Introduction to Data Engineering: Setting the Stage for Data Science

Chapter 2

Michael Tsiang

Stats 167: Introduction to Databases

UCLA

Purpose and Pipelines

Prerequisite: How Data Engineering Works

First watch the video “How Data Engineering Works” by AltexSoft:

https://youtu.be/qWru-b6m030

For those who need transcripts, click here.

This video covers a clear, concise, and well motivated overview of data engineering.

After watching the video, we want to summarize the main takeaways and expand on some of the concepts mentioned.

The Purpose of Data Engineering

From the AltexSoft video:

“The sole purpose of data engineering is to take data from the source and save it to make it available for analysis.”

To efficiently take data from the source and save it into storage, data engineers build infrastructure and automate data processing.

Data Pipelines

Data engineers and data scientists use data pipelines to move data from a source to a destination. So what is a data pipeline?

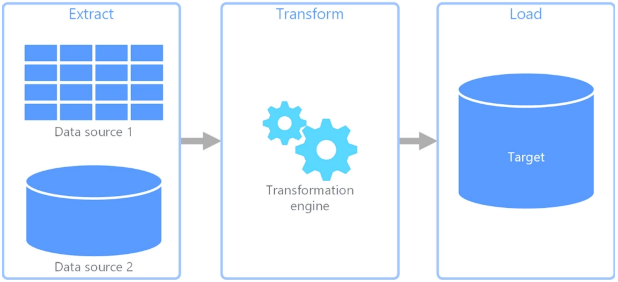
A **data pipeline** is a series of data processing steps.

Popular tools that data engineers use to manage and schedule (i.e., automate) data pipelines are Apache Airflow and Luigi.

https://airflow.apache.org/
https://luigi.readthedocs.io/en/stable/index.html

ETL Pipelines

A common type of data pipeline is an **ETL** pipeline, shown below[1]:



1. **Extract** the data from data sources.
2. **Transform** the data into a usable and clean format.
3. **Load** the data into a storage location (e.g., a database or data warehouse).

[1]Source: https://estuary.dev/what-is-an-etl-pipeline/

Data Structures

The Four Vs of Big Data

Four key properties of big data are **volume**, **variety**, **veracity**, and **velocity**.

- **Volume** refers to the massive amount of data that is too large to be stored or analyzed on a single computer.
- **Variety** refers to the varied sources and types of data.
- **Veracity** refers to the quality of the data. Is the data accurate and meaningful?
- **Velocity** refers to the speed at which data is generated.

Examples of Big Data

Some examples[2] of big data:

- The New York Stock Exchange generates roughly one terabyte of new trade data per day.
- More than 500 terabytes of data is generated on Facebook every day (e.g., posts, comments, photo/video uploads, messages on Facebook Messenger). In 2016, users were uploading about 900 million photos per day[3].
- A single jet engine can generate over 10 terabytes of data in 30 minutes of flight time.

[2] Source: https://ijcat.com/archieve/volume9/issue3/ijcatr09031002.pdf
[3] Source: https://datacenterfrontier.com/inside-facebooks-blu-ray-cold-storage-data-center/

Databases

To handle big data storage, we need to generalize our idea of what datasets look like to understand how we can store, access, and understand larger amounts of data.

A **database** is an organized collection of data.

Intuitively, a database is a collection of data tables (though not always). Since the database may be extremely large and the structure may be complicated, we use a **database management system (DBMS)** to organize and access the data efficiently.

Relational Databases

There are many different types of databases, depending on the type of data and its purpose.

One of the most popular types is the **relational database**, in which observations from separate tables are *related* to each other based on a common unique identifier (called a **key**).

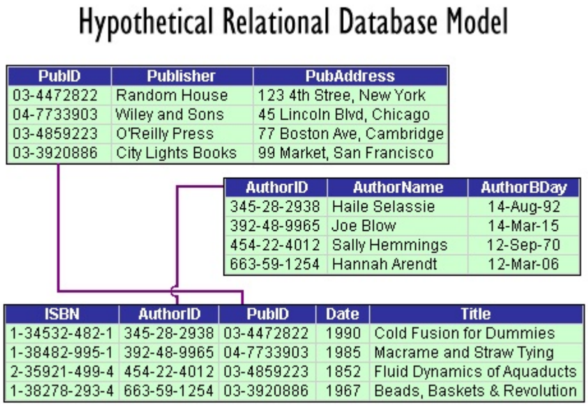
Relational databases use **structured query language (SQL)** for writing and querying data, which is why relational databases are also called **SQL databases**.

Examples of relational database management systems (RDBMS) are SQL Server, MySQL, and PostgreSQL. SQL Server is developed by Microsoft, while MySQL and PostgreSQL are open source. As their names suggest, SQL is the main language used to interact with relational databases through the RDBMS.

Side Note: SQL is sometimes pronounced as the letters "S-Q-L" or as "sequel."

Example of a Relational Database

An example of a relational database is shown below.



PubID	Publisher	PubAddress
03-4472822	Random House	123 4th Stree, New York
04-7733903	Wiley and Sons	45 Lincoln Blvd, Chicago
03-4859223	O'Reilly Press	77 Boston Ave, Cambridge
03-3920886	City Lights Books	99 Market, San Francisco

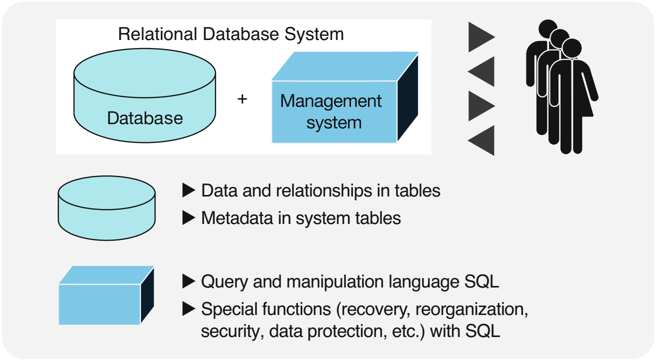
AuthorID	AuthorName	AuthorBDay
345-28-2938	Haile Selassie	14-Aug-92
392-48-9965	Joe Blow	14-Mar-15
454-22-4012	Sally Hemmings	12-Sep-70
663-59-1254	Hannah Arendt	12-Mar-06

ISBN	AuthorID	PubID	Date	Title
1-34532-482-1	345-28-2938	03-4472822	1990	Cold Fusion for Dummies
1-38482-995-1	392-48-9965	04-7733903	1985	Macrame and Straw Tying
2-35921-499-4	454-22-4012	03-4859223	1852	Fluid Dynamics of Aquaducts
1-38278-293-4	663-59-1254	03-3920886	1967	Beads, Baskets & Revolution

Source: https://gnosis.cx/publish/programming/xml_matters_8.html

Schematic of a Relational Database Management System

The basic structure of an RDBMS is shown below.


Relational Database System
Database
+
Management system
▶ Data and relationships in tables
▶ Metadata in system tables
▶ Query and manipulation language SQL
▶ Special functions (recovery, reorganization, security, data protection, etc.) with SQL

Source: Figure 1.6 in SQL and NoSQL Databases by Kaufmann and Meier, Second Edition, Springer, 2023.

Non-Relational Databases

Non-relational databases, also called **NoSQL** databases, are databases that do not store data in relational tables.

Note: NoSQL stands for “not only SQL,” not “no SQL.”

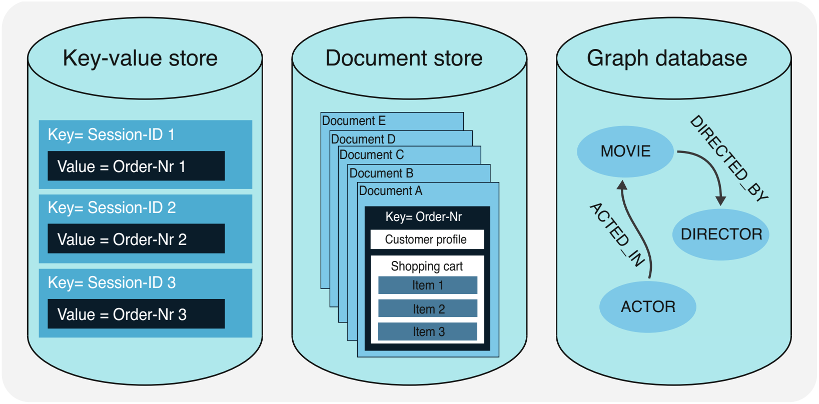
NoSQL databases cover a wide range of database types, the most popular of which include:

- Document databases
- Key-value databases (also called key-value stores)
- Wide-column databases
- Graph databases
- Vector databases

Examples of NoSQL database management systems are MongoDB, Redis, Cassandra, and Pinecone.

Examples of NoSQL Databases

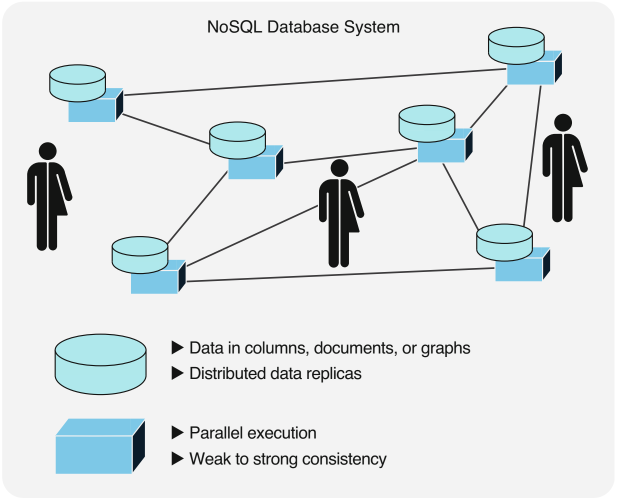
Some examples of NoSQL databases are shown below.



Source: Figure 1.9 in SQL and NoSQL Databases by Kaufmann and Meier, Second Edition, Springer, 2023.

Schematic of a Non-Relational DBMS

The basic structure of a NoSQL DBMS is shown below.


NoSQL Database System
▶ Data in columns, documents, or graphs
▶ Distributed data replicas
▶ Parallel execution
▶ Weak to strong consistency

Source: Figure 1.8 in SQL and NoSQL Databases by Kaufmann and Meier, Second Edition, Springer, 2023.

Transactional Databases

Many relational database systems are called **transactional** because they are optimized for simple **transactions**, such as queries, insertions, updates, or deletions of information in the database.

A standard data processing system for many scenarios is the **OLTP (online transaction processing)** system, which uses a relational database to process a large number of simple database transactions (usually online).

A common example is an ATM (automated teller machine), which needs to process a huge number of transactions (e.g., deposit, withdraw, or query) to many accounts while keeping the exact amounts in each account correct.

Oh CRUD!

OLTP systems are optimized for transactions that involve **CRUD** operations. CRUD stands for:

- **Create**: Adding new data to the database.
  - Example: Creating a new user account on a social media site.
- **Read**: Retrieving existing data from the database.
  - Example: Viewing a user profile.
- **Update**: Modifying existing data in the database.
  - Example: Changing a user's contact information.
- **Delete**: Removing data from the database.
  - Example: Deleting a user account.

CRUD operations are relatively small and simple, so OLTP systems are designed to process a high volume of these types of transactions quickly and efficiently.

OLTP Versus OLAP

Because the relational database used in the OLTP system is organized to be efficient in storing data in tables, it is not optimized for more complex data analytics.

In contrast to OLTP, an **OLAP (online analytical processing)** system is designed to optimize processing for data analysis and business intelligence applications.

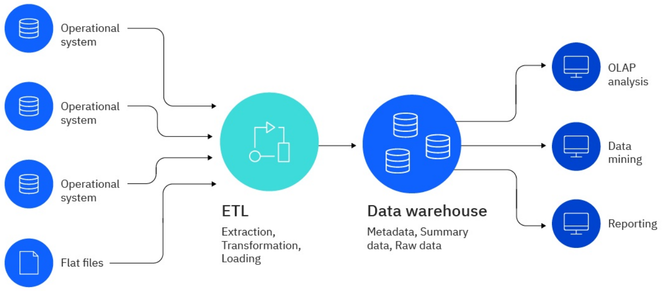
For more on the differences and use cases of OLTP and OLAP systems:

- https://www.sprinkledata.com/blogs/oltp-vs-olap
- https://www.guru99.com/what-is-oltp.html
- https://stackoverflow.com/questions/21900185/what-are-oltp-and-olap-what-is-the-difference-between-them
- https://www.analyticsvidhya.com/blog/2020/11/oltp-vs-olap/
- https://www.fivetran.com/blog/oltp-vs-olap-what-is-the-difference

Data Warehouses

When analytics are prioritized, data is typically stored in a data warehouse rather than a relational database.

A **data warehouse** aggregates data from different sources into a single centralized location, stored for efficiency in data analysis.



Source: https://www.ibm.com/think/topics/data-warehouse

Structured and Unstructured Data

Databases and data warehouses typically store **structured** data, i.e., data that follows a consistent and organized format. This structure makes the data easily organized and searchable. Most structured data is stored in relational databases (i.e., data tables).

Some data is considered **semi-structured**, which has some consistent organization but is not necessarily stored in tabular form, e.g., JSON or XML data.

However, a lot of (in fact most) data is now **unstructured**, which does not always follow a well organized format, e.g., PDFs, images, or audio/video files.

How does one store unstructured data?

References for Structured vs Unstructured Data

For some comparisons and examples:

- https://www.datamation.com/big-data/structured-vs-unstructured-data/
- https://www.serverwatch.com/storage/structured-vs-unstructured-data/
- https://www.geeksforgeeks.org/difference-between-structured-semi-structured-and-unstructured-data/

Data Lakes

A **data lake** is a repository for storing all structured, semi-structured, and unstructured data in its raw form.

When data needs to be used or analyzed, a data engineer or data scientist often use an **ELT (Extract, Load, and Transform)** pipeline (rather than ETL) to efficiently access data from a data lake.

For more comparisons and uses cases of databases, data warehouses, and data lakes:

- https://www.confluent.io/learn/databases-data-lakes-and-data-warehouses-compared/
- https://panoply.io/data-warehouse-guide/the-difference-between-a-database-and-a-data-warehouse/
- https://www.guru99.com/data-lake-vs-data-warehouse.html

Data Lakehouses

A **data lakehouse** is the (currently) most modern data management paradigm that tries to unify the benefits data lakes and data warehouses.

The excellent video “Data Storage for Analytics and Machine Learning” by AltexSoft covers comparisons, use cases, and connections between databases, data warehouses, data lakes, *and* data lakehouses:

https://youtu.be/zASIPr4FwGM

More information:

- https://cloud.google.com/discover/what-is-a-data-lakehouse
- https://www.databricks.com/glossary/data-lakehouse
- https://www.montecarlodata.com/blog-data-warehouse-vs-data-lake-vs-data-lakehouse-definitions-similarities-and-differences/

Data Streaming

Batch vs Streaming Data

Batch data is data that is retrieved or generated at specific times, usually at regular time intervals (every hour, week, month, etc.).

Streaming data is data that is retrieved or generated constantly.

Some examples of batch vs streaming data processing:

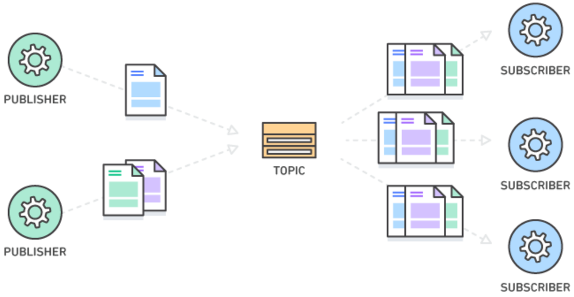
https://www.precisely.com/blog/big-data/big-data-101-batch-stream-processing

For more information on data streaming:

- https://youtu.be/ag9jIVxM_18

Pub/Sub

Pub/Sub, or **Publish/Subscribe** messaging, is a system of asynchronous communication between systems that generate a lot of data simultaneously.



Source: https://aws.amazon.com/pub-sub-messaging/

Pub/Sub Resources

For more information on Pub/Sub messaging:

- https://youtu.be/jLI-84UjZLE
- https://cloud.google.com/pubsub/docs/overview

Apache Kafka

A popular Pub/Sub tool for streaming data processing is Apache Kafka.

More about Apache Kafka:

- https://kafka.apache.org/
- https://www.tibco.com/glossary/what-is-apache-kafka

Distributed Computing

Distributed Computing

When data is too large to be stored on a single computer, we use **distributed storage**: The data is distributed across several computers, collectively called a **cluster**. A **distributed computer system** allows multiple components to function as a single system.

Two benefits of a distributed computer system are:

- **Scalability**: Increasing storage can be done by adding computers (also called **nodes**) to the cluster.
- **Redundancy**: Components in a distributed system are duplicated so that the whole system will still be available and reliable even if one (or more) nodes is unavailable.

Hadoop

A long-standing popular framework for distributed storage of big data is **Hadoop**.

https://hadoop.apache.org/

The three components of the Hadoop framework are:

- The **Hadoop Distributed File System (HDFS)**: How Hadoop distributes storage and incorporates scalability and redundancy.
- **MapReduce**: Hadoop's data processing method which splits tasks into a Map phase and a Reduce phase.
- **YARN (Yet Another Resource Negotiator)**: Hadoop's resource management and job scheduling system.

Hadoop and MapReduce Resources

For more about Hadoop:

- A great overview from Simplilearn about how Hadoop stores and processes data: https://youtu.be/aReuLtY0YMI
- A clear and concise explanation from Computerphile of how MapReduce works: https://youtu.be/cvhKoniK5Uo

Apache Spark and Databricks

In more recent years, Hadoop and MapReduce (though still very widely used) have lost favor to **Apache Spark**, a newer framework for large-scale distributed computing.

Databricks is a web-based platform to run Apache Spark in the cloud with automatic cluster management and an interactive notebook environment (similar to Jupyter and RStudio notebooks).

https://spark.apache.org/

https://www.databricks.com/