done varies; Azure handles this scaling for you, and you don't have to manually add or remove virtual machines, or perform any other form of configuration.

Azure offers several PaaS solutions for relational databases, include Azure SQL Database, Azure Database for PostgreSQL, Azure Database for MySQL, and Azure Database for MariaDB. These services run managed versions of the database management systems on your behalf. You just connect to them, create your databases, and upload your data. However, you may find that there are some functional restrictions in place, and not every feature of your selected database management system may be available. These restrictions are often due to security issues. For example, they might not expose the underlying operating system and hardware to your applications. In these cases, you may need to rework your applications to remove any dependencies on these features.

# The image below illustrates the benefits and tradeoffs when running a database management system (in this case, SQL Server) on-premises, using virtual machines in Azure (IaaS), or using Azure SQL Database (PaaS). The same generalized considerations are true for other database management systems. Explore characteristics of non-relational data

Completed100 XP

* 7 minutes

Relational databases are an excellent tool for storing and retrieving data that has a well-known structure, containing fields that you can define in advance. In some situations, you might not have the required knowledge of the structure of your data, in advance of it arriving in your database, to record it as a neat set of rows and columns in a tabular format. This is a common scenario in systems that consume data from a wide variety of sources, such as data ingestion pipelines. In these situations, a non-relational database can prove extremely useful.

In this unit, you'll look in more detail at the common characteristics of non-relational databases. You'll learn how they enable you to capture data quickly, and model data that can vary in structure.

## What are the characteristics of non-relational data?

You use a database to model some aspect of the real-world. Entities in the real-world often have highly variable structures. For example, in an ecommerce database that stores information about customers, how many telephone numbers does a customer have? A customer might have a landline and a mobile number, but some customers might have a business number, an additional home number, and maybe several mobile numbers. Similarly, the addresses of customers might not always follow the same format; addresses for customers in different states and regions might contain different elements, such as zip codes or postal codes.

In another scenario, if you are ingesting data rapidly, you want to capture the data and save it very quickly. Processing the data and manipulating it into a set of rows in different tables in a relational database might not be appropriate at this point; you can perform these tasks at a later date. At the time of ingestion, you simply need to store the data in its original state and format.

A key aspect of non-relational databases is that they enable you to store data in a very flexible manner. Non-relational databases don't impose a schema on data. Instead, they focus on the data itself rather than how to structure it. This approach means that you can store information in a natural format, that mirrors the way in which you would consume, query and use it.

In a non-relational system, you store the information for entities in collections or containers rather than relational tables. Two entities in the same collection can have a different set of fields rather than a regular set of columns found in a relational table. The lack of a fixed schema means that each entity must be self-describing. Often this is achieved by labeling each field with the name of the data that it represents. For example, a non-relational collection of customer entities might look like this:

textCopy

## Customer 1

ID: 1

Name: Mark Hanson

Telephone: [ Home: 1-999-9999999, Business: 1-888-8888888, Cell: 1-777-7777777 ]

Address: [ Home: 121 Main Street, Some City, NY, 10110,

Business: 87 Big Building, Some City, NY, 10111 ]

## Customer 2

ID: 2

Title: Mr

Name: Jeff Hay

Telephone: [ Home: 0044-1999-333333, Mobile: 0044-17545-444444 ]

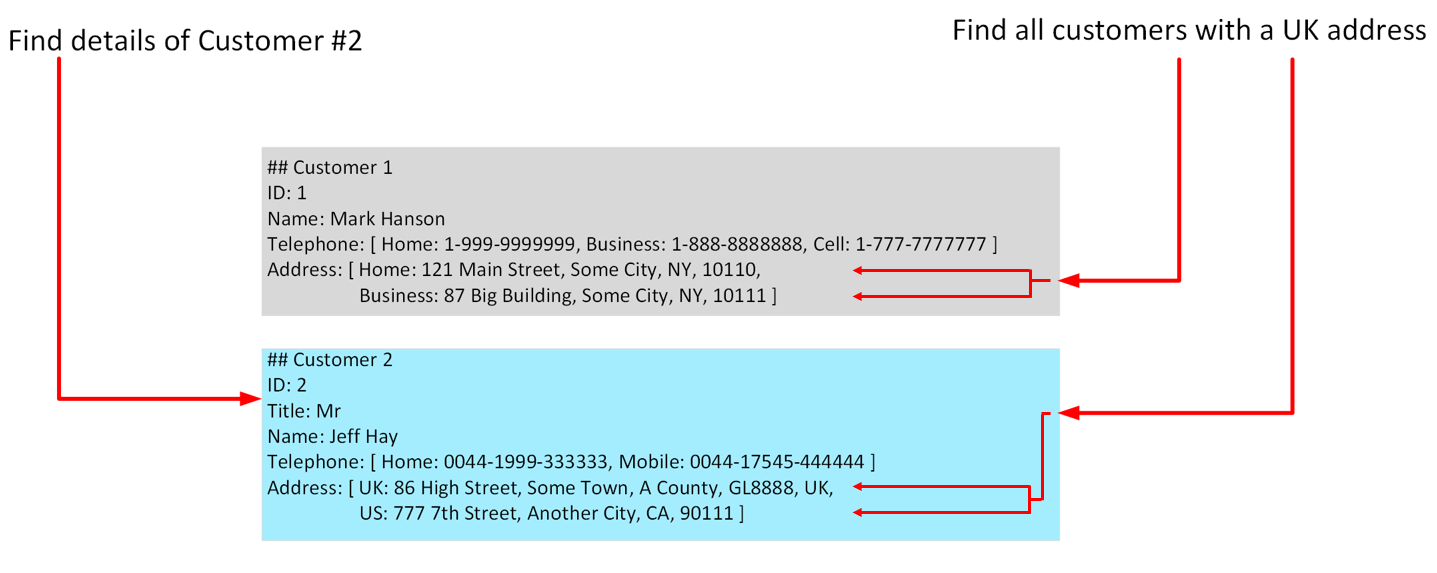
Address: [ UK: 86 High Street, Some Town, A County, GL8888, UK,

US: 777 7th Street, Another City, CA, 90111 ]

In this example, fields are prefixed with a name. Fields might also have multiple subfields, also with names. In the example, multiple subfields are denoted by enclosing them between square brackets.

Adding a new customer is a matter of inserting an entity with its fields labeled in a meaningful way. An application that queries this data must be prepared to parse the information in the entity that it retrieves.

The data retrieval capabilities of a non-relational database can vary. Each entity should have a unique key value. The entities in a collection are usually stored in key-value order. In the example above, the unique key is the ID field. The simplest type of non-relational database enables an application to either specify the unique key, or a range of keys as query criteria. In the customers example, the database would enable an application to query customers by ID only. Filtering data on other fields would require scanning the entire collection of entities, parsing each entity in turn, and then applying any query criteria to each entity to find any matches. In the example below, a query that fetches the details of a customer by ID can quickly identify which entity to retrieve. A query that attempts to find all customers with a UK address would have to iterate through every entity, and for each entity examine each field in turn. If the database contains many millions of entities, this query could take a considerable time to run.



More advanced non-relational systems support indexing, in a similar manner to an index in a relational database. Queries can then use the index to identify and fetch data based on non-key fields. Non-relational systems such as Azure Cosmos DB (a non-relational database management system available in Azure), support indexing even when the structure of the indexed data can vary from record to record. For more information, read [Indexing in Azure Cosmos DB - Overview](https://docs.microsoft.com/en-us/azure/cosmos-db/index-overview).

When you design a non-relational database, it's important to understand the capabilities of the database management system and the types of query it will have to support.

**Note**

Non-relational databases often provide their own proprietary language for managing and querying data. This language may be procedural (follow a set of commands, in order), or it may be similar to SQL; it depends on how the database is implemented by the database management system.

## Identify non-relational database use cases

Non-relational databases are highly suitable for the following scenarios:

* IoT and telematics. These systems typically ingest large amounts of data in frequent bursts of activity. Non-relational databases can store this information very quickly. The data can then be used by analytics services such as Azure Machine Learning, Azure HDInsight, and Microsoft Power BI. Additionally, you can process the data in real-time using Azure Functions that are triggered as data arrives in the database.
* Retail and marketing. Microsoft uses CosmosDB for its own ecommerce platforms that run as part of Windows Store and Xbox Live. It's also used in the retail industry for storing catalog data and for event sourcing in order processing pipelines.
* Gaming. The database tier is a crucial component of gaming applications. Modern games perform graphical processing on mobile/console clients, but rely on the cloud to deliver customized and personalized content like in-game stats, social media integration, and high-score leaderboards. Games often require single-millisecond latencies for reads and write to provide an engaging in-game experience. A game database needs to be fast and be able to handle massive spikes in request rates during new game launches and feature updates.
* Web and mobile applications. A non-relational database such as Azure Cosmos DB is commonly used within web and mobile applications, and is well suited for modeling social interactions, integrating with third-party services, and for building rich personalized experiences. The Cosmos DB SDKs (software development kits) can be used to build rich iOS and Android applications using the popular Xamarin framework.

A relational database restructures the data into a fixed format that is designed to answer specific queries. When data needs to be ingested very quickly, or the query is unknown and unconstrained, a relational database can be less suitable than a non-relational database.

# Describe types of non-relational data

Completed100 XP

* 4 minutes

Non-relational data generally falls into two categories; semi-structured and non-structured. In this unit, you'll learn about what these terms mean, and see some examples.

## What is semi-structured data?

Semi-structured data is data that contains fields. The fields don't have to be the same in every entity. You only define the fields that you need on a per-entity basis. The Customer entities shown in the previous unit are examples of semi-structured data. The data must be formatted in such a way that an application can parse and process it. One common way of doing this is to store the data for each entity as a JSON document. The term JSON stands for JavaScript Object Notation; it's the format used by JavaScript applications to store data in memory, but can also be used to read and write documents to and from files.

A JSON document is enclosed in curly brackets ({ and }). Each field has a name (a label), followed by a colon, and then the value of the field. Fields can contain simple values, or subdocuments (each starting and ending with curly brackets). Fields can also have multiple values, held as arrays and surrounded with square brackets ([ and ]). Literals, or fixed values, in a field are enclosed in quotes, and fields are separated with commas.

The example below shows the customers from the previous unit, formatted as JSON documents:

JSONCopy

{

"ID": "1",

"Name": "Mark Hanson",

"Telephone": [

{ "Home": "1-999-9999999" },

{ "Business": "1-888-8888888" },

{ "Cell": "1-777-7777777" }

],

"Address": [

{ "Home": [

{ "StreetAddress": "121 Main Street" },

{ "City": "Some City" },

{ "State": "NY" },

{ "Zip": "10110" }

] },

{ "Business": [

{ "StreetAddress": "87 Big Building" },

{ "City": "Some City" },

{ "State": "NY" },

{ "Zip": "10111" }

] }

]

}

{

"ID": "2",

"Title": "Mr",

"Name": "Jeff Hay",

"Telephone": [

{ "Home": "0044-1999-333333" },

{ "Mobile": "0044-17545-444444" }

],

"Address": [

{ "UK": [

{ "StreetAddress": "86 High Street" },

{ "Town": "Some Town" },

{ "County": "A County" },

{ "Postcode": "GL8888" },

{ "Region": "UK" }

] },

{ "US": [

{ "StreetAddress": "777 7th Street" },

{ "City": "Another City" },

{ "State": "CA" },

{ "Zip": "90111" }

] }

]

}

You're free to define whatever fields you like. The important point is that the data follows the JSON grammar. When an application reads a document, it can use a JSON parser to break up the document into its component fields and extract the individual pieces of data.

Other formats you might see include Avro, ORC, and Parquet:

* Avro is a row-based format. It was created by Apache. Each record contains a header that describes the structure of the data in the record. This header is stored as JSON. The data is stored as binary information. An application uses the information in the header to parse the binary data and extract the fields it contains. Avro is a very good format for compressing data and minimizing storage and network bandwidth requirements. This example is a subset of the header information for the previous example, formatted as Avro:

AvroCopy

{

"type": "record",

"name": "contact\_schema",

"fields": [

{

"name": "id",

"type": "int",

"doc": "ID of the contact"

},

{

"name": "name",

"type": "string",

"doc": "Name of the contact"

},

{

"name": "telephone",

"type": [

"null",

{

"type": "array",

"items": {

"type": "record",

"name": "contact\_schema.telephone",

"fields": [

{

"name": "phoneid",

"type": "int"

},

{

"name": "phonetype",

"type": [ "null", "string" ]

}

]

}

}

]

}

]

}

* ORC (Optimized Row Columnar format) organizes data into columns rather than rows. It was developed by HortonWorks for optimizing read and write operations in Apache Hive. Hive is a data warehouse system that supports fast data summarization and querying over very large datasets. Hive supports SQL-like queries over unstructured data. An ORC file contains stripes of data. Each stripe holds the data for a column or set of columns. A stripe contains an index into the rows in the stripe, the data for each row, and a footer that holds statistical information (count, sum, max, min, and so on) for each column.
* Parquet is another columnar data format. It was created by Cloudera and Twitter. A Parquet file contains row groups. Data for each column is stored together in the same row group. Each row group contains one or more chunks of data. A Parquet file includes metadata that describes the set of rows found in each chunk. An application can use this metadata to quickly locate the correct chunk for a given set of rows, and retrieve the data in the specified columns for these rows. Parquet specializes in storing and processing nested data types efficiently. It supports very efficient compression and encoding schemes.

## What is unstructured data?

Unstructured data is data that doesn't naturally contain fields. Examples include video, audio, and other media streams. Each item is an amorphous blob of binary data. You can't search for specific elements in this data.

You might choose to store data such as this in storage that is specifically designed for the purpose. In Azure, you would probably store video and audio data as block blobs in an Azure Storage account. (The term blob stands for Binary Large Object\*). A block blob only supports basic read and write operations.

You could also consider files as a form of unstructured data, although in some cases a file might include metadata that indicates what type of file it is (photograph, Word document, Excel spreadsheet, and so on), owner, and other elements that could be stored as fields. However, the main content of the file is unstructured.

# Describe types of non-relational and NoSQL databases

Completed100 XP

* 10 minutes

Non-relational data is an all-encompassing term that means anything not structured as a set of tables. There are many different types of non-structured data, and the information is used for a wide variety of purposes. Consequently, there are many different types of non-relational database management systems, each oriented towards a specific set of scenarios.

In this unit, you'll learn about some of the most common types of non-relational databases.

## What is NoSQL?

You might see the term NoSQL when reading about non-relational databases. NoSQL is a rather loose term that simply means non-relational. There's some debate about whether it's intended to imply Not SQL, or Not Only SQL; some non-relational databases support a version of SQL adapted for documents rather than tables (examples include Azure Cosmos DB).

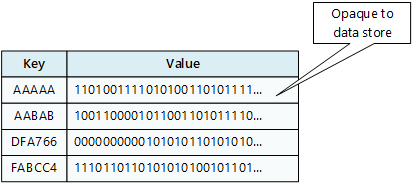
NoSQL (non-relational) databases generally fall into four categories: key-value stores, document databases, column family databases, and graph databases. The following sections discuss these types of NoSQL databases.

## What is a key-value store?

A key-value store is the simplest (and often quickest) type of NoSQL database for inserting and querying data. Each data item in a key-value store has two elements, a key and a value. The key uniquely identifies the item, and the value holds the data for the item. The value is opaque to the database management system. Items are stored in key order.

**Note**

The term opaque means that the database management system just sees the value as an unstructured block. Only the application understands how the data in the value is structured and what fields it contains. The opposite of opaque is transparent. If the data is transparent, the database management system understands how the fields in the data are organized. A relational table is an example of a transparent structure.



A query specifies the keys to identify the items to be retrieved. You can't search on values. An application that retrieves data from a key-value store is responsible for parsing the contents of the values returned.

Write operations are restricted to inserts and deletes. If you need to update an item, you must retrieve the item, modify it in memory (in the application), and then write it back to the database, overwriting the original (effectively a delete and an insert).

The focus of a key-value store is the ability to read and write data very quickly. Search capabilities are secondary. A key-value store is an excellent choice for data ingestion, when a large volume of data arrives as a continual stream and must be stored immediately.

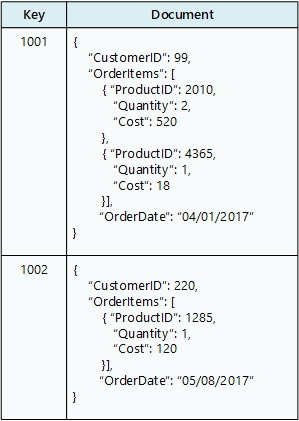
Azure Table storage is an example of a key-value store. Cosmos DB also implements a key-value store using the [**Table API**](https://docs.microsoft.com/en-us/azure/cosmos-db/table-introduction).

## What is a document database?

A document database represents the opposite end of the NoSQL spectrum from a key-value store. In a document database, each document has a unique ID, but the fields in the documents are transparent to the database management system. Document databases typically store data in JSON format, as described in the previous unit, or they could be encoded using other formats such as XML, YAML, JSON, BSON. Documents could even be stored as plain text. The fields in documents are exposed to the storage management system, enabling an application to query and filter data by using the values in these fields.

Typically, a document contains the entire data for an entity. What items constitute an entity are application-specific. For example, an entity could contain the details of a customer, an order, or a combination of both. A single document may contain information that would be spread across several relational tables in an RDBMS (relational database management system).

A document store does not require that all documents have the same structure. This free-form approach provides a great deal of flexibility. Applications can store different data in documents as business requirements change.



An application can retrieve documents by using the document key. The key is a unique identifier for the document. Some document databases create the document key automatically. Others enable you to specify an attribute of the document to use as the key. The application can also query documents based on the value of one or more fields. Some document databases support indexing to facilitate fast lookup of documents based on one or more indexed fields.

Some document database management systems support in-place updates, enabling an application to modify the values of specific fields in a document without rewriting the entire document. Other document database management systems (such as Cosmos DB) can only read and write entire documents. In these cases, an update replaces the entire document with a new version. This approach helps to reduce fragmentation in the database, which can, in turn, improve performance.

Most document databases will ingest large volumes of data more rapidly than a relational database, but aren't as optimal as a key-value store for this type of processing. The focus of a document database is its query capabilities.

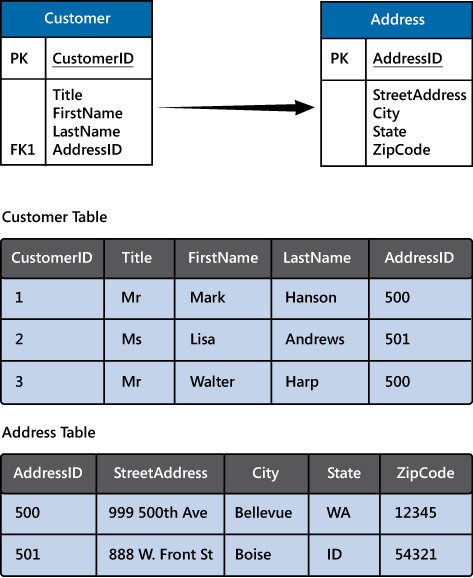
Azure Cosmos DB implements a document database approach in its Core (SQL) API.

## What is a column family database?

A column family database organizes data into rows and columns. Examples of this structure include ORC and Parquet files, described in the previous unit.

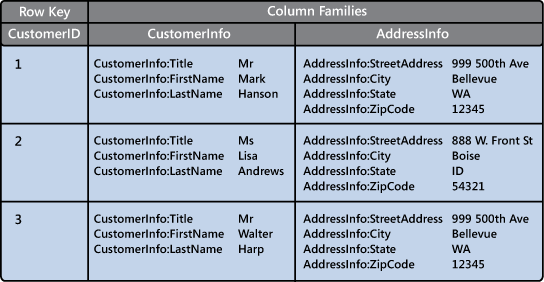
In its simplest form, a column family database can appear very similar to a relational database, at least conceptually. The real power of a column family database lies in its denormalized approach to structuring sparse data.

For example, if you need to store information about customers and their addresses in a relational database (ignoring the need to maintain historical data as described in the previous section), you might design a schema similar to that shown below. This diagram also shows some sample data. In this example, customer 1 and customer 3 share the same address, and the schema ensures that this address information is not duplicated. This is a standard way of implementing a one-to-many relationship.



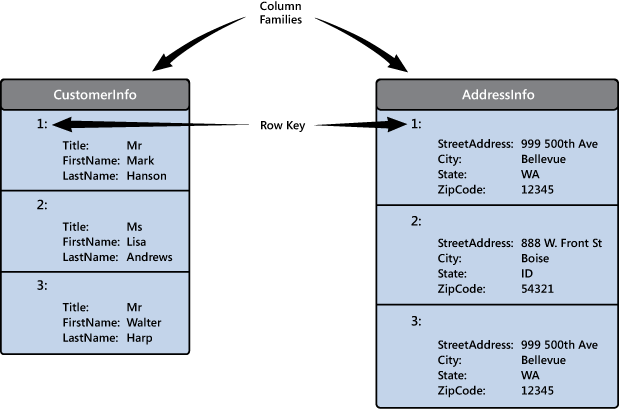
The relational model supports a very generalized approach to implementing this type of relationship, but to find the address of any given customer an application needs to run a query that joins two tables. If this is the most common query performed by the application, then the overhead associated with performing this join operation can quickly become significant if there are a large number of requests and the tables themselves are large.

The purpose of a column family database is to efficiently handle situations such as this. You can think of a column family database as holding tabular data comprising rows and columns, but you can divide the columns into groups known as column-families. Each column family holds a set of columns that are logically related together. The image below shows one way of structuring the same information as the previous image, by using a column family database to group the data into two column-families holding the customer name and address information. Other ways of organizing the columns are possible, but you should implement your column-families to optimize the most common queries that your application performs. In this case, queries that retrieve the addresses of customers can fetch the data with fewer reads than would be required in the corresponding relational database; these queries can fetch the data directly from the **AddressInfo** column family.



The illustration above is conceptual rather than physical, and is intended to show the logical structure of the data rather than how it might be physically organized. Each row in a column family database contains a key, and you can fetch the data for a row by using this key.

In most column family databases, the column-families are stored separately. In the previous example, the CustomerInfo column family might be held in one area of physical storage and the AddressInfo column family in another, in a simple form of vertical partitioning. You should really think of the structure in terms of column-families rather than rows. The data for a single entity that spans multiple column-families will have the same row key in each column family. As an alternative to the conceptual layout shown previously, you can visualize the data shown as the following pair of physical structures.

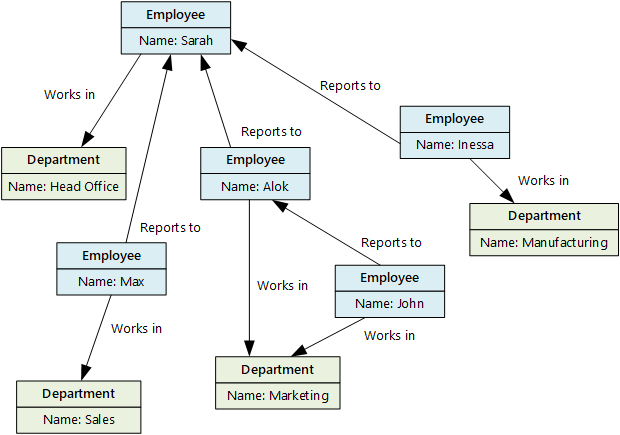


The most widely used column family database management system is Apache Cassandra. Azure Cosmos DB supports the column-family approach through the Cassandra API.

## What is a graph database?

Graph databases enable you to store entities, but the main focus is on the relationships that these entities have with each other. A graph database stores two types of information: nodes that you can think of as instances of entities, and edges, which specify the relationships between nodes. Nodes and edges can both have properties that provide information about that node or edge (like columns in a table). Additionally, edges can have a direction indicating the nature of the relationship.

The purpose of a graph database is to enable an application to efficiently perform queries that traverse the network of nodes and edges, and to analyze the relationships between entities. The image below shows an organization's personnel database structured as a graph. The entities are the employees and the departments in the organization, and the edges indicate reporting lines and the department in which employees work. In this graph, the arrows on the edges show the direction of the relationships.



A structure such as this makes it straightforward to conduct inquiries such as "Find all employees who directly or indirectly work for Sarah" or "Who works in the same department as John?" For large graphs with lots of entities and relationships, you can perform very complex analyses very quickly, and many graph databases provide a query language that you can use to traverse a network of relationships efficiently. You can often store the same information in a relational database, but the SQL required to query this information might require many expensive recursive join operations and nested subqueries.

Azure Cosmos DB supports graph databases using the [**Gremlin API**](https://docs.microsoft.com/en-us/azure/cosmos-db/graph-introduction). The Gremlin API is a standard language for creating and querying graphs.

# Describe data ingestion and processing

Completed100 XP

* 5 minutes

Data analytics is concerned with taking data and finding meaningful information and inferences from it. This could be as wide ranging as selecting the ideal range of products for a retailer, or selecting the best vaccine candidates for a biotechnology company.

For example, in a company data analytics could be concerned with taking the data that your organization produces, and using it to establish a picture of how your organization is performing, and what you can do to maintain business performance. Data analytics could help you to identify strengths and weaknesses in your organization, and enable you to make appropriate business decisions.

The data a company uses can come from many sources. There could be a mass of historical data to comb through, and fresh data continuing to arrive all the time. This data could be the result of customer purchases, bank transactions, stock price movements, real-time weather data, monitoring devices, or even cameras. In a data analytics solution, you combine this data and construct a data warehouse that you can use to ask (and answer) questions about your business operations. Building a data warehouse requires that you can capture the data that you need and wrangle it into an appropriate format. You can then use analysis tools and visualizations to examine the information, and identify trends and their causes.

**Note**

Wrangling is the process by which you transform and map raw data into a more useful format for analysis. It can involve writing code to capture, filter, clean, combine, and aggregate data from many sources.

In this unit, you'll learn about two important stages in data analytics: data ingestion, and data processing. The diagram below shows how these stages fit together.

## What is data ingestion?

Data ingestion is the process of obtaining and importing data for immediate use or storage in a database. The data can arrive as a continuous stream, or it may come in batches, depending on the source. The purpose of the ingestion process is to capture this data and store it. This raw data can be held in a repository such as a database management system, a set of files, or some other type of fast, easily accessible storage.

The ingestion process might also perform filtering. For example, ingestion might reject suspicious, corrupt, or duplicated data. Suspicious data might be data arriving from an unexpected source. Corrupt or duplicated data could be due to a device error, transmission failure, or tampering.

It may also be possible to perform some transformations at this stage, converting data into a standard form for later processing. For example, you might want to reformat all date and time data to use the same date and time representations, and convert all measurement data to use the same units. However, these transformations must be quick to perform. Don't attempt to run any complex calculations or aggregations on the data at this stage.

## What is data processing?

The data processing stage occurs after the data has been ingested and collected. Data processing takes the data in its raw form, cleans it, and converts it into a more meaningful format (tables, graphs, documents, and so on). The result is a database of data that you can use to perform queries and generate visualizations, giving it the form and context necessary to be interpreted by computers and used by employees throughout an organization.

**Note**

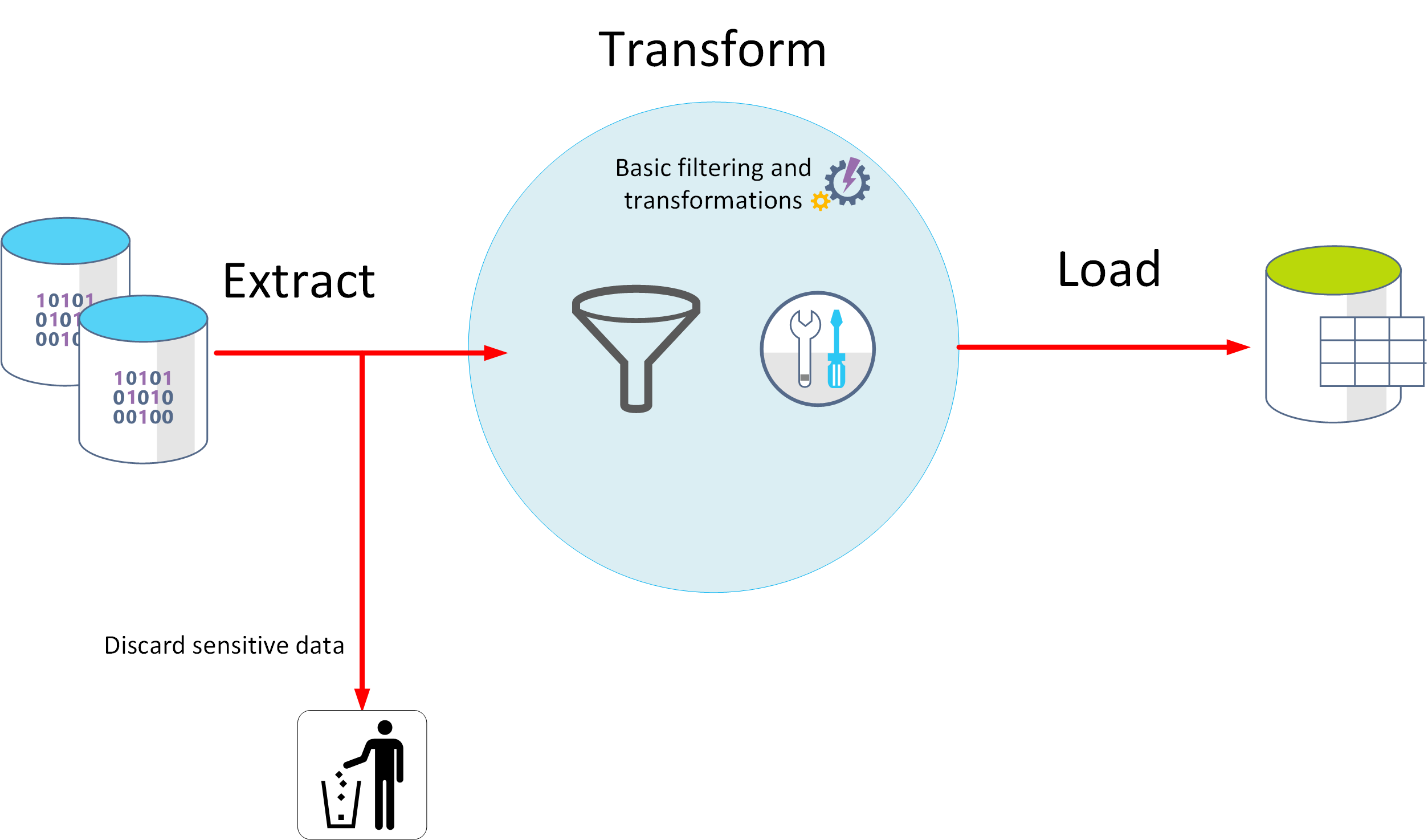
Data cleaning is a generalized term that encompasses a range of actions, such as removing anomalies, and applying filters and transformations that would be too time-consuming to run during the ingestion stage.

The aim of data processing is to convert the raw data into one or more business models. A business model describes the data in terms of meaningful business entities, and may aggregate items together and summarize information. The data processing stage could also generate predictive or other analytical models from the data. Data processing can be complex, and may involve automated scripts, and tools such as Azure Databricks, Azure Functions, and Azure Cognitive Services to examine and reformat the data, and generate models. A data analyst could use machine learning to help determine future trends based on these models.

## What is ELT and ETL?

The data processing mechanism can take two approaches to retrieving the ingested data, processing this data to transform it and generate models, and then saving the transformed data and models. These approaches are known as ETL and ELT.

ETL stands for Extract, Transform, and Load. The raw data is retrieved and transformed before being saved. The extract, transform, and load steps can be performed as a continuous pipeline of operations. It is suitable for systems that only require simple models, with little dependency between items. For example, this type of process is often used for basic data cleaning tasks, deduplicating data, and reformatting the contents of individual fields.

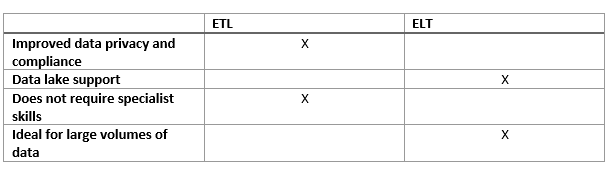


An alternative approach is ELT. ELT is an abbreviation of Extract, Load, and Transform. The process differs from ETL in that the data is stored before being transformed. The data processing engine can take an iterative approach, retrieving and processing the data from storage, before writing the transformed data and models back to storage. ELT is more suitable for constructing complex models that depend on multiple items in the database, often using periodic batch processing.

ELT is a scalable approach that is suitable for the cloud because it can make use of the extensive processing power available. The more stream-oriented approach of ETL places more emphasis on throughput. However, ETL can filter data before it's stored. In this way, ETL can help with data privacy and compliance, removing sensitive data before it arrives in your analytical data models.

Azure provides several options that you can use to implement the ELT and ETL approaches. For example, if you are storing data in Azure SQL Database, you can use SQL Server Integration Services. Integration Services can extract and transform data from a wide variety of sources such as XML data files, flat files, and relational data sources, and then load the data into one or more destinations.

This is a simple table showing the advantages of ETL and ELT in most cases.



Another more generalized approach is to use Azure Data Factory. Azure Data Factory is a cloud-based data integration service that allows you to create data-driven workflows for orchestrating data movement and transforming data at scale. Using Azure Data Factory, you can create and schedule data-driven workflows (called pipelines) that can ingest data from disparate data stores. You can build complex ETL processes that transform data visually with data flows, or by using compute services such as Azure HDInsight Hadoop, Azure Databricks, and Azure SQL Database.

# Explore data visualization

Completed100 XP

* 4 minutes

A business model can contain an enormous amount of information. The purpose of producing a model such as this is to help you reason over the information it contains, ask questions, and hopefully obtain answers that can help you drive your business forward.

This unit discusses some of the techniques you can use to analyze and understand the information in your models.

## What is reporting?

Reporting is the process of organizing data into informational summaries to monitor how different areas of an organization are performing. Reporting helps companies monitor their online business, and know when data falls outside of expected ranges. Good reporting should raise questions about the business from its end users. Reporting shows you what has happened, while analysis focuses on explaining why it happened and what you can do about it.

## What is business intelligence?

The term Business Intelligence (BI) refers to technologies, applications, and practices for the collection, integration, analysis, and presentation of business information. The purpose of business intelligence is to support better decision making.

Business intelligence systems provide historical, current, and predictive views of business operations, most often using data that has been gathered into a data warehouse, and occasionally working from live operational data. Software elements support reporting, interactive “slice-and-dice” pivot table analysis, visualization, and statistical data mining. Applications tackle sales, production, financial, and many other sources of business data for purposes that include business performance management. Information is often gathered about other companies in the same industry for comparison. This process of comparison with other companies in the same industry is known as benchmarking.

## What is data visualization?

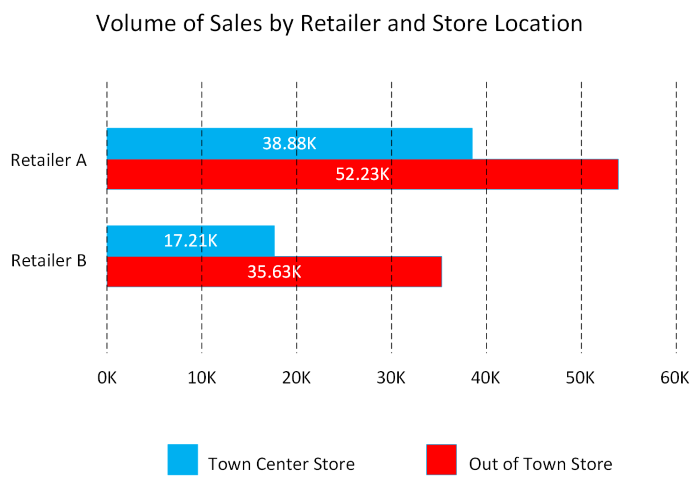
Data visualization is the graphical representation of information and data. By using visual elements like charts, graphs, and maps, data visualization tools provide an accessible way to spot and understand trends, outliers, and patterns in data. If you are using Azure, the most popular data visualization tool is Power BI.

Using Power BI, you can connect to multiple different sources of data, and combine them into a data model. This data model lets you build visuals, and collections of visuals you can share as reports, with other people inside your organization.

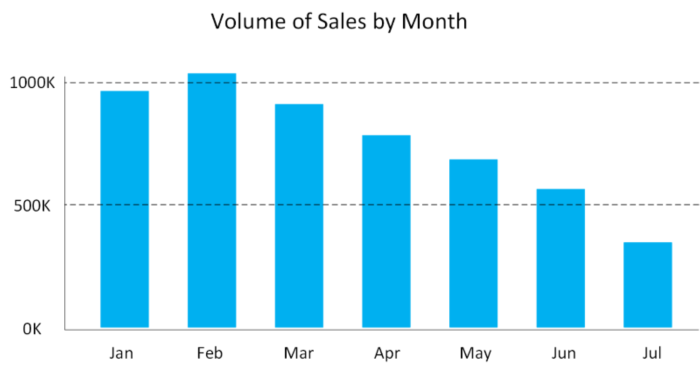
### Explore visualization options to represent data

Data visualization helps you to focus on the meaning of data, rather than looking at the data itself. A good data visualization enables you to quickly spot trends, anomalies, and potential issues. The most common forms of visualizations are:

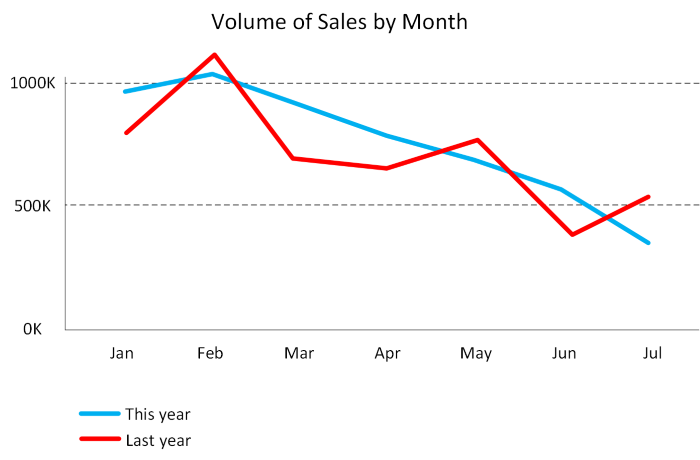
* Bar and column charts: Bar and column charts enable you to see how a set of variables changes across different categories. For example, the first chart below shows how sales for a pair of fictitious retailers vary between store sites.



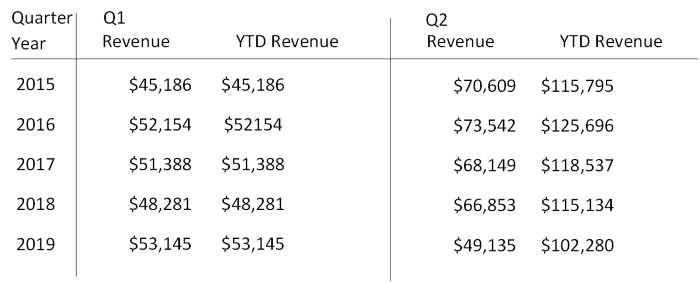
This chart shows how sales vary by month.



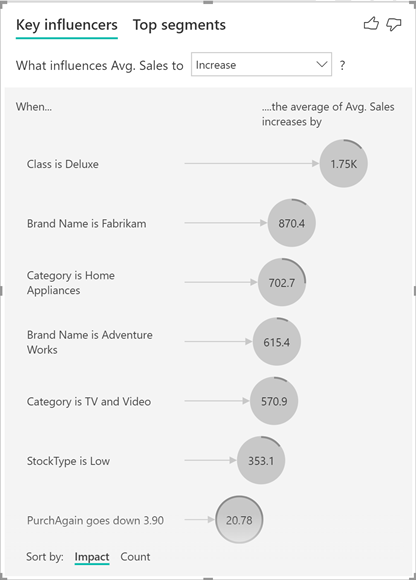
* Line charts: Line charts emphasize the overall shape of an entire series of values, usually over time.



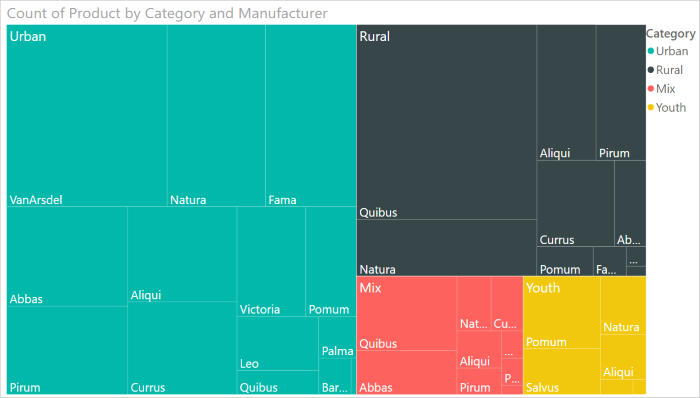
* Matrix: A matrix visual is a tabular structure that summarizes data. Often, report designers include matrixes in reports and dashboards to allow users to select one or more element (rows, columns, cells) in the matrix to cross-highlight other visuals on a report page.



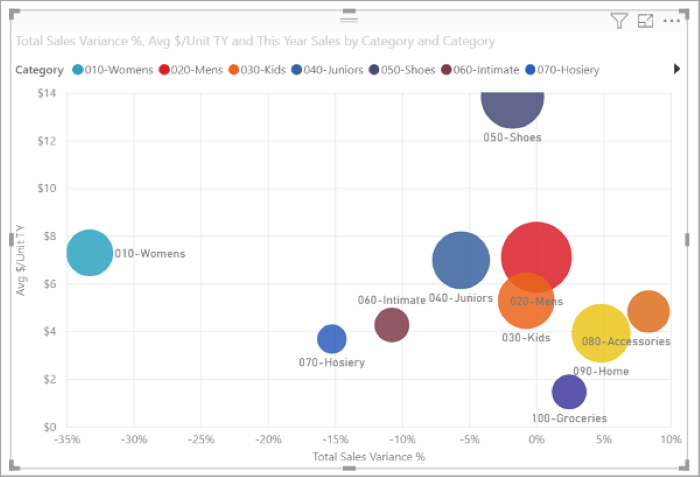
* Key influencers: A key influencer chart displays the major contributors to a selected result or value. Key influencers are a great choice to help you understand the factors that influence a key metric. For example, what influences customers to place a second order or why sales were so high last June.



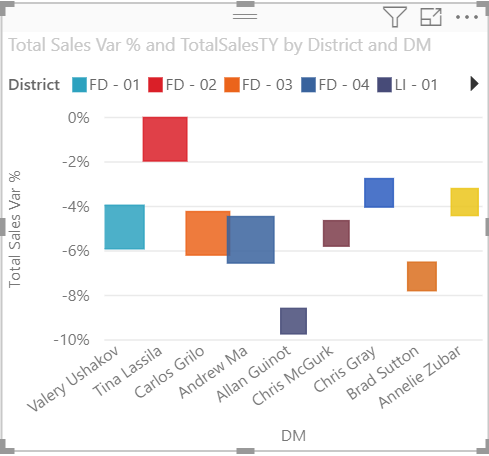
* Treemap: Treemaps are charts of colored rectangles, with size representing the relative value of each item. They can be hierarchical, with rectangles nested within the main rectangles.



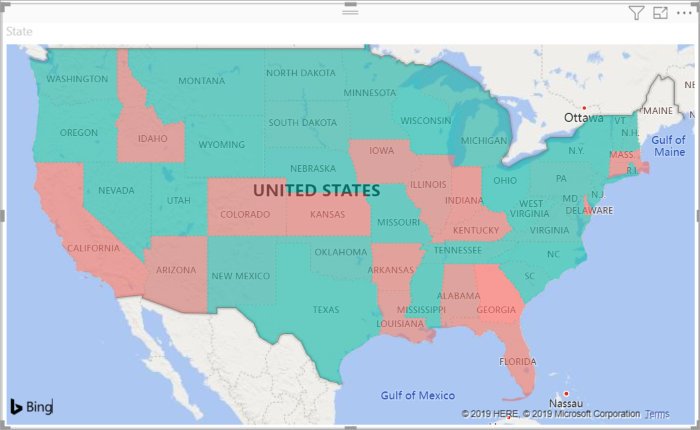
* Scatter: A scatter chart shows the relationship between two numerical values. A bubble chart is a scatter chart that replaces data points with bubbles, with the bubble size representing an additional third data dimension.



A dot plot chart is similar to a bubble chart and scatter chart, but can plot categorical data along the X-Axis.



* Filled map. If you have geographical data, you can use a filled map to display how a value differs in proportion across a geography or region. You can see relative differences with shading that ranges from light (less-frequent/lower) to dark (more-frequent/more).



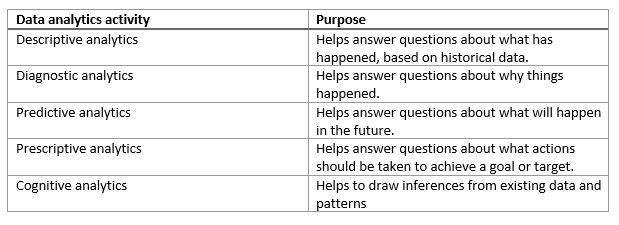
# Explore data analytics

Completed100 XP

* 3 minutes

Data analytics is concerned with examining, transforming, and arranging data so that you can study it and extract useful information. Data analytics is a discipline that covers the entire range of data management tasks. These tasks not only include analysis, but also data collection, organization, storage, and all the tools and techniques used.

The term data analytics is a catch-all that covers a range of activities, each with its own focus and goals. You can categorize these activities as descriptive, diagnostic, predictive, prescriptive, and cognitive analytics.



In this unit, you'll learn about these categories of data analytics.

## Descriptive analytics

Descriptive analytics helps answer questions about what has happened, based on historical data. Descriptive analytics techniques summarize large datasets to describe outcomes to stakeholders.

By developing KPIs (Key Performance Indicators), these strategies can help track the success or failure of key objectives. Metrics such as return on investment (ROI) are used in many industries. Specialized metrics are developed to track performance in specific industries.

Examples of descriptive analytics include generating reports to provide a view of an organization's sales and financial data.

## Diagnostic analytics

Diagnostic analytics helps answer questions about why things happened. Diagnostic analytics techniques supplement more basic descriptive analytics. They take the findings from descriptive analytics and dig deeper to find the cause. The performance indicators are further investigated to discover why they got better or worse. This generally occurs in three steps:

1. Identify anomalies in the data. These may be unexpected changes in a metric or a particular market.
2. Collect data that's related to these anomalies.
3. Use statistical techniques to discover relationships and trends that explain these anomalies.

## Predictive analytics

Predictive analytics helps answer questions about what will happen in the future. Predictive analytics techniques use historical data to identify trends and determine if they're likely to recur. Predictive analytical tools provide valuable insight into what may happen in the future. Techniques include a variety of statistical and machine learning techniques such as neural networks, decision trees, and regression.

## Prescriptive analytics

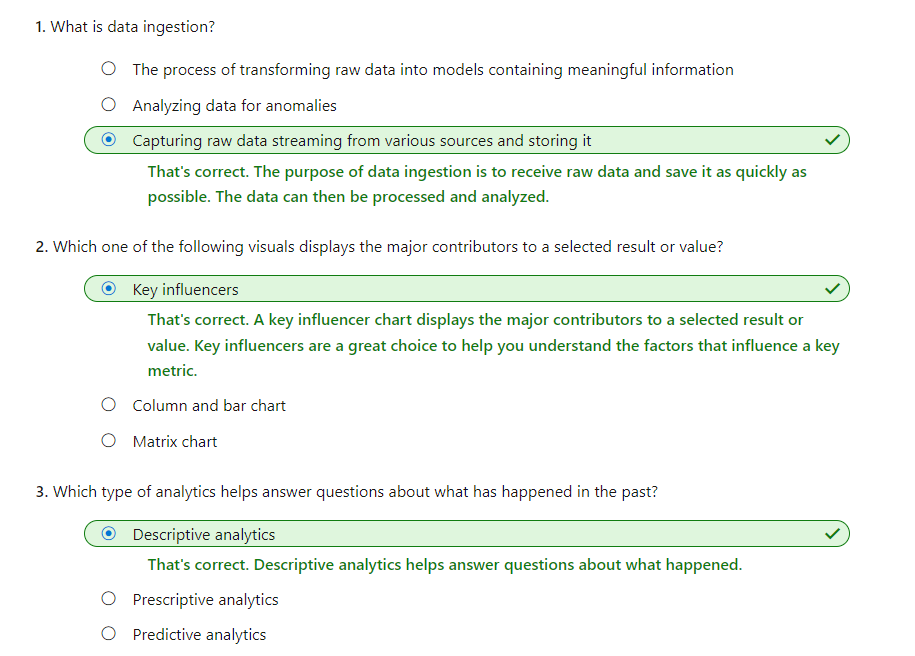
Prescriptive analytics helps answer questions about what actions should be taken to achieve a goal or target. By using insights from predictive analytics, data-driven decisions can be made. This technique allows businesses to make informed decisions in the face of uncertainty. Prescriptive analytics techniques rely on machine learning strategies to find patterns in large datasets. By analyzing past decisions and events, the likelihood of different outcomes can be estimated.

## Cognitive analytics

Cognitive analytics attempts to draw inferences from existing data and patterns, derive conclusions based on existing knowledge bases, and then add these findings back into the knowledge base for future inferences--a self-learning feedback loop. Cognitive analytics helps you to learn what might happen if circumstances change, and how you might handle these situations.

Inferences aren't structured queries based on a rules database, rather they're unstructured hypotheses gathered from a number of sources, and expressed with varying degrees of confidence. Effective cognitive analytics depends on machine learning algorithms. It uses several NLP (Natural Language Processing) concepts to make sense of previously untapped data sources, such as call center conversation logs and product reviews.

Theoretically, by tapping the benefits of massive parallel/distributed computing and the falling costs of data storage and computing power, there's no limit to the cognitive development that these systems can achieve.



# **2nd Chapter:Explore Relational Data In Azure** Explore relational Azure data services

Completed100 XP

* 6 minutes

Azure offers a range of options for running a database management system in the cloud. For example, you can migrate your on-premises systems to a collection of Azure virtual machines. This approach still requires that you manage your DBMS carefully. Alternatively, you can take advantage of the various Azure relational data services available. These data services manage the DBMS for you, leaving you free to concentrate on the data they contain and the applications that use them.

## Understand IaaS, PaaS, and SaaS

Before delving into Azure Data Services, you need to understand some common terms used to describe the different ways in which you can host a database in Azure.

**IaaS** is an acronym for Infrastructure-as-a-Service. Azure enables you to create a virtual infrastructure in the cloud that mirrors the way an on-premises data center might work. You can create a set of virtual machines, connect them together using a virtual network, and add a range of virtual devices. You take responsibility for installing and configuring the software, such as the DBMS, on these virtual machines. In many ways, this approach is similar to the way in which you run your systems inside an organization, except that you don't have to concern yourself with buying or maintaining the hardware.

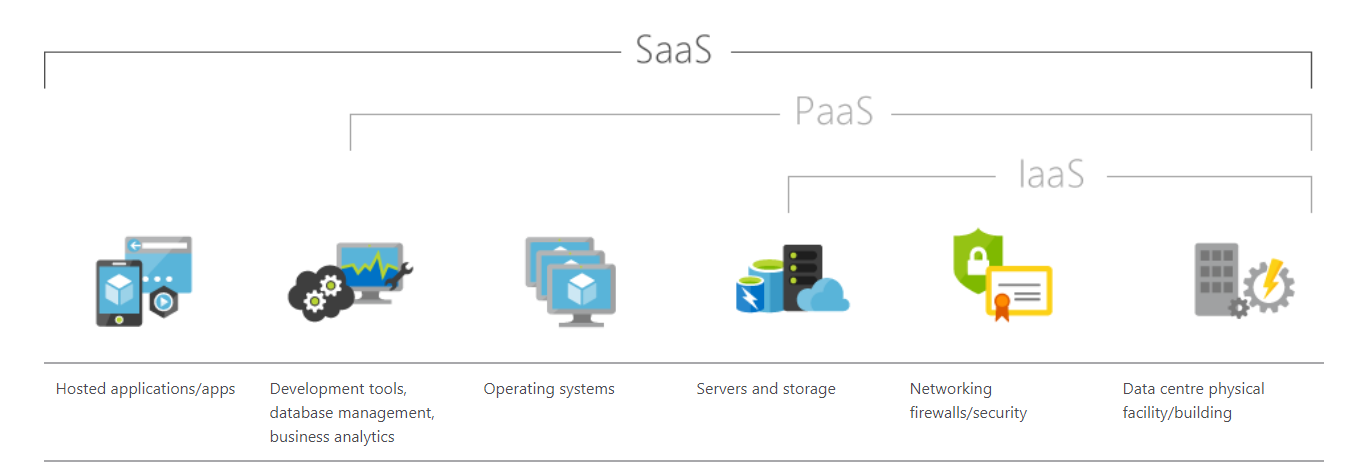
**Note**

An Azure Virtual Network is a representation of your own network in the cloud. A virtual network enables you to connect virtual machines and Azure services together, in much the same way that you might use a physical network on-premises. Azure ensures that each virtual network is isolated from other virtual networks created by other users, and from the Internet. Azure enables you to specify which machines (real and virtual), and services, are allowed to access resources on the virtual network, and which ports they can use.

**PaaS** stands for Platform-as-a-service. Rather than creating a virtual infrastructure, and installing and managing the database software yourself, a PaaS solution does this for you. You specify the resources that you require (based on how large you think your databases will be, the number of users, and the performance you require), and Azure automatically creates the necessary virtual machines, networks, and other devices for you. You can usually scale up or down (increase or decrease the size and number of resources) quickly, as the volume of data and the amount of work being done varies; Azure handles this scaling for you, and you don't have to manually add or remove virtual machines, or perform any other form of configuration.

**SaaS** is short for Software-as-a-Service. SaaS services are typically specific software packages that are installed and run on virtual hardware in the cloud. SaaS packages are typically hosted applications rather than more generalized software such as a DBMS. Common SaaS packages available on Azure include Microsoft 365 (formerly Office 365).

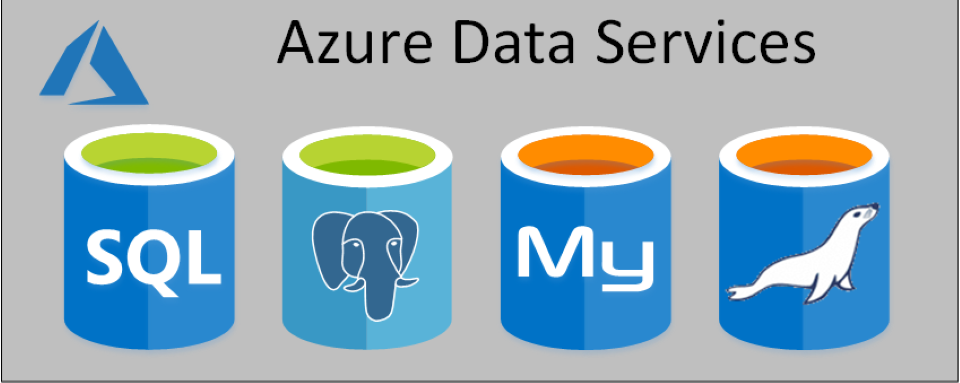
| **TABLE 1** | | |
| --- | --- | --- |
|  | **Example** | **Includes** |
| IaaS | Azure virtual network | Servers, storage, networking, and physical data center. |
| PaaS | Azure SQL Databases | IaaS plus database management (or other server systems), and operating systems. |
| SaaS | Office 365 | PaaS plus apps. |



## What are Azure Data Services?

Azure Data Services fall into the PaaS category. These services are a series of DBMSs managed by Microsoft in the cloud. Each data service takes care of the configuration, day-to-day management, software updates, and security of the databases that it hosts. All you do is create your databases under the control of the data service.

Azure Data Services are available for several common relational database management systems. The most well-known service is Azure SQL Database. The others currently available are Azure Database for MySQL servers, Azure Database for MariaDB servers, and Azure Database for PostgreSQL servers. The remaining units in this module describe the features provided by these services.



**Note**

Microsoft also provides data services for non-relational database management systems, such as Cosmos DB.

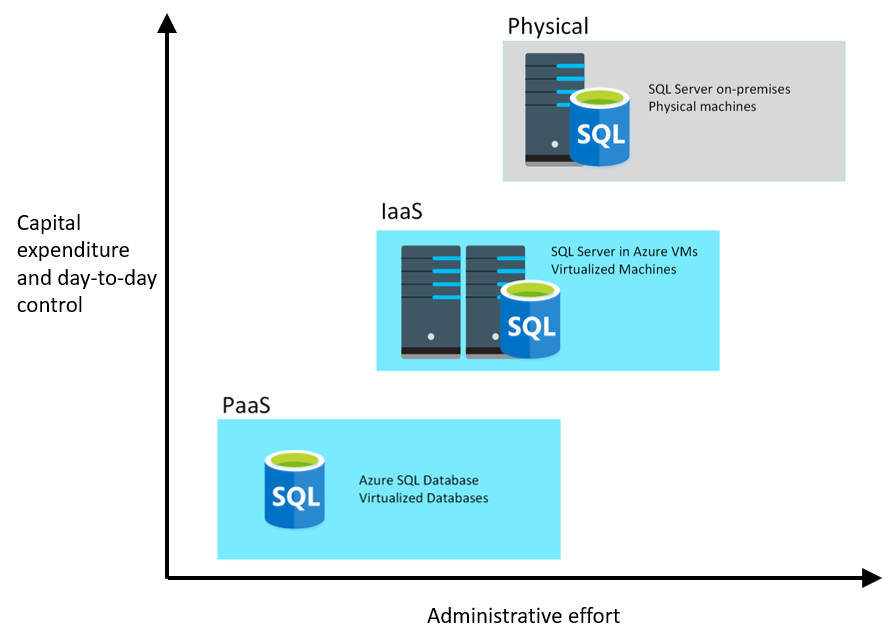
Using Azure Data Services reduces the amount of time that you need to invest to administer a DBMS. However, these services can also limit the range of custom administration tasks that you can perform, because manually performing some tasks might risk compromising the way in which the service runs. For example, some DBMSs enable you to install custom software into a database, or run scripts as part of a database operation. This software might not be supported by the data service, and allowing an application to run a script from a database could affect the security of the service. You must be prepared to work with these restrictions in mind.

Apart from reducing the administrative workload, Azure Data Services ensure that your databases are available for at least 99.99% of the time.

There are costs associated with running a database in Azure Data Services. The base price of each service covers underlying infrastructure and licensing, together with the administration charges. Additionally, these services are designed to be always on. This means that you can't shut down a database and restart it later.

Not all features of a database management system are available in Azure Data Services. This is because Azure Data Services takes on the task of managing the system and keeping it running using hardware situated in an Azure datacenter. Exposing some administrative functions might make the underlying platform vulnerable to misuse, and even open up some security concerns. Therefore, you have no direct control over the platform on which the services run. If you need more control than Azure Data Services allow, you can install your database management system on a virtual machine that runs in Azure. The next unit examines this approach in more detail for SQL Server, although the same issues apply for the other database management systems supported by Azure Data Services.

The image below highlights the different ways in which you could run a DBMS such as SQL Server, starting with an on-premises system in the top-right-hand corner, to PaaS in the bottom left. The diagram illustrates the benefits of moving to the PaaS approach.



# SQL Server on Azure virtual machines

Completed100 XP

* 4 minutes

Microsoft SQL Server is a popular relational DBMS. It has a long history, and has features that provide database management to organizations of all sizes. In the past, organizations have run SQL Server on-premises. However, many organizations are now looking to shift operations on-line to take advantage of services available in the cloud. SQL Server offers several ways to run a database in Azure. In this unit, you'll look at moving SQL Server to an Azure Virtual Machine.

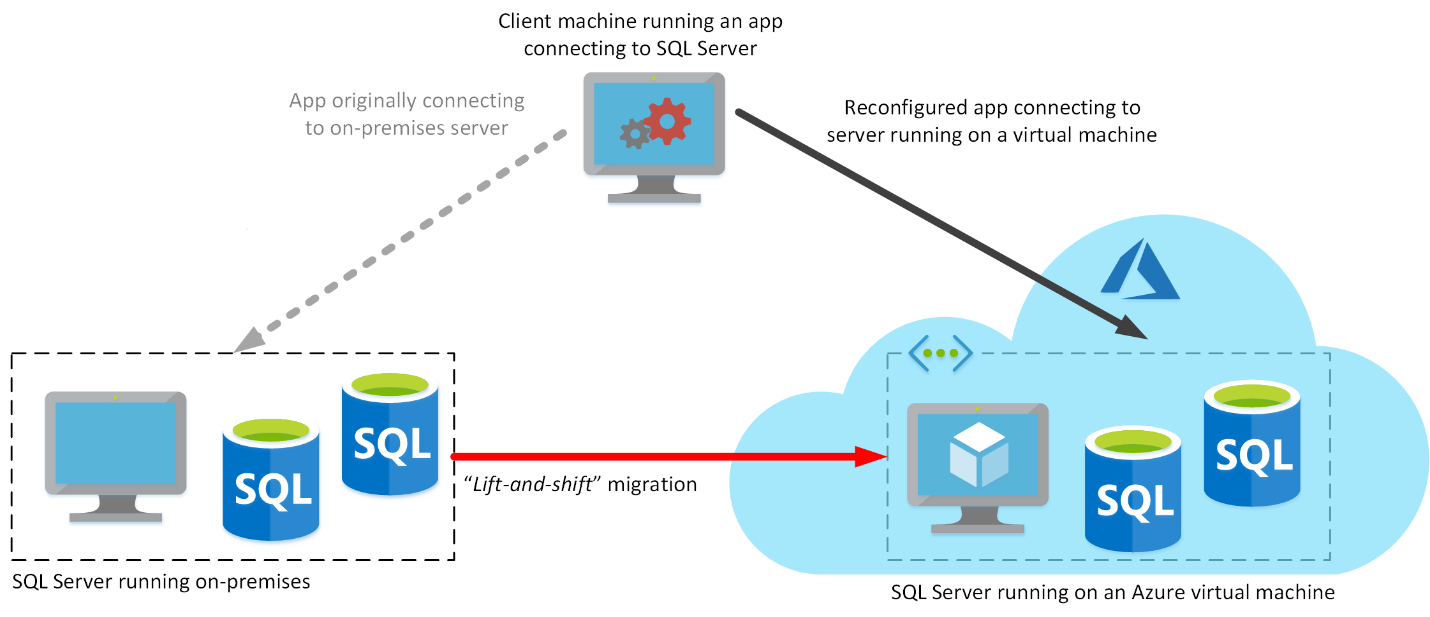
## What is SQL Server on Azure Virtual Machines?

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.

In the example scenario described in the introduction, the database runs stored procedures and scripts as part of the database workload. If these stored procedures and scripts depend on features that are restricted by following a PaaS approach, then running SQL Server on your own virtual machines might be a good option. However, you remain responsible for maintaining the SQL Server software and performing the various administrative tasks to keep the database running from day-to-day.

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.



**Note**

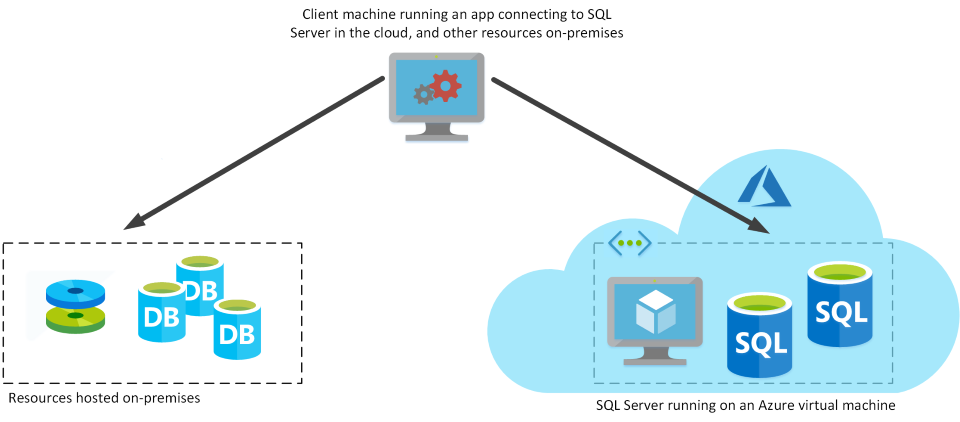
The term lift-and-shift refers to the way in which you can move a database directly from an on-premises server to an Azure virtual machine without requiring that you make any changes to it. Applications that previously connected to the on-premises database can be quickly reconfigured to connect to the database running on the virtual machine, but should otherwise remain unchanged.

### Use cases

This approach is optimized for migrating existing applications to Azure, or extending 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 do not 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 does not eliminate the need to administer your DBMS as carefully as you would on-premises.

# Azure SQL Database

Completed100 XP

* 5 minutes

If you don't want to incur the management overhead associated with running SQL Server on a virtual machine, you can use Azure SQL Database.

## What is 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.

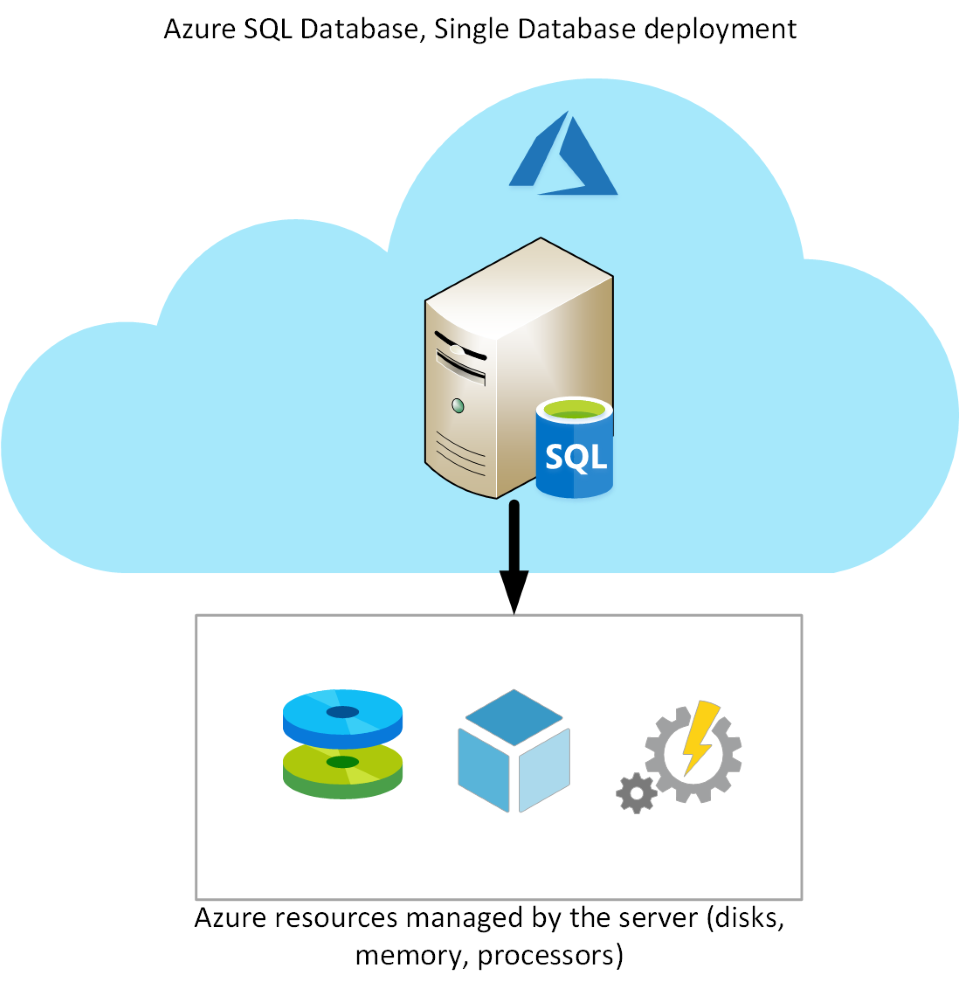
**Note**

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 with several options: Single Database, Elastic Pool, and Managed Instance. The following sections describe Single Instance and Elastic Pool. Managed Instance is the subject of the next unit.

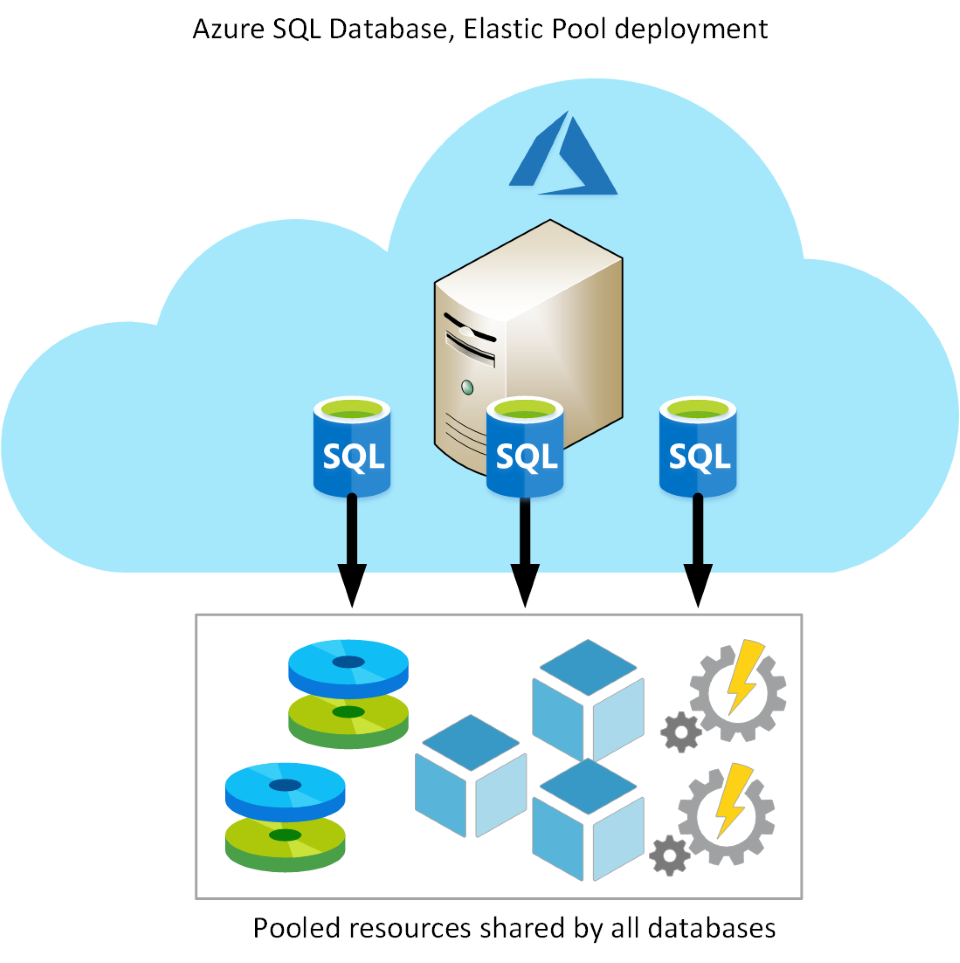
### Single Database

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 additional storage space, memory, or processing power. By default, resources are pre-allocated, 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 a number of 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. For more information, read [What is a single database in Azure SQL Database](https://docs.microsoft.com/en-us/azure/sql-database/sql-database-single-database).



### 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 is not fully compatible with on-premises SQL Server installations. It is 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**](https://docs.microsoft.com/en-us/sql/dma/dma-overview).

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 are 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 99.99% 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 additional assurance 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. For data in motion, it uses Transport Layer Security. For data at rest, it uses Transparent Data Encryption. For data in use, it uses Always Encrypted. For more information on Transport Layer Security, Transparent Data Encryption, and Always Encrypted, see the links in the Summary unit.

In the Wide World Importers scenario, linked servers are used to perform distributed queries. However, neither Single Database nor Elastic Pool support linked servers. If you want to use Single Database or Elastic Pool, you may need to modify the queries that use linked servers and rework the operations that depend on these features.

# Azure SQL Database Managed Instance

Completed100 XP

* 4 minutes

A business may want to eliminate as much management overhead as possible from administering databases and servers, but the limitations of the Single Database and Elastic Pool options may mean that those options aren't suitable. In these situations. Azure SQL Database managed instance may be a good choice to consider.

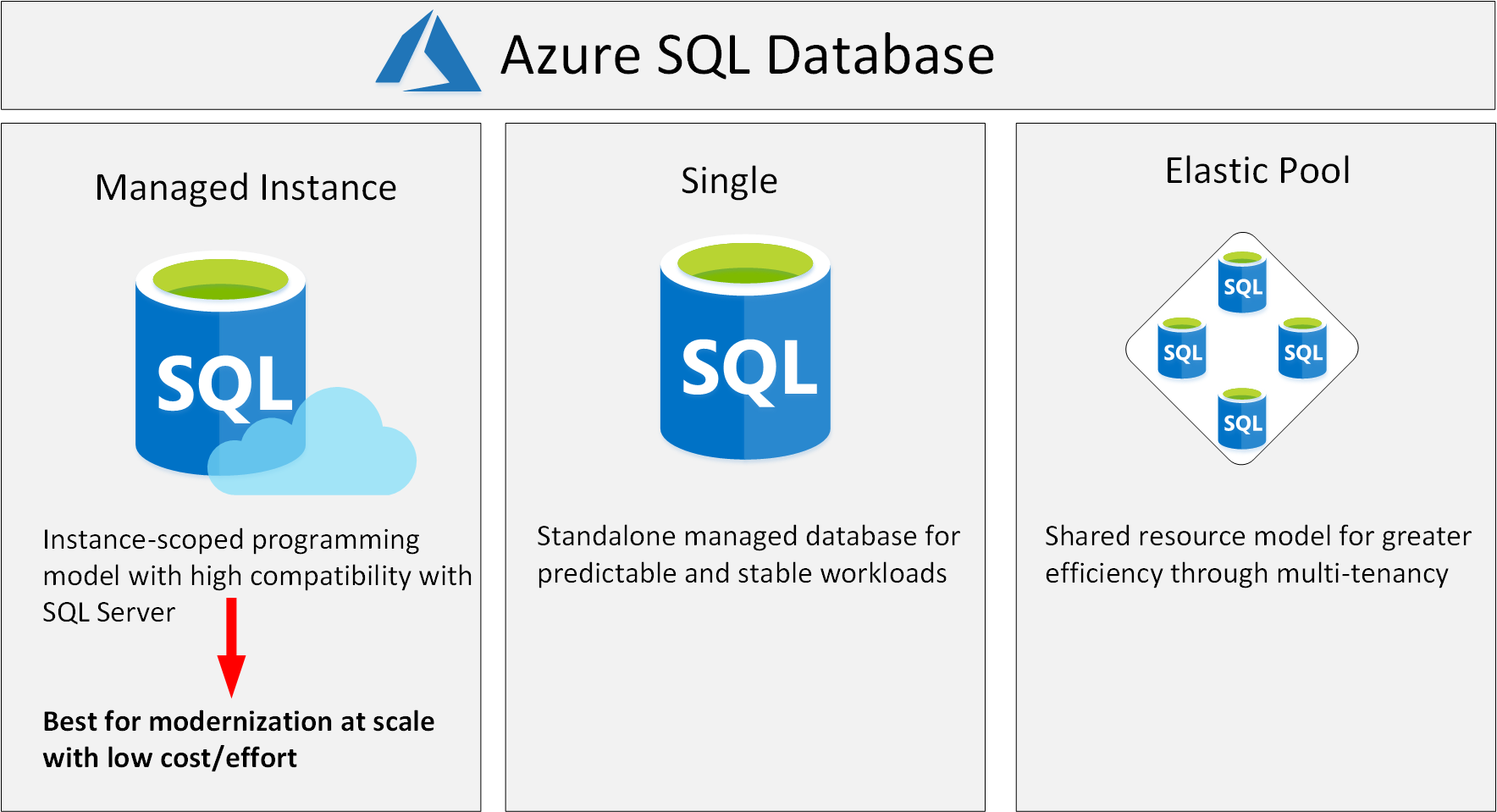
## What is Azure SQL Database managed instance?

The Single Database and Elastic Pool options restrict some of the administrative features available to SQL Server. 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. The Managed instance service 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 Database managed instance?](https://docs.microsoft.com/en-us/azure/sql-database/sql-database-managed-instance).

Managed instances depend on other Azure services such as Azure Storage for backups, Azure Event Hubs for telemetry, Azure Active Directory 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.

The following image summarizes the differences between SQL Database managed instance, Single Database, and Elastic Pool



## Use cases

Consider Azure SQL Database 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.

SQL Database managed instance provides features not available with the Single Database or Elastic Pool options. 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)](https://www.microsoft.com/download/details.aspx?id=53595). This tool analyzes your databases on SQL Server and reports any issues that could block migration to a managed instance.

## Business benefits

SQL Database managed instance provides all the management and security benefits available when using Single Database and Elastic Pool. Managed instance deployment enables a system administrator to spend less time on administrative tasks because the SQL Database 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.

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

The SQL Database managed instance deployment option supports traditional SQL Server Database engine logins and logins integrated with Azure Active Directory (AD). Traditional SQL Server Database engine logins include a username and a password. You must enter your credentials each time you connect to the server. Azure AD 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.

In the Wide World Importers scenario, SQL Database managed instance may be a more suitable choice than Single Database or Elastic Pool. SQL Database managed instance supports linked servers, although some of the other advanced features required by the database might not be available. If you want a complete match, then running SQL Server on a virtual machine may be your only option, but you need to balance the benefits of complete functionality against the administrative and maintenance overhead required.

# PostgreSQL, MariaDB, and MySQL

Completed100 XP

* 6 minutes

As well as Azure SQL Database, Azure Data Services are available for other popular SQL-based database solutions. Currently, data services are available for PostgreSQL, MySQL, and MariaDB. The primary reason for these services is to enable organizations running PostgreSQL, MySQL, or MariaDB to move to Azure quickly, without making wholesale changes to their applications.

## What are MySQL, MariaDB, and PostgreSQL

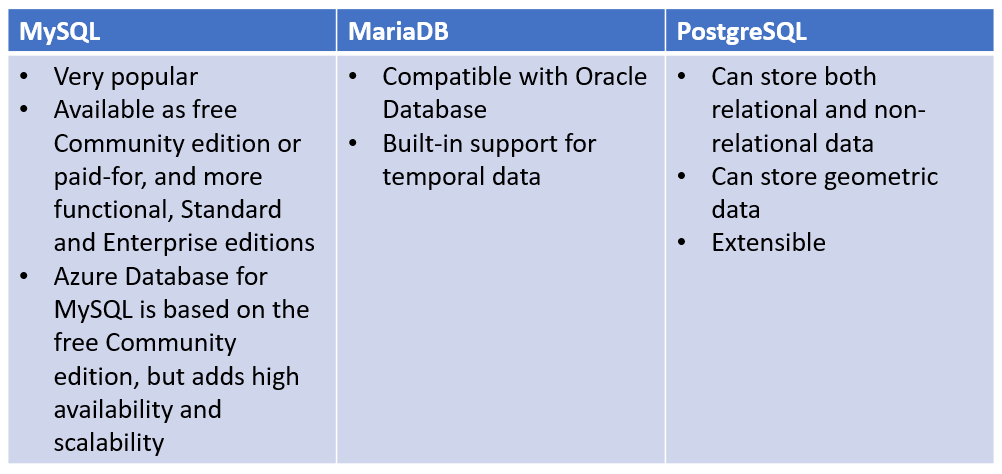
PostgreSQL, MariaDB, and MySQL are relational database management systems that are tailored for different specializations.

MySQL started life as a simple-to-use open-source database management system. It is 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.

MariaDB is a newer database management system, created by the original developers of MySQL. The database engine has since been rewritten and optimized to improve performance. MariaDB offers compatibility with Oracle Database (another popular commercial database management system). One notable feature of MariaDB is its built-in support for temporal data. A table can hold several versions of data, enabling an application to query the data as it appeared at some point in the past.

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.



## What is 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.

## What is Azure Database for MariaDB?

Azure Database for MariaDB is an implementation of the MariaDB database management system adapted to run in Azure. It's based on the MariaDB Community Edition.

The database is fully managed and controlled by Azure. Once you've provisioned the service and transferred your data, the system requires almost no additional administration.

### Benefits of Azure Database for MariaDB

Azure Database for MariaDB delivers:

* Built-in high availability with no additional cost.
* Predictable performance, using inclusive pay-as-you-go pricing.
* Scaling as needed within seconds.
* Secured protection of sensitive data at rest and in motion.
* Automatic backups and point-in-time-restore for up to 35 days.
* Enterprise-grade security and compliance.

## What is 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 are not available in Azure Database for PostgreSQL. These features are mainly 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 pgsql, 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 has two deployment options: Single-server and Hyperscale.

### Azure Database for PostgreSQL single-server

The single-server deployment option for PostgreSQL provides similar benefits as Azure Database for MySQL. You choose from three pricing tiers: Basic, General Purpose, and Memory Optimized. Each tier supports different numbers of CPUs, memory, and storage sizes—you select one based on the load you expect to support.

### Azure Database for PostgreSQL Hyperscale (Citus)

Hyperscale (Citus) is a deployment option that scales queries across multiple server nodes to support large database loads. Your database is split across nodes. Data is split into chunks based on the value of a partition key or sharding key. Consider using this deployment option for the largest database PostgreSQL deployments in the Azure Cloud.

### 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 will be 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, are not available because the server is managed and maintained by Microsoft.

Azure Database for PostgreSQL servers records information about the 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.

## Migrate data to Azure

If you have existing MySQL, MariaDB, or PostgreSQL databases running on premises, and you want to move the data to a database running the corresponding data services in Azure, you can use the [Azure Database Migration Service (DMS)](https://docs.microsoft.com/en-us/azure/dms/tutorial-postgresql-azure-postgresql-online).

The Database Migration Service enables you to restore a backup of your on-premises databases directly to databases running in Azure Data Services. You can also configure replication from an on-premises database, so that any changes made to data in that database are copied to the database running in Azure Data Services. This strategy enables you to reconfigure users and applications to connect to the database in the cloud while the on-premises system is still active; you don't have to shut down the on-premises system while you transfer users to the cloud.

# Describe provisioning relational data services

Completed100 XP

* 4 minutes

In the sample scenario, Contoso has decided that the organization will require several different relational stores. As the data engineer, you've been asked to set up data stores using Azure SQL Database, PostgreSQL, and MySQL.

In this module, you'll learn how to provision these services.

## What is provisioning?

Provisioning is the act of running series of tasks that a service provider, such as Azure SQL Database, performs to create and configure a service. Behind the scenes, the service provider will set up the various resources (disks, memory, CPUs, networks, and so on) required to run the service. You'll be assigned these resources, and they remain allocated to you (and charged to you), until you delete the service.

How the service provider provisions resources is opaque, and you don't need to be concerned with how this process works. All you do is specify parameters that determine the size of the resources required (how much disk space, memory, computing power, and network bandwidth). These parameters are determined by estimating the size of the workload that you intend to run using the service. In many cases, you can modify these parameters after the service has been created, perhaps increasing the amount of storage space or memory if the workload is greater than you initially anticipated. The act of increasing (or decreasing) the resources used by a service is called scaling.

This video summarizes the process that Azure performs when you provision a service:

Azure provides several tools you can use to provision services:

* The Azure portal. This is the most convenient way to provision a service for most users. The Azure portal displays a series of service-specific pages that prompt you for the settings required, and validates these settings, before actually provisioning the service.
* The Azure command-line interface (CLI). The CLI provides a set of commands that you can run from the operating system command prompt or the Cloud Shell in the Azure portal. You can use these commands to create and manage Azure resources. The CLI is suitable if you need to automate service creation; you can store CLI commands in scripts, and you can run these scripts programmatically. The CLI can run on Windows, macOS, and Linux computers. For detailed information about the Azure CLI, read [What is Azure CLI](https://docs.microsoft.com/en-us/cli/azure/what-is-azure-cli).
* Azure PowerShell. Many administrators are familiar with using PowerShell commands to script and automate administrative tasks. Azure provides a series of commandlets (Azure-specific commands) that you can use in PowerShell to create and manage Azure resources. You can find further information about Azure PowerShell online, at [Azure PowerShell documentation](https://docs.microsoft.com/en-us/powershell/azure). Like the CLI, PowerShell is available for Windows, macOS, and Linux.
* Azure Resource Manager templates. An Azure Resource Manager template describes the service (or services) that you want to deploy in a text file, in a format known as JSON (JavaScript Object Notation). The example below shows a template that you can use to provision an instance of Azure SQL Database.

JSONCopy

"resources": [

{

"name": "sql-server-dev",

"type": "Microsoft.Sql/servers",

"apiVersion": "2014-04-01-preview",

"location": "[parameters('location')]",

"tags": {

"displayName": "SqlServer"

},

"properties": {}

}

]

You send the template to Azure using the az deployment group create command in the Azure CLI, or New-AzResourceGroupDeployment command in Azure PowerShell. For more information about creating and using Azure Resource Manager templates to provision Azure resources, see [What are Azure Resource Manager templates?](https://docs.microsoft.com/en-us/azure/azure-resource-manager/templates/overview)

# Describe configuring relational data services

Completed100 XP

* 6 minutes

After you've provisioned a resource, you'll often need to configure it to meet the needs of your applications and environment. For example, you might need to set up network access, or open a firewall port to enable your applications to connect to the resource.

In this unit, you'll learn how to enable network access to your resources, and how you can prevent accidental exposure of your resources to third parties. You'll see how to use authentication and access control to protect the data managed by your resources.

## Configure connectivity and firewalls

The default connectivity for Azure relational data services is to disable access to the world.

### Configure connectivity to virtual networks and on-premises computers

To enable connectivity, use the **Firewalls and virtual networks** page for a service. To enable connectivity, choose **Selected networks**. Three further sections will appear, labeled **Virtual network**, **Firewall**, and **Exceptions**.

**Note**

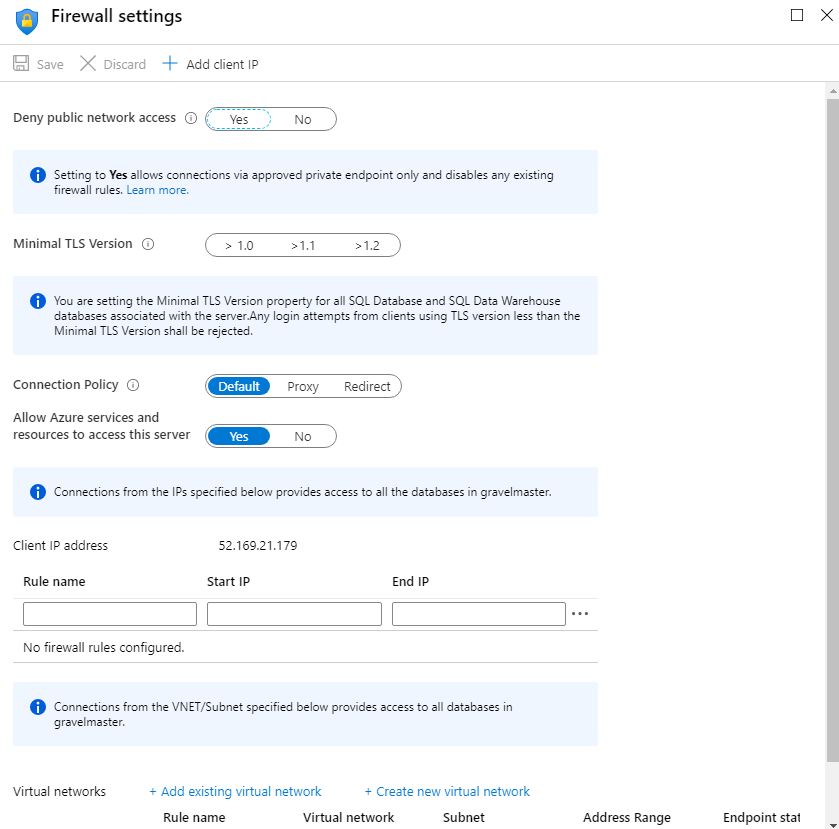
An Azure Virtual Network is a representation of your own network in the cloud. A virtual network enables you to connect virtual machines and Azure services together, in much the same way that you might use a physical network on-premises. Azure ensures that each virtual network is isolated from other virtual networks created by other users, and from the Internet. Azure enables you to specify which machines (real and virtual), and services, are allowed to access resources on the virtual network, and which ports they can use.

In the **Virtual networks** section, you can specify which virtual networks are allowed to route traffic to the service. When you create items such as web applications and virtual machines, you can add them to a virtual network. If these applications and virtual machines require access to your resource, add the virtual network containing these items to the list of allowed networks.

If you need to connect to the service from an on-premises computer, in the **Firewall** section, add the IP address of the computer. This setting creates a firewall rule that allows traffic from that address to reach the service.

The **Exceptions** setting allows you to enable access to any other services that cannot be uniquely isolated through virtual network or IP address rules.

The image below shows the **Firewalls and virtual networks** page for an Azure SQL database. MySQL and PostgreSQL have a similar page.



**Note**

Azure SQL Database communicates over port 1433. If you're trying to connect from within a corporate network, outbound traffic over port 1433 might not be allowed by your network's firewall. If so, you can't connect to your Azure SQL Database server unless your IT department opens port 1433.

**Important**

A firewall rule of 0.0.0.0 enables all Azure services to pass through the server-level firewall rule and attempt to connect to a single or pooled database through the server.

### Configure connectivity from private endpoints.

**Azure Private Endpoint** is a network interface that connects you privately and securely to a service powered by Azure Private Link. Private Endpoint uses a private IP address from your virtual network, effectively bringing the service into your virtual network. The service could be an Azure service such as Azure App Service, or your own Private Link Service. For detailed information, read [What is Azure Private Endpoint?](https://docs.microsoft.com/en-us/azure/private-link/private-endpoint-overview).

The **Private endpoint connections** page for a service allows you to specify which private endpoints, if any, are permitted access to your service. You can use the settings on this page, together with the **Firewalls and virtual networks** page, to completely lock down users and applications from accessing public endpoints to connect to your Azure SQL Database account.

## Configure authentication

With Azure Active Directory (AD) authentication, you can centrally manage the identities of database users and other Microsoft services in one central location. Central ID management provides a single place to manage database users and simplifies permission management.

You can use these identities and configure access to your relational data services.

For detailed information on using Azure AD with Azure SQL database, visit the page [What is Azure Active Directory authentication for SQL database](https://docs.microsoft.com/en-us/azure/sql-database/sql-database-aad-authentication) on the Microsoft website. You can also authenticate users connecting to [Azure Database for PostgreSQL](https://docs.microsoft.com/en-us/azure/postgresql/concepts-aad-authentication) and [Azure Database for MySQL](https://docs.microsoft.com/en-us/azure/mysql/concepts-azure-ad-authentication) with AD.

## Configure access control

Azure AD enables you to specify who, or what, can access your resources. Access control defines what a user or application can do with your resources once they've been authenticated.

Access management for cloud resources is a critical function for any organization that is using the cloud. Azure role-based access control (Azure RBAC) helps you manage who has access to Azure resources, and what they can do with those resources. For example, using RBAC you could:

* Allow one user to manage virtual machines in a subscription and another user to manage virtual networks.
* Allow a database administrator group to manage SQL databases in a subscription.
* Allow a user to manage all resources in a resource group, such as virtual machines, websites, and subnets.
* Allow an application to access all resources in a resource group.

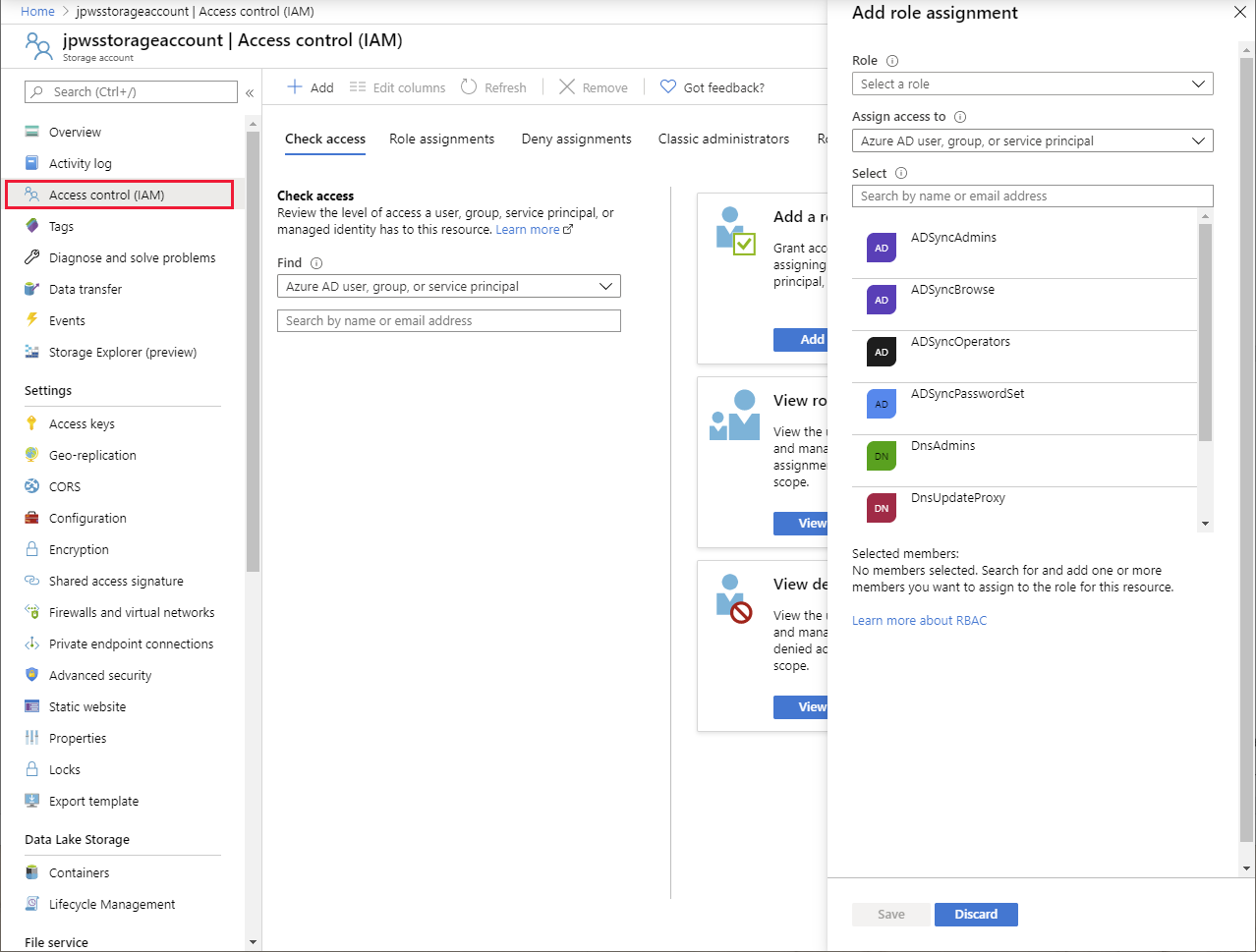
You control access to resources using Azure RBAC to create role assignments. A role assignment consists of three elements: a security principal, a role definition, and a scope.

* A **security principal** is an object that represents a user, group, service principal, or managed identity that is requesting access to Azure resources.
* A **role definition**, often abbreviated to role, is a collection of permissions. A role definition lists the operations that can be performed, such as read, write, and delete. Roles can be given high-level names, like owner, or specific names, like virtual machine reader. Azure includes several built-in roles that you can use, including:
  + **Owner** - Has full access to all resources including the right to delegate access to others.
  + **Contributor** - Can create and manage all types of Azure resources but can't grant access to others.
  + **Reader**- Can view existing Azure resources.
  + **User Access Administrator** - Lets you manage user access to Azure resources.

You can also create your own custom roles. For detailed information, see [Create or update Azure custom roles using the Azure portal](https://docs.microsoft.com/en-us/azure/role-based-access-control/custom-roles-portal) on the Microsoft website.

* A **scope** lists the set of resources that the access applies to. When you assign a role, you can further limit the actions allowed by defining a scope. This is helpful if, for example, you want to make someone a Website Contributor, but only for one resource group.

You add role assignments to a resource in the Azure portal using the **Access control (IAM)** page. The **Role assignments** tab enables you to associate a role with a security principal, defining the level of access the role has to the resource. For further information, read [Add or remove Azure role assignments using the Azure portal](https://docs.microsoft.com/en-us/azure/role-based-access-control/role-assignments-portal).



## Configure advanced data security

Apart from authentication and authorization, many services provide additional protection through advanced data security.

Advanced data security implements threat protection and assessment. Threat protection adds security intelligence to your service. This intelligence monitors the service and detects unusual patterns of activity that could be harmful, or compromise the data managed by the service. Assessment identifies potential security vulnerabilities and recommends actions to mitigate them.

The image below shows the **Advanced data security** page for SQL database. The corresponding pages for MySQL and PostgreSQL are similar.

