# Microsoft Azure Data Fundamentals: Explore core data concepts

**Identify data formats**

Data is a collection of facts such as numbers, descriptions, and observations used to record information. Data structures in which this data is organized often represents *entities* that are important to an organization (such as customers, products, sales orders, and so on). Each entity typically has one or more *attributes*, or characteristics (for example, a customer might have a name, an address, a phone number, and so on).

You can classify data as *structured*, *semi-structured*, or *unstructured*.

1. **Structured data**

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. For example, the following image shows tabular data representations for *Customer* and *Product* entities.

Structured data is often stored in a database in which multiple tables can reference one another by using key values in a *relational* model; which we'll explore in more depth later.

1. **Semi-structured data**

*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). The example below shows a pair of JSON documents that represent customer information. Each customer document includes address and contact information, but the specific fields vary between customers.

1. **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 stores**

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

We'll explore both of these types of data store in subsequent topics.

# Explore file storage

The ability to store data in files is a core element of any computing system. Files can be stored in local file systems on the hard disk of your personal computer, and on removable media such as USB drives; but in most organizations, important data files are stored centrally in some kind of shared file storage system. Increasingly, that central storage location is hosted in the cloud, enabling cost-effective, secure, and reliable storage for large volumes of data.

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.

Some common file formats are discussed below.

## 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.

The following example shows customer data in comma-delimited format:

Copy

FirstName,LastName,Email

Joe,Jones,joe@litware.com

Samir,Nadoy,samir@northwind.com

## 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.

The following example shows a JSON document containing a collection of customers. Each customer has three attributes (firstName, lastName, and contact), and the contact attribute contains a collection of objects that represent one or more contact methods (email or phone). Note that objects are enclosed in braces (**{..}**) and collections are enclosed in square brackets (**[..]**). Attributes are represented by name **:** value pairs and separated by commas (**,**).

JSONCopy

{

"customers":

[

{

"firstName": "Joe",

"lastName": "Jones",

"contact":

[

{

"type": "home",

"number": "555 123-1234"

},

{

"type": "email",

"address": "joe@litware.com"

}

]

},

{

"firstName": "Samir",

"lastName": "Nadoy",

"contact":

[

{

"type": "email",

"address": "samir@northwind.com"

}

]

}

]

}

## 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 tags enclosed in angle-brackets (**<../>**) to define elements and attributes, as shown in this example:

XMLCopy

<Customers>

<Customer name="Joe" lastName="Jones">

<ContactDetails>

<Contact type="home" number="555 123-1234"/>

<Contact type="email" address="joe@litware.com"/>

</ContactDetails>

</Customer>

<Customer name="Samir" lastName="Nadoy">

<ContactDetails>

<Contact type="email" address="samir@northwind.com"/>

</ContactDetails>

</Customer>

</Customers>

## 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 through 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.

# Explore databases

A database is used to define a central system in which data can be stored and queried. In a simplistic sense, the file system on which files are stored is a kind of database; but when we use the term in a professional data context, we usually mean a dedicated system for managing data records rather than files.

## Relational databases

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

## Non-relational databases

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

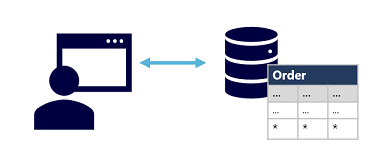
There are four common types of Non-relational database commonly in use.

* **Key-value databases** in which each record consists of a unique key and an associated value, which can be in any format.
* **Document databases**, which are a specific form of key-value database in which the value is a JSON document (which the system is optimized to parse and query)
* **Column family databases**, which store tabular data comprising rows and columns, but you can divide the columns into groups known as column-families. Each column family holds a set of columns that are logically related together.
* **Graph databases**, which store entities as nodes with links to define relationships between them.

# Explore transactional data processing

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

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



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

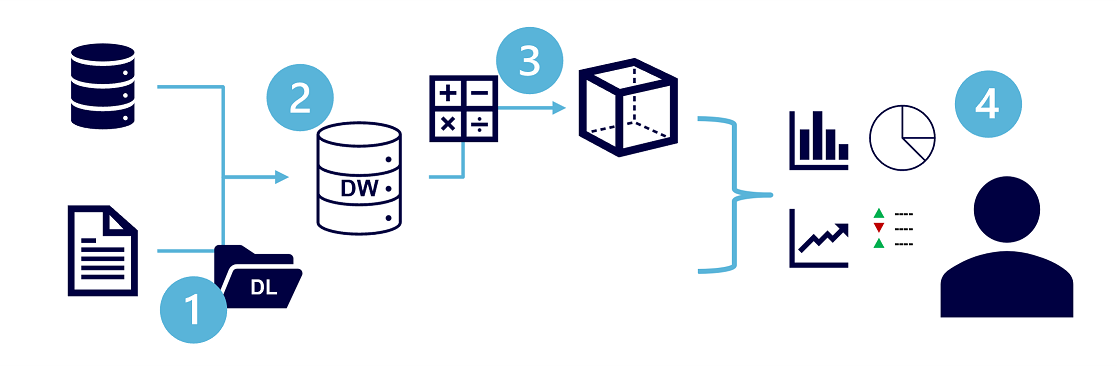
* **Atomicity** – each transaction is treated as a single unit, which succeeds completely or fails completely. For example, a transaction that involved debiting funds from one account and crediting the same amount to another account must complete both actions. If either action can't be completed, then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

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

# Explore analytical data processing

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

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



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

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

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

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

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

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

# Explore job roles in the world of data

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

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

* **Database administrators** manage databases, assigning permissions to users, storing backup copies of data and restore data in the event of a failure.
* **Data engineers** manage infrastructure and processes for data integration across the organization, applying data cleaning routines, identifying data governance rules, and implementing pipelines to transfer and transform data between systems.
* **Data analysts** explore and analyze data to create visualizations and charts that enable organizations to make informed decisions.

**Note**

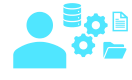
The job roles define differentiated tasks and responsibilities. In some organizations, the same person might perform multiple roles; so in their role as database administrator they might provision a transactional database, and then in their role as a data engineer they might create a pipeline to transfer data from the database to a data warehouse for analysis.

## 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.

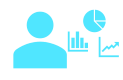
The database administrator is also responsible for managing the security of the data in the database, granting privileges over the data, granting or denying access to users as appropriate.

## 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.

**Note**

The roles described here represent the key data-related roles found in most medium to large organizations. There are additional data-related roles not mentioned here, such as data scientist and data architect; and there are other technical professionals that work with data, including application developers and software engineers.

# Identify data services

Microsoft Azure is a cloud platform that powers the applications and IT infrastructure for some of the world's largest organizations. It includes many services to support cloud solutions, including transactional and analytical data workloads.

Some of the most commonly used cloud services for data are described below.

**Note**

This topic covers only some of the most commonly used data services for modern transactional and analytical solutions. Additional services are also available.

## 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.

## Microsoft Purview

Azure Purview logo Microsoft Purview provides a solution for enterprise-wide data governance and discoverability. You can use Microsoft 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 Microsoft Purview to enforce data governance across the enterprise and ensure the integrity of data used to support analytical workloads.

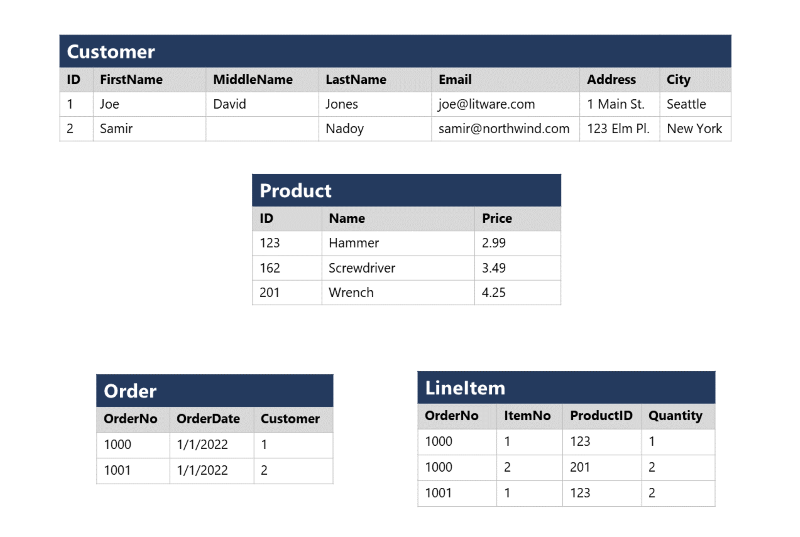
## Microsoft Power BI

 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.

# Microsoft Azure Data Fundamentals: Explore 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.



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

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

# Understand normalization

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

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

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

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

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

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

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

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

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

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

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

# Explore SQL

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

**Note**

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

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

Some popular dialects of SQL include:

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

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

**Note**

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

## SQL statement types

SQL statements are grouped into three main logical groups:

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

### DDL statements

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

The most common DDL statements are:

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

**Warning**

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

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

**Note**

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

SQLCopy

CREATE TABLE Product

(

ID INT PRIMARY KEY,

Name VARCHAR(20) NOT NULL,

Price DECIMAL NULL

);

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

### DCL statements

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

The three main DCL statements are:

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

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

SQLCopy

GRANT SELECT, INSERT, UPDATE

ON Product

TO user1;

### DML statements

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

The four main DML statements are:

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

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

**Warning**

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

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

SQLCopy

SELECT \*

FROM Customer

WHERE City = 'Seattle';

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

SQLCopy

SELECT FirstName, LastName, Address, City

FROM Customer

WHERE City = 'Seattle';

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

SQLCopy

SELECT FirstName, LastName, Address, City

FROM Customer

WHERE City = 'Seattle'

ORDER BY LastName;

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

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

SQLCopy

SELECT o.OrderNo, o.OrderDate, c.Address, c.City

FROM Order AS o

JOIN Customer AS c

ON o.Customer = c.ID

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

SQLCopy

UPDATE Customer

SET Address = '123 High St.'

WHERE ID = 1;

**Warning**

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

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

SQLCopy

DELETE FROM Product

WHERE ID = 162;

**Warning**

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

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

SQLCopy

INSERT INTO Product(ID, Name, Price)

VALUES (99, 'Drill', 4.99);

**Note**

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

# Describe database objects

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

## What is a view?

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

SQLCopy

CREATE VIEW Deliveries

AS

SELECT o.OrderNo, o.OrderDate,

c.FirstName, c.LastName, c.Address, c.City

FROM Order AS o JOIN Customer AS c

ON o.CustomerID = c.ID;

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

SQLCopy

SELECT OrderNo, OrderDate, LastName, Address

FROM Deliveries

WHERE City = 'Seattle';

## What is a stored procedure?

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

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

SQLCopy

CREATE PROCEDURE RenameProduct

@ProductID INT,

@NewName VARCHAR(20)

AS

UPDATE Product

SET Name = @NewName

WHERE ID = @ProductID;

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

SQLCopy

EXEC RenameProduct 201, 'Spanner';

## What is an index?

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

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

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

SQLCopy

CREATE INDEX idx\_ProductName

ON Product(Name);

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

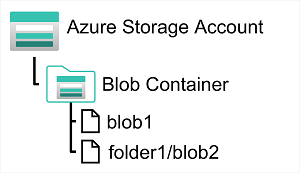
For a table containing few rows, using the index is probably not any more efficient than simply reading the entire table and finding the rows requested by the query (in which case the query optimizer will ignore the index). However, when a table has many rows, indexes can dramatically improve the performance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

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

# Microsoft Azure Data Fundamentals: Explore 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.

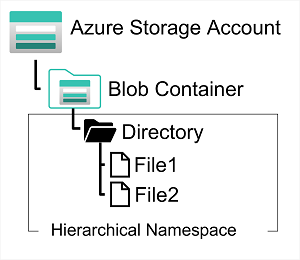
Blob storage provides three access tiers, which help to balance access latency and storage cost:

* 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 Gen**2** 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.



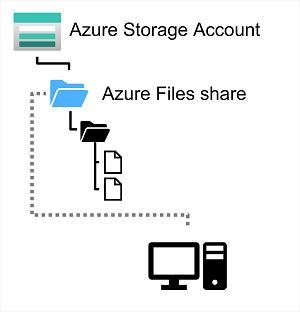
Systems like Hadoop in Azure HDInsight, Azure Databricks, and Azure Synapse Analytics can mount a distributed file system hosted in Azure Data Lake Store Gen2 and use it to process huge volumes of data.

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.

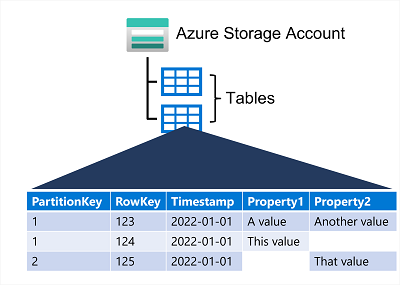
Azure File Storage offers two performance tiers. The Standard tier uses hard disk-based hardware in a datacenter, and the Premium tier uses solid-state disks. The Premium tier offers greater throughput, but is charged at a higher rate.

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.

# 4. Microsoft Azure Data Fundamentals: Explore data analytics in Azure

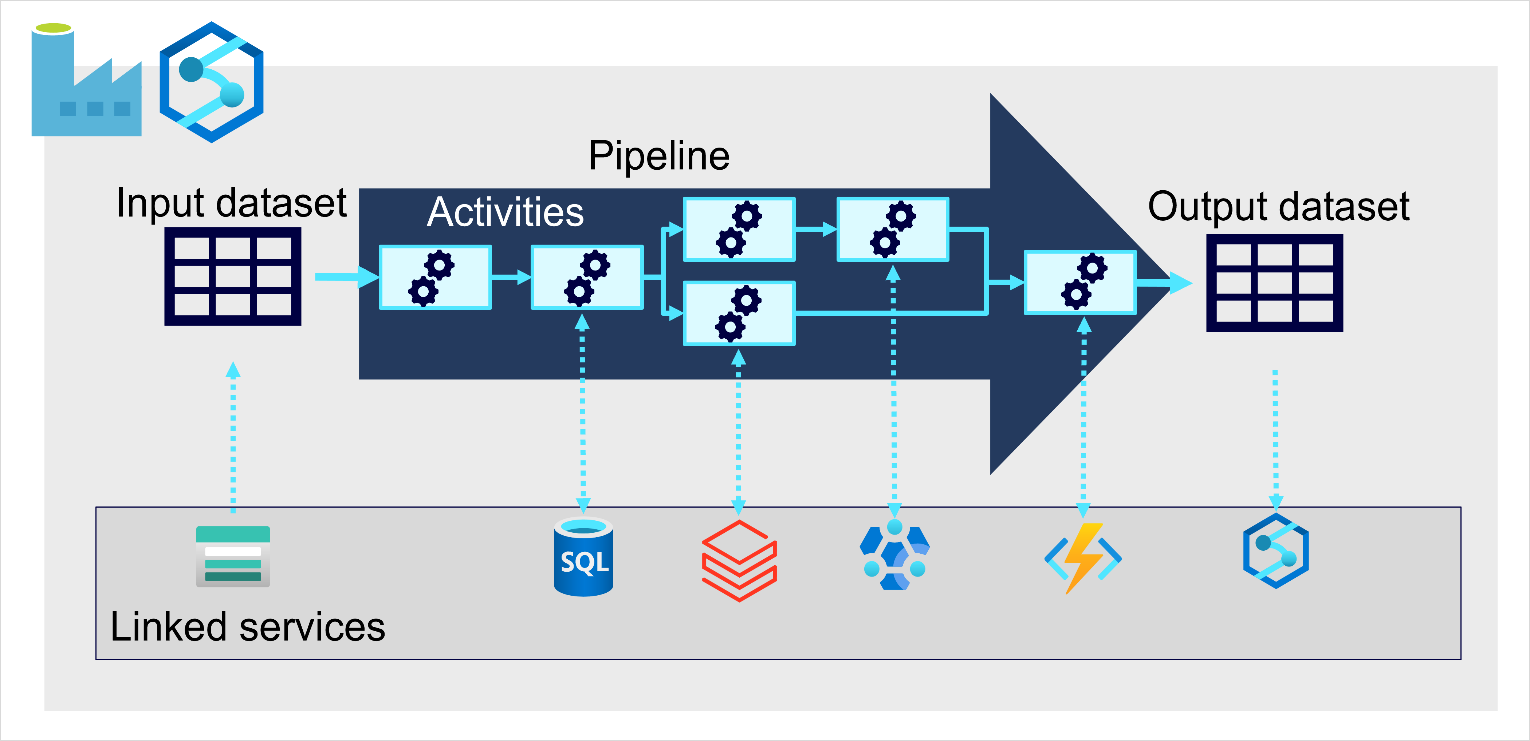
# Describe modern data warehousing

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](https://azure.microsoft.com/services/data-factory), or you can use the same pipeline engine in [Azure Synapse Analytics](https://azure.microsoft.com/services/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.

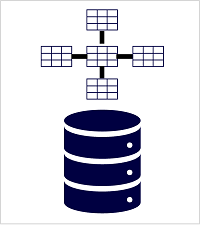
# Explore analytical data stores

Completed100 XP

* 8 minutes

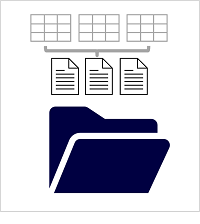
There are two common types of analytical data store.

## 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**](https://azure.microsoft.com/services/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**](https://azure.microsoft.com/services/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**](https://azure.microsoft.com/services/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.