Data Engineering

Shabab Akhter
External Lecturer
RUC

Contents

Contents

Background

In the previous class you learned about how to work with data – analyze it, transform it, clean it, etc. The data was easily accessible and ready for you to use so you could easily load it in a notebook and start working with it, ***BUT in most real-world settings, that is not the case.***

Real world situation:

1. What if the data was not in just one file but rather in 10 different databases?
2. What is the data was in an external system?
3. What if some of the data was numerical, while some text and some images?

In today's class we are going to explore how to answer these questions and work with data in the real life

Data and its types

What is Data?

Data is raw facts and figures collected from various sources, waiting to be organized and interpreted.

Structured: Highly organized and easily stored in tables or databases with predefined schemas.
Examples: Customer records, financial transactions, inventory logs.
Characteristics:

- Follows a strict format (e.g., rows and columns).
- Suitable for SQL databases and relational models.

Semi-Structured: Data with some level of organization but lacking a strict schema.
Examples: JSON files, XML files, and NoSQL database records.
Characteristics:

- Contains tags or markers to separate elements.
- Combines flexibility of unstructured data with some structure.

Unstructured: Data that lacks any predefined structure or organization.
Examples: Text files, images, videos, social media posts
Characteristics:

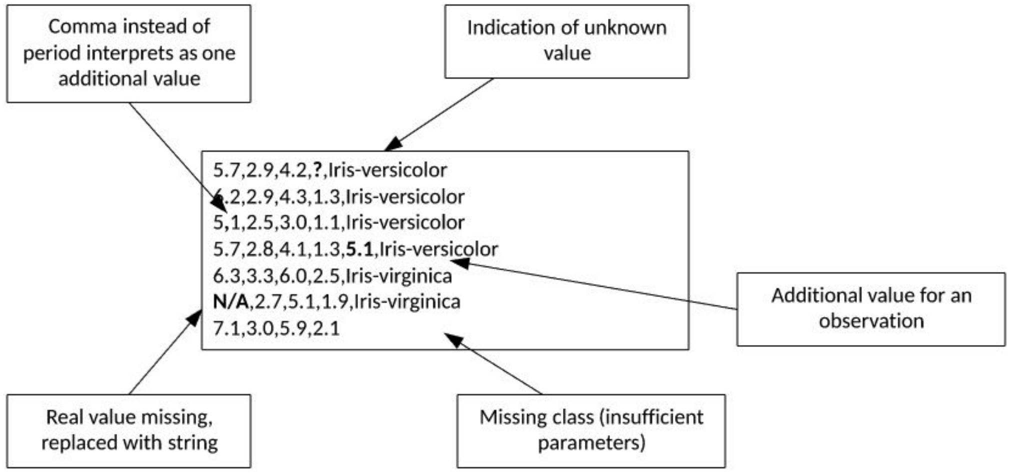
- Requires advanced tools like AI and machine learning for processing.
- Stored in data lakes or NoSQL databases for flexibility.

Importance of Data

- **Data Drives Decision-Making**
  - Enables businesses and organizations to make informed choices backed by evidence and trends
    - *E.g., Companies use customer data to predict future market demands.*
- **Enhances Customer Experience**
  - Powers personalized recommendations, improving user satisfaction and engagement.
    - *E.g., Platforms like Netflix and Amazon use data to tailor recommendations to individual preferences.*
- **Fuels Innovation and AI**
  - Serves as the foundation for advancements in AI, machine learning, and predictive analytics.
    - *E.g., Self-driving cars use real-time sensor data to navigate safely and efficiently.*
- **Improves Efficiency and Reduces Costs**
  - Optimizes processes across industries, increasing productivity while lowering operational costs.
    - *E.g., Predictive maintenance in manufacturing prevents equipment breakdowns and reduces downtime.*

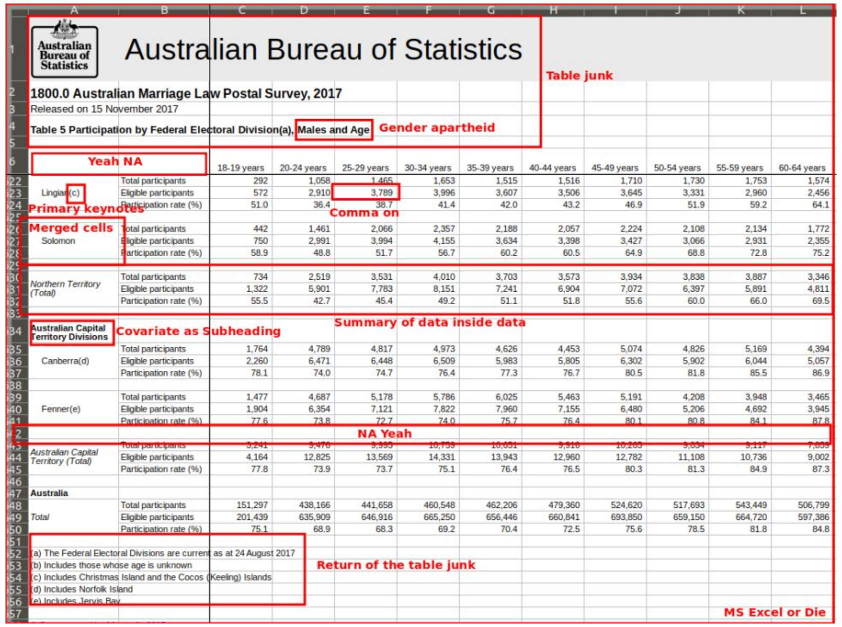
The Challenge with Data

- **Data is Often Messy**
  - Raw data contains errors, duplicates, and inconsistencies.
    - *E.g., Customer entries with incomplete or incorrect information.*
- **Data is Siloed**
  - Data is spread across multiple systems and formats, making integration complex.
    - *E.g., Sales data is in one database and marketing data is in another, with no unified view.*
- **Data is Inaccessible**
  - Lack of proper systems and tools can make it hard to retrieve and use data efficiently.
    - *E.g., Legacy systems that don't support modern analytics platforms.*
- **Data Volume is Overwhelming**
  - The sheer scale of data generated daily requires robust storage and processing solutions.



```
5.7,2.9,4.2,?,Iris-versicolor
6.2,2.9,4.3,1.3,Iris-versicolor
5,1,2.5,3.0,1.1,Iris-versicolor
5.7,2.8,4.1,1.3,5.1,Iris-versicolor
6.3,3.3,6.0,2.5,Iris-virginica
N/A,2.7,5.1,1.9,Iris-virginica
7.1,3.0,5.9,2.1
```

Comma instead of period interprets as one additional value
Indication of unknown value
Additional value for an observation
Real value missing, replaced with string
Missing class (insufficient parameters)



Australian Bureau of Statistics

1800.0 Australian Marriage Law Postal Survey, 2017

Released on 15 November 2017

Table 5 Participation by Federal Electoral Division(a), Males and Age

		18-19 years	20-24 years	25-29 years	30-34 years	35-39 years	40-44 years	45-49 years	50-54 years	55-59 years	60-64 years
Lingiari(c)	Total participants	292	1,058	1,465	1,653	1,515	1,516	1,710	1,730	1,753	1,574
	Eligible participants	572	2,910	3,789	3,996	3,607	3,506	3,645	3,331	2,960	2,456
	Participation rate (%)	51.0	36.4	38.7	41.4	42.0	43.2	46.9	51.9	59.2	64.1
Solomon	Total participants	442	1,461	2,066	2,357	2,188	2,057	2,224	2,108	2,134	1,772
	Eligible participants	750	2,991	3,994	4,155	3,634	3,398	3,427	3,066	2,931	2,355
	Participation rate (%)	58.9	48.8	51.7	56.7	60.2	60.5	64.9	68.8	72.8	75.2
Northern Territory (Total)	Total participants	734	2,519	3,531	4,010	3,703	3,573	3,934	3,838	3,887	3,346
	Eligible participants	1,322	5,901	7,783	8,151	7,241	6,904	7,072	6,397	5,891	4,811
	Participation rate (%)	55.5	42.7	45.4	49.2	51.1	51.8	55.6	60.0	66.0	69.5
Australian Capital Territory Divisions											
Canberra(d)	Total participants	1,764	4,789	4,817	4,973	4,626	4,453	5,074	4,826	5,169	4,394
	Eligible participants	2,260	6,471	6,448	6,509	5,983	5,805	6,302	5,902	6,044	5,057
	Participation rate (%)	78.1	74.0	74.7	76.4	77.3	76.7	80.5	81.8	85.5	86.9
Fenner(e)	Total participants	1,477	4,687	5,178	5,786	6,025	5,463	5,191	4,208	3,948	3,465
	Eligible participants	1,904	6,354	7,121	7,822	7,960	7,155	6,480	5,206	4,692	3,945
	Participation rate (%)	77.6	73.8	72.7	74.0	75.7	76.4	80.1	80.8	84.1	87.8
Australian Capital Territory (Total)	Total participants	3,241	9,476	9,995	10,759	10,651	9,916	10,265	9,034	9,117	7,059
	Eligible participants	4,164	12,825	13,569	14,331	13,943	12,960	12,782	11,108	10,736	9,002
	Participation rate (%)	77.8	73.9	73.7	75.1	76.4	76.5	80.3	81.3	84.9	87.3
Australia											
Total	Total participants	151,297	438,166	441,658	460,548	462,206	479,360	524,620	517,693	543,449	506,799
	Eligible participants	201,439	635,909	646,916	665,250	656,446	660,841	693,850	659,150	664,720	597,386
	Participation rate (%)	75.1	68.9	68.3	69.2	70.4	72.5	75.6	78.5	81.8	84.8

(a) The Federal Electoral Divisions are current as at 24 August 2017
(b) Includes those whose age is unknown
(c) Includes Christmas Island and the Cocos (Keeling) Islands
(d) Includes Norfolk Island
(e) Includes Jervis Bay

The Solution: Data Engineering

What is Data Engineering?

- Data Engineering is the practice of designing, building, and maintaining systems to handle data (whether large or small) effectively.
- It bridges the gap between raw data and actionable insights, ensuring data is accessible, clean, and ready for analysis or decision-making.

Why it is important?

- Enable organizations to process, store, and analyze data efficiently.
- Lay the foundation for reliable and scalable data-driven insights.

The Data Life Cycle

1. Data Storage and Management
2. Data Transformation and Preparation
3. Data Acquisition

The Data Life Cycle ensures that data is properly collected, organized, cleaned, and transformed, making it ready for analysis, decision-making, and machine learning applications.

Data Storage and Management

Organizing and Storing Data:

- Structured data is stored in SQL databases like PostgreSQL or MySQL.
- Unstructured and semi-structured data are stored in NoSQL databases like MongoDB, Cassandra, or in cloud-based data lakes.

Tools Used: Apache HDFS, AWS S3, Azure Data Lake, Google BigQuery.

Ensuring Data Accessibility and Security:

- Data is organized and indexed for easy retrieval.
- Access controls (RBAC) ensure only authorized users can view or manipulate the data.

Tools Used: AWS IAM, Azure Active Directory, Google Cloud Identity, and data encryption protocols.

Data Transformation and Preparation

Cleaning and Enriching Data:

- Raw data is cleaned to remove duplicates, fill missing values, and handle errors.
- Data is then enriched by merging additional sources or transforming it into usable formats.

Tools Used: Apache Spark, pandas, Talend, dbt (data build tool).

Data Pipelines:

- Data engineering automates the movement and transformation of data using ETL (Extract, Transform, Load) pipelines.
- Data is extracted from sources, transformed according to business logic, and loaded into a destination like a data warehouse or lake.

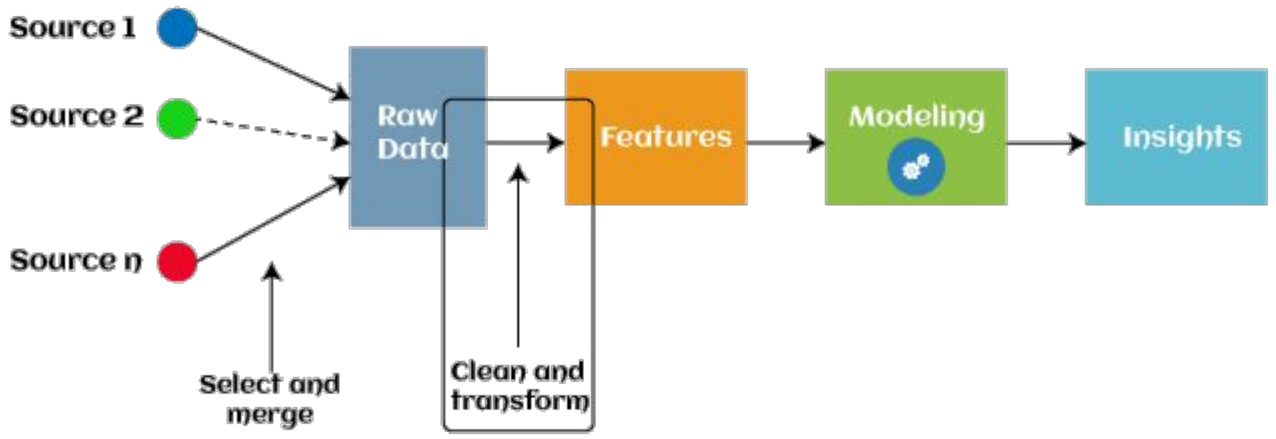
Tools Used: Apache Airflow, AWS Glue, Talend, Prefect.

Data Transformation and Preparation

Feature Engineering and Preprocessing:

- Data is transformed into features suitable for analysis or machine learning (e.g., scaling, encoding, feature selection).
- Data is normalized, categorized, and aggregated to create meaningful features for predictive models.

Tools Used: Scikit-learn, pandas, Spark MLlib.



Data Acquisition

Collecting and Ingesting Data:

- Data is collected from multiple sources like external databases, IoT devices, internal databases, logs, and third-party systems.
- Data can be ingested in batches or in real-time depending on the use case.

Tools Used: Kafka, Flume, Apache NiFi, and cloud-based services (AWS Kinesis, Google Pub/Sub).

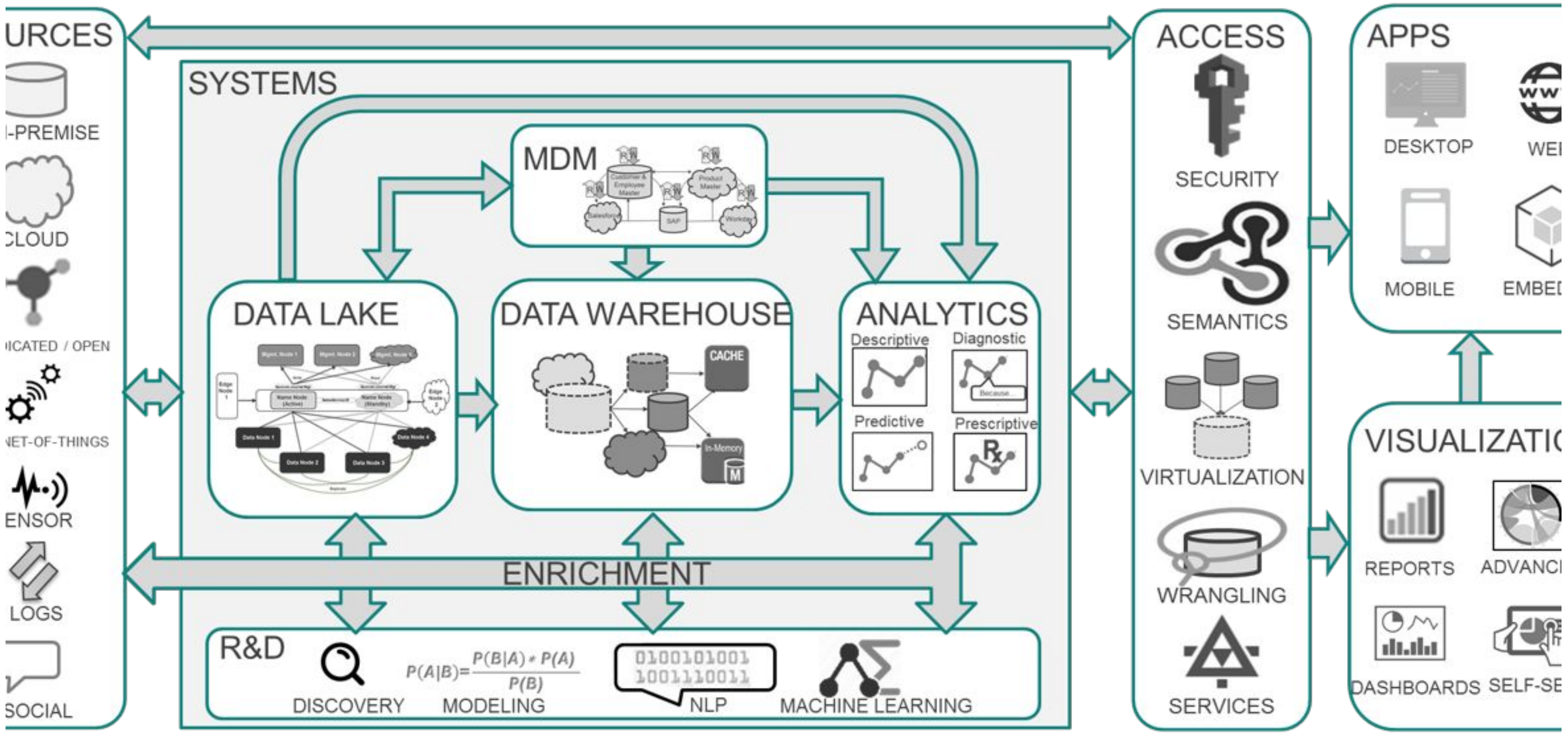
Real-time vs Batch Ingestion:

- **Real-time:** Streaming data for immediate processing (e.g., sensor data sent every second).
- **Batch:** Periodic collection (e.g., daily transactional data collected from different systems).

A data engineer works across the entire data lifecycle

1. Data Acquisition
2. Data storage and management
3. Data transformation & preparation

Modern Data Architecture



Contents

Introduction to databases

- Databases are organized collections of data that allow for efficient storage, retrieval, and management of information.

- Deployment models:
    - On-Premise Databases: Hosted on local servers within the organization (e.g., Oracle, MySQL).
    - Cloud Databases: Fully managed by cloud providers, offering scalability and ease of use (e.g., AWS RDS, Google Cloud SQL).
    - Hybrid Databases: Combine on-premise and cloud deployments, enabling flexible data storage and migration.

Types of Databases

SQL databases store structured data in tables with predefined schemas

Other databases that a data engineer might use include **Time-series database** and **Geospatial database**

NoSQL databases handle unstructured or semi-structured data with flexible schemas.

Vector databases are designed to store high-dimensional vectors, enabling similarity searches for machine learning and AI applications.

Graph databases use graph structures for semantic queries with nodes, edges, and properties to represent and store data.

SQL Database

SQL databases store data in structured tables with defined schemas and use relational models to maintain data relationships.

They are mostly used for applications such as:

Transactional Systems: E-commerce, banking, and financial systems for consistent, reliable data storage.

Data Warehousing: Analytical tasks where structured data and complex querying are necessary.

ERP and CRM Systems: Managing enterprise resources and customer information efficiently.

Pros

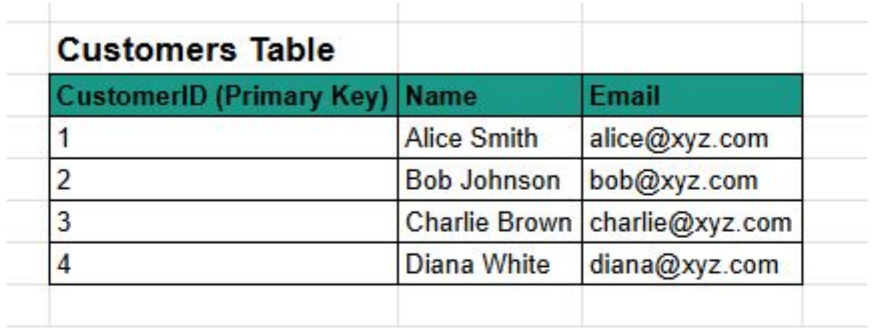
- Excellent for structured data.
- Supports complex queries.
- Ensures data consistency (ACID).

Cons

- Rigid schema
- Challenging to scale horizontally.
- Not suitable for unstructured data.

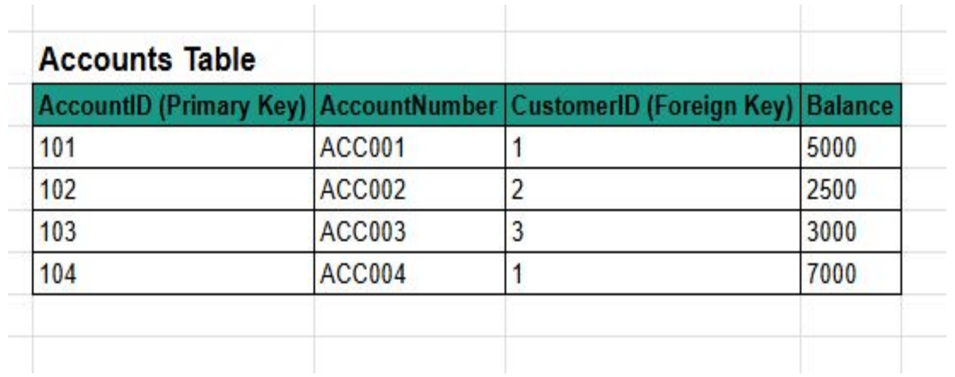
Commonly Used SQL Databases

- MySQL
- PostgreSQL
- Microsoft SQL Server
- Oracle Database
- SQLite



Customers Table

CustomerID (Primary Key)	Name	Email
1	Alice Smith	alice@xyz.com
2	Bob Johnson	bob@xyz.com
3	Charlie Brown	charlie@xyz.com
4	Diana White	diana@xyz.com



Accounts Table

AccountID (Primary Key)	AccountNumber	CustomerID (Foreign Key)	Balance
101	ACC001	1	5000
102	ACC002	2	2500
103	ACC003	3	3000
104	ACC004	1	7000

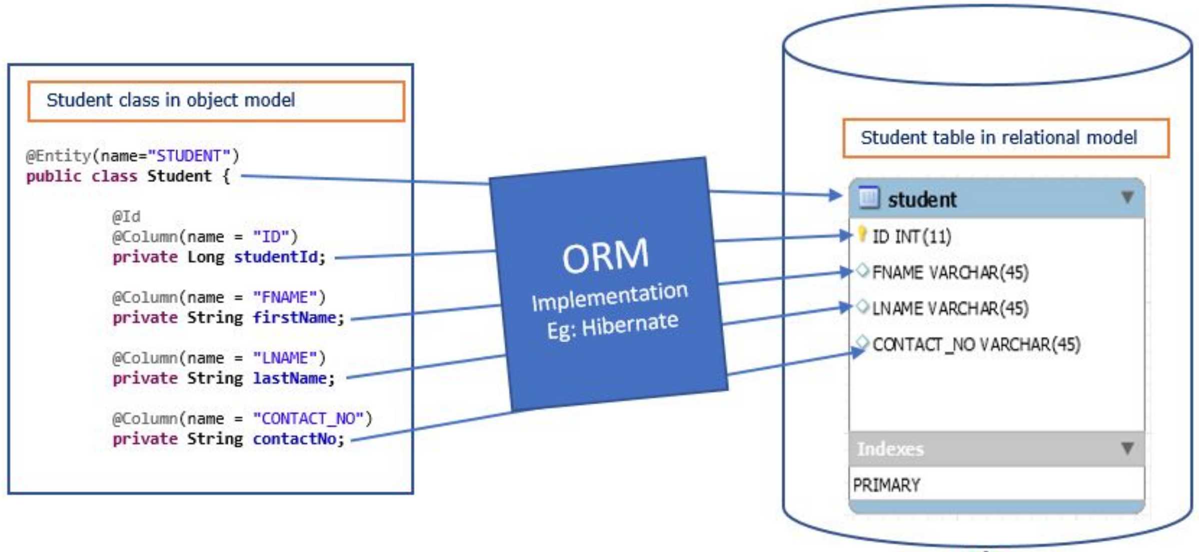
Difficulties with relational DBs

- Big data and data from Web 2.0 are often unstructured or semi-structured.
- Relational DB design is not easy.
- Relational DBs don't naturally scale out (e.g., to a cluster of machines).

Object–relational impedance mismatch:

Mismatch between OOP and relational DBs

- A set of conceptual and technical difficulties between OOP and RDBs
- A translation layer is needed between the objects in the application code and the data (rows and columns) in the tables.
- Object-relational mapping (ORM)



```
@Entity(name="STUDENT")
public class Student {

        @Id
        @Column(name = "ID")
        private Long studentId;

        @Column(name = "FNAME")
        private String firstName;

        @Column(name = "LNAME")
        private String lastName;

        @Column(name = "CONTACT_NO")
        private String contactNo;
```

ORM implements responsibility of mapping the Object to Relational Model. JavaByDeveloper

NoSQL Database

NoSQL databases store data in flexible formats like key-value pairs, documents, graphs, or wide columns, enabling schema-less or dynamic schema designs for scalability and performance.

They are mostly used for applications such as:

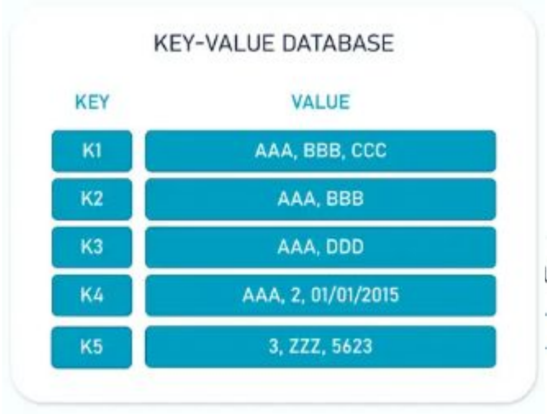
Real-Time Analytics: Social media platforms, IoT systems, and recommendation engines where high-speed data processing is crucial.

Big Data Applications: Handling unstructured or semi-structured data, such as logs, JSON, or XML files.

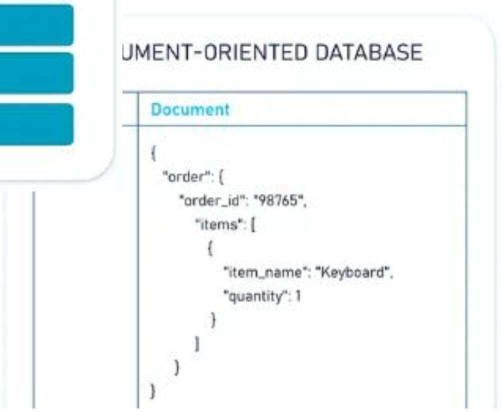
Scalable Web Apps: Applications with dynamic data models or those requiring horizontal scalability.

Commonly Used NoSQL Databases

- MongoDB
- Cassandra
- Redis
- Couchbase
- Amazon DynamoDB


KEY-VALUE DATABASE

KEY	VALUE
K1	AAA, BBB, CCC
K2	AAA, BBB
K3	AAA, DDD
K4	AAA, 2, 01/01/2015
K5	3, ZZZ, 5623


UMENT-ORIENTED DATABASE

Document

```
{
  "order": {
    "order_id": "98765",
      "items": [
        {
          "item_name": "Keyboard",
          "quantity": 1
        }
      ]
    }
}
```

Pros

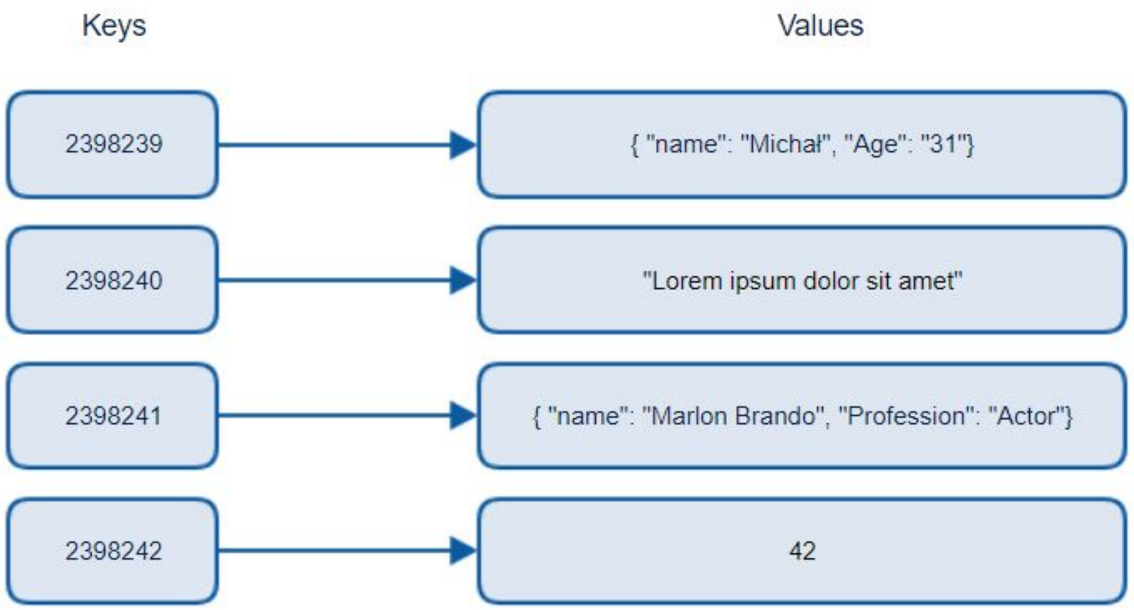
- Flexible schema
- High scalability for large datasets.
- Optimized for specific use cases like caching or graph queries.

Cons

- Lack of standardization.
- Less suitable for complex relationships.
- Limited support for ACID compliance.

How do Key-Value stores work

- Key–value stores use an associative array (a.k.a. map or dictionary)
  - The key is represented by an arbitrary string
  - The value can be any kind of data like an image, file, text or document
- Data is represented as a collection of key–value pairs
  - Each pair forms a ‘row’
  - Keys are unique but values are not.
  - Values across keys may be heterogeneous.
  - Given a key, the value could be composite.


Keys
Values
2398239
{ "name": "Michał", "Age": "31"}
2398240
"Lorem ipsum dolor sit amet"
2398241
{ "name": "Marlon Brando", "Profession": "Actor"}
2398242
42

Graph Database

Graph databases store data as nodes (entities), edges (relationships), and properties, making them ideal for analyzing interconnected data.

They are mostly used for applications such as:

Social Networks: To model user connections, interactions, and relationships effectively.

Recommendation Systems: Providing personalized suggestions by analyzing user-item relationships.

Fraud Detection: Identifying suspicious patterns and anomalies in financial or transactional networks.

Pros

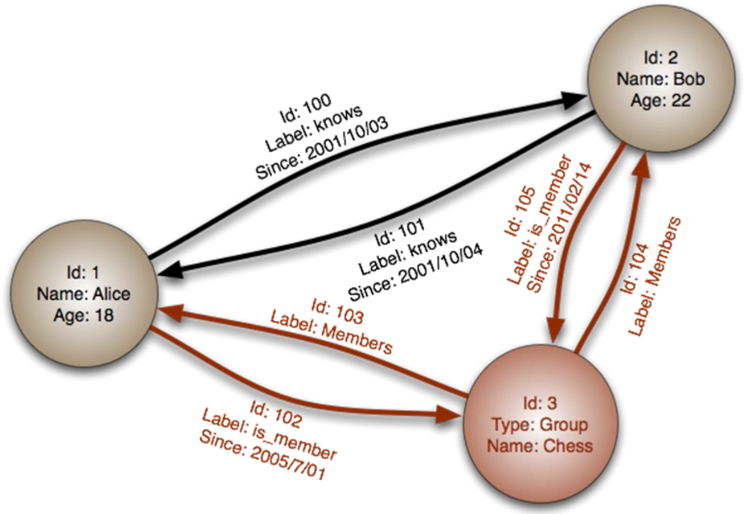
- More descriptive queries
- Greater flexibility in adapting your model
- Greater performance when traversing data relationships

Cons

- Difficult to scale, e.g., partitioning the data
- No standard language

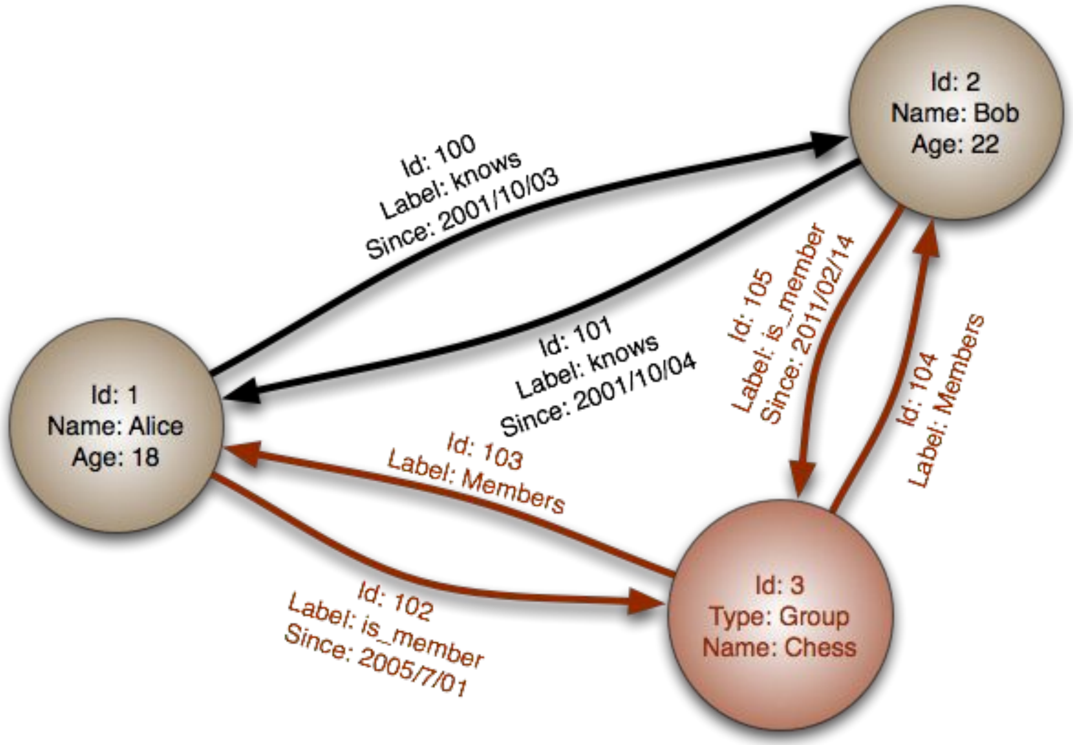
Commonly Used Graph Databases

- Neo4j
- Amazon Neptune
- ArangoDB
- OrientDB
- TigerGraph



How do graph databases work

- **Nodes** represent entities or instances such as people, businesses, accounts, or others. Each node is roughly the equivalent of a record in a relational database, or a document in a document-store database.
  - E.g., Alice, Bob and Chess Group
- **Properties** are information associated to nodes.
  - E.g., Name, Age, …
- **Edges** represent the (directed or undirected) relationship between nodes.
  - The key concept in graph databases
  - An abstraction that is not directly implemented in a relational model or a document-store model.
  - Different types of relationships in the example.



Property model

- A graph database uses graph structures for semantic queries with nodes, edges, and properties to represent and store data.
- It is designed to treat the relationships between data as equally important to the data itself.
  - Accessing nodes and relationships in a native graph database is equally efficient.
- Graph data models
  - Property Graph Model
  - Triple-Store Model
- A graph is formally defined as G = (V, E)
  - V: Vertex/node set. Each node may have a number of properties.
  - E: Edge set. A edge captures the relationship between two nodes. It may be directed or not, and/or have a weight.
- Examples:
  - Social networks: Facebook, Twitter, Instagram
  - Web: Hyperlinks
  - Transportation: Road networks or railway networks
- Graph data can be stored in Relational Databases, but (native) Graph Databases are more efficient at managing graph data.

Vector Database

Vector databases store data as vectors, representing high-dimensional data points, ideal for machine learning tasks such as similarity search, recommendations, and natural language processing.

They are mostly used for applications such as:

Search Engines: Enhancing search with semantic matching and nearest neighbor searches.

Natural Language Processing (NLP): Storing and querying word embeddings or document embeddings.

Image and Video Search: Searching for similar images or videos based on feature vectors.

Pros

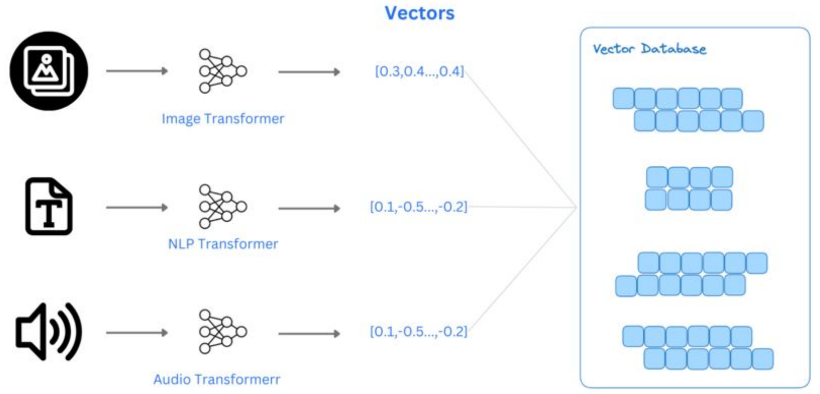
- Optimized for high-dimensional data and similarity searches.
- Scalable and efficient for machine learning applications.
- Fast indexing and retrieval of large datasets.

Cons

- Not suited for traditional relational data.
- Requires advanced setup and knowledge.
- Limited tooling and support compared to other database types.

Commonly Used Vector Databases

- PGVector
- Pinecone
- Faiss (Facebook AI Similarity Search)
- Weaviate
- Milvus



Some more databases

Time-series databases are designed to store and manage data points that are indexed by time, optimized for high-frequency data collection and querying.

They are mostly used for applications such as:

IoT Data: Storing and analyzing data from IoT devices, like smart meters or temperature sensors.

Financial Market Data: Tracking and analyzing stock prices, trading volumes, and other economic indicators.

Server and Application Monitoring: Collecting metrics related to CPU usage, memory consumption, and network traffic.

Commonly Used Time-Series Databases

- InfluxDB
- Prometheus
- TimescaleDB
- OpenTSDB
- Graphite

Geospatial databases are specialized for storing, querying, and analyzing spatial data supporting operations like proximity searches and spatial relationships.

They are mostly used for applications such as:

Geographical Information Systems (GIS): Analyzing and visualizing spatial data for urban planning, environmental monitoring, and resource management.

Location-Based Services: Enabling applications like navigation, ride-hailing, and proximity-based recommendations.

Commonly Used Geospatial Databases

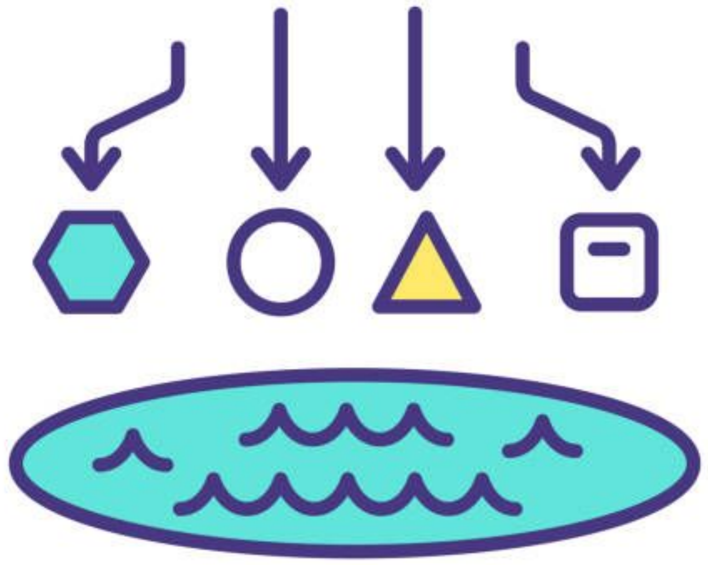
- PostGIS (extension for PostgreSQL)
- MongoDB with GeoJSON
- Oracle Spatial
- CartoDB
- Spatialite

Data Lake

A data lake is a centralized repository that stores large amounts of raw, unstructured, semi-structured, and structured data at scale. It enables organizations to store data in its original form for future processing and analysis.

Key Features

- Data Variety: Stores all types of data—text, images, videos, logs, and more.
- Schema-on-Read: Data is not structured until it is accessed, offering flexibility.
- Scalability: Can handle massive volumes of data with distributed storage systems.

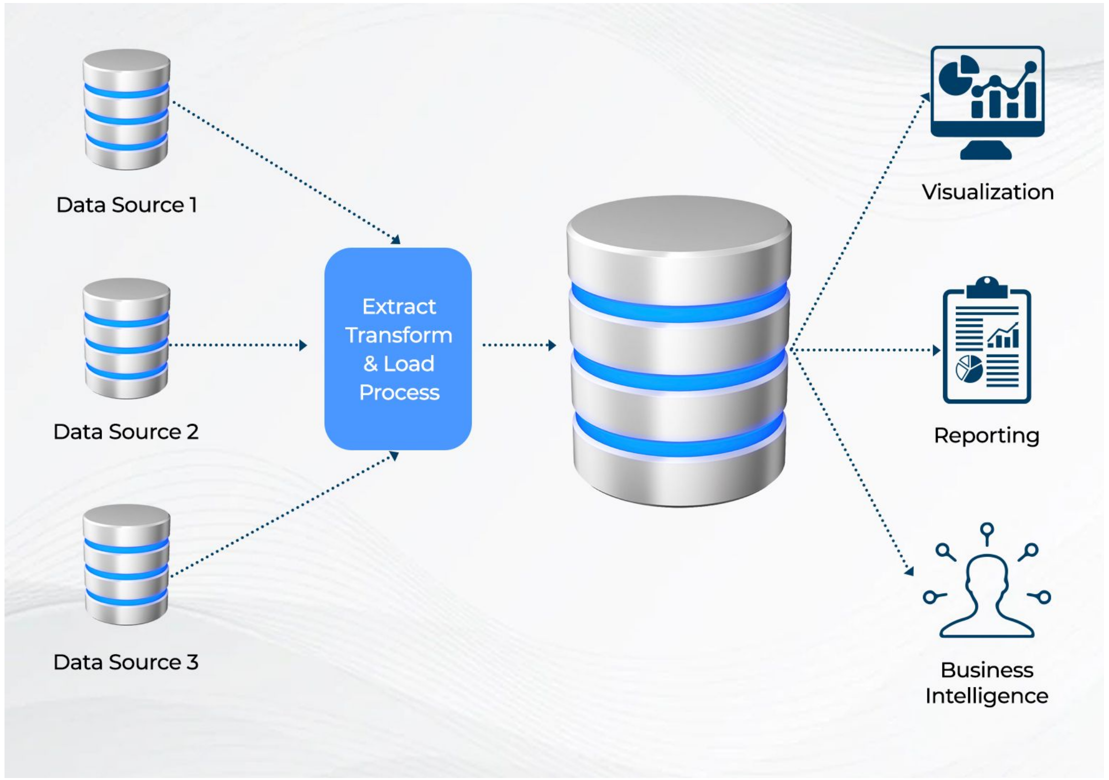


Data Warehouse

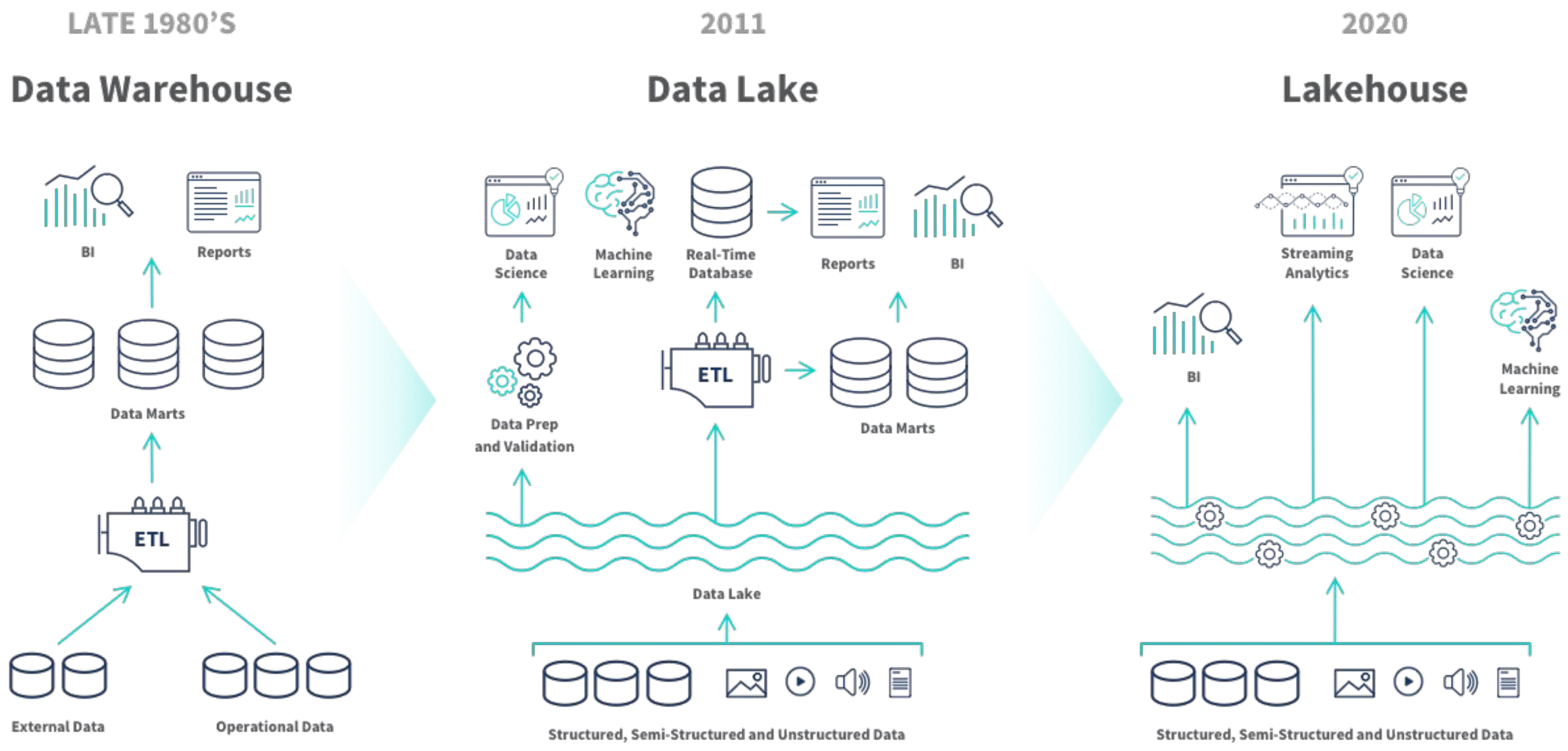
A data warehouse is a centralized repository designed to store structured and pre-processed data, optimized for query performance and business intelligence (BI) tasks.

Key Features

- Structured Data: Organizes data into tables with predefined schemas.
- Schema-on-Write: Data is processed and structured before storage.
- Optimized for Analytics: Provides fast query performance for reporting and decision-making.



How Data Lake and Data Warehouse Differ



Azure Data Lake Storage (ADLS)

❓What is ADLS?

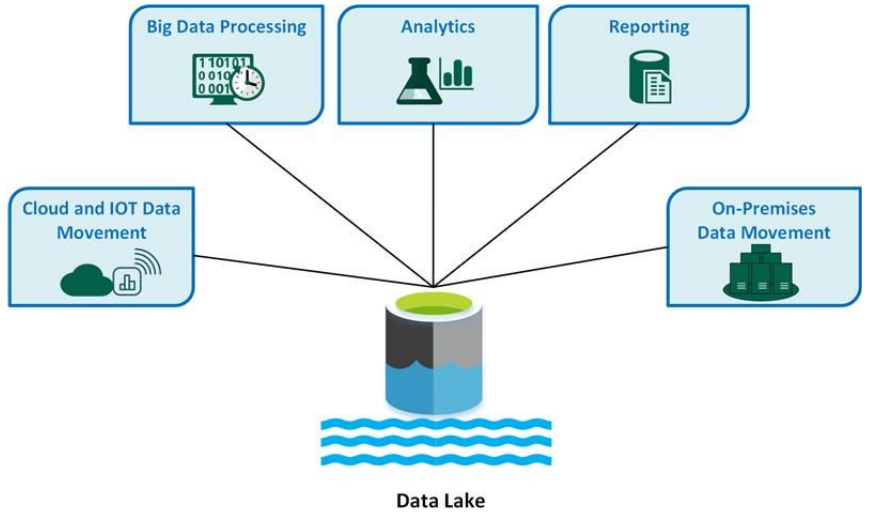
A scalable, cloud-based data lake solution that integrates with the Azure ecosystem.

Key Features:

- Stores vast amounts of raw data.
- Secure with encryption and fine-grained access control.
- Integrates with Azure tools (Databricks, Synapse).

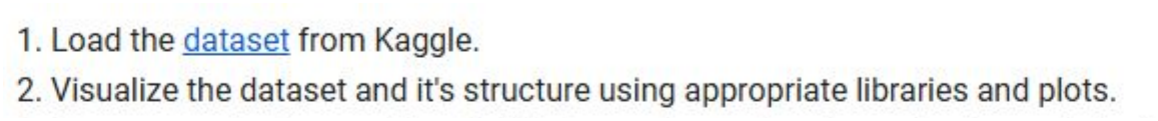
Example Use:

- Store raw sensor data in ADLS, process with **Azure Databricks**, and analyze with **Azure Synapse Analytics**.



Exercise

- Let us look at some examples
- Complete the following tasks from exercise notebook:



1. Load the dataset from Kaggle.
2. Visualize the dataset and it's structure using appropriate libraries and plots.

Contents

Introduction to Data Transformation & Techniques

- Data transformation involves converting raw data into a format suitable for analysis, machine learning, and business intelligence.
- Key Techniques in Data Transformation
  - Preprocessing:
    - Handling missing values (e.g., replacing, dropping).
    - Normalizing or standardizing data.
    - Detecting and handling outliers.
- **Feature Engineering**:
  Creating new features from raw data that make machine learning models more effective.
  Example: Converting dates into year, month, day, or creating features like total transaction amount.

Look at all of this data! Tons of it!
Yes, but what does it all mean? Too much of a good thing isn't always a good thing. Data without meaning... is just data.
Copyright © MMXV Shep Hyken.

Preprocessing Data

What is preprocessing?

Preprocessing is the process of cleaning, transforming, and organizing raw data to prepare it for analysis or modeling. It ensures the data is consistent, accurate, and usable.

Why is Preprocessing Important?

Improves Data Quality: Handles missing, noisy, or inconsistent data.

Enhances Model Performance: Provides well-structured data for machine learning models.

Ensures Compatibility: Aligns data with the requirements of analytical tools and techniques.

Feature Engineering

What is feature engineering?

Feature Engineering is the process of using domain knowledge to extract features from raw data. These features help improve the performance of machine learning models.

Key Aspects of feature engineering:

1. Feature Creation
2. Feature Selection
3. Feature Encoding

Why is Feature Engineering Important?

- Improves Model Accuracy: Relevant features lead to better predictions and insights.
- Reduces Overfitting: Properly selected features help the model generalize better on unseen data.
- Speeds Up Computation: Efficient feature sets reduce the complexity of models, leading to faster training and inference.

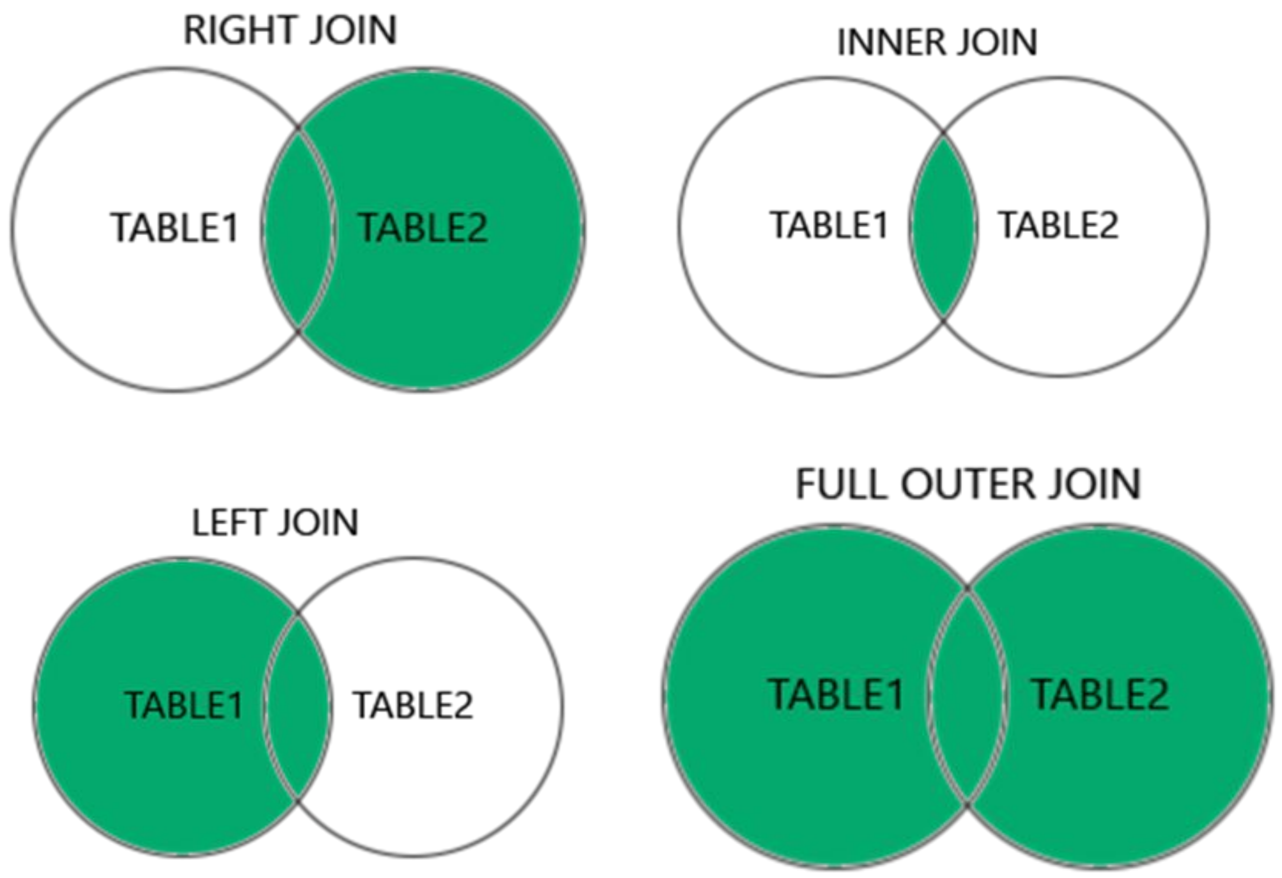
Join

Joining is used to combine multiple DataFrames based on common columns (keys).

Syntax: *df1.join(df2, on='key_column', how='join_type')*

Types of Joins:

- (INNER) JOIN: Returns records that have matching values in both tables
- LEFT (OUTER) JOIN: Returns all records from the left table, and the matched records from the right table
- RIGHT (OUTER) JOIN: Returns all records from the right table, and the matched records from the left table
- FULL (OUTER) JOIN: Returns all records when there is a match in either left or right table



GroupBy

GroupBy is used to group data based on certain features and apply aggregate functions (e.g., sum, mean) to each group.

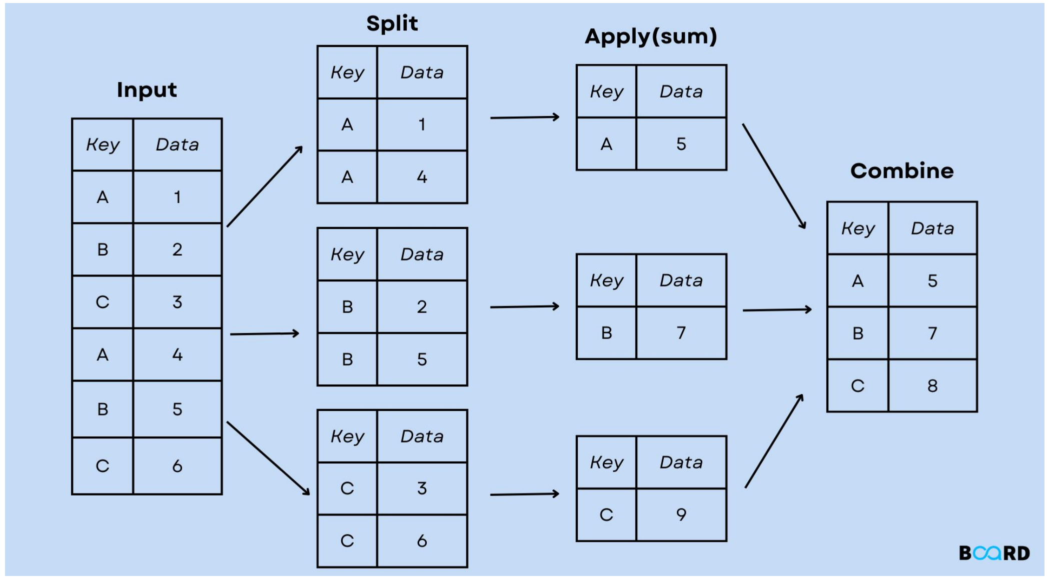
Syntax: *df.groupby('column_name').agg({'column_to_aggregate': 'aggregation_function'})*

Steps in GroupBy:

- Split: Divides the data into groups based on a column or condition.
- Apply: Applies a function (e.g., sum, mean) to each group.
- Combine: Combines the results back into a DataFrame.

Common Aggregate Functions:

- sum(): Returns the sum of each group.
- mean(): Returns the mean of each group.
- count(): Counts the number of entries in each group.
- agg(): Allows custom aggregation functions.



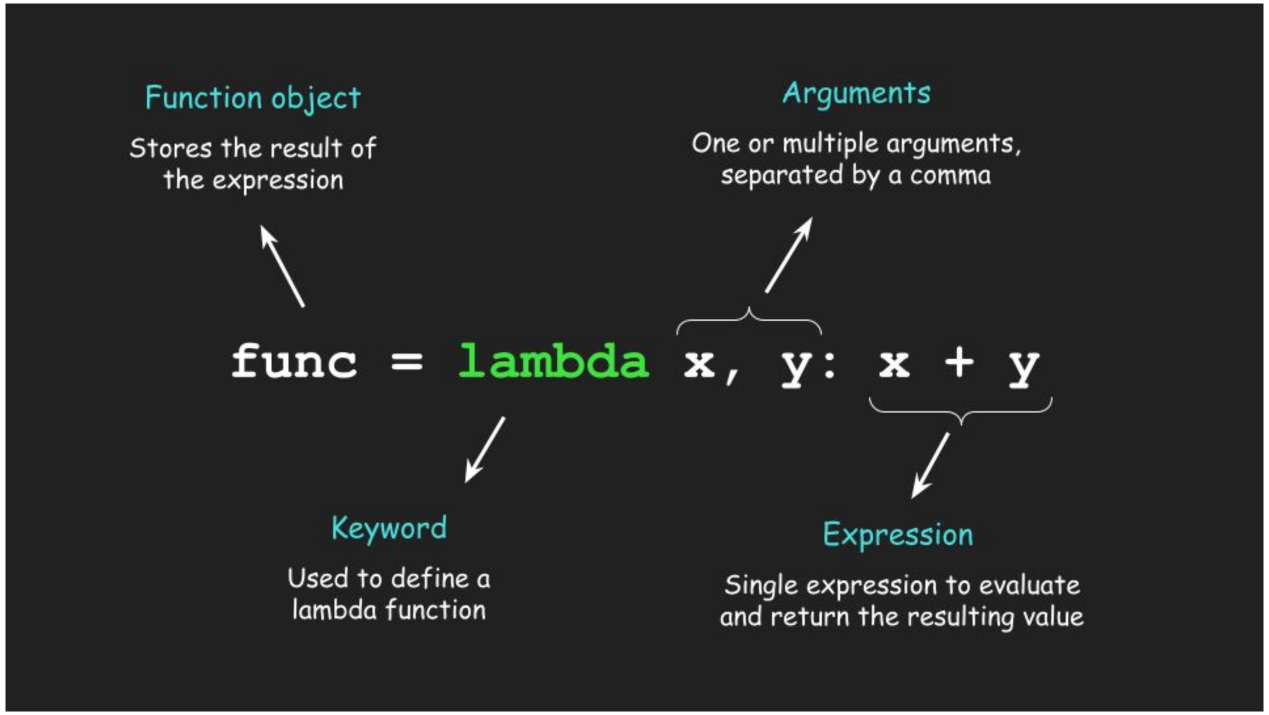
Lambda Functions

Lambda functions are small, anonymous functions defined using the lambda keyword.

Use Cases:

- **Apply to DataFrame:** Lambda functions are often used with **apply()**, **map()**, or **filter()** to modify or operate on DataFrame columns.
- **Sorting or Filtering:** For custom sorting or filtering of data.

Lambda functions provide much faster execution time.


Function object
Stores the result of the expression
Arguments
One or multiple arguments, separated by a comma
func = lambda x, y: x + y
Keyword
Used to define a lambda function
Expression
Single expression to evaluate and return the resulting value

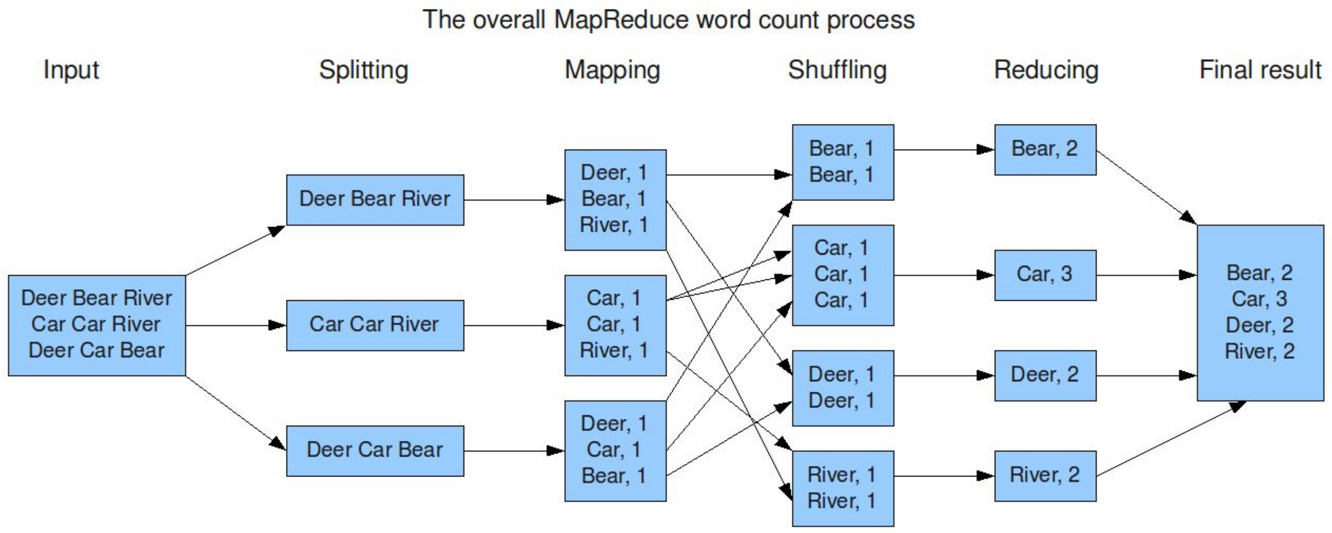
MapReduce: A Distributed Data Processing Model

MapReduce is a programming model for processing large datasets in a distributed computing environment.

It splits tasks into small, manageable chunks and processes them in parallel

Three concepts of MapReduce

1. Programming model
   - How are developers supposed to program?
2. Execution framework
   - Runtime that coordinates the execution of programs written in the MapReduce paradigm
3. Software implementation
   - The implementation of the previous two: Google, Hadoop and others.



MapReduce Programming Model

- **Input:** A set of key/value pairs
- **Output:** Another set of key/value pairs
- Keys and values can be primitives or complex types
- A programmer implements two functions: map and reduce
- **map:** `(k1, v1)  list(k2, v2)`
  - Takes an input pair and produces a set of intermediate key/value pairs. MapReduce groups all intermediate pairs with the same key and gives them to *reduce.*
- **reduce:** `(k2,list(v2))  list(k3,v3)(Hadoop)`
  `list(v2)(Google)`
  - Takes an intermediate key and the set of all values for that key.

    Merges the values to form a smaller set (typically empty or with a single value)

Example: WordCount

- Data: A large collection of documents
- Expected output: #occurrence of *each* word across *all* documents.

- Parallelism could be easily achieved
  - A number of mappers can work on different documents in parallel
    - For each word they see, they output an intermediate count (of 1) for the word
  - When the mappers all finish, a number of reducers can work in parallel
    - A reducer gets all counts for a certain word and ‘reduces’ the list of counts (i.e., 1’s) to a single value

Map and Reduce for WordCount

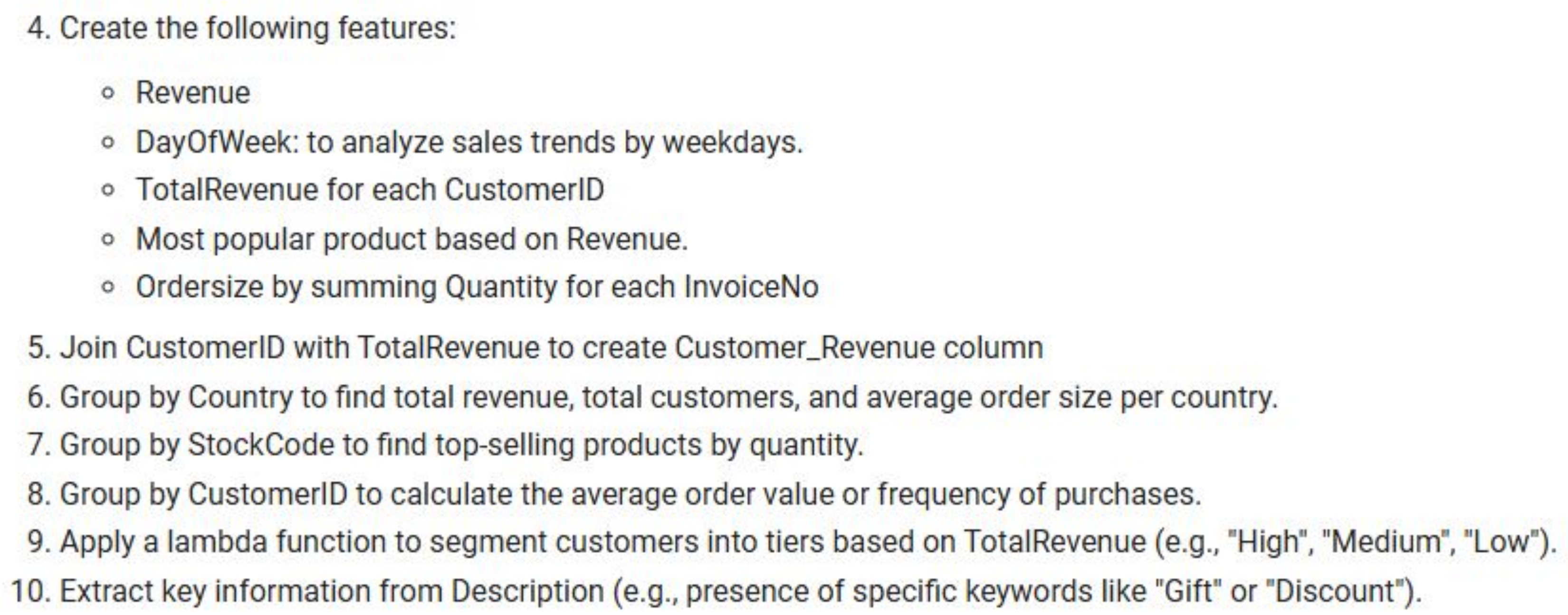
```
map(String key, String value):
  // key: document id; value: document contents
  for each word w in value:
     emit(w, 1)

reduce(String key, Iterator values):
  // key: a word; values: a list of counts
  int result = 0;

  for each v in values:

     result += v;
  emit(result);
```

Execution Framework

- MapReduce separates the *what* of distributed processing from the *how*.
- A MapReduce program (a job) consists of
    - Code for mappers and reducers
        - And combiners and partitioners to be covered shortly
    - Configuration parameters
        - E.g., where the input is and where the output should go
- The developer submits the job to the *submission node* of a cluster
    - It’s called jobtracker in Hadoop
- **Execution framework** (a.k.a. runtime) takes care of everything else *transparently*:
    - All other aspects of distributed code execution
    - On clusters ranging from a single node to thousands of nodes.

Exercise

- **Let us look at some examples**
- **Complete the following tasks from exercise notebook**:



4. Create the following features:
   - Revenue
   - DayOfWeek: to analyze sales trends by weekdays.
   - TotalRevenue for each CustomerID
   - Most popular product based on Revenue.
   - Ordersize by summing Quantity for each InvoiceNo
5. Join CustomerID with TotalRevenue to create Customer_Revenue column
6. Group by Country to find total revenue, total customers, and average order size per country.
7. Group by StockCode to find top-selling products by quantity.
8. Group by CustomerID to calculate the average order value or frequency of purchases.
9. Apply a lambda function to segment customers into tiers based on TotalRevenue (e.g., "High", "Medium", "Low").
10. Extract key information from Description (e.g., presence of specific keywords like "Gift" or "Discount").

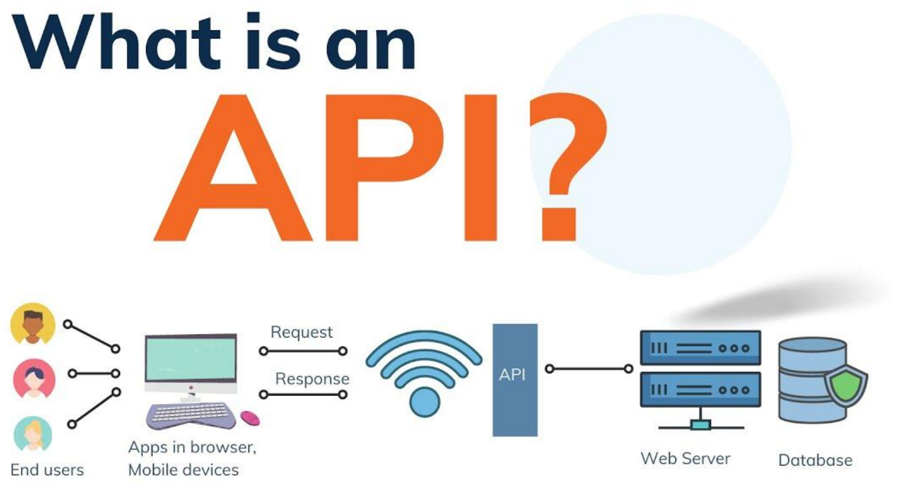
Contents

Introduction to Data Acquisition Techniques

- **Data Acquisition** refers to the process of collecting and gathering data from different sources, which can then be processed, analyzed, and used for decision-making or machine learning models. This involves several techniques and tools to capture data effectively.
- **Key Techniques:**
    - **Batch Data Collection:** Data is collected periodically and stored in bulk (e.g., hourly, daily).
    - **Real-Time Data Collection:** Data is continuously collected as events occur (e.g., IoT devices, financial transactions).
    - **Data Streaming:** Continuous data flow that can be processed on the fly (e.g., sensor data, social media feeds).
- **Common Data Sources:**
    - APIs
    - Databases (SQL/NoSQL)
    - External Data Providers
    - IoT Sensors

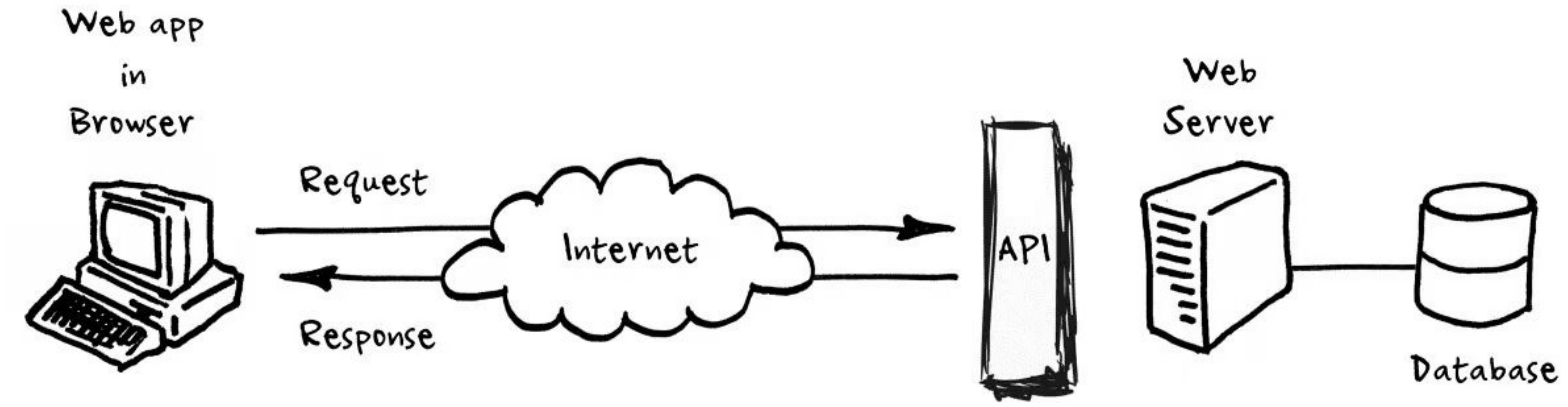
APIs (Application Programming Interfaces)

- APIs are one of the most common and efficient ways to acquire data from external sources or services. They allow applications to communicate and share data seamlessly.
- **How it works**:
  An API provides endpoints that allow you to send requests to retrieve or push data from external systems or services (e.g., weather data, financial data, social media data).
- **Example**:
  - **Twitter API**: Retrieve tweets, user data, trends, etc.
  - **Google Maps API**: Fetch geolocation data or maps for embedding in applications.



FastAPI

- **FastAPI** is a modern, high-performance, web framework for building APIs with Python.
- It is built on top of **Starlette** for the web parts and **Pydantic** for data validation, making it easy to create robust and fast web applications and APIs.
- FastAPI is known for its speed, ease of use, and automatic documentation.

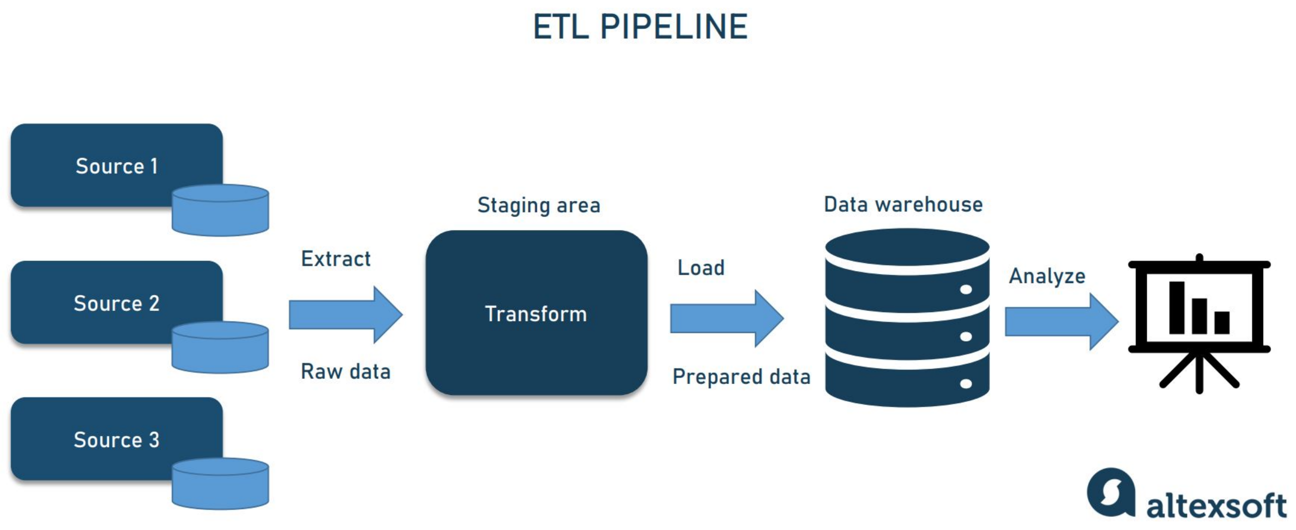


Example of FastAPI

```
from fastapi import FastAPI
from pydantic import BaseModel
import requests

app = FastAPI()

# Example: Collecting weather data via API
class WeatherRequest(BaseModel):
   city: str

@app.post("/get_weather")
def get_weather(request: WeatherRequest):
   response = 
requests.get(f"https://api.openweathermap.org/data/2.5/weather?q={request.city}&appid=YOUR_API_KEY")
   return response.json()
```

Data Pipelines – batch processing

Data pipelines automate the flow of data between systems, collecting, processing, and storing data efficiently. The pipeline can be both real time or batch, in a batch setting, we use scripts triggered at fixed frequency transfer data from one place to another



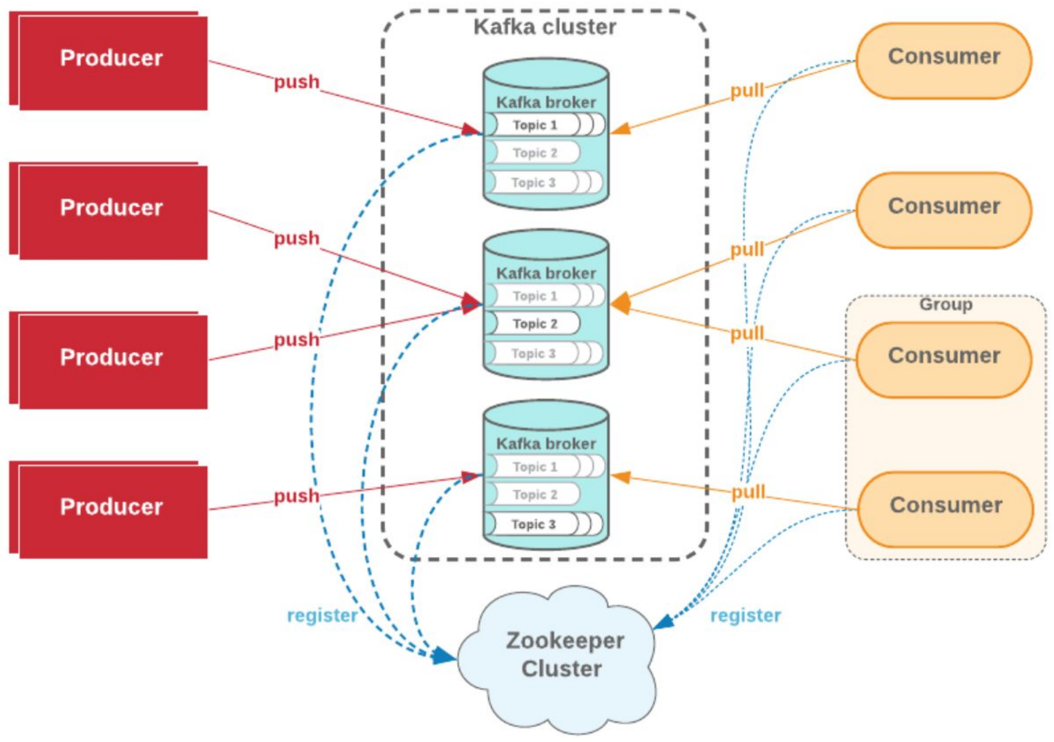
- **How it works**:
  Data is extracted from various sources (e.g., databases, APIs, files), transformed as needed, and loaded into storage systems (e.g., data warehouses, data lakes) for analysis or further processing.
- **Example**:
  - **ETL (extract transform load) Pipelines**: Extract data from a source, transform it (clean, enrich, aggregate), and load it into a destination system like a data warehouse.

Data pipelines – real time processing

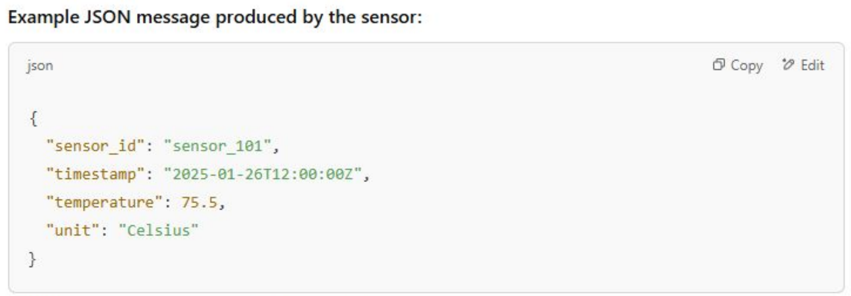
Data pipelines automate the flow of data between systems, collecting, processing, and storing data efficiently. The pipeline can be both real time or batch, in a real time setting, we use technology such as Kafka to push and pull data in real time.

1. **Producer**: a sensor pushing data to such as temperature
2. **Cluster**: Stores the data in topic in paritions
3. **Consumer**: pulls the data

Apache Kafka architecture



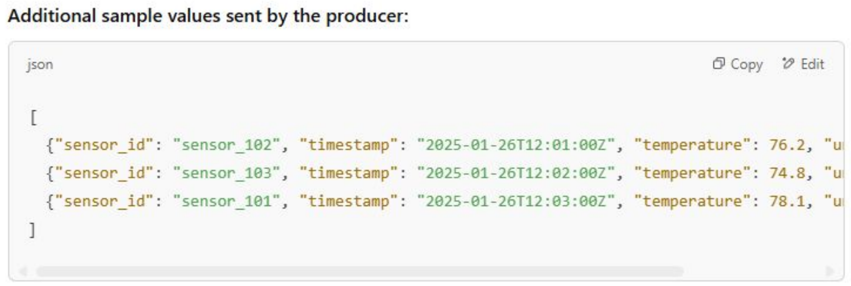
Real time processing examples



Example JSON message produced by the sensor:

json Copy Edit

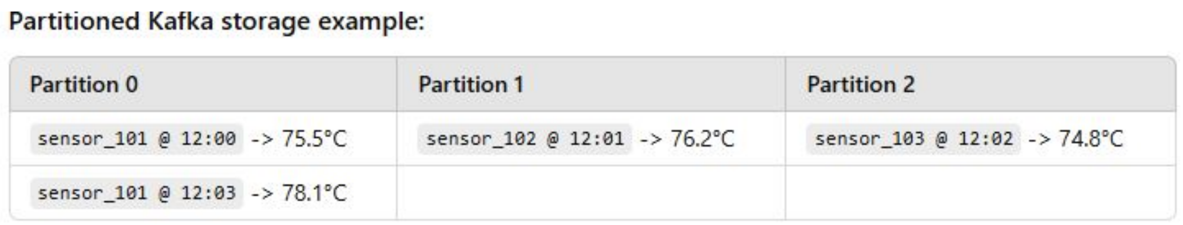
```json
{
  "sensor_id": "sensor_101",
  "timestamp": "2025-01-26T12:00:00Z",
  "temperature": 75.5,
  "unit": "Celsius"
}
```



Additional sample values sent by the producer:

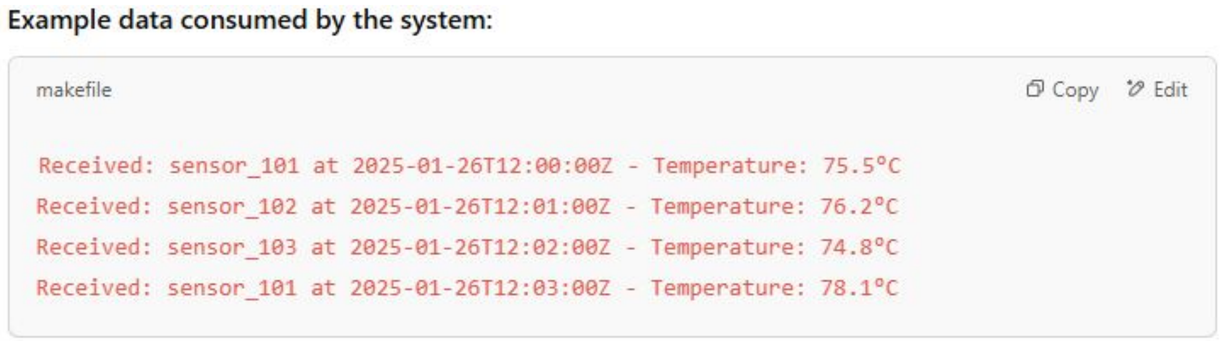
json Copy Edit

```json
[
  {"sensor_id": "sensor_102", "timestamp": "2025-01-26T12:01:00Z", "temperature": 76.2, "u
  {"sensor_id": "sensor_103", "timestamp": "2025-01-26T12:02:00Z", "temperature": 74.8, "u
  {"sensor_id": "sensor_101", "timestamp": "2025-01-26T12:03:00Z", "temperature": 78.1, "u
]
```



Partitioned Kafka storage example:

Partition 0	Partition 1	Partition 2
`sensor_101 @ 12:00` -> 75.5°C	`sensor_102 @ 12:01` -> 76.2°C	`sensor_103 @ 12:02` -> 74.8°C
`sensor_101 @ 12:03` -> 78.1°C		



Example data consumed by the system:

makefile Copy Edit

```makefile
Received: sensor_101 at 2025-01-26T12:00:00Z - Temperature: 75.5°C
Received: sensor_102 at 2025-01-26T12:01:00Z - Temperature: 76.2°C
Received: sensor_103 at 2025-01-26T12:02:00Z - Temperature: 74.8°C
Received: sensor_101 at 2025-01-26T12:03:00Z - Temperature: 78.1°C
```

Data Systems

- Data systems are computer (hardware and software) systems that store, manage and process massive, often dynamic, data accessed by data-intensive applications.
- Building blocks (unnecessary to be all available in a single system)
  - Persistent storage (databases)
  - Specification of user requests (queries)
  - Efficient data search (indexes)
  - Reuse of the results of an expensive operation (caching)
  - Sending a message to be handled asynchronously (stream processing)
  - Periodically handle a big amount of accumulated data (batch processing)

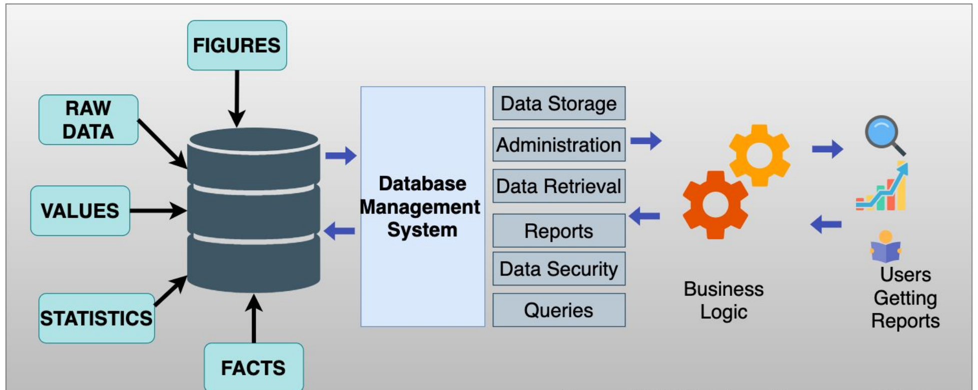


Figure: DBMS(Database Management Systems)

A Possible Data System Architecture

- The whole system may be built with different tools or available libraries
  - They are *stitched* together via APIs called in application code
- To build such data systems, you need first to know the overall *design principles*
  - **Reliability**
  - **Scalability**
  - **Maintainability**

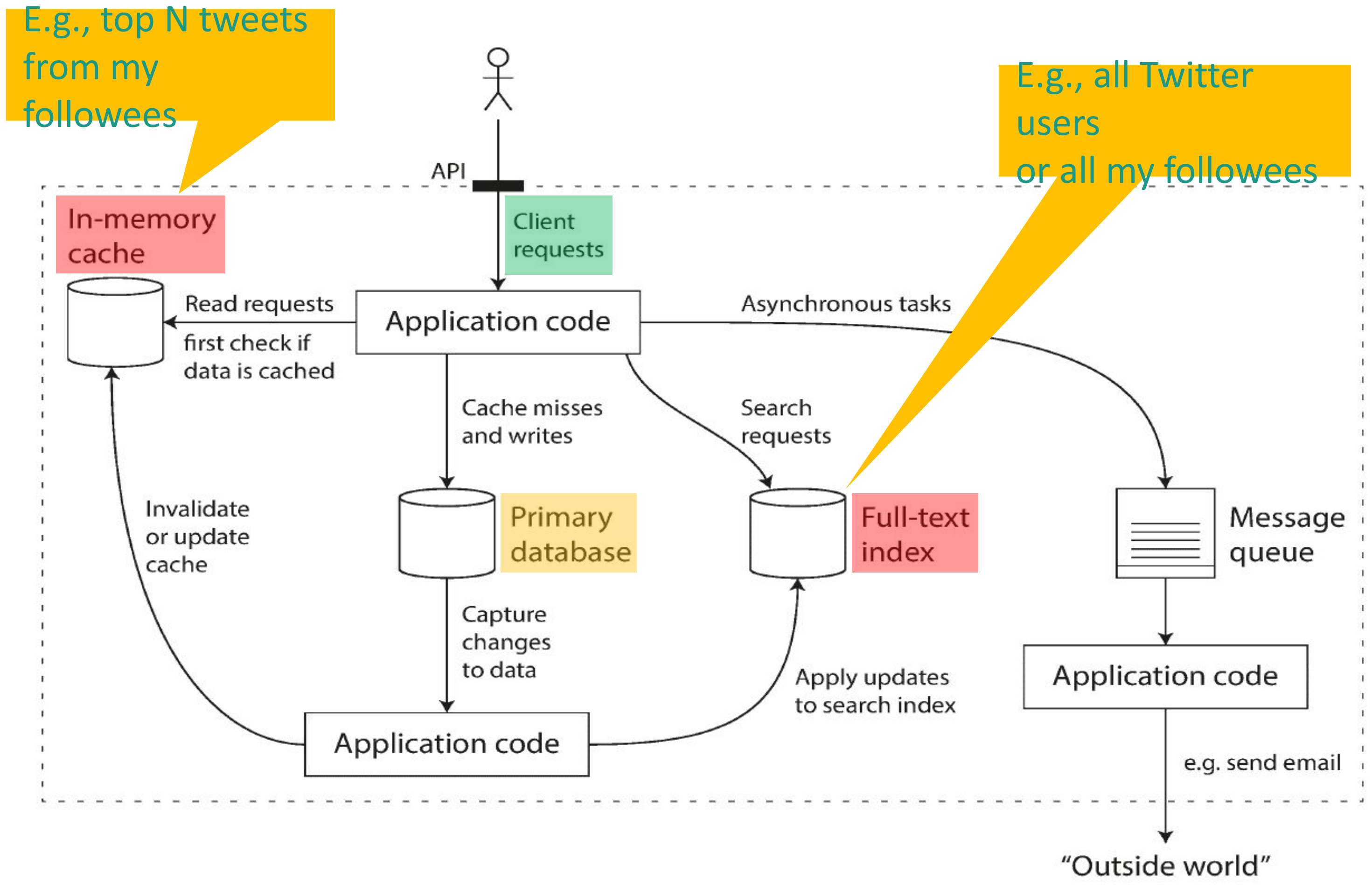
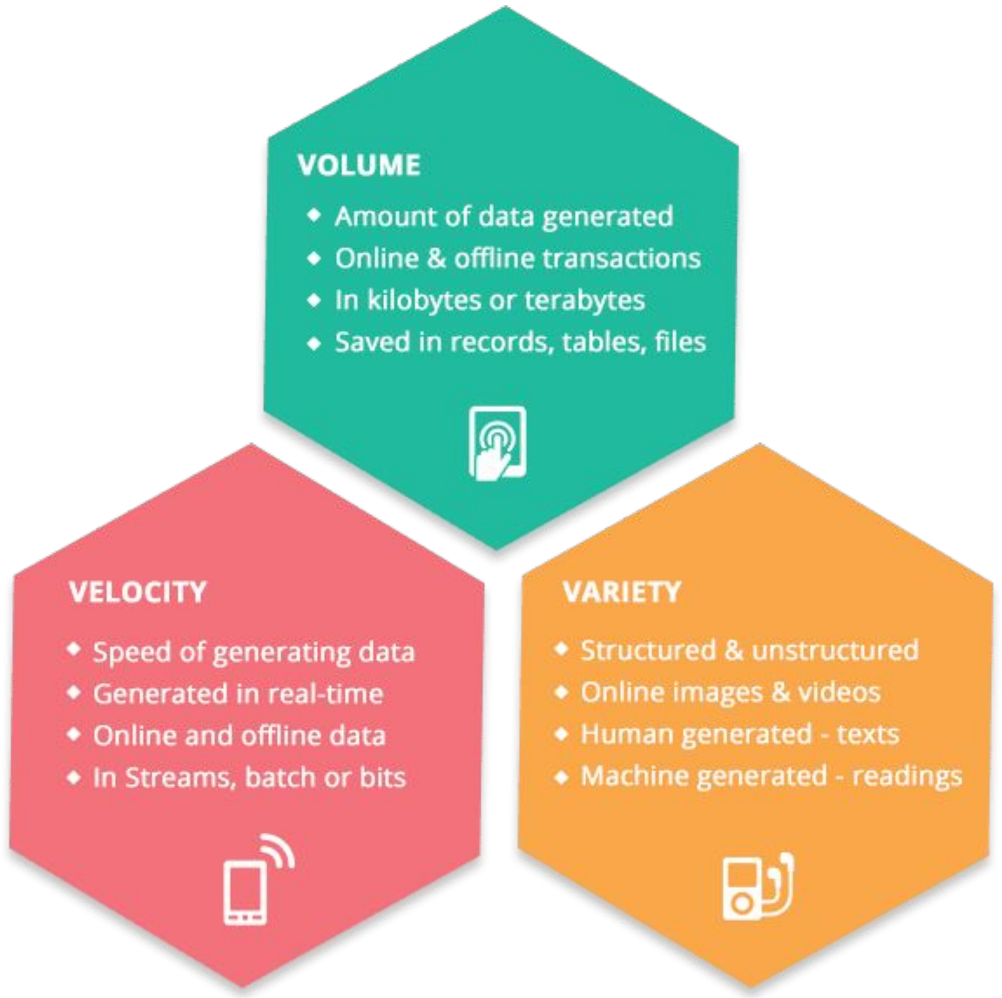


Figure from Martin Kleppmann

Contents

What is Big Data?



VOLUME
Amount of data generated
Online & offline transactions
In kilobytes or terabytes
Saved in records, tables, files
VELOCITY
Speed of generating data
Generated in real-time
Online and offline data
In Streams, batch or bits
VARIETY
Structured & unstructured
Online images & videos
Human generated - texts
Machine generated - readings

In recent years, a fourth and fifth V have been introduced:

- **Veracity**: The uncertainty of data; ensuring the data quality and reliability.
- **Value**: The usefulness of the data after processing and analysis.

Challenges of Big Data

Storage: Storing large volumes of data requires vast amounts of storage space, which can be costly.

Processing: Traditional data processing methods struggle with large-scale data, making parallelization necessary.

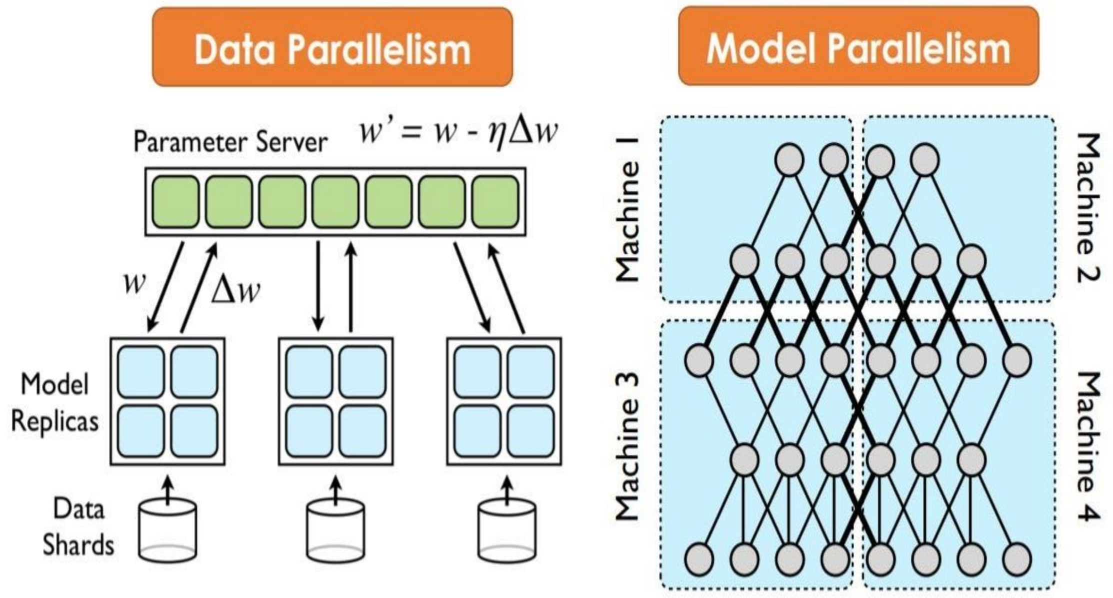
Analysis: Analyzing Big Data requires sophisticated tools and algorithms, often utilizing machine learning and AI.

Data Integration: Data often comes from multiple sources, and integrating these diverse data types can be difficult.

The Role of Parallelization in Big Data

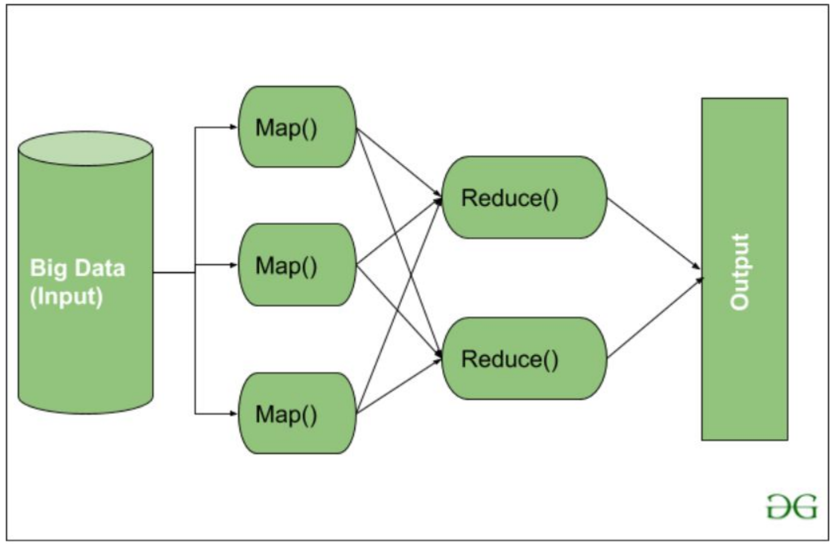
- Parallelization refers to dividing a large task into smaller tasks that can be processed simultaneously. This is crucial for handling the massive computational load that Big Data presents.

- **Key Concepts of Parallelization:**
    - **Task Parallelism**: Different tasks are executed simultaneously on multiple processors. Each processor performs a different operation on the data.
    - **Data Parallelism**: The same task is executed on different parts of the dataset. Each processor works on a subset of the data (e.g., partitioning a large dataset into smaller chunks).



Apache Hadoop

- **Hadoop** is an open-source framework that enables distributed storage and processing of large datasets across clusters of computers. It uses the **MapReduce** paradigm to process data in parallel.
- **HDFS (Hadoop Distributed File System)**: A distributed storage system that splits data into blocks and stores them across different nodes.
- **MapReduce**: A programming model for processing large data sets with a distributed algorithm.



Summary

1. Introduction & background
    a. Background of Data Engineering - challenges & relevance
    b. The data lifecycle & a data engineers role in it
2. Data store & management - focus on databases
    a. Intro to databases - onprem, cloud & hybrid
    b. SQL, NoSQL, Graph, Vector & other databases with some notebook tasks
    c. Data lake & data warehouse with intro to ADLS
    d. Exercise: load data, store in pandas dataframe, visualize
3. Data Transformation -
    a. Feature engineering with focus on Lambda functions & 'Apply'
    b. Exercise: Lambda function with Apply
4. Data acquisition & system overview
    a. Intro to data acquisition techniques
    b. APIs & Real time vs batch processing
    c. Overview of a data system with various pipelines
5. Big data & parallelization