DP-900 – Microsoft docs

Spis treści

[Data core concepts 2](#_Toc99708831)

[Relational data in Azure 12](#_Toc99708832)

# Data core concepts

**Structured data** is data that adheres to a fixed schema, so all of the data has the same fields or properties. Most commonly, the schema for structured data entities is tabular - in other words, the data is represented in one or more tables that consist of rows to represent each instance of a data entity, and columns to represent attributes of the entity.

**Semi-structured data** is information that has some structure, but which allows for some variation between entity instances. For example, while most customers may have an email address, some might have multiple email addresses, and some might have none at all. One common format for semi-structured data is JavaScript Object Notation (JSON).

**Unstructured data** - Not all data is structured or even semi-structured. For example, documents, images, audio and video data, and binary files might not have a specific structure. This kind of data is referred to as unstructured data.

**Data storage** - Organizations typically store data in structured, semi-structured, or unstructured format to record details of entities (for example, customers and products), specific events (such as sales transactions), or other information in documents, images, and other formats. The stored data can then be retrieved for analysis and reporting later.

There are two broad categories of data store in common use:

* File stores
* Databases

The specific file format used to store data depends on a number of factors, including:

* The type of data being stored (structured, semi-structured, or unstructured).
* The applications and services that will need to read, write, and process the data.
* The need for the data files to be readable by humans, or optimized for efficient storage and processing.

**Delimited text files** - data is often stored in plain text format with specific field delimiters and row terminators. The most common format for delimited data is comma-separated values (CSV) in which fields are separated by commas, and rows are terminated by a carriage return / new line. Optionally, the first line may include the field names. Other common formats include tab-separated values (TSV) and space-delimited (in which tabs or spaces are used to separate fields), and fixed-width data in which each field is allocated a fixed number of characters. Delimited text is a good choice for structured data that needs to be accessed by a wide range of applications and services in a human-readable format.

**JavaScript Object Notation (JSON)** - JSON is a ubiquitous format in which a hierarchical document schema is used to define data entities (objects) that have multiple attributes. Each attribute might be an object (or a collection of objects); making JSON a flexible format that's good for both structured and semi-structured data.

**Extensible Markup Language (XML)** - XML is a human-readable data format that was popular in the 1990s and 2000s. It's largely been superseded by the less verbose JSON format, but there are still some systems that use XML to represent data. XML uses t-ags enclosed in angle-brackets (<../>) to define elements and attributes.

**Binary Large Object (BLOB) -** Ultimately, all files are stored as binary data (1's and 0's), but in the human-readable formats discussed above, the bytes of binary data are mapped to printable characters (typically though a character encoding scheme such as ASCII or Unicode). Some file formats however, particularly for unstructured data, store the data as raw binary that must be interpreted by applications and rendered. Common types of data stored as binary include images, video, audio, and application-specific documents.When working with data like this, data professionals often refer to the data files as *BLOBs* (Binary Large Objects).

**Optimized file formats**

While human-readable formats for structured and semi-structured data can be useful, they're typically not optimized for storage space or processing. Over time, some specialized file formats that enable compression, indexing, and efficient storage and processing have been developed.

Some common optimized file formats you might see include *Avro*, *ORC*, and *Parquet*:

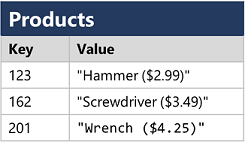
* *Avro* is a row-based format. It was created by Apache. Each record contains a header that describes the structure of the data in the record. This header is stored as JSON. The data is stored as binary information. An application uses the information in the header to parse the binary data and extract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

**Relational databases -** Relational databases are commonly used to store and query structured data. The data is stored in tables that represent entities, such as customers, products, or sales orders. Each instance of an entity is assigned a *primary key* that uniquely identifies it; and these keys are used to reference the entity instance in other tables. For example, a customer's primary key can be referenced in a sales order record to indicate which customer placed the order. This use of keys to reference data entities enables a relational database to be *normalized*; which in part means the elimination of duplicate data values so that, for example, the details of an individual customer are stored only once; not for each sales order the customer places. The tables are managed and queried using Structured Query Language (SQL), which is based on an ANSII standard, so it's similar across multiple database systems.

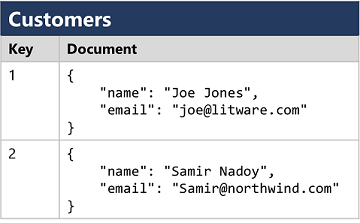
**Non-relational databases -** Non-relational databases are data management systems that don’t apply a relational schema to the data. Non-relational databases are often referred to as NoSQL database, even though some support a variant of the SQL language.

There are four common types of Non-relational database commonly in use.

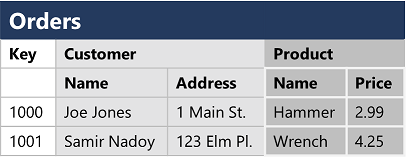
* **Key-value databases** in which each record consists of a unique key and an associated value, which can be in any format.



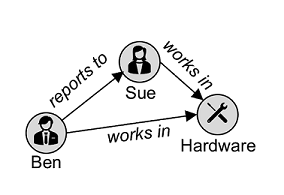
* **Document databases**, which are a specific form of key-value database in which the value is a JSON document (which the system is optimized to parse and query)



* **Column family databases**, which store 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.

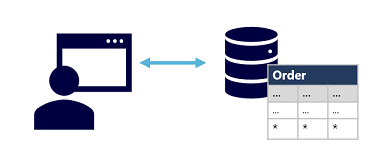


* **Graph databases**, which store entities as nodes with links to define relationships between them.



**Explore transactional data processing -** A transactional data processing system is what most people consider the primary function of business computing. A transactional system records *transactions* that encapsulate specific events that the organization wants to track. 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).



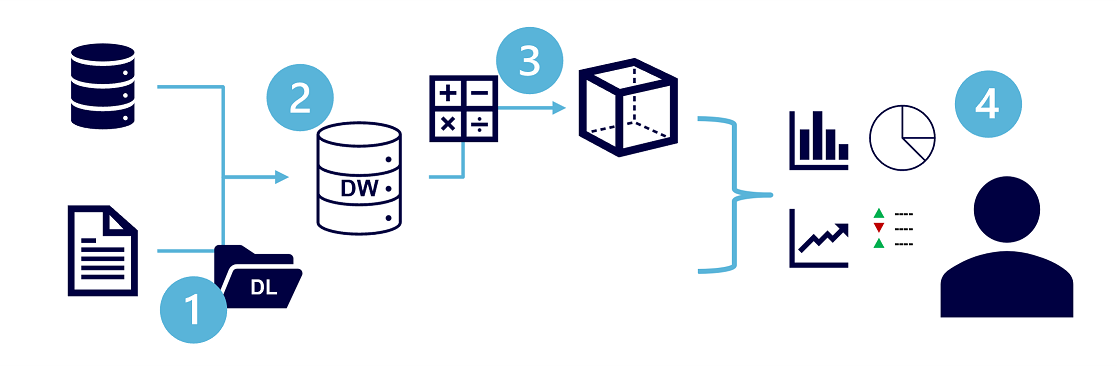
OLTP solutions rely on a database system in which data storage is optimized for both read and write operations in order to support transactional workloads in which data records are created, retrieved, updated, and deleted (often referred to as *CRUD* operations). These operations are applied transactionally, in a way that ensures the integrity of the data stored in the database. To accomplish this, OLTP systems enforce transactions that support so-called ACID semantics:

* **Atomicity** – each transaction is treated as a single unit, which succeeds completely or fails completely. For example, a transaction that involved debiting funds from one account and crediting the same amount to another account must complete both actions. If either action can't be completed, then the other action must fail.
* **Consistency** – transactions can only take the data in the database from one valid state to another. To continue the debit and credit example above, the completed state of the transaction must reflect the transfer of funds from one account to the other.
* **Isolation** – concurrent transactions cannot interfere with one another, and must result in a consistent database state. For example, while the transaction to transfer funds from one account to another is in-process, another transaction that checks the balance of these accounts must return consistent results - the balance-checking transaction can't retrieve a value for one account that reflects the balance *before* the transfer, and a value for the other account that reflects the balance *after* the transfer.
* **Durability** – when a transaction has been committed, it will remain committed. After the account transfer transaction has completed, the revised account balances are persisted so that even if the database system were to be switched off, the committed transaction would be reflected when it is switched on again.

OLTP systems are typically used to support live applications that process business data - often referred to as *line of business* (LOB) applications.

**Explore analytical data processing -** Analytical data processing typically uses read-only (or read-*mostly*) systems that store vast volumes of historical data or business metrics. Analytics can be based on a snapshot of the data at a given point in time, or a series of snapshots.

The specific details for an analytical processing system can vary between solutions, but a common architecture for enterprise-scale analytics looks like this:



1. Data files may be stored in a central data lake for analysis.
2. An extract, transform, and load (ETL) process copies data from files and OLTP databases into a data warehouse that is optimized for read activity. Commonly, a data warehouse schema is based on *fact* tables that contain numeric values you want to analyze (for example, sales amounts), with related *dimension* tables that represent the entities by which you want to measure them (for example, customer or product),
3. Data in the data warehouse may be aggregated and loaded into an online analytical processing (OLAP) model, or *cube*. Aggregated numeric values (*measures*) from fact tables are calculated for intersections of *dimensions* from dimension tables. For example, sales revenue might be totaled by date, customer, and product.
4. The data in the data lake, data warehouse, and analytical model can be queried to produce reports, visualizations, and dashboards.

*Data lakes* are common in modern data analytical processing scenarios, where a large volume of file-based data must be collected and analyzed.

*Data warehouses* are an established way to store data in a relational schema that is optimized for read operations – primarily queries to support reporting and data visualization. The data warehouse schema may require some denormalization of data in an OLTP data source (introducing some duplication to make queries perform faster).

An OLAP model is an aggregated type of data storage that is optimized for analytical workloads. Data aggregations are across dimensions at different levels, enabling you to *drill up/down* to view aggregations at multiple hierarchical levels; for example to find total sales by region, by city, or for an individual address. Because OLAP data is pre-aggregated, queries to return the summaries it contains can be run quickly.

Different types of user might perform data analytical work at different stages of the overall architecture. For example:

* Data scientists might work directly with data files in a data lake to explore and model data.
* Data Analysts might query tables directly in the data warehouse to produce complex reports and visualizations.
* Business users might consume pre-aggregated data in an analytical model in the form of reports or dashboards.

**Database Administrator**

 A database administrator is responsible for the design, implementation, maintenance, and operational aspects of on-premises and cloud-based database systems. They're responsible for the overall availability and consistent performance and optimizations of databases. 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**

 A data engineer collaborates with stakeholders to design and implement data-related workloads, including 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 non-relational 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 own the management and monitoring of data pipelines to ensure that data loads perform as expected.

**Data Analyst**

 A data analyst enables businesses to maximize the value of their data assets. They're responsible for exploring data to identify trends and relationships, designing and building analytical models, 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.

**Azure SQL**

 *Azure SQL* is the collective name for a family of relational database solutions based on the Microsoft SQL Server database engine. Specific Azure SQL services include:

* **Azure SQL Database** – a fully managed platform-as-a-service (PaaS) database hosted in Azure
* **Azure SQL Managed Instance** – a hosted instance of SQL Server with automated maintenance, which allows more flexible configuration than Azure SQL DB but with more administrative responsibility for the owner.
* **Azure SQL VM** – a virtual machine with an installation of SQL Server, allowing maximum configurability with full management responsibility.

Database administrators typically provision and manage Azure SQL database systems to support line of business (LOB) applications that need to store transactional data.

Data engineers may use Azure SQL database systems as sources for data pipelines that perform *extract*, *transform*, and *load* (ETL) operations to ingest the transactional data into an analytical system.

Data analysts may query Azure SQL databases directly to create reports, though in large organizations the data is generally combined with data from other sources in an analytical data store to support enterprise analytics.

**Azure Database for open-source relational databases**

 Azure includes managed services for popular open-source relational database systems, including:

* **Azure Database for MySQL** - a simple-to-use open-source database management system that is commonly used in *Linux*, *Apache*, *MySQL*, and *PHP* (LAMP) stack apps.
* **Azure Database for MariaDB** - 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).
* **Azure Database for PostgreSQL** - 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.

As with Azure SQL database systems, open-source relational databases are managed by database administrators to support transactional applications, and provide a data source for data engineers building pipelines for analytical solutions and data analysts creating reports.

**Azure Cosmos DB**

 Azure Cosmos DB is a global-scale non-relational (*NoSQL*) database system that supports multiple application programming interfaces (APIs), enabling you to store and manage data as JSON documents, key-value pairs, column-families, and graphs.

In some organizations, Cosmos DB instances may be provisioned and managed by a database administrator; though often software developers manage NoSQL data storage as part of the overall application architecture. Data engineers often need to integrate Cosmos DB data sources into enterprise analytical solutions that support modeling and reporting by data analysts.

**Azure Storage**

 Azure Storage is a core Azure service that enables you to store data in:

* **Blob containers** - scalable, cost-effective storage for binary files.
* **File shares** - network file shares such as you typically find in corporate networks.
* **Tables** - key-value storage for applications that need to read and write data values quickly.

Data engineers use Azure Storage to host *data lakes* - blob storage with a hierarchical namespace that enables files to be organized in folders in a distributed file system.

**Azure Data Factory**

 Azure Data Factory is an Azure service that enables you to define and schedule data pipelines to transfer and transform data. You can integrate your pipelines with other Azure services, enabling you to ingest data from cloud data stores, process the data using cloud-based compute, and persist the results in another data store.

Azure Data Factory is used by data engineers to build *extract*, *transform*, and *load* (ETL) solutions that populate analytical data stores with data from transactional systems across the organization.

**Azure Synapse Analytics**

 Azure Synapse Analytics is a comprehensive, unified data analytics solution that provides a single service interface for multiple analytical capabilities, including:

* **Pipelines** - based on the same technology as Azure Data Factory.
* **SQL** - a highly scalable SQL database engine, optimized for data warehouse workloads.
* **Apache Spark** - an open-source distributed data processing system that supports multiple programming languages and APIs, including Java, Scala, Python, and SQL.
* **Azure Synapse Data Explorer** - a high-performance data analytics solution that is optimized for real-time querying of log and telemetry data using Kusto Query Language (KQL).

Data engineers can use Azure Synapse Analytics to create a unified data analytics solution that combines data ingestion pipelines, data warehouse storage, and data lake storage through a single service.

Data analysts can use SQL and Spark pools through interactive notebooks to explore and analyze data, and take advantage of integration with services such as Azure Machine Learning and Microsoft Power BI to create data models and extract insights from the data.

**Azure Databricks**

 Azure Databricks is an Azure-integrated version of the popular Databricks platform, which combines the Apache Spark data processing platform with SQL database semantics and an integrated management interface to enable large-scale data analytics.

Data engineers can use existing Databricks and Spark skills to create analytical data stores in Azure Databricks.

Data Analysts can use the native notebook support in Azure Databricks to query and visualize data in an easy to use web-based interface.

**Azure HDInsight**

 Azure HDInsight is an Azure service that provides Azure-hosted clusters for popular Apache open-source big data processing technologies, including:

* **Apache Spark** - a distributed data processing system that supports multiple programming languages and APIs, including Java, Scala, Python, and SQL.
* **Apache Hadoop** - a distributed system that uses *MapReduce* jobs to process large volumes of data efficiently across multiple cluster nodes. MapReduce jobs can be written in Java or abstracted by interfaces such as Apache Hive - a SQL-based API that runs on Hadoop.
* **Apache HBase** - an open-source system for large-scale NoSQL data storage and querying.
* **Apache Kafka** - a message broker for data stream processing.
* **Apache Storm** - an open-source system for real-time data processing through a topology of *spouts* and *bolts*.

Data engineers can use Azure HDInsight to support big data analytics workloads that depend on multiple open-source technologies.

**Azure Stream Analytics**

 Azure Stream Analytics is a real-time stream processing engine that captures a stream of data from an input, applies a query to extract and manipulate data from the input stream, and writes the results to an output for analysis or further processing.

Data engineers can incorporate Azure Stream Analytics into data analytics architectures that capture streaming data for ingestion into an analytical data store or for real-time visualization.

**Azure Data Explorer**

 Azure Data Explorer is a standalone service that offers the same high-performance querying of log and telemetry data as the Azure Synapse Data Explorer runtime in Azure Synapse Analytics.

Data analysts can use Azure Data Explorer to query and analyze data that includes a timestamp attribute, such as is typically found in log files and *Internet-of-things* (IoT) telemetry data.

**Azure Purview**

Azure Purview logo Azure Purview provides a solution for enterprise-wide data governance and discoverability. You can use Azure Purview to create a map of your data and track data lineage across multiple data sources and systems, enabling you to find trustworthy data for analysis and reporting.

Data engineers can use Azure Purview to enforce data governance across the enterprise and ensure the integrity of data used to support analytical workloads.

**Microsoft Power BI**

 Microsoft Power BI is a platform for analytical data modeling and reporting that data analysts can use to create and share interactive data visualizations. Power BI reports can be created by using the Power BI Desktop application, and the published and delivered through web-based reports and apps in the Power BI service, as well as in the Power BI mobile app.

# Relational data in Azure

Understand relational data

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

Obraz zawierający stół

Opis wygenerowany automatycznie

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

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

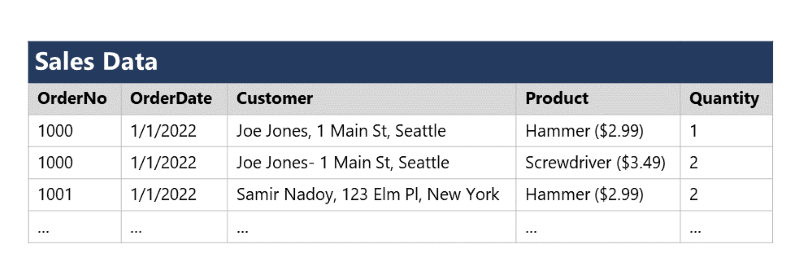
**Understand normalization**

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

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

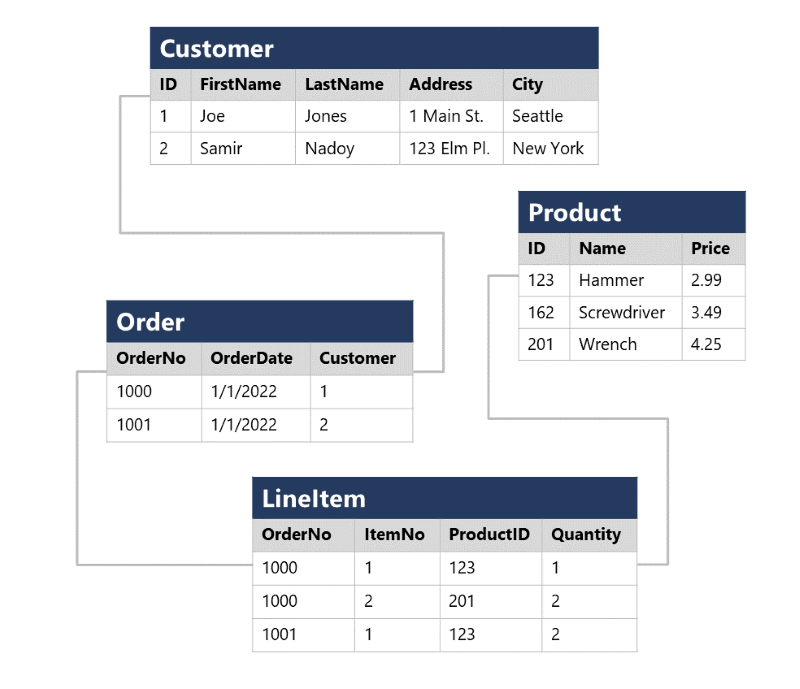
1. Separate each *entity* into its own table.
2. Separate each discrete *attribute* into its own column.
3. Uniquely identify each entity instance (row) using a *primary key*.
4. Use *foreign key* columns to link related entities.

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



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

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



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

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

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

Instances of each entity are uniquely identified by an ID or other key value, known as a *primary key*; and when one entity references another (for example, an order has an associated customer), the primary key of the related entity is stored as a *foreign key*. You can look up the address of the customer (which is stored only once) for each record in the **Order** table by referencing the corresponding record in the **Customer** table. Typically, a relational database management system (RDBMS) can enforce referential integrity to ensure that a value entered into a foreign key field has an existing corresponding primary key in the related table – for example, preventing orders for non-existent customers.

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

**Explore SQL**

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

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

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

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

Some popular dialects of SQL include:

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

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

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

**SQL statement types**

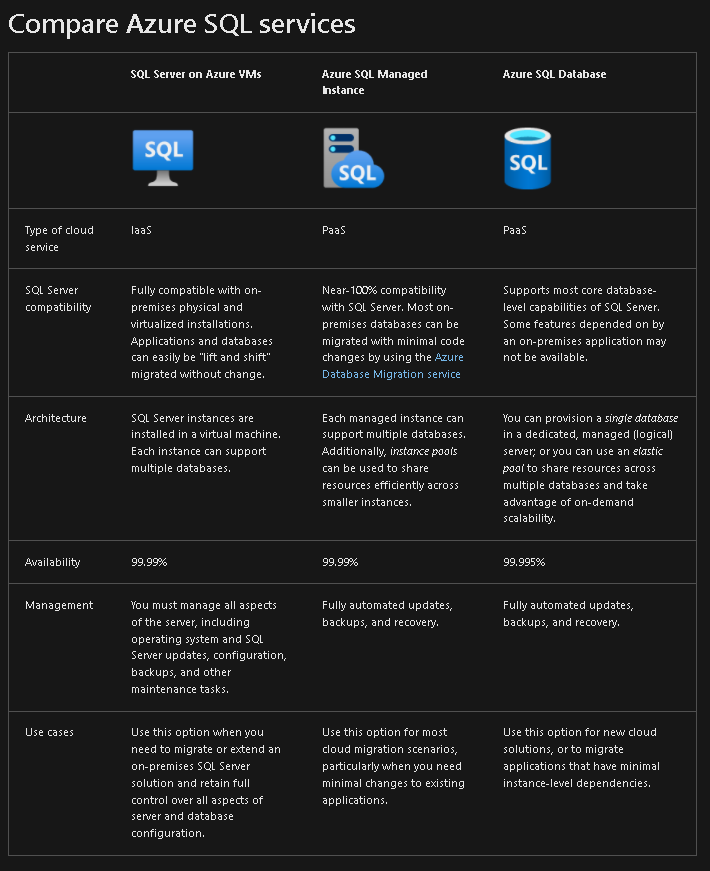
SQL statements are grouped into three main logical groups:

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

**Describe Azure SQL services and capabilities**

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

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



**SQL Server on Azure Virtual Machines**

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

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

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

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

**These capabilities enable you to:**

* Create rapid development and test scenarios when you don't want to buy on-premises non-production SQL Server hardware.
* Become lift-and-shift ready for existing applications that require fast migration to the cloud with minimal changes or no changes.
* Scale up the platform on which SQL Server is running, by allocating more memory, CPU power, and disk space to the virtual machine. You can quickly resize an Azure virtual machine without the requirement that you reinstall the software that is running on it.

**Azure SQL Database Managed Instance**

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

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

**Use cases**

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

Azure SQL Managed Instance provides features not available in Azure SQL Database (discussed below). If your system uses features such as linked servers, Service Broker (a message processing system that can be used to distribute work across servers), or Database Mail (which enables your database to send email messages to users), then you should use managed instance. To check compatibility with an existing on-premises system, you can install [Data Migration Assistant (DMA)](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**

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

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

Azure SQL Managed Instance supports SQL Server Database engine logins and logins integrated with Azure Active Directory (AD). 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.

**Azure SQL Database**

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

**Note**

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

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

**Single Database**

This option enables you to quickly set up and run a single SQL Server database. You create and run a database server in the cloud, and you access your database through this server. Microsoft manages the server, so all you have to do is configure the database, create your tables, and populate them with your data. You can scale the database if you need more storage space, memory, or processing power. By default, resources are 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 databases belonging to other Azure subscribers. Microsoft ensures the privacy of your database. Your database automatically scales and resources are allocated or deallocated as required.

**Elastic Pool**

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

**Use cases**

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

**Note**

You can use the Data Migration Assistant to detect compatibility issues with your databases that can impact database functionality in Azure SQL Database. For more information, see [**Overview of Data Migration Assistant**](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're always running the latest and most secure version of the service.

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

The service provides high availability guarantees, to ensure that your databases are available at least 99.99% 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 more resiliency and disaster recovery

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

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

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

**Describe Azure services for open-source databases**

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

**What are MySQL, MariaDB, and PostgreSQL?**

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

MySQL started life as a simple-to-use open-source database management system. It's the leading open source relational database for *Linux, Apache, MySQL, and PHP* (LAMP) stack apps. It's available in several editions; Community, Standard, and Enterprise. The Community edition is available free-of-charge, and has historically been popular as a database management system for web applications, running under Linux. Versions are also available for Windows. Standard edition offers higher performance, and uses a different technology for storing data. Enterprise edition provides a comprehensive set of tools and features, including enhanced security, availability, and scalability. The Standard and Enterprise editions are the versions most frequently used by commercial organizations, although these versions of the software aren't free.

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.

**Azure Database for MySQL**

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

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

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

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

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

**Benefits of Azure Database for MySQL**

You get the following features with Azure Database for MySQL:

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

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

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

**Azure Database for 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.

**Azure Database for PostgreSQL**

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

Some features of on-premises PostgreSQL databases aren't available in Azure Database for PostgreSQL. These features are mostly concerned with the extensions that users can add to a database to perform specialized tasks, such as writing stored procedures in various programming languages (other than 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 three deployment options: Single Server, Flexible 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 Flexible Server**

The flexible-server deploymnet option for PostgreSQL is a fully managed database service. It provides more control and server configuration customizations, and has better cost optimization controls.

**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, aren't available because the server is managed and maintained by Microsoft.

Azure Database for PostgreSQL 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.

# Non-relational data in Azure