

DP-900 – Microsoft docs

Spis treści

Data core concepts.....	2
Relational data in Azure	12
Non-relational data in Azure	23
Data analytics in Azure	28

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 ANSI 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.

Products	
Key	Value
123	"Hammer (\$2.99)"
162	"Screwdriver (\$3.49)"
201	"Wrench (\$4.25)"

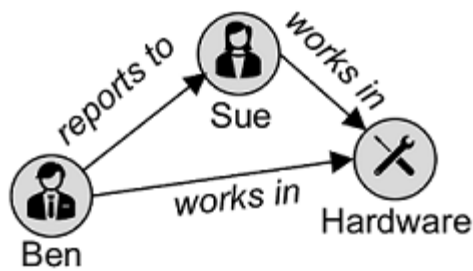
- **Document databases**, which are a specific form of key-value database in which the value is a JSON document (which the system is optimized to parse and query)

Customers	
Key	Document
1	{ "name": "Joe Jones", "email": "joe@litware.com" }
2	{ "name": "Samir Nadoy", "email": "Samir@northwind.com" }

- **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.

Orders				
Key	Customer		Product	
	Name	Address	Name	Price
1000	Joe Jones	1 Main St.	Hammer	2.99
1001	Samir Nadoy	123 Elm Pl.	Wrench	4.25

- **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

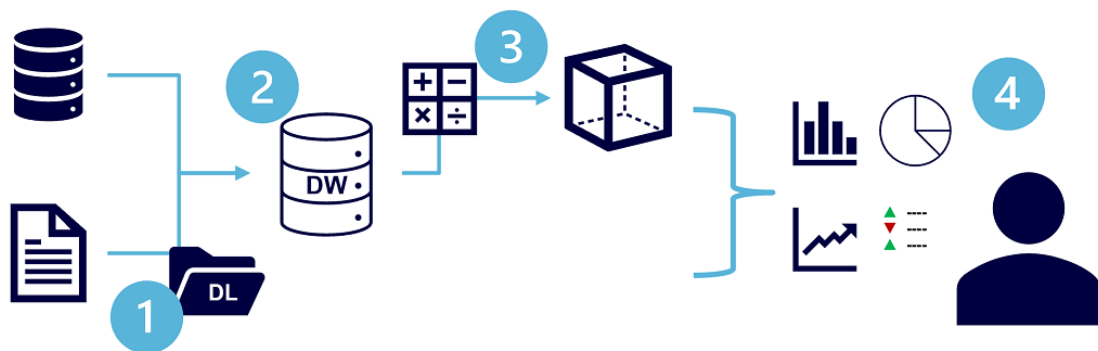
value for one account that reflects the balance *before* the transfer, and a value for the other account that reflects the balance *after* the transfer.

- **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 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.

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

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

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

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

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

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

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:

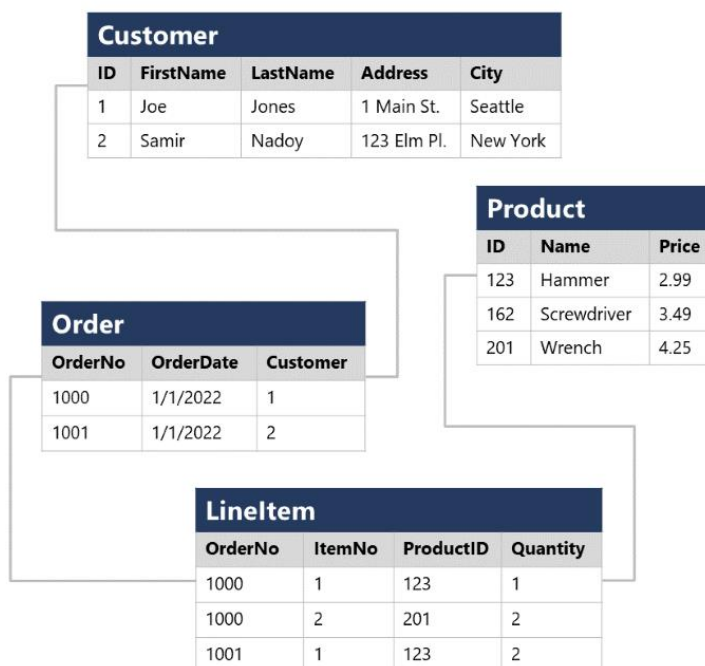
1. Separate each *entity* into its own table.
2. Separate each discrete *attribute* into its own column.
3. Uniquely identify each entity instance (row) using a *primary key*.
4. Use *foreign key* columns to link related entities.

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

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

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

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



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

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

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

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

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

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.

Compare Azure SQL services

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

SQL Server on Azure Virtual Machines

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

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

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

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 \(DMA\)](#). 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](#).

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 deployment 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 (Citrus)

Hyperscale (Citrus) 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.

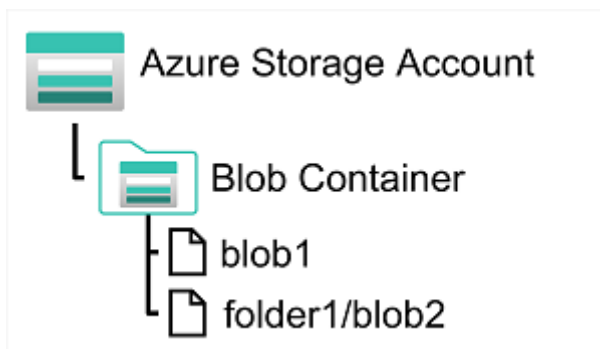
Azure Database for PostgreSQL records information about the queries run against databases on the server, and saves them in a database named *azure_sys*. You query the *query_store.qs_view* view to

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

Explore Azure blob storage

Azure Blob Storage is a service that enables you to store massive amounts of unstructured data as binary large objects, or *blobs*, in the cloud. Blobs are an efficient way to store data files in a format that is optimized for cloud-based storage, and applications can read and write them by using the Azure blob storage API.



In an Azure storage account, you store blobs in *containers*. A container provides a convenient way of grouping related blobs together. You control who can read and write blobs inside a container at the container level.

Within a container, you can organize blobs in a hierarchy of virtual folders, similar to files in a file system on disk. However, by default, these folders are simply a way of using a "/" character in a blob name to organize the blobs into namespaces. The folders are purely virtual, and you can't perform folder-level operations to control access or perform bulk operations.

Azure Blob Storage supports three different types of blob:

- **Block blobs.** A block blob is handled as a set of blocks. Each block can vary in size, up to 100 MB. A block blob can contain up to 50,000 blocks, giving a maximum size of over 4.7 TB. The block is the smallest amount of data that can be read or written as an individual unit. Block blobs are best used to store discrete, large, binary objects that change infrequently.
- **Page blobs.** A page blob is organized as a collection of fixed size 512-byte pages. A page blob is optimized to support random read and write operations; you can fetch and store data for a single page if necessary. A page blob can hold up to 8 TB of data. Azure uses page blobs to implement virtual disk storage for virtual machines.
- **Append blobs.** An append blob is a block blob optimized to support append operations. You can only add blocks to the end of an append blob; updating or deleting existing blocks isn't supported. Each block can vary in size, up to 4 MB. The maximum size of an append blob is just over 195 GB.

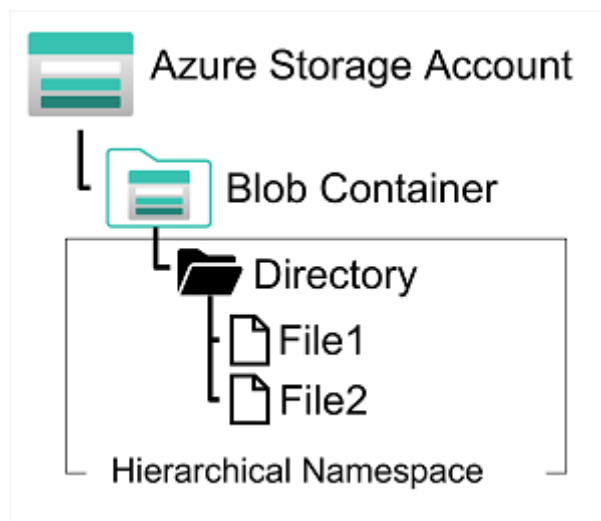
Blob storage provides three access tiers, which help to balance access latency and storage cost:

- The *Hot* tier is the default. You use this tier for blobs that are accessed frequently. The blob data is stored on high-performance media.
- The *Cool* tier has lower performance and incurs reduced storage charges compared to the Hot tier. Use the Cool tier for data that is accessed infrequently. It's common for newly created blobs to be accessed frequently initially, but less so as time passes. In these situations, you can create the blob in the Hot tier, but migrate it to the Cool tier later. You can migrate a blob from the Cool tier back to the Hot tier.
- The *Archive* tier provides the lowest storage cost, but with increased latency. The Archive tier is intended for historical data that mustn't be lost, but is required only rarely. Blobs in the Archive tier are effectively stored in an offline state. Typical reading latency for the Hot and Cool tiers is a few milliseconds, but for the Archive tier, it can take hours for the data to become available. To retrieve a blob from the Archive tier, you must change the access tier to Hot or Cool. The blob will then be rehydrated. You can read the blob only when the rehydration process is complete.

You can create lifecycle management policies for blobs in a storage account. A lifecycle management policy can automatically move a blob from Hot to Cool, and then to the Archive tier, as it ages and is used less frequently (policy is based on the number of days since modification). A lifecycle management policy can also arrange to delete outdated blobs.

Explore Azure DataLake Storage Gen2

Azure Data Lake Store (Gen1) is a separate service for hierarchical data storage for analytical data lakes, often used by so-called *big data* analytical solutions that work with structured, semi-structured, and unstructured data stored in files. Azure Data Lake Storage Gen2 is a newer version of this service that is integrated into Azure Storage; enabling you to take advantage of the scalability of blob storage and the cost-control of storage tiers, combined with the hierarchical file system capabilities and compatibility with major analytics systems of Azure Data Lake Store.



Systems like Hadoop in Azure HDInsight, Azure Databricks, and Azure Synapse Analytics can mount a distributed file system hosted in Azure Data Lake Store Gen2 and use it to process huge volumes of data.

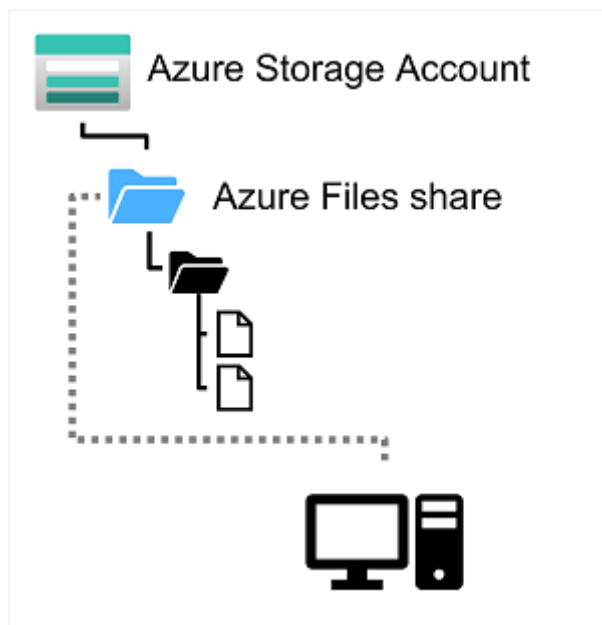
To create an Azure Data Lake Store Gen2 files system, you must enable the **Hierarchical Namespace** option of an Azure Storage account. You can do this when initially creating the storage

account, or you can upgrade an existing Azure Storage account to support Data Lake Gen2. Be aware however that upgrading is a one-way process – after upgrading a storage account to support a hierarchical namespace for blob storage, you can't revert it to a flat namespace.

Explore Azure Files

Many on-premises systems comprising a network of in-house computers make use of file shares. A file share enables you to store a file on one computer, and grant access to that file to users and applications running on other computers. This strategy can work well for computers in the same local area network, but doesn't scale well as the number of users increases, or if users are located at different sites.

Azure Files is essentially a way to create cloud-based network shares, such as you typically find in on-premises organizations to make documents and other files available to multiple users. By hosting file shares in Azure, organizations can eliminate hardware costs and maintenance overhead, and benefit from high availability and scalable cloud storage for files.



You create Azure File storage in a storage account. Azure Files enables you to share up to 100 TB of data in a single storage account. This data can be distributed across any number of file shares in the account. The maximum size of a single file is 1 TB, but you can set quotas to limit the size of each share below this figure. Currently, Azure File Storage supports up to 2000 concurrent connections per shared file.

After you've created a storage account, you can upload files to Azure File Storage using the Azure portal, or tools such as the *AzCopy* utility. You can also use the Azure File Sync service to synchronize locally cached copies of shared files with the data in Azure File Storage.

Azure File Storage offers two performance tiers. The *Standard* tier uses hard disk-based hardware in a datacenter, and the *Premium* tier uses solid-state disks. The *Premium* tier offers greater throughput, but is charged at a higher rate.

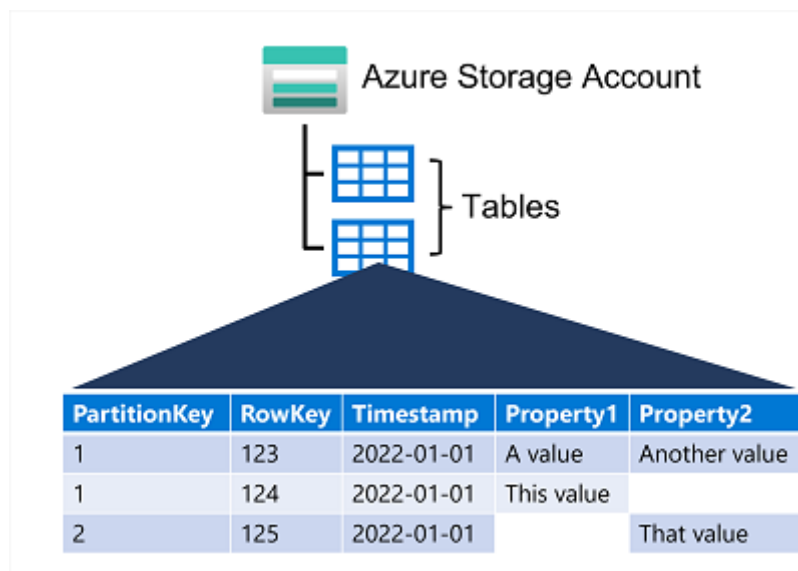
Azure Files supports two common network file sharing protocols:

- *Server Message Block* (SMB) file sharing is commonly used across multiple operating systems (Windows, Linux, macOS).

- *Network File System (NFS)* shares are used by some Linux and macOS versions. To create an NFS share, you must use a premium tier storage account and create and configure a virtual network through which access to the share can be controlled.

Explore Azure Tables

Azure Table Storage is a NoSQL storage solution that makes use of tables containing *key/value* data items. Each item is represented by a row that contains columns for the data fields that need to be stored.



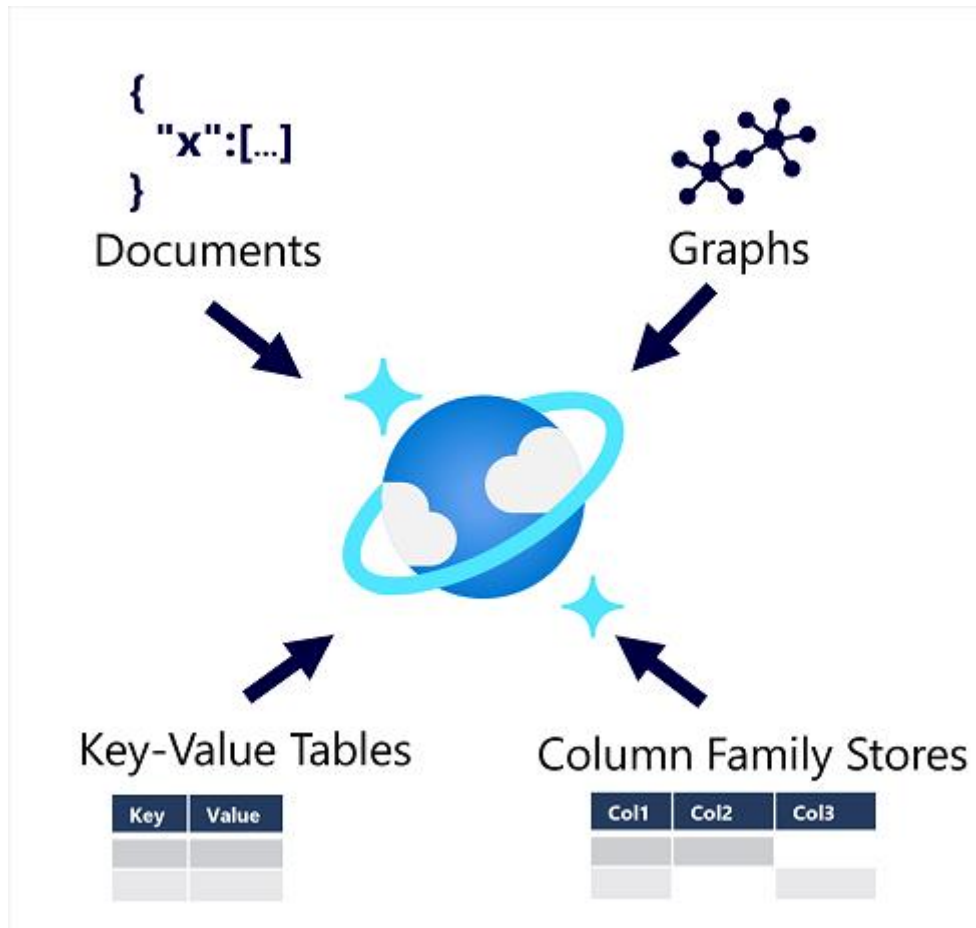
However, don't be misled into thinking that an Azure Table Storage table is like a table in a relational database. An Azure Table enables you to store semi-structured data. All rows in a table must have a unique key (composed of a partition key and a row key), and when you modify data in a table, a *timestamp* column records the date and time the modification was made; but other than that, the columns in each row can vary. Azure Table Storage tables have no concept of foreign keys, relationships, stored procedures, views, or other objects you might find in a relational database. Data in Azure Table storage is usually denormalized, with each row holding the entire data for a logical entity. For example, a table holding customer information might store the first name, last name, one or more telephone numbers, and one or more addresses for each customer. The number of fields in each row can be different, depending on the number of telephone numbers and addresses for each customer, and the details recorded for each address. In a relational database, this information would be split across multiple rows in several tables.

To help ensure fast access, Azure Table Storage splits a table into partitions. Partitioning is a mechanism for grouping related rows, based on a common property or partition key. Rows that share the same partition key will be stored together. Partitioning not only helps to organize data, it can also improve scalability and performance in the following ways:

- Partitions are independent from each other, and can grow or shrink as rows are added to, or removed from, a partition. A table can contain any number of partitions.
- When you search for data, you can include the partition key in the search criteria. This helps to narrow down the volume of data to be examined, and improves performance by reducing the amount of I/O (input and output operations, or *reads* and *writes*) needed to locate the data.

The key in an Azure Table Storage table comprises two elements; the partition key that identifies the partition containing the row, and a row key that is unique to each row in the same partition. Items in the same partition are stored in row key order. If an application adds a new row to a table, Azure ensures that the row is placed in the correct position in the table. This scheme enables an application to quickly perform *point* queries that identify a single row, and *range* queries that fetch a contiguous block of rows in a partition.

Describe Azure Cosmos DB



Azure Cosmos DB supports multiple application programming interfaces (APIs) that enable developers to use the programming semantics of many common kinds of data store to work with data in a Cosmos DB database. The internal data structure is abstracted, enabling developers to use Cosmos DB to store and query data using APIs with which they're already familiar.

Note

An *API* is an *Application Programming Interface*. Database management systems (and other software frameworks) provide a set of APIs that developers can use to write programs that need to access data. The APIs vary for different database management systems.

Cosmos DB uses indexes and partitioning to provide fast read and write performance and can scale to massive volumes of data. You can enable multi-region writes, adding the Azure regions of your choice to your Cosmos DB account so that globally distributed users can each work with data in their local replica.

When to use Cosmos DB

Cosmos DB is a highly scalable database management system. Cosmos DB automatically allocates space in a container for your partitions, and each partition can grow up to 10 GB in size. Indexes are created and maintained automatically. There's virtually no administrative overhead.

Cosmos DB is a foundational service in Azure. Cosmos DB has been used by many of Microsoft's products for mission critical applications at global scale, including Skype, Xbox, Microsoft 365, Azure, and many others. Cosmos DB is highly suitable for the following scenarios:

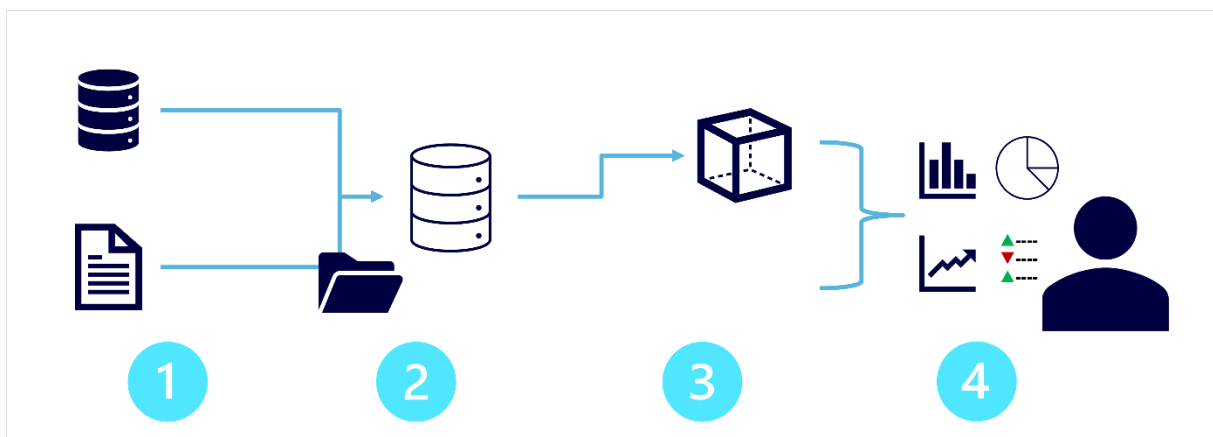
- *IoT and telematics.* These systems typically ingest large amounts of data in frequent bursts of activity. Cosmos DB can accept and store this information quickly. The data can then be used by analytics services, such as Azure Machine Learning, Azure HDInsight, and Power BI. Additionally, you can process the data in real-time using Azure Functions that are triggered as data arrives in the database.
- *Retail and marketing.* Microsoft uses Cosmos DB for its own e-commerce 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.* 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 can be used to build rich iOS and Android applications using the popular Xamarin framework.

For additional information about uses for Cosmos DB, read [Common Azure Cosmos DB use cases](#).

Data analytics in Azure

Describe modern data warehousing

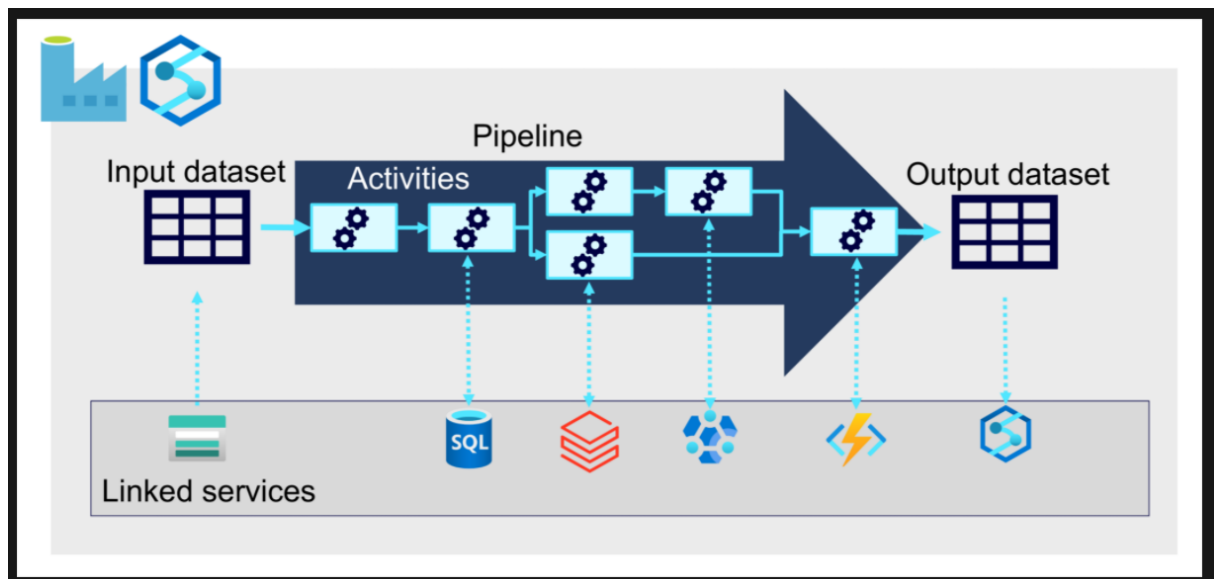
Modern data warehousing architecture can vary, as can the specific technologies used to implement it; but in general, the following elements are included:



1. **Data ingestion and processing** – data from one or more transactional data stores, files, real-time streams, or other sources is loaded into a data lake or a relational data warehouse. The load operation usually involves an *extract, transform, and load* (ETL) or *extract, load, and transform* (ELT) process in which the data is cleaned, filtered, and restructured for analysis. In ETL processes, the data is transformed before being loaded into an analytical store, while in an ELT process the data is copied to the store and then transformed. Either way, the resulting data structure is optimized for analytical queries. The data processing is often performed by distributed systems that can process high volumes of data in parallel using multi-node clusters. Data ingestion includes both batch processing of static data and real-time processing of streaming data.
2. **Analytical data store** – data stores for large scale analytics include relational *data warehouses*, file-system based *data lakes*, and hybrid architectures that combine features of data warehouses and data lakes (sometimes called *data lakehouses* or *lake databases*). We'll discuss these in more depth later.
3. **Analytical data model** – while data analysts and data scientists can work with the data directly in the analytical data store, it's common to create one or more data models that pre-aggregate the data to make it easier to produce reports, dashboards, and interactive visualizations. Often these data models are described as *cubes*, in which numeric data values are aggregated across one or more dimensions (for example, to determine total sales by product and region). The model encapsulates the relationships between data values and dimensional entities to support "drill-up/drill-down" analysis.
4. **Data visualization** – data analysts consume data from analytical models, and directly from analytical stores to create reports, dashboards, and other visualizations. Additionally, users in an organization who may not be technology professionals might perform self-service data analysis and reporting. The visualizations from the data show trends, comparisons, and key performance indicators (KPIs) for a business or other organization, and can take the form of printed reports, graphs and charts in documents or PowerPoint presentations, web-based dashboards, and interactive environments in which users can explore data visually.

Explore data ingestion pipelines

Now that you understand a little about the architecture of a modern data warehousing solution, and some of the distributed processing technologies that can be used to handle large volumes of data, it's time to explore how data is ingested into an analytical data store from one or more sources.



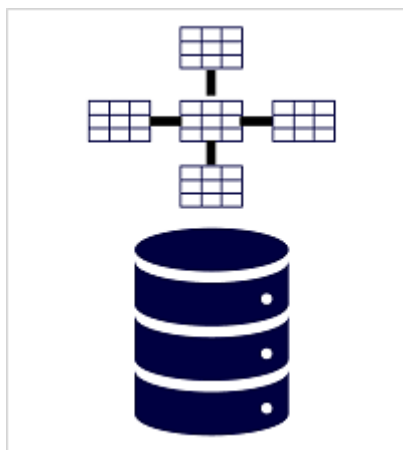
On Azure, large-scale data ingestion is best implemented by creating pipelines that orchestrate ETL processes. You can create and run pipelines using Azure Data Factory, or you can use the same pipeline engine in Azure Synapse Analytics if you want to manage all of the components of your data warehousing solution in a unified workspace.

In either case, pipelines consist of one or more activities that operate on data. An input dataset provides the source data, and activities can be defined as a data flow that incrementally manipulates the data until an output dataset is produced. Pipelines use linked services to load and process data – enabling you to use the right technology for each step of the workflow. For example, you might use an Azure Blob Store linked service to ingest the input dataset, and then use services such as Azure SQL Database to run a stored procedure that looks up related data values, before running a data processing task on Azure Databricks or Azure HDInsight, or apply custom logic using an Azure Function. Finally, you can save the output dataset in a linked service such as Azure Synapse Analytics. Pipelines can also include some built-in activities, which don't require a linked service.

Explore analytical data stores

There are two common types of analytical data store.

Data warehouses



A *data warehouse* is a relational database in which the data is stored in a schema that is optimized for data analytics rather than transactional workloads. Commonly, the data from a transactional store is denormalized into a schema in which numeric values are stored in central *fact* tables, which are related to one or more *dimension* tables that represent entities by which the data can be aggregated. For example a fact table might contain sales order data, which can be aggregated by customer, product, store, and time dimensions (enabling you, for example, to easily find monthly total sales revenue by product for each store). This kind of fact and dimension table schema is called a *star schema*; though it's often extended into a *snowflake schema* by adding additional tables related to the dimension tables to represent dimensional hierarchies (for example, product might be related to product categories). A data warehouse is a great choice when you have transactional data that can be organized into a structured schema of tables, and you want to use SQL to query them.

Data lakes



A *data lake* is a file store, usually on a distributed file system for high performance data access. Technologies like Spark or Hadoop are often used to process queries on the stored files and return data for reporting and analytics. These systems often apply a *schema-on-read* approach to define tabular schemas on semi-structured data files at the point where the data is read for analysis, without applying constraints when it's stored. Data lakes are great for supporting a mix of structured, semi-structured, and even unstructured data that you want to analyze without the need for schema enforcement when the data is written to the store.

Hybrid approaches

You can use a hybrid approach that combines features of data lakes and data warehouses in a *lake database* or *data lakehouse*. The raw data is stored as files in a data lake, and a relational storage layer abstracts the underlying files and expose them as tables, which can be queried using SQL. SQL pools in Azure Synapse Analytics include *PolyBase*, which enables you to define external tables based on files in a datalake (and other sources) and query them using SQL. Synapse Analytics also supports a Lake Database approach in which you can use database templates to define the relational schema of your data warehouse, while storing the underlying data in data lake storage – separating the storage and compute for your data warehousing solution. Data lakehouses are a relatively new approach in Spark-based systems, and are enabled through technologies like *Delta Lake*; which adds relational storage capabilities to Spark, so you can define tables that enforce schemas and transactional consistency, support batch-loaded and streaming data sources, and provide a SQL API for querying.

Azure services for analytical stores

On Azure, there are three main services that you can use to implement a large-scale analytical store



[Azure Synapse Analytics](#) is a unified, end-to-end solution for large scale data analytics. It brings together multiple technologies and capabilities, enabling you to combine the data integrity and reliability of a scalable, high-performance SQL Server based relational data warehouse with the flexibility of a data lake and open-source Apache Spark. It also includes native support for log and telemetry analytics with Azure Synapse Data Explorer pools, as well as built in data pipelines for data ingestion and transformation. All Azure Synapse Analytics services can be managed through a single, interactive user interface called Azure Synapse Studio, which includes the ability to create interactive notebooks in which Spark code and markdown content can be combined. Synapse Analytics is a great choice when you want to create a single, unified analytics solution on Azure.



[Azure Databricks](#) is an Azure implementation of the popular Databricks platform. Databricks is a comprehensive data analytics solution built on Apache Spark, and offers native SQL capabilities as well as workload-optimized Spark clusters for data analytics and data science. Databricks provides an interactive user interface through which the system can be managed and data can be explored in interactive notebooks. Due to its common use on multiple cloud platforms, you might want to consider using Azure Databricks as your analytical store if you want to use existing expertise with the platform or if you need to operate in a multi-cloud environment or support a cloud-portable solution.



[Azure HDInsight](#) is an Azure service that supports multiple open-source data analytics cluster types. Although not as user-friendly as Azure Synapse Analytics and Azure Databricks, it can be a suitable option if your analytics solution relies on multiple open-source frameworks or if you need to migrate an existing on-premises Hadoop-based solution to the cloud.

Note

Each of these services can be thought of as an analytical data *store*, in the sense that they provide a schema and interface through which the data can be queried. In many cases however, the data is actually stored in a data lake and the service is used to *process* the data and run queries. Some solutions might even combine the use of these services. An *extract, load, and transform* (ELT) ingestion process might copy data into the data lake, and then use one of these services to transform the data, and another to query it. For example, a pipeline might use a MapReduce job running in HDInsight or a notebook running in Azure Databricks to process a large volume of data in the data lake, and then load it into tables in a SQL pool in Azure Synapse Analytics.

Understand batch and stream processing

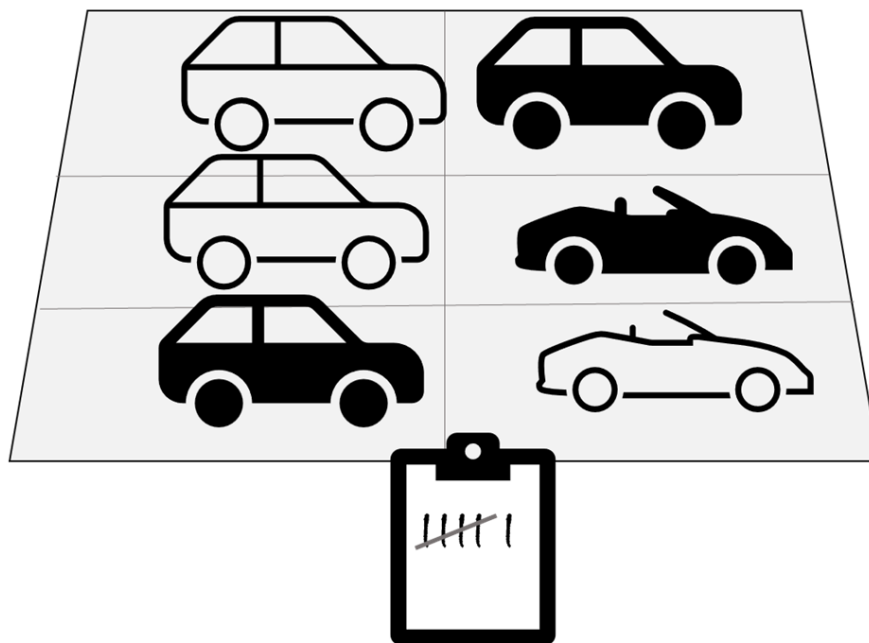
Data processing is simply the conversion of raw data to meaningful information through a process. There are two general ways to process data:

- *Batch processing*, in which multiple data records are collected and stored before being processed together in a single operation.
- *Stream processing*, in which a source of data is constantly monitored and processed in real time as new data events occur.

Understand batch processing

In batch processing, newly arriving data elements are collected and stored, and the whole group is processed together 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.

For example, suppose you want to analyze road traffic by counting the number of cars on a stretch of road. A batch processing approach to this would require that you collect the cars in a parking lot, and then count them in a single operation while they're at rest.



If the road is busy, with a large number of cars driving along at frequent intervals, this approach may be impractical; and note that you don't get any results until you have parked a batch of cars and counted them.

A real world example of batch processing is the way that credit card companies handle billing. The customer doesn't receive a bill for each separate credit card purchase but one monthly bill for all of that month's purchases.

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.

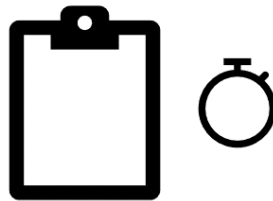
Disadvantages of batch processing include:

- 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 can prevent a batch job from running.

Understand stream processing

In stream processing, each new piece of data is processed when it arrives. Unlike batch processing, there's no waiting until the next batch processing interval - data is processed as individual units in real-time rather than being processed a batch at a time. Stream data processing is beneficial in scenarios where new, dynamic data is generated on a continual basis.

For example, a better approach to our hypothetical car counting problem might be to apply a *streaming* approach, by counting the cars in real-time as they pass:



In this approach, you don't need to wait until all of the cars have parked to start processing them, and you can aggregate the data over time intervals; for example, by counting the number of cars that pass each minute.

Real world 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 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:

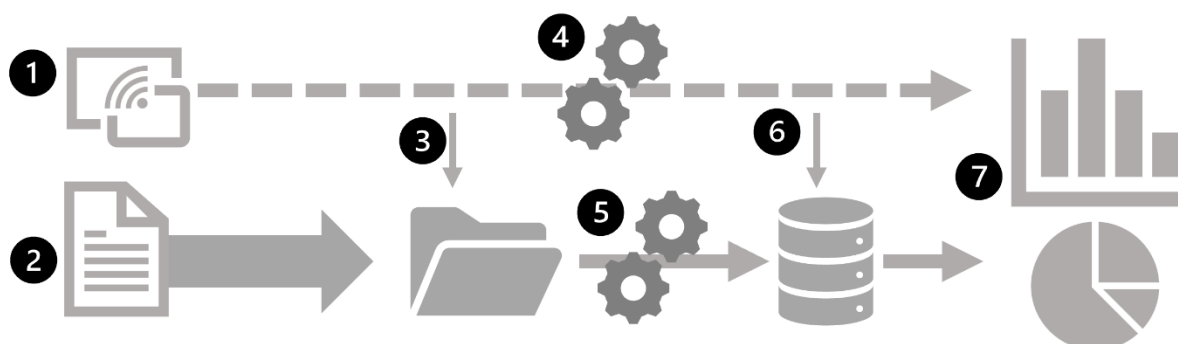
- **Data scope:** Batch processing can process all the data in the dataset. Stream processing typically only has access to the most recent data received, or within a rolling time window (the last 30 seconds, for example).
- **Data size:** Batch processing is suitable for handling large datasets efficiently.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

Combine batch and stream processing

Many large-scale analytics solutions include a mix of batch and stream processing, enabling both historical and real-time data analysis. It's common for stream processing solutions to capture real-time data, process it by filtering or aggregating it, and present it through real-time dashboards and visualizations (for example, showing the running total of cars that have passed along a road within the current hour), while also persisting the processed results in a data store for historical analysis alongside batch processed data (for example, to enable analysis of traffic volumes over the past year).

Even when real-time analysis or visualization of data is not required, streaming technologies are often used to capture real-time data and store it in a data store for subsequent batch processing (this is the equivalent of redirecting all of the cars that travel along a road into a parking lot before counting them).

The following diagram shows some ways in which batch and stream processing can be combined in a large-scale data analytics architecture.



1. Data events from a streaming data source are captured in real-time.
2. Data from other sources is ingested into a data store (often a *data lake*) for batch processing.

3. If real-time analytics is not required, the captured streaming data is written to the data store for subsequent batch processing.
4. When real-time analytics is required, a stream processing technology is used to prepare the streaming data for real-time analysis or visualization; often by filtering or aggregating the data over temporal windows.
5. The non-streaming data is periodically batch processed to prepare it for analysis, and the results are persisted in an analytical data store (often referred to as a *data warehouse*) for historical analysis.
6. The results of stream processing may also be persisted in the analytical data store to support historical analysis.
7. Analytical and visualization tools are used to present and explore the real-time and historical data.

Note

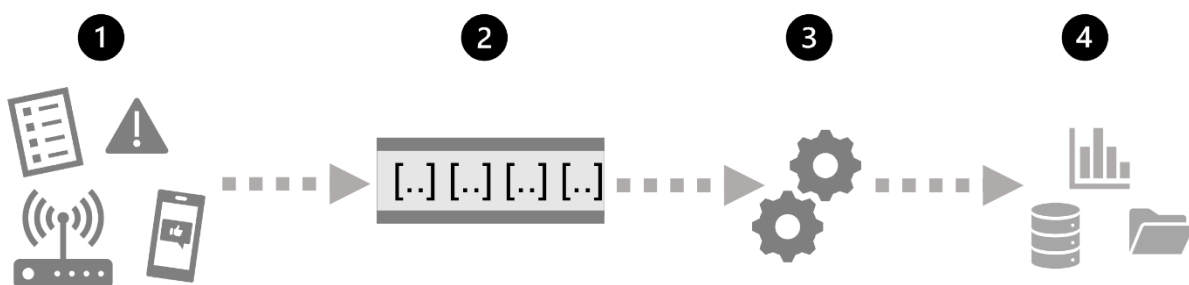
Commonly used solution architectures for combined batch and stream data processing include *lambda* and *delta* architectures. Details of these architectures are beyond the scope of this course, but they incorporate technologies for both large-scale batch data processing and real-time stream processing to create an end-to-end analytical solution.

Explore common elements of stream processing architecture

There are many technologies that you can use to implement a stream processing solution, but while specific implementation details may vary, there are common elements to most streaming architectures.

A general architecture for stream processing

At its simplest, a high-level architecture for stream processing looks like this:



1. An event generates some data. This might be a signal being emitted by a sensor, a social media message being posted, a log file entry being written, or any other occurrence that results in some digital data.
2. The generated data is captured in a streaming *source* for processing. In simple cases, the source may be a folder in a cloud data store or a table in a database. In more robust streaming solutions, the source may be a "queue" that encapsulates logic to ensure that event data is processed in order and that each event is processed only once.

3. The event data is processed, often by a perpetual query that operates on the event data to select data for specific types of events, project data values, or aggregate data values over temporal (time-based) periods (or *windows*) - for example, by counting the number of sensor emissions per minute.
4. The results of the stream processing operation are written to an output (or *sink*), which may be a file, a database table, a real-time visual dashboard, or another queue for further processing by a subsequent downstream query.

Real-time analytics in Azure

Microsoft Azure supports multiple technologies that you can use to implement real-time analytics of streaming data, including:

- **Azure Stream Analytics:** A platform-as-a-service (PaaS) solution that you can use to define *streaming jobs* that ingest data from a streaming source, apply a perpetual query, and write the results to an output.
- **Spark Structured Streaming:** An open-source library that enables you to develop complex streaming solutions on Apache Spark based services, including **Azure Synapse Analytics**, **Azure Databricks**, and **Azure HDInsight**.
- **Azure Data Explorer:** A high-performance database and analytics service that is optimized for ingesting and querying batch or streaming data with a time-series element, and which can be used as a standalone Azure service or as an **Azure Synapse Data Explorer** runtime in an Azure Synapse Analytics workspace.

Sources for stream processing

The following services are commonly used to ingest data for stream processing on Azure:

- **Azure Event Hubs:** A data ingestion service that you can use to manage queues of event data, ensuring that each event is processed in order, exactly once.
- **Azure IoT Hub:** A data ingestion service that is similar to Azure Event Hubs, but which is optimized for managing event data from *Internet-of-things* (IoT) devices.
- **Azure Data Lake Store Gen 2:** A highly scalable storage service that is often used in *batch processing* scenarios, but which can also be used as a source of streaming data.
- **Apache Kafka:** An open-source data ingestion solution that is commonly used together with Apache Spark. You can use Azure HDInsight to create a Kafka cluster.

Sinks for stream processing

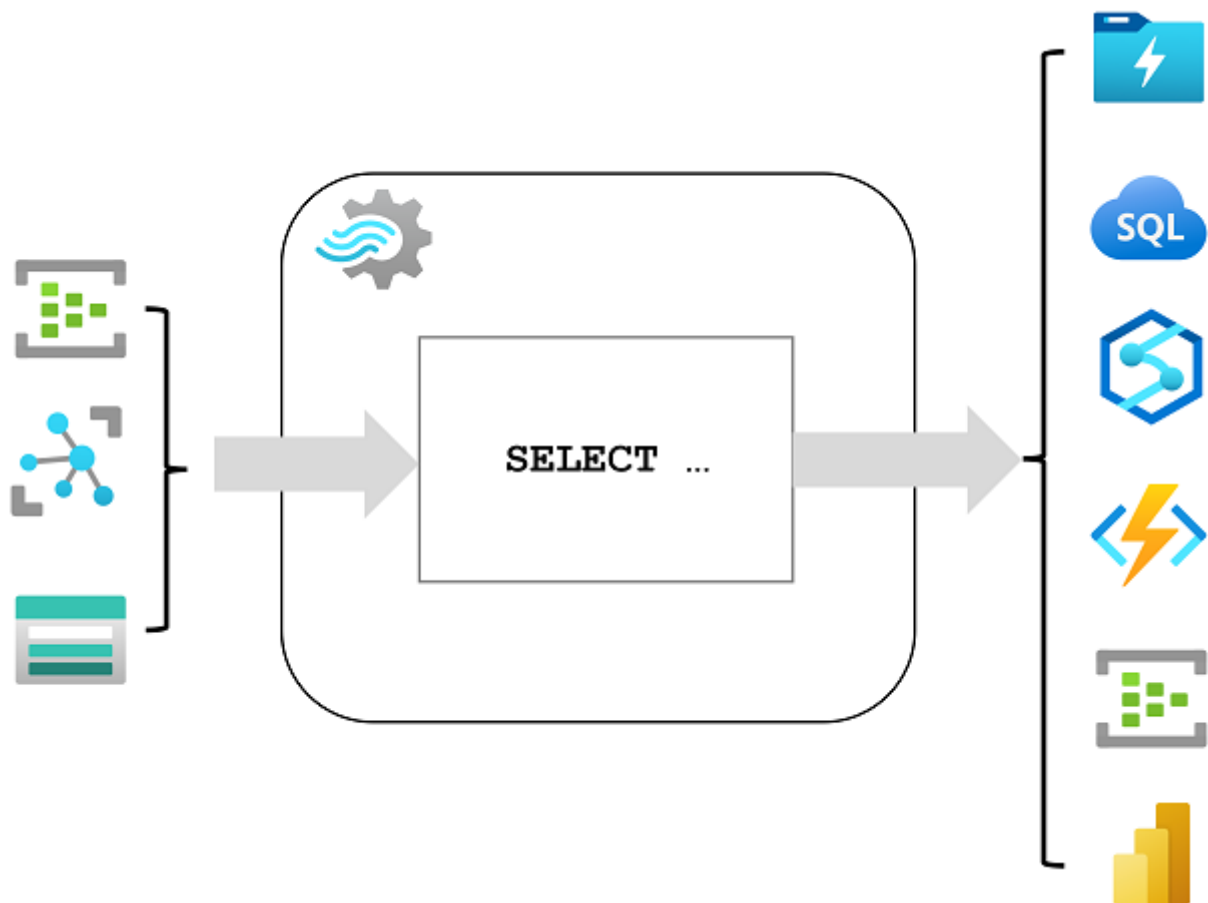
The output from stream processing is often sent to the following services:

- **Azure Event Hubs:** Used to queue the processed data for further downstream processing.
- **Azure Data Lake Store Gen 2 or Azure blob storage:** Used to persist the processed results as a file.
- **Azure SQL Database or Azure Synapse Analytics, or Azure Databricks:** Used to persist the processed results in a database table for querying and analysis.
- **Microsoft Power BI:** Used to generate real time data visualizations in reports and dashboards.

Explore Azure Stream Analytics

Azure Stream Analytics is a service for complex event processing and analysis of streaming data. Stream Analytics is used to:

- Ingest data from an *input*, such as an Azure event hub, Azure IoT Hub, or Azure Storage blob container.
- Process the data by using a *query* to select, project, and aggregate data values.
- Write the results to an *output*, such as Azure Data Lake Gen 2, Azure SQL Database, Azure Synapse Analytics, Azure Functions, Azure event hub, Microsoft Power BI, or others.



Once started, a Stream Analytics query will run perpetually, processing new data as it arrives in the input and storing results in the output.

Azure Stream Analytics is a great technology choice when you need to continually capture data from a streaming source, filter or aggregate it, and send the results to a data store or downstream process for analysis and reporting.

Azure Stream Analytics jobs and clusters

The easiest way to use Azure Stream Analytics is to create a Stream Analytics *job* in an Azure subscription, configure its input(s) and output(s), and define the query that the job will use to process the data. The query is expressed using structured query language (SQL) syntax, and can incorporate static reference data from multiple data sources to supply lookup values that can be combined with the streaming data ingested from an input.

If your stream process requirements are complex or resource-intensive, you can create a Stream Analysis *cluster*, which uses the same underlying processing engine as a Stream Analytics job, but in a dedicated tenant (so your processing is not affected by other customers) and with configurable scalability that enables you to define the right balance of throughput and cost for your specific scenario.

Note

To learn more about the capabilities of Azure Stream Analytics, see the [Azure Stream Analytics documentation](#).

Explore Apache Spark on Microsoft Azure

Apache Spark is a distributed processing framework for large scale data analytics. You can use Spark on Microsoft Azure in the following services:

- Azure Synapse Analytics
- Azure Databricks
- Azure HDInsight

Spark can be used to run code (usually written in Python, Scala, or Java) in parallel across multiple cluster nodes, enabling it to process very large volumes of data efficiently. Spark can be used for both batch processing and stream processing.

Spark Structured Streaming

To process streaming data on Spark, you can use the *Spark Structured Streaming* library, which provides an application programming interface (API) for ingesting, processing, and outputting results from perpetual streams of data.

Spark Structured Streaming is built on a ubiquitous structure in Spark called a *dataframe*, which encapsulates a table of data. You use the Spark Structured Streaming API to read data from a real-time data source, such as a Kafka hub, a file store, or a network port, into a "boundless" dataframe that is continually populated with new data from the stream. You then define a query on the dataframe that selects, projects, or aggregates the data - often in temporal windows. The results of the query generate another dataframe, which can be persisted for analysis or further processing.



Spark Structured Streaming is a great choice for real-time analytics when you need to incorporate streaming data into a Spark based data lake or analytical data store.

Note

For more information about Spark Structured Streaming, see the [Spark Structured Streaming programming guide](#).

Delta Lake

Delta Lake is an open-source storage layer that adds support for transactional consistency, schema enforcement, and other common data warehousing features to data lake storage. It also unifies storage for streaming and batch data, and can be used in Spark to define relational tables for both batch and stream processing. When used for stream processing, a Delta Lake table can be used as a streaming source for queries against real-time data, or as a sink to which a stream of data is written.

The Spark runtimes in Azure Synapse Analytics and Azure Databricks include support for Delta Lake.

Delta Lake combined with Spark Structured Streaming is a good solution when you need to abstract batch and stream processed data in a data lake behind a relational schema for SQL-based querying and analysis.

Note

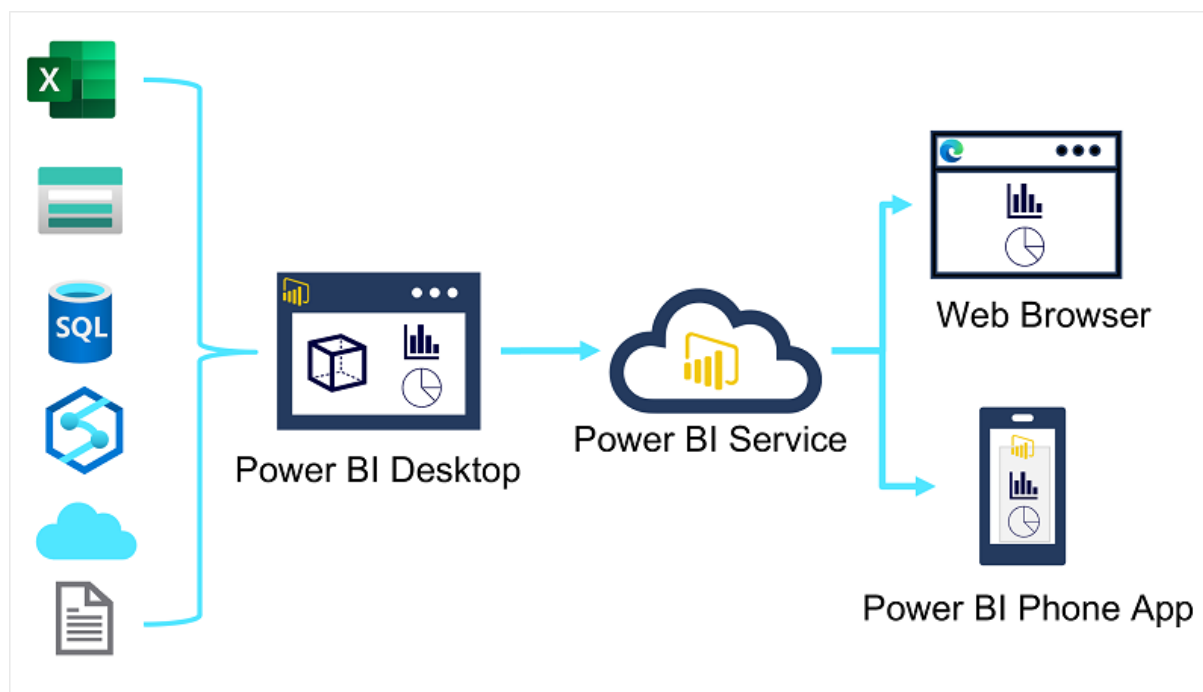
For more information about Delta Lake, see [What is Delta Lake?](#)

Describe Power BI tools and workflow

There are many data visualization tools that data analysts can use to explore data and summarize insights visually; including chart support in productivity tools like Microsoft Excel and built-in data visualization widgets in notebooks used to explore data in services such as Azure Synapse Analytics and Azure Databricks. However, for enterprise-scale business analytics, an integrated solution that can support complex data modeling, interactive reporting, and secure sharing is often required.

Microsoft Power BI

Microsoft Power BI is a suite of tools and services that data analysts can use to build interactive data visualizations for business users to consume.



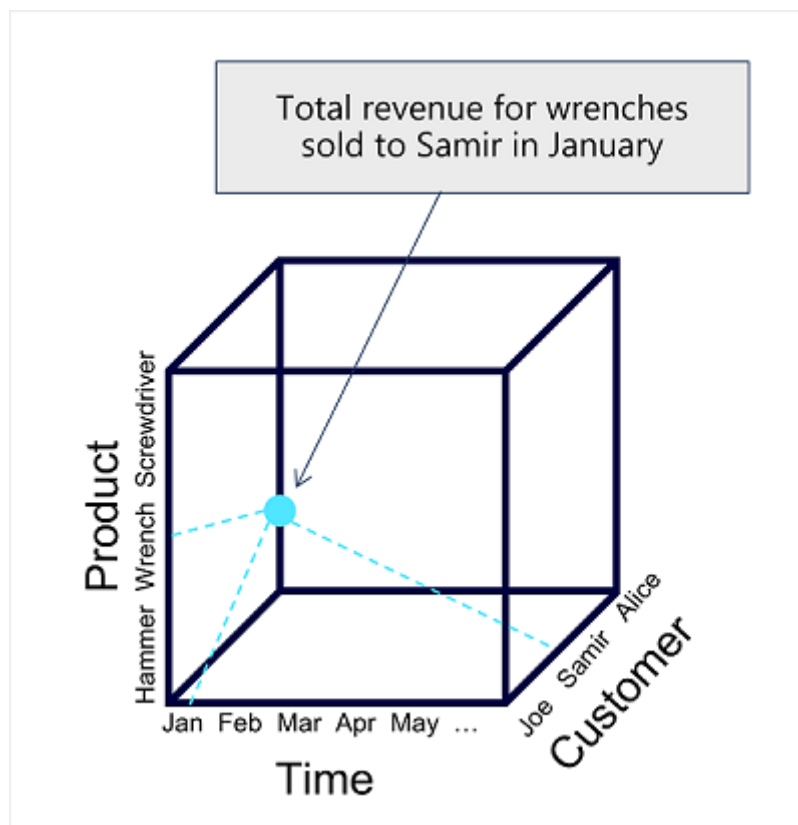
A typical workflow for creating a data visualization solution starts with **Power BI Desktop**, a Microsoft Windows application in which you can import data from a wide range of data sources, combine and organize the data from these sources in an analytics data model, and create reports that contain interactive visualizations of the data.

After you've created data models and reports, you can publish them to the **Power BI service**; a cloud service in which reports can be published and interacted with by business users. You can also do some basic data modeling and report editing directly in the service using a web browser, but the functionality for this is limited compared to the Power BI Desktop tool. You can use the service to schedule refreshes of the data sources on which your reports are based, and to share reports with other users. You can also define dashboards and apps that combine related reports in a single, easy to consume location.

Users can consume reports, dashboards, and apps in the Power BI service through a web browser, or on mobile devices by using the **Power BI phone app**.

Describe core concepts of data modeling

Analytical models enable you to structure data to support analysis. Models are based on related tables of data and define the numeric values that you want to analyze or report (known as *measures*) and the entities by which you want to aggregate them (known as *dimensions*). For example, a model might include a table containing numeric measures for sales (such as revenue or quantity) and dimensions for products, customers, and time. This would enable you to aggregate sale measures across one or more dimensions (for example, to identify total revenue by customer, or total items sold by product per month). Conceptually, the model forms a multidimensional structure, which is commonly referred to as a *cube*, in which any point where the dimensions intersect represents an aggregated measure for those dimensions.



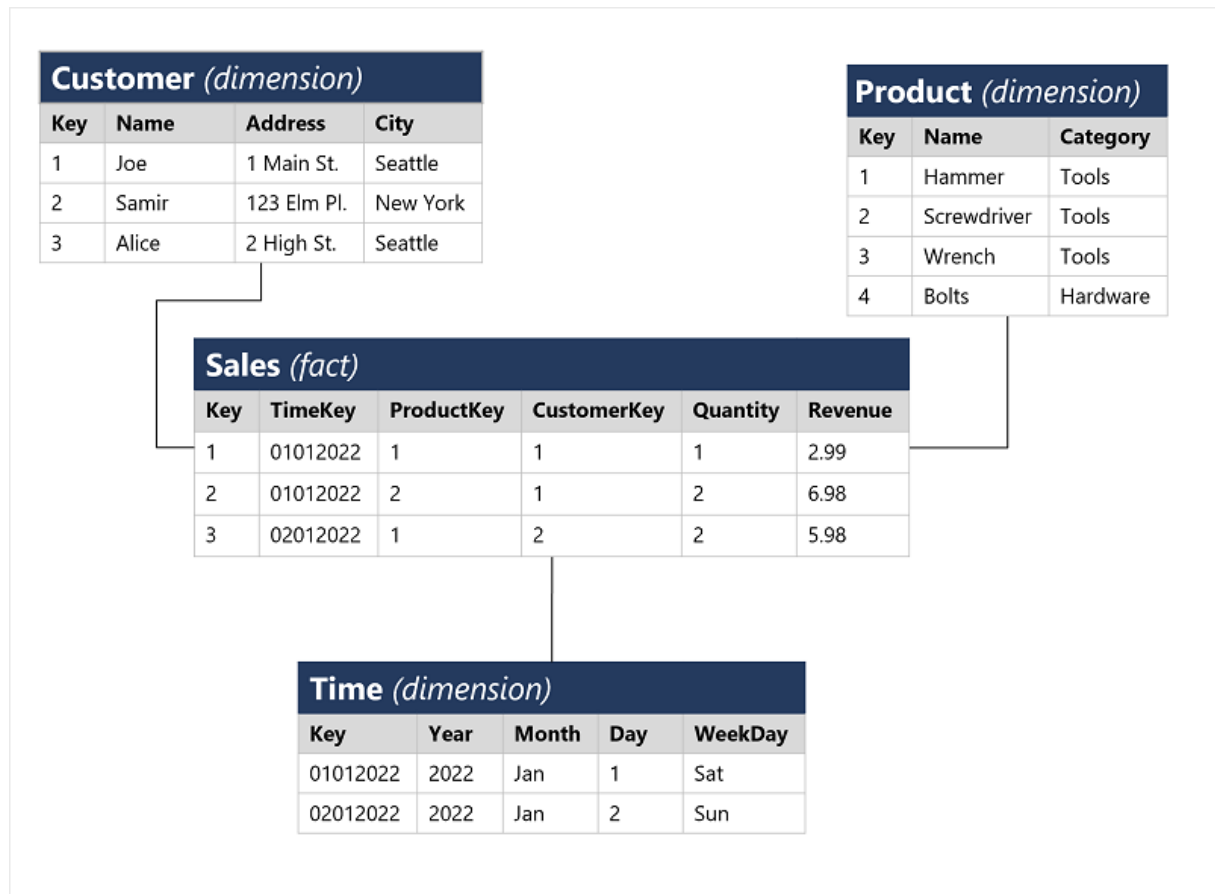
Note

Although we commonly refer to an analytical model as a *cube*, there can be more (or fewer) than three dimensions – it's just not easy for us to visualize more than three!

Tables and schema

Dimension tables represent the entities by which you want to aggregate numeric measures – for example product or customer. Each entity is represented by a row with a unique key value. The remaining columns represent attributes of an entity – for example, products have names and categories, and customers have addresses and cities. It's common in most analytical models to include a *Time* dimension so that you can aggregate numeric measures associated with events over time.

The numeric measures that will be aggregated by the various dimensions in the model are stored in *Fact* tables. Each row in a fact table represents a recorded event that has numeric measures associated with it. For example, the **Sales** table in the schema below represents sales transactions for individual items, and includes numeric values for quantity sold and revenue.

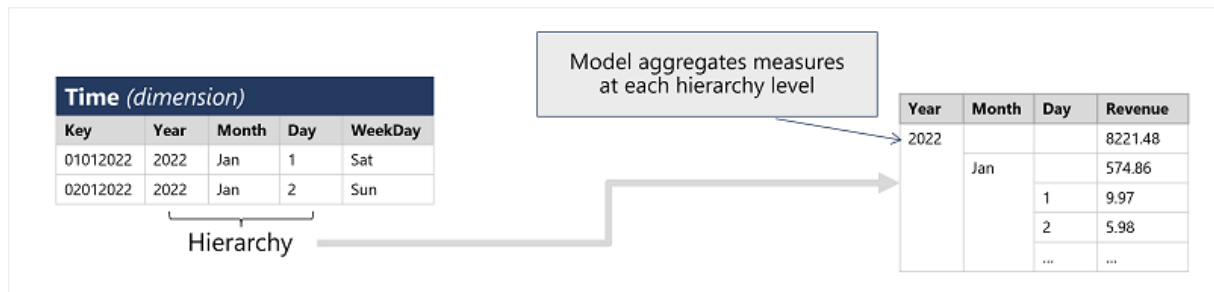


This type of schema, where a fact table is related to one or more dimension tables, is referred to as a star schema (imagine there are five dimensions related to a single fact table – the schema would form a five-pointed star!). You can also define more a complex schema in which dimension tables are related to additional tables containing more details (for example, you could represent attributes of product categories in a separate **Category** table that is related to the **Product** table – in which case the design is referred to as a snowflake schema. The schema of fact and dimension tables is used to create an analytical model, in which measure aggregations across all dimensions are pre-calculated; making performance of analysis and reporting activities much faster than calculating the aggregations each time.

Attribute hierarchies

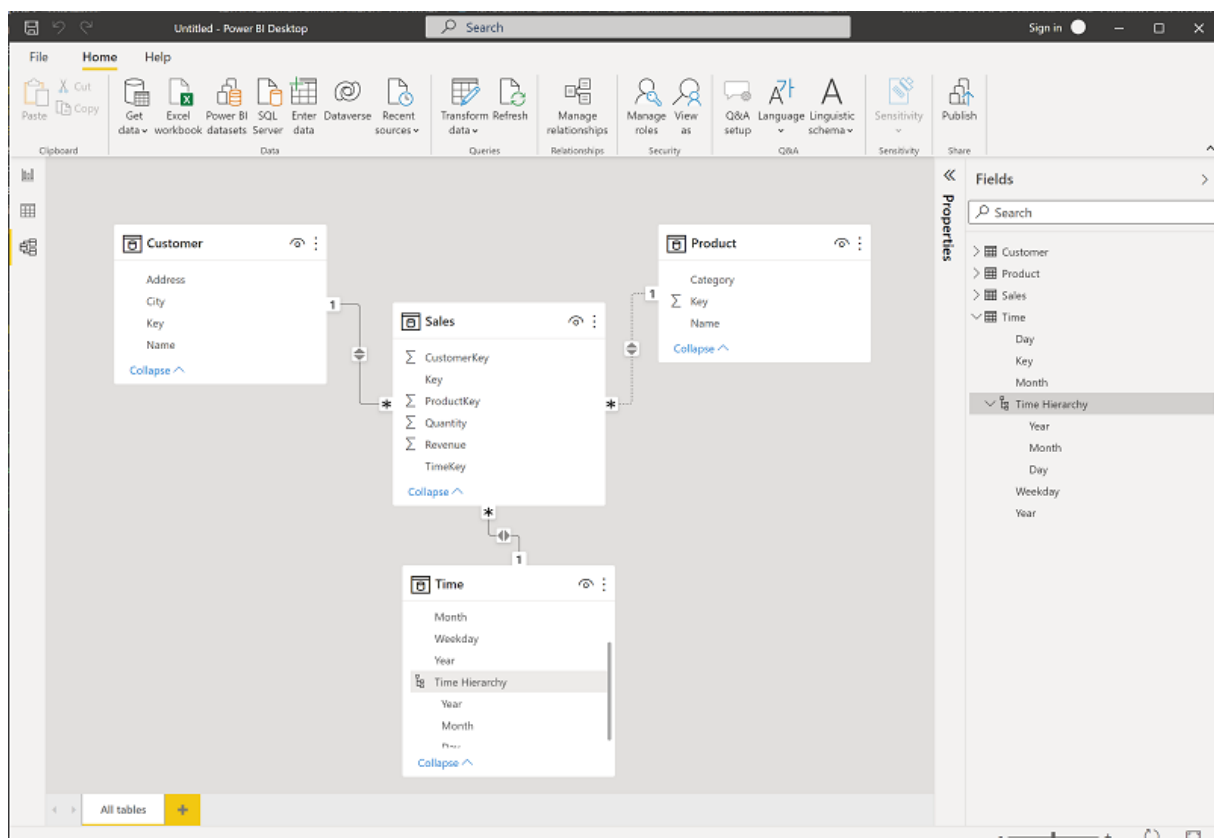
One final thing worth considering about analytical models is the creation of attribute *hierarchies* that enable you to quickly *drill-up* or *drill-down* to find aggregated values at different levels in a hierarchical dimension. For example, consider the attributes in the dimension tables we've discussed so far. In

the **Product** table, you can form a hierarchy in which each category might include multiple named products. Similarly, in the **Customer** table, a hierarchy could be formed to represent multiple named customers in each city. Finally, in the **Time** table, you can form a hierarchy of year, month, and day. The model can be built with pre-aggregated values for each level of a hierarchy, enabling you to quickly change the scope of your analysis – for example, by viewing total sales by year, and then drilling down to see a more detailed breakdown of total sales by month.



Analytical modeling in Microsoft Power BI

You can use Power BI to define an analytical model from tables of data, which can be imported from one or more data source. You can then use the data modeling interface on the **Model** tab of Power BI Desktop to define your analytical model by creating relationships between fact and dimension tables, defining hierarchies, setting data types and display formats for fields in the tables, and managing other properties of your data that help define a rich model for analysis.



Describe considerations for data visualization

After you've created a model, you can use it to generate data visualizations that can be included in a report.

There are many kinds of data visualization, some commonly used and some more specialized. Power BI includes an extensive set of built-in visualizations, which can be extended with custom and third-party visualizations. The rest of this unit discusses some common data visualizations but is by no means a complete list.

Tables and text

Tables and text are often the simplest way to communicate data. Tables are useful when numerous related values must be displayed, and individual text values in cards can be a useful way to show important figures or metrics.

Bar and column charts

Bar and column charts are a good way to visually compare numeric values for discrete categories.

Line charts

Line charts can also be used to compare categorized values and are useful when you need to examine trends, often over time.

Pie charts

Pie charts are often used in business reports to visually compare categorized values as proportions of a total.

Scatter plots

Scatter plots are useful when you want to compare two numeric measures and identify a relationship or correlation between them.

Maps

Maps are a great way to visually compare values for different geographic areas or locations.

Interactive reports in Power BI

In Power BI, the visual elements for related data in a report are automatically linked to one another and provide interactivity. For example, selecting an individual category in one visualization will automatically filter and highlight that category in other related visualizations in the report. In the image above, the city *Seattle* has been selected in the **Sales by City and Category** column chart, and the other visualizations are filtered to reflect values for Seattle only.