Data Fundamentals (DP-900)

Self-study Notes by Neil bagchi

Describe core data concepts (25–30%)

Describe ways to represent data
- describe features of structured data
- describe features of semi-structured
- describe features of unstructured data

Identify options for data storage
- describe common formats for data files
- describe types of databases

Describe common data workloads
- describe features of transactional workloads
- describe features of analytical workloads

Identify roles and responsibilities for data workloads
- describe responsibilities for database administrators
- describe responsibilities for data engineers
- describe responsibilities for data analysts

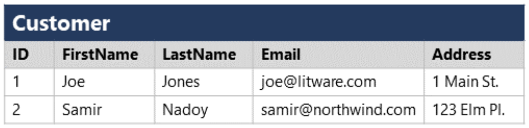
Core Data Concepts

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 represent *entities* that are important to an organization (such as customers, products, sales orders, and so on). Each entity type 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*.

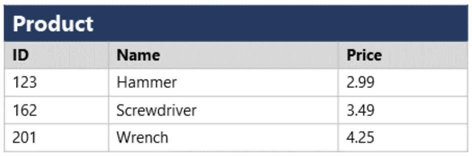
1. Structured Data

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.



Customer

ID	FirstName	LastName	Email	Address
1	Joe	Jones	joe@litware.com	1 Main St.
2	Samir	Nadoy	samir@northwind.com	123 Elm Pl.



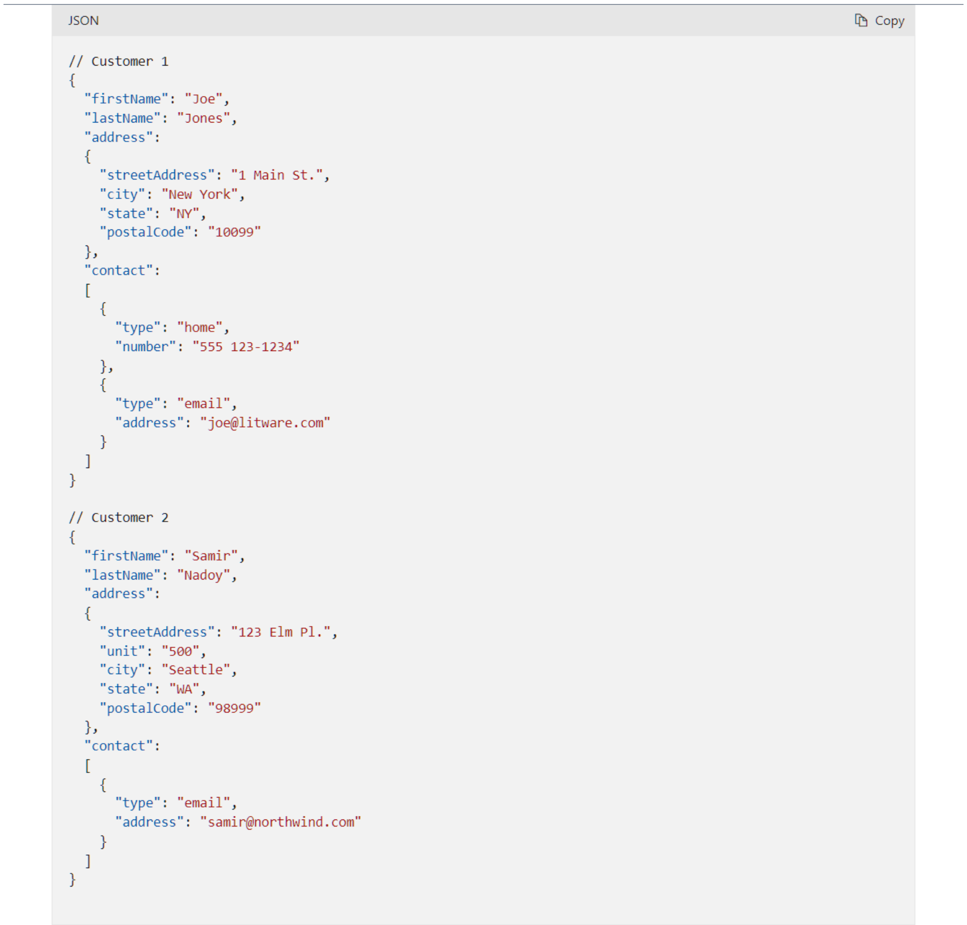
Product

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

2. Semi-Structured Data

Semi-structured data is information that has some structure but 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 shows a pair of JSON documents that represent customer information. JSON is just one of many ways in which semi-structured data can be represented.



```json
// Customer 1
{
  "firstName": "Joe",
  "lastName": "Jones",
  "address":
  {
    "streetAddress": "1 Main St.",
    "city": "New York",
    "state": "NY",
    "postalCode": "10099"
  },
  "contact":
  [
    {
      "type": "home",
      "number": "555 123-1234"
    },
    {
      "type": "email",
      "address": "joe@litware.com"
    }
  ]
}

// Customer 2
{
  "firstName": "Samir",
  "lastName": "Nadoy",
  "address":
  {
    "streetAddress": "123 Elm Pl.",
    "unit": "500",
    "city": "Seattle",
    "state": "WA",
    "postalCode": "98999"
  },
  "contact":
  [
    {
      "type": "email",
      "address": "samir@northwind.com"
    }
  ]
}
```

3. Unstructured Data

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

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

A. File stores

Delimited text files

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 return/newline.

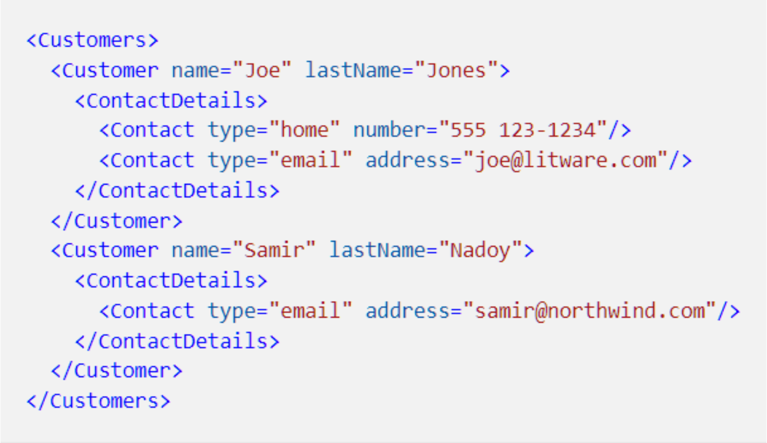
Other common formats include tab-separated values (TSV) and space-delimited (in which tabs or spaces are used to separate fields), and fixed-width data in which each field is allocated a fixed number of characters.

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 primarily 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.*



```
<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)

Technically, 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 a raw binary that must be interpreted by applications and rendered such as images, video, audio, and application-specific documents.

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.

B. Databases

A database is used to define a central system in which data can be stored and queried.

Relational Database

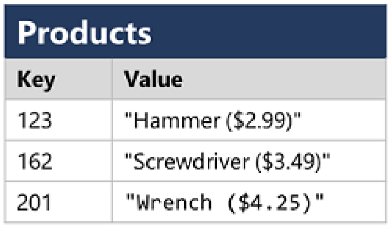
Relational databases are commonly used to store and query structured data. The data is stored in tables that represent entities. Each instance of an entity is assigned a *primary key* that uniquely identifies it, and these keys are used to reference the entity instance in other tables. The use of keys to reference data entities enables a relational database to be *normalized*; which in part means the elimination of duplicate data values. The tables are managed and queried using Structured Query Language (SQL)

Non-Relational Database

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 databases, even though some support a variant of the SQL language.

There are four common types of Non-relational databases commonly used.

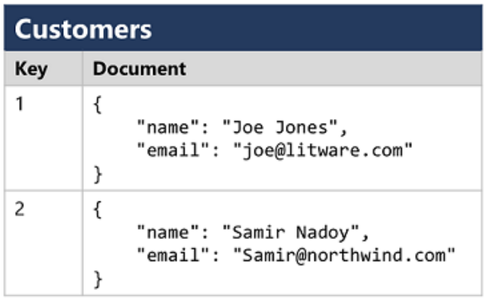
- **Key-value databases** in which each record consists of a unique key and an associated value, which can be in any format.



Products

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

- **Document databases** 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)



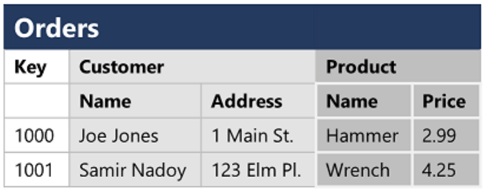
Customers

Key	Document

1	{
	"name": "Joe Jones",
	"email": "joe@litware.com"
	}

2	{
	"name": "Samir Nadoy",
	"email": "Samir@northwind.com"
	}

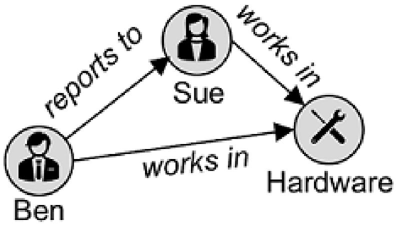
- **Column family databases**, store tabular data comprising rows and columns, but you can divide the columns into groups known as column families. Each column family holds a set of columns that are logically related together.



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** store entities as nodes with links to define relationships between them.



Transactional Data Processing

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.

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 reading 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 been 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.

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.

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 reading 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 reading 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

Job Roles

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

- **Database administrators** manage databases, assign permissions to users, store 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.

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*.

Azure Data Services

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

Service Group	Service Name	Description
Azure SQL	**Azure SQL Database**	a fully managed platform-as-a-service (PaaS) database hosted in Azure

Service Group	Service Name	Description
	Azure SQL Managed Instance	a hosted instance of SQL Server (PaaS) with automated maintenance, which allows a 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 (IaaS), allowing maximum configurability with full management responsibility.
	Azure SQL Edge	A SQL engine that is optimized for Internet-of-things (IoT) scenarios that need to work with streaming time-series data.
Azure Service for open-source relational databases	**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 MySQL has two deployment options: Single Server and Flexible Server. Flexible Server provides more granular control and flexibility over database management functions and configuration settings
	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
	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. Azure Database for PostgreSQL has three deployment options: Single Server, Flexible Server, and Hyperscale. Hyperscale (Citus) is a deployment option that scales queries across multiple server nodes to support large database loads.
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.

Service Group	Service Name	Description
Azure Storage	**Blob containers**	scalable, cost-effective storage for binary files
(Non-Relational)	**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
Azure Data Factory		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	**Pipelines**	based on the same technology as Azure Data Factory.
comprehensive, unified data analytics solution that provides a single service interface for multiple analytical capabilities	**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)

Service Group	Service Name	Description
Azure Databricks		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.
Azure HDInsight	**Apache Spark**	a distributed data processing system that supports multiple programming languages and APIs, including Java, Scala, Python, and SQL
Azure service that provides Azure-hosted clusters for popular Apache open-source big data processing technologies	**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*
Azure Stream Analytics		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.
Azure Data Explorer		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.
Microsoft Purview		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

Service Group	Service Name	Description
Microsoft Power BI		platform for analytical data modeling and reporting that data analysts can use to create and share interactive data visualizations

Identify considerations for relational data on Azure (20–25%)

Describe relational concepts

- identify features of relational data
- describe normalization and why it is used
- identify common structured query language (SQL) statements
- identify common database objects

Describe relational Azure data services

- describe the Azure SQL family of products including Azure SQL Database, Azure SQL
- Managed Instance, and SQL Server on Azure Virtual Machines
- identify Azure database services for open-source database system

Relational Data in Azure

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. A table contains rows, and each row represents a single instance of an entity. 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.

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.

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 updating data in a database or retrieving data from a database. Some standard relational database management systems that use SQL include Microsoft SQL Server, MySQL, PostgreSQL, MariaDB, and Oracle.

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.

SQL statements are grouped into three main logical groups:

- Data Definition Language (DDL) *[Create, Alter, Drop, Rename]*
- Data Control Language (DCL) *[Grant, Deny, Revoke]*
- Data Manipulation Language (DML) *[Select, Insert, Update, Delete]*

View

A view is a virtual table based on the results of a **SELECT** query. You can think

```
CREATE VIEW Deliveries
AS
```

of a view as a window on specified rows in one or more underlying tables.

```
SELECT o.OrderNo, o.OrderDate,
       c.FirstName, c.LastName, c.Addres
s, c.City
FROM Order AS o JOIN Customer AS c
ON o.CustomerID = c.ID;
```

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.

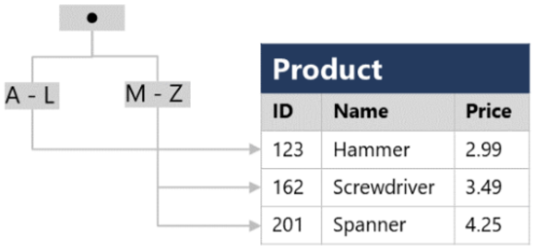
```
CREATE PROCEDURE RenameProduct
  @ProductID INT,
  @NewName VARCHAR(20)
AS
UPDATE Product
SET Name = @NewName
WHERE ID = @ProductID;

EXEC RenameProduct 201, 'Spanner';
```

Index

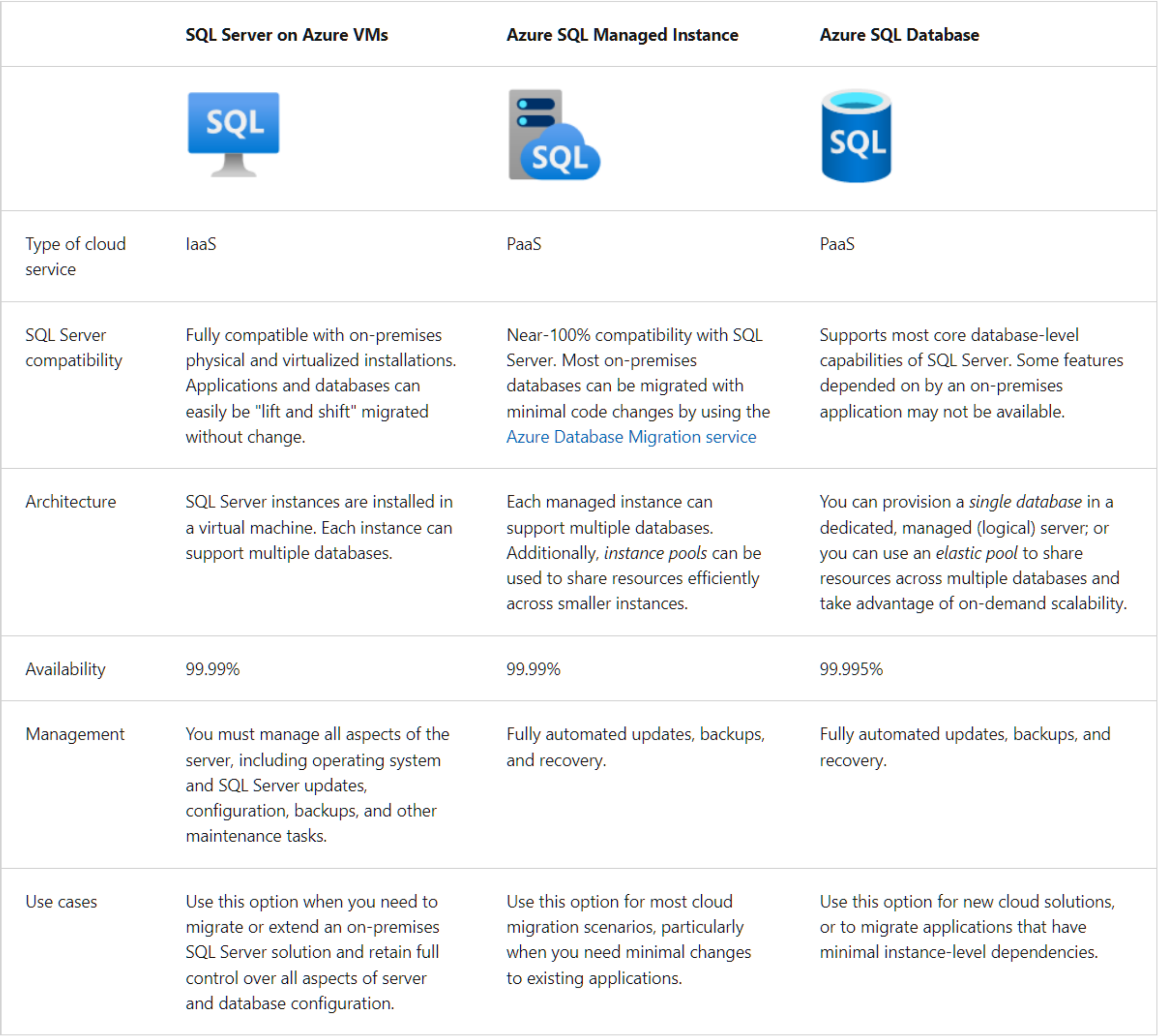
An index helps you search for data in a table. When you create an index in a database, you specify a column from the table, and the index contains a copy of this data in sorted order, with pointers to the corresponding rows in the table. When the user runs a query that specifies this column in the **WHERE** 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.

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

For a table containing few rows, using the index is probably not any more efficient than simply reading the entire table



ID	Name	Price
123	Hammer	2.99
162	Screwdriver	3.49
201	Spanner	4.25



	SQL Server on Azure VMs	Azure SQL Managed Instance	Azure SQL Database
	SQL	SQL	SQL
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, *instance pools* can be used to share resources efficiently across smaller instances.	You can provision a *single database* in a dedicated, managed (logical) server; or you can use an *elastic pool* 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.

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). Suppose 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). In that case, 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.

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

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 been completed.

Describe considerations for working with non-relational data on Azure (15–20%)

Describe capabilities of Azure storage

• describe Azure Blob storage
• describe Azure File storage
• describe Azure Table storage

Describe capabilities and features of Azure Cosmos DB

• identify use cases for Azure Cosmos DB
• describe Azure Cosmos DB APIs

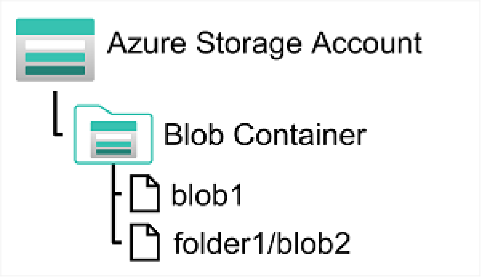
Non-Relational Data in Azure

Blob Storage

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.

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 a 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**.

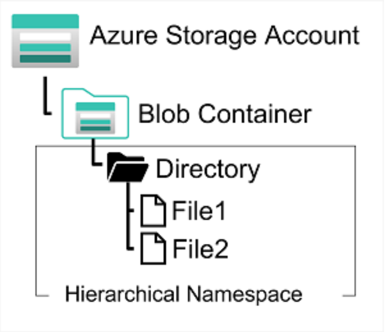
- **Page blobs**. A page blob is organized as a collection of **fixed-size 512-byte page**s. A page blob is **optimized to support random read and write operations**; you can fetch and store data for a single page if necessary. A page blob can hold **up to 8 TB** of data. Azure uses page blobs to implement 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**.

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.

Azure DataLake Storage Gen2

Azure Data Lake Storage 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.

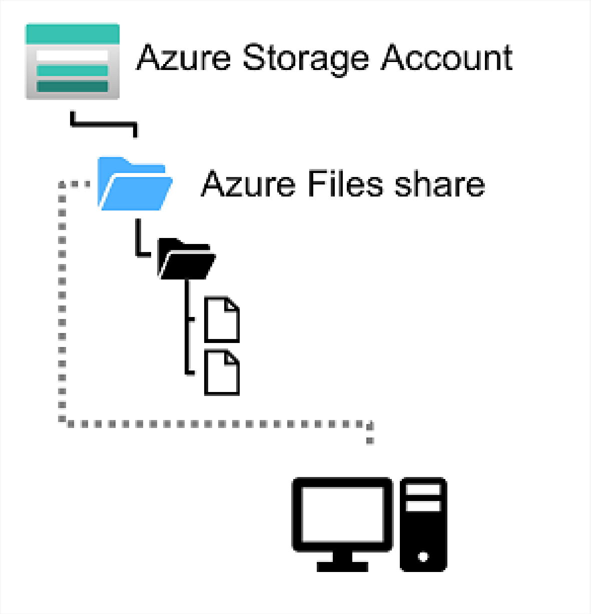


To create an Azure Data Lake Store Gen2 files system, you must enable the **Hierarchical Namespace** option of an Azure Storage account. You can do this when initially creating the storage account, or you can upgrade an existing Azure

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.***

Azure Files

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.



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**. Azure File Storage offers two performance tiers. *The Standard tier* uses hard disk-based hardware in a data center, and the *Premium tier* uses solid-state disks

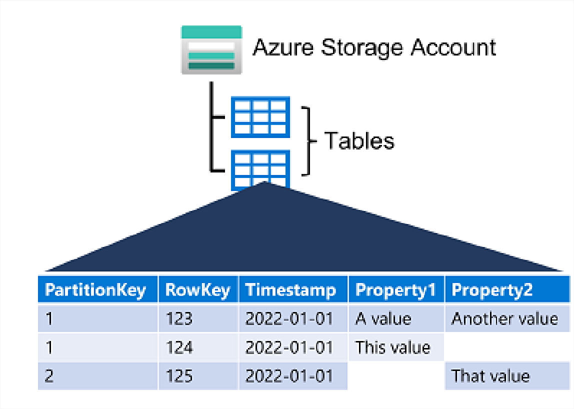
Azure Files supports two common network file sharing protocols:

- ***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.

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.



PartitionKey	RowKey	Timestamp	Property1	Property2
1	123	2022-01-01	A value	Another value
1	124	2022-01-01	This value	
2	125	2022-01-01		That value

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

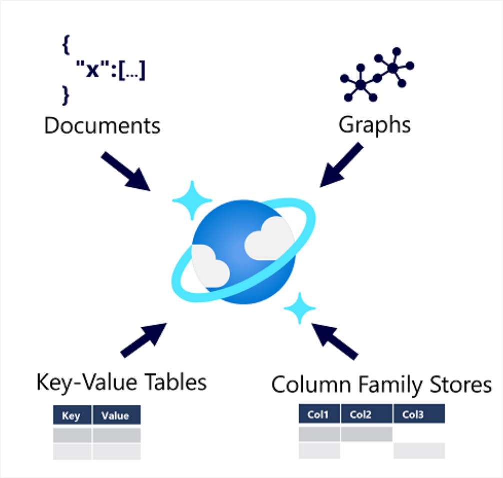
- Partitions are independent of each other and can grow or shrink as rows are added to, or removed from, a partition. A table can contain any number of partitions.
- When you search for data, you can include the partition key in the search criteria. This helps to narrow down the volume of data to be examined and improves

performance by reducing the amount of I/O (input and output operations, or *reads* and *writes*) needed to locate the data.

Cosmos DB

Azure Cosmos DB is a highly scalable cloud database service for NoSQL data.

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. Cosmos DB uses indexes and partitioning to provide fast read and write performance and can scale to massive volumes of data.



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.

When you provision a new Cosmos DB instance, you select the API that you want to use. The choice of API depends on many factors including, the type of data to be stored, the need to support existing applications, and the API skills of the developers who will work with the data store.

Core (SQL) API

The native API in Cosmos DB manages data in JSON document format, and despite being a NoSQL data storage solution, uses SQL syntax to work with the data.

```
Input (SQL)
SELECT *
FROM customers c
WHERE c.id = "joe@litware.com"
```

```
Output (JSON)
{
   "id": "joe@litware.com",
   "name": "Joe Jones",
   "address": {
        "street": "1 Main St.",
        "city": "Seattle"
    }
}
```

MongoDB API

MongoDB is a popular open-source database in which data is stored in Binary JSON (BSON) format. MongoDB Query Language (MQL) uses a compact, object-oriented syntax in which developers use ***objects* to call *methods***. For example, the following query uses the **find** method to query the **products** collection in the **db** object:

```
Input (Javascript)
db.products.find({id: 123})

Output (BSON)
{
   "id": 123,
   "name": "Hammer",
   "price": 2.99}
}
```

Table API

The Table API is used to work with data in key-value tables, similar to Azure Table Storage. The Azure Cosmos DB Table API offers greater scalability and performance than Azure Table Storage.

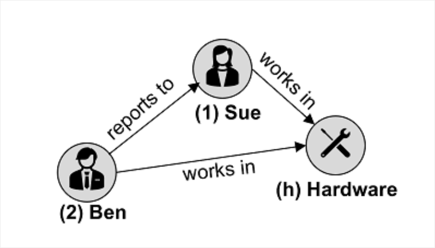
```
https://endpoint/Customers(PartitionKey='1',RowKey='124')
```

Cassandra API

The Cassandra API is compatible with Apache Cassandra, which is a popular open-source database that uses a column-family storage structure. Column families are tables, similar to those in a relational database, with the exception that it's not mandatory for every row to have the same columns. Cassandra supports a syntax based on SQL.

Gremlin API

The Gremlin API is used with data in a graph structure; in which entities are defined as *vertices* that form nodes in a connected graph. Nodes are connected by *edges* that represent relationships, like this:



The example in the image shows two kinds of vertex (employee and department) and edges that connect them (employee "Ben" reports to employee "Sue", and both employees work in the "Hardware" department).

Gremlin syntax includes functions to operate on vertices and edges, enabling you to insert, update, delete, and query data in the graph. For example, you could use the following code to add a new employee named *Alice* who reports to the employee with ID **1** (*Sue*)

```
g.addV('employee').property('id', '3').property('firstName', 'Alice')
g.V('3').addE('reports to').to(g.V('1'))
```

Describe an analytics workload on Azure (25–30%)

Describe common elements of a modern data warehouse

- describe considerations for data ingestion and processing
- describe options for analytical data stores
- describe Azure services for data warehousing, including Azure Synapse Analytics, Azure Databricks, Azure HDInsight, and Azure Data Factory

Describe consideration for real-time data analytics

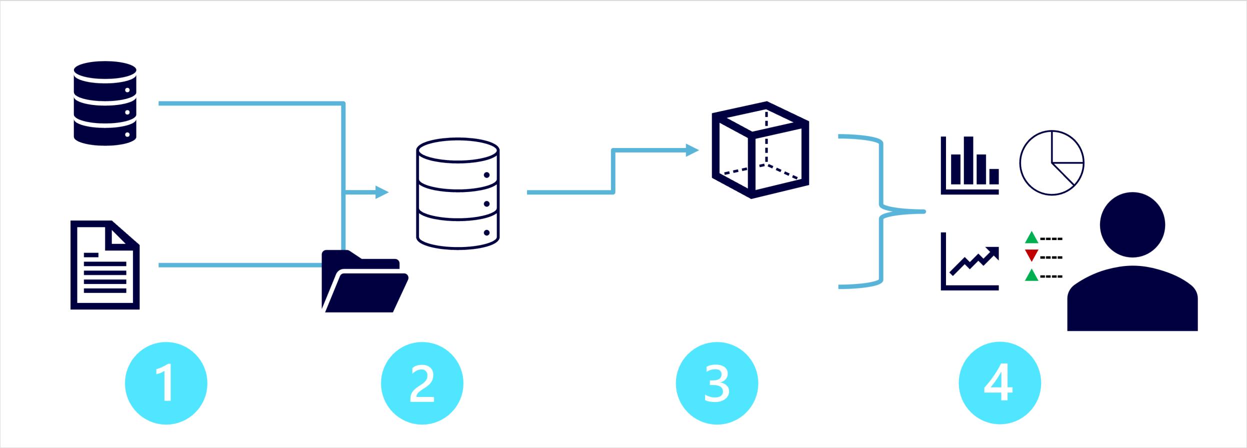
- describe the difference between batch and streaming data
- describe technologies for real-time analytics including Azure Stream Analytics, Azure Synapse Data Explorer, and Spark structured streaming

Describe data visualization in Microsoft Power BI

- identify capabilities of Power BI
- describe features of data models in Power BI
- identify appropriate visualizations for data

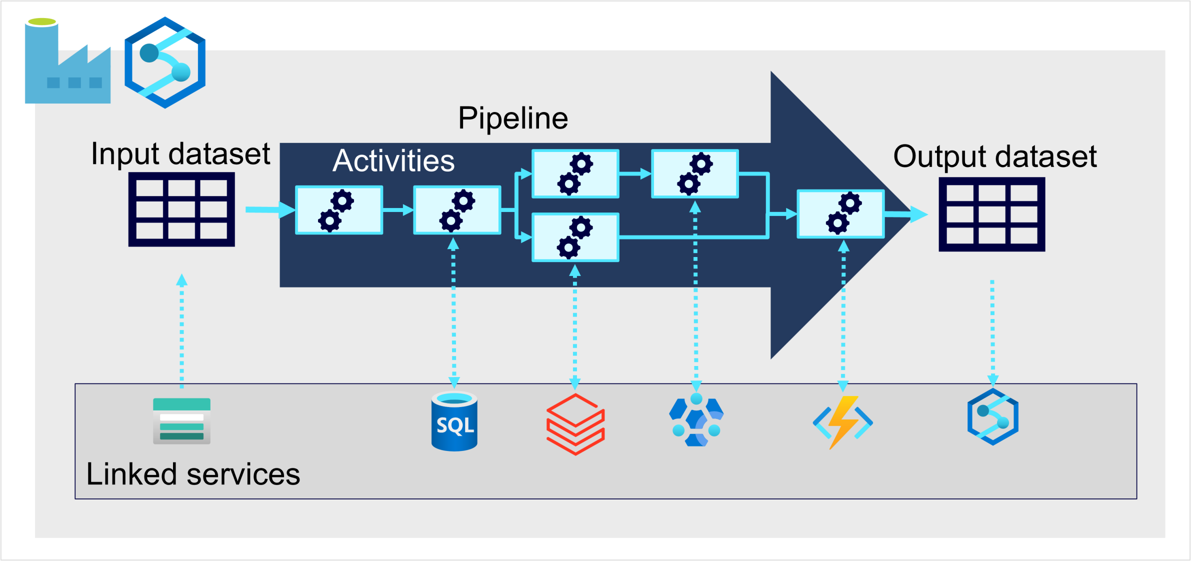
Data Analytics In Azure

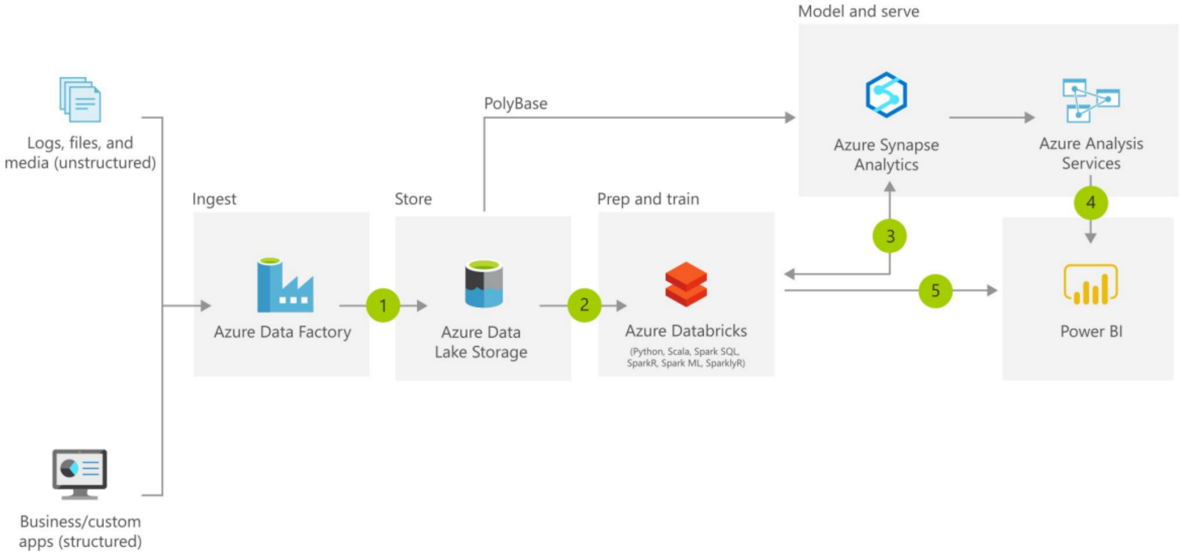
Data Warehousing



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. 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.

   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.





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*).

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 report. 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.

Data Warehouse

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. 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.

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.

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 exposes 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 data lake (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 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.

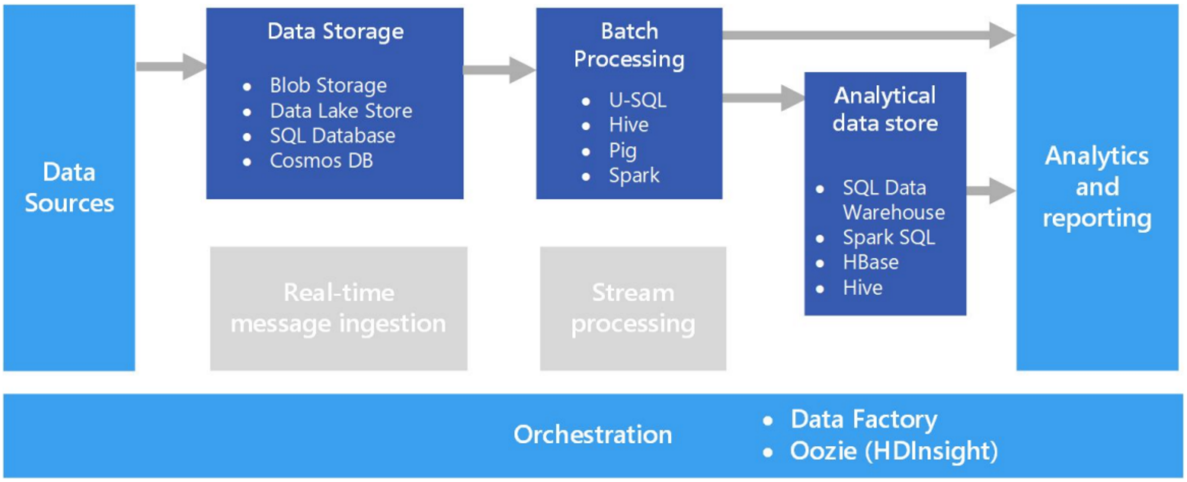
Azure Databricks is an Azure implementation of the popular Databricks platform. Databricks is a comprehensive data analytics solution built on Apache Spark and offers native SQL capabilities as well as workload-optimized Spark clusters for data analytics and data science. Databricks provides an interactive user interface through which the system can be managed and data can be explored in interactive notebooks.

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.

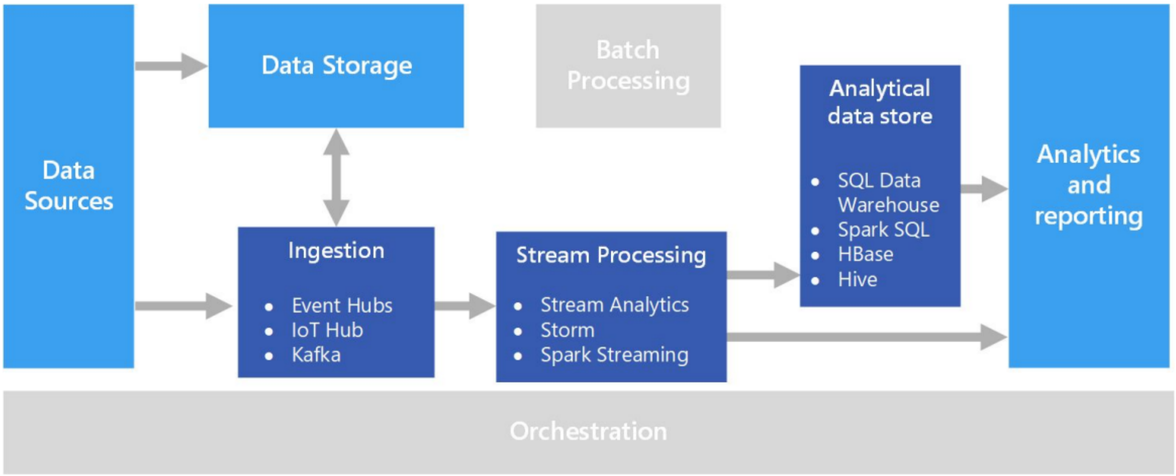
Real-time Analytics

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.

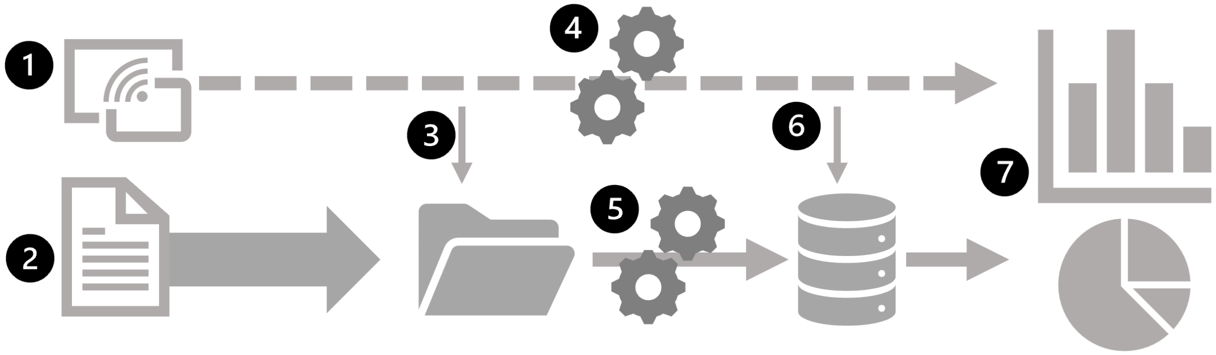


Apart from the way in which batch processing and streaming processing handle data, there are other differences:

- ***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.

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

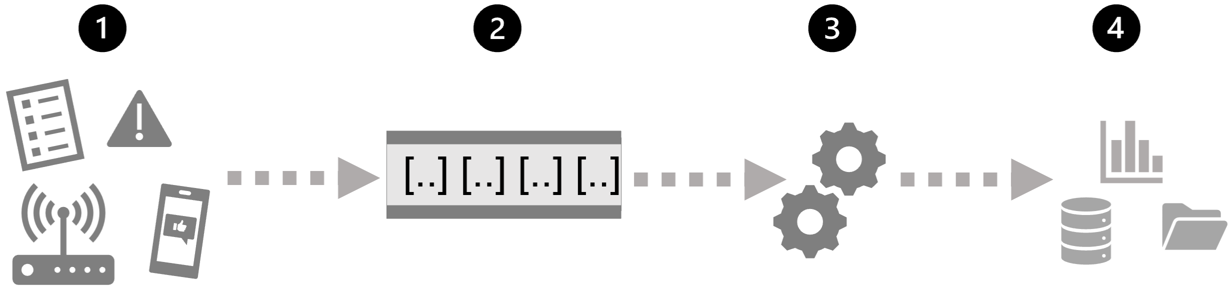


1. Data events from a streaming data source are captured in real-time.
2. Data from other sources are ingested into a data store (often a *data lake*) for batch processing.
3. If real-time analytics is not required, the captured streaming data is written to the data store for subsequent batch processing.
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 real-time and historical data.

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 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:

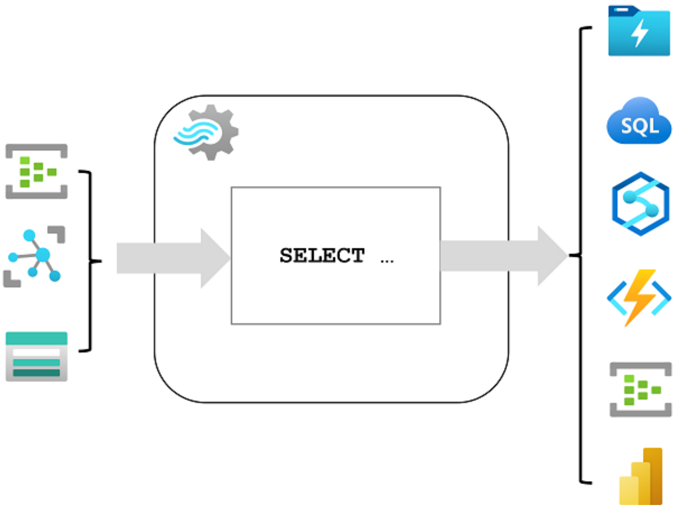
- **Azure Event Hubs**: Used to queue the processed data for further downstream processing.
- **Azure Data Lake Store Gen 2** or **Azure blob storage**: Used to persist the processed results as a file.
- **Azure SQL Database** or **Azure Synapse Analytics**, or **Azure Databricks**: Used to persist the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

- **Microsoft Power BI**: Used to generate real-time data visualizations in reports and dashboards.

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.

Apache Spark on 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.

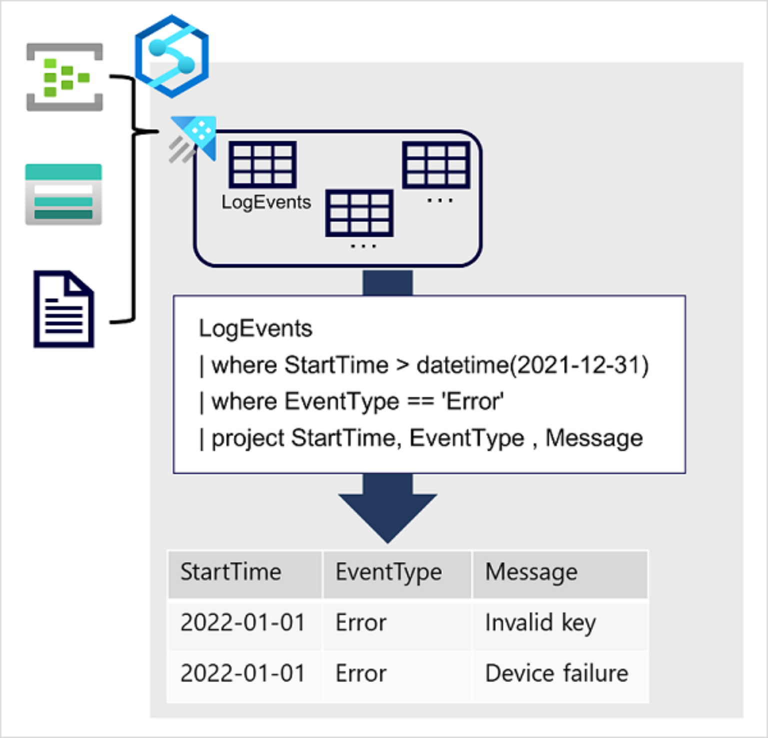
Delta Lake

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.

Azure Data Explorer

Azure Data Explorer is a standalone Azure service for efficiently analyzing data. You can use the service as the output for analyzing large volumes of diverse data from data sources such as websites, applications, IoT devices, and more. For example, by outputting Azure Stream Analytics logs to Azure Data Explorer, you can complement Stream Analytics low latency alert handling with Data Explorer's deep investigation capabilities. The service is also encapsulated as a runtime in Azure Synapse Analytics, where it is referred to as Azure Synapse Data Explorer; enabling you to build and manage analytical solutions that combine SQL, Spark, and Data Explorer analytics in a single workspace.

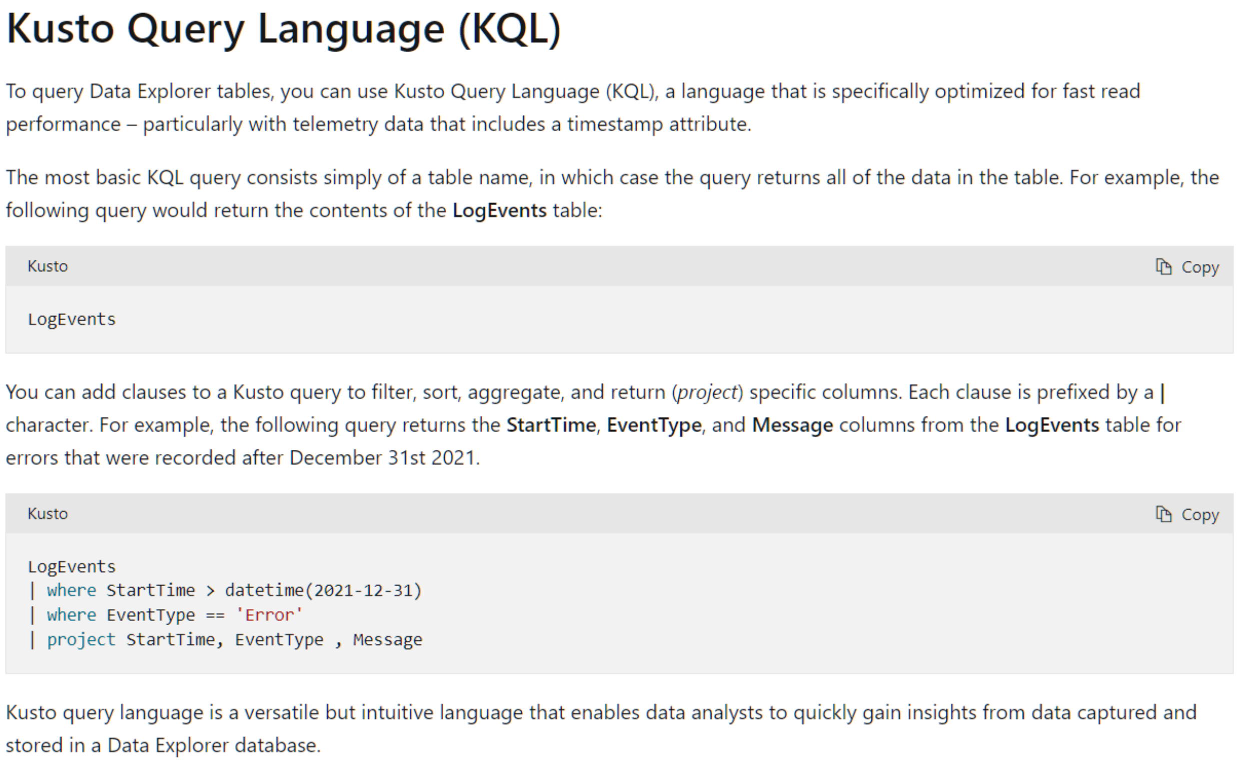


```
LogEvents
| where StartTime > datetime(2021-12-31)
| where EventType == 'Error'
| project StartTime, EventType , Message
```

StartTime	EventType	Message
2022-01-01	Error	Invalid key
2022-01-01	Error	Device failure

Azure Data Explorer is a great choice of technology when you need to:

- Capture and analyze real-time or batch data that includes a time-series element; such as log telemetry or values emitted by Internet-of-things (IoT) devices.
- Explore, filter, and aggregate data quickly by using the intuitive and powerful Kusto Query Language (KQL)



Kusto Query Language (KQL)

To query Data Explorer tables, you can use Kusto Query Language (KQL), a language that is specifically optimized for fast read performance – particularly with telemetry data that includes a timestamp attribute.

The most basic KQL query consists simply of a table name, in which case the query returns all of the data in the table. For example, the following query would return the contents of the **LogEvents** table:

```
LogEvents
```

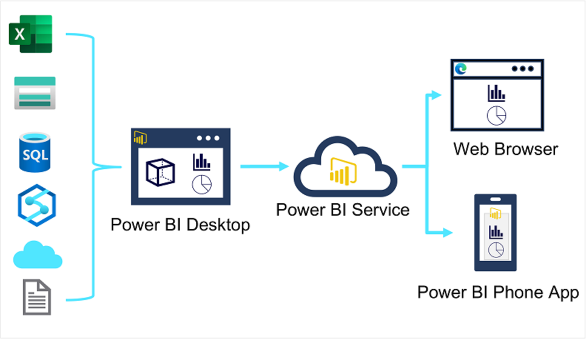
You can add clauses to a Kusto query to filter, sort, aggregate, and return (*project*) specific columns. Each clause is prefixed by a **|** character. For example, the following query returns the **StartTime**, **EventType**, and **Message** columns from the **LogEvents** table for errors that were recorded after December 31st 2021.

```
LogEvents
| where StartTime > datetime(2021-12-31)
| where EventType == 'Error'
| project StartTime, EventType , Message
```

Kusto query language is a versatile but intuitive language that enables data analysts to quickly gain insights from data captured and stored in a Data Explorer database.

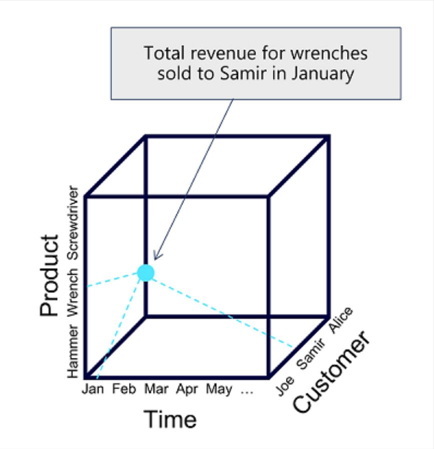
Data Visualization (Read more from Learn)

Data modeling and visualization is at the heart of business intelligence (BI) workloads that are supported by modern data analytics solutions. Essentially, data visualization powers reporting and decision-making that helps organizations succeed. 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.



Concept 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 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.


Total revenue for wrenches sold to Samir in January
Product
Hammer
Wrench
Screwdriver
Jan
Feb
Mar
Apr
May
...
Time
Joe
Samir
Alice
Customer