All Data Engineering

* **Workloads -** Many systems implement relational consistency and isolation by applying locks to data when it is updated. Extensive locking can lead to poor performance, while applications wait for locks to be released.
* Processing data as it arrives is called streaming. Buffering and processing the data in groups is called batch processing.
* **Batch Data** – Static data i.e. sitting in place e.g. csv/json/db/blob(Blob is an acronym for Binary Large Object)
* **Streaming Data** – Always being generated – IoT,Youtube, Apache Kafka,blob log processing
* **Relational Database** – Table,views,keys -Primary & foreign, referential integrity A key-value database stores Associative arrays
* **Analytics -** read-only systems that store vast volumes of historical data

1. Descriptive – What Happened? – Summary – existing records – Hindsight
2. Diagnostic – Why Happened?
3. Predictive
4. Prescriptive – Search Engine Optimize Tool
5. Cognitive

* **Transactional System** - The work performed by transactional systems is often referred to as Online Transactional Processing (OLTP)
* Splitting tables out into separate groups of columns like this is called normalization.
* **Data Ingestion**: Data ingestion is the process of capturing the raw data.
* **Data Transformation/Data Processing:** anomalies that should be filtered out and other Key Performance Indicators (KPIs).
* **Data Querying: T**o perform ad-hoc queries against your data and generate regular reports.
* **Data Visualization**: Data represented in tables such as rows and columns, or as documents, aren't always intuitive.

**First normal form**

1. Eliminate repeating groups in individual tables.
2. Create a separate table for each set of related data.
3. Identify each set of related data with a primary key.

## Second normal form

* Create separate tables for sets of values that apply to multiple records.
* Relate these tables with a foreign key.

## Third normal form

* Eliminate fields that do not depend on the key.

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

* *Atomicity* guarantees that each transaction is treated as a single *unit*, which either succeeds completely, or fails completely. If any of the statements constituting a transaction fails to complete, the entire transaction fails and the database is left unchanged. An atomic system must guarantee atomicity in each and every situation, including power failures, errors, and crashes.
* *Consistency* ensures that a transaction can only take the data in the database from one valid state to another. A consistent database should never *lose* or *create* data in a manner that can't be accounted for. In the bank transfer example described earlier, if you add funds to an account, there must be a corresponding deduction of funds somewhere, or a record that describes where the funds have come from if they have been received externally. You can't suddenly create (or lose) money.
* *Isolation* ensures that concurrent execution of transactions leaves the database in the same state that would have been obtained if the transactions were executed sequentially. A concurrent process can't see the data in an inconsistent state (for example, the funds have been deducted from one account, but not yet credited to another.)
* *Durability* guarantees that once a transaction has been committed, it will remain committed even if there's a system failure such as a power outage or crash.
* **Distributed** databases are widely used in many organizations. A distributed database is a database in which data is stored across different physical locations. If you require transactional consistency in this scenario, locks may be retained for a very long time, especially if there's a network failure between databases at a critical point in time. To counter this problem, many distributed database management systems relax the strict isolation requirements of transactions and implement "eventual consistency." In this form of consistency, as an application writes data, each change is recorded by one server and then propagated to the other servers in the distributed database system asynchronously. While this strategy helps to minimize latency, it can lead to temporary inconsistencies in the data. Eventual consistency is ideal where the application doesn't require any ordering guarantees. Examples include counts of shares, likes, or non-threaded comments in a social media system.
* Data Scope: Batch processing can process all the data in the dataset. Stream processing typically only has access to the most recent data received, or within a rolling time window (the last 30 seconds, for example).
* Data Size: Batch processing is suitable for handling large datasets efficiently. Stream processing is intended for individual records or micro batches consisting of few records.
* Performance: The latency for batch processing is typically a few hours. Stream processing typically occurs immediately, with latency in the order of seconds or milliseconds. Latency is the time taken for the data to be received and processed.
* Analysis: You typically use batch processing for performing complex analytics. Stream processing is used for simple response functions, aggregates, or calculations such as rolling averages

**Azure SQL Database** is a fully managed platform as a service (PaaS) database engine

* highly available and high-performance data storage layer
* At the server level, you can administer [logins](https://docs.microsoft.com/en-us/azure/azure-sql/database/logins-create-manage), [firewall rules](https://docs.microsoft.com/en-us/azure/azure-sql/database/firewall-configure), [auditing rules](https://docs.microsoft.com/en-us/azure/azure-sql/database/auditing-overview), [threat detection policies](https://docs.microsoft.com/en-us/azure/azure-sql/database/threat-detection-configure), and [auto-failover groups](https://docs.microsoft.com/en-us/azure/azure-sql/database/auto-failover-group-overview)
* [SQL authentication](https://docs.microsoft.com/en-us/sql/relational-databases/security/choose-an-authentication-mode#connecting-through-sql-server-authentication).

With this authentication method, the user submits a user account name and associated password to establish a connection. This password is stored in the master database for user accounts linked to a login or stored in the database containing the user accounts not linked to a login.

* [Azure Active Directory Authentication](https://docs.microsoft.com/en-us/azure/azure-sql/database/authentication-aad-overview)

With this authentication method, the user submits a user account name and requests that the service use the credential information stored in Azure Active Directory (Azure AD).

| **Purchasing model** | **Description** | **Best for** |
| --- | --- | --- |
| DTU-based | This model is based on a bundled measure of compute, storage, and I/O resources. Compute sizes are expressed in DTUs for single databases and in elastic database transaction units (eDTUs) for elastic pools. For more information about DTUs and eDTUs, see [What are DTUs and eDTUs?](https://docs.microsoft.com/en-us/azure/azure-sql/database/purchasing-models#dtu-purchasing-model). | Customers who want simple, preconfigured resource options |
| vCore-based | This model allows you to independently choose compute and storage resources. The vCore-based purchasing model also allows you to use [Azure Hybrid Benefit](https://azure.microsoft.com/pricing/hybrid-benefit/) for SQL Server to save costs. | Customers who value flexibility, control, and transparency |

## Database transaction units (DTUs)

A database transaction unit (DTU) represents a blended measure of CPU, memory, reads, and writes.

* [Single database](https://docs.microsoft.com/en-us/azure/azure-sql/database/single-database-overview) represents a fully managed, isolated database. You might use this option if you have modern cloud applications and microservices that need a single reliable data source. A single database is similar to a [contained database](https://docs.microsoft.com/en-us/sql/relational-databases/databases/contained-databases?toc=%2fazure%2fsql-database%2ftoc.json) in the [SQL Server database engine](https://docs.microsoft.com/en-us/sql/sql-server/sql-server-technical-documentation?toc=%2fazure%2fsql-database%2ftoc.json).
* [Elastic pool](https://docs.microsoft.com/en-us/azure/azure-sql/database/elastic-pool-overview) is a collection of [single databases](https://docs.microsoft.com/en-us/azure/azure-sql/database/single-database-overview) with a shared set of resources, such as CPU or memory. Single databases can be moved into and out of an elastic pool.

Azure Blob storage is Microsoft's object storage solution for the cloud. Blob storage is optimized for storing massive amounts of unstructured data.

Blob storage is designed for:

* Serving images or documents directly to a browser.
* Storing files for distributed access.
* Streaming video and audio.
* Writing to log files.
* Storing data for backup and restore, disaster recovery, and archiving.
* Storing data for analysis by an on-premises or Azure-hosted service.

# Azure Cosmos DB

To achieve low latency and high availability, Azure Cosmos DB is a fully managed NoSQL database. non-relational field, you might use [Azure Cosmos DB](https://docs.microsoft.com/en-us/azure/cosmos-db/introduction) as your primary data store

* Deeply integrated with key Azure services used in modern (cloud-native) app development including Azure Functions, IoT Hub, AKS (Azure Kubernetes Service), App Service, and more.
* Choose from multiple database APIs including the native Core (SQL) API, API for MongoDB, Cassandra API, Gremlin API, and Table API.
* Build apps on Core (SQL) API using the languages of your choice with SDKs for .NET, Java, Node.js and Python. Or your choice of drivers for any of the other database APIs.
* Change feed makes it easy to track and manage changes to database containers and create triggered events with Azure Functions.
* Azure Cosmos DB’s schema-less service automatically indexes all your data, regardless of the data model, to deliver blazing fast queries.
* **Database Administrators** manage databases, assigning permissions to users, storing backup copies of data and restore data in case of any failures.
* **Data Engineers** are vital in working with data, applying data cleaning routines, identifying business rules, and turning data into useful information.
* **Data Analysts** explore and analyze data to create visualizations and charts to enable organizations to make informed decisions.

# Azure Databricks

* Azure Databricks is a data analytics platform optimized for the Microsoft Azure cloud services platform. Azure Databricks offers three environments for developing data intensive applications: Databricks SQL, Databricks Data Science & Engineering, and Databricks Machine Learning.
* **Databricks SQL** provides an easy-to-use platform for analysts who want to run SQL queries on their data lake, create multiple visualization types to explore query results from different perspectives, and build and share dashboards.
* **Databricks Data Science & Engineering** provides an interactive workspace that enables collaboration between data engineers, data scientists, and machine learning engineers. For a big data pipeline, the data (raw or structured) is ingested into Azure through Azure Data Factory in batches, or streamed near real-time using Apache Kafka, Event Hub, or IoT Hub. This data lands in a data lake for long term persisted storage, in Azure Blob Storage or Azure Data Lake Storage. As part of your analytics workflow, use Azure Databricks to read data from multiple data sources and turn it into breakthrough insights using Spark.
* **Databricks Machine Learning** is an integrated end-to-end machine learning environment incorporating managed services for experiment tracking, model training, feature development and management, and feature and model serving.

# Azure HDInsight

Azure HDInsight is a managed, full-spectrum, open-source analytics service in the cloud for enterprises. With HDInsight, you can use open-source frameworks such as Hadoop, Apache Spark, Apache Hive, LLAP, Apache Kafka, Apache Storm, R, and more, in your Azure environment

* **ELT/ETL –** Extraction Loading Transformation

**Power BI** is a collection of software services, apps, and connectors that work together to turn your unrelated sources of data into coherent, visually immersive, and interactive insights.

* A Windows desktop application called **Power BI Desktop**.
* An online SaaS (*Software as a Service*) service called the **Power BI service**.
* Power BI **mobile apps** for Windows, iOS, and Android devices.

**Relational Database**

Relational databases are commonly used in ecommerce systems, but one of the major use cases for using relational databases is Online Transaction Processing (OLTP).

OLTP applications are focused on transaction-oriented tasks that process a very large number of transactions per minute.

 of SQL makes it easy for users to perform ad-hoc queries over data. Insert delete or update

Examples of OLTP applications that use relational databases are:

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

 If a table is queried infrequently, but subject to a large number of inserts, updates, and deletes (such as a table involved in OLTP), then creating **indexes** on that table can slow your system down.

A clustered index physically reorganizes a table by the index key.

A view is a virtual table based on the result set of a query.

CREATE VIEW CustomersProducts AS

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

FROM Customers JOIN Orders

ON Customers.CustomerID = Orders.CustomerID

JOIN Products

ON Orders.ProductID = Products.ProductID

**IaaS** is an acronym for Infrastructure-as-a-Service.

**PaaS** stands for Platform-as-a-service.