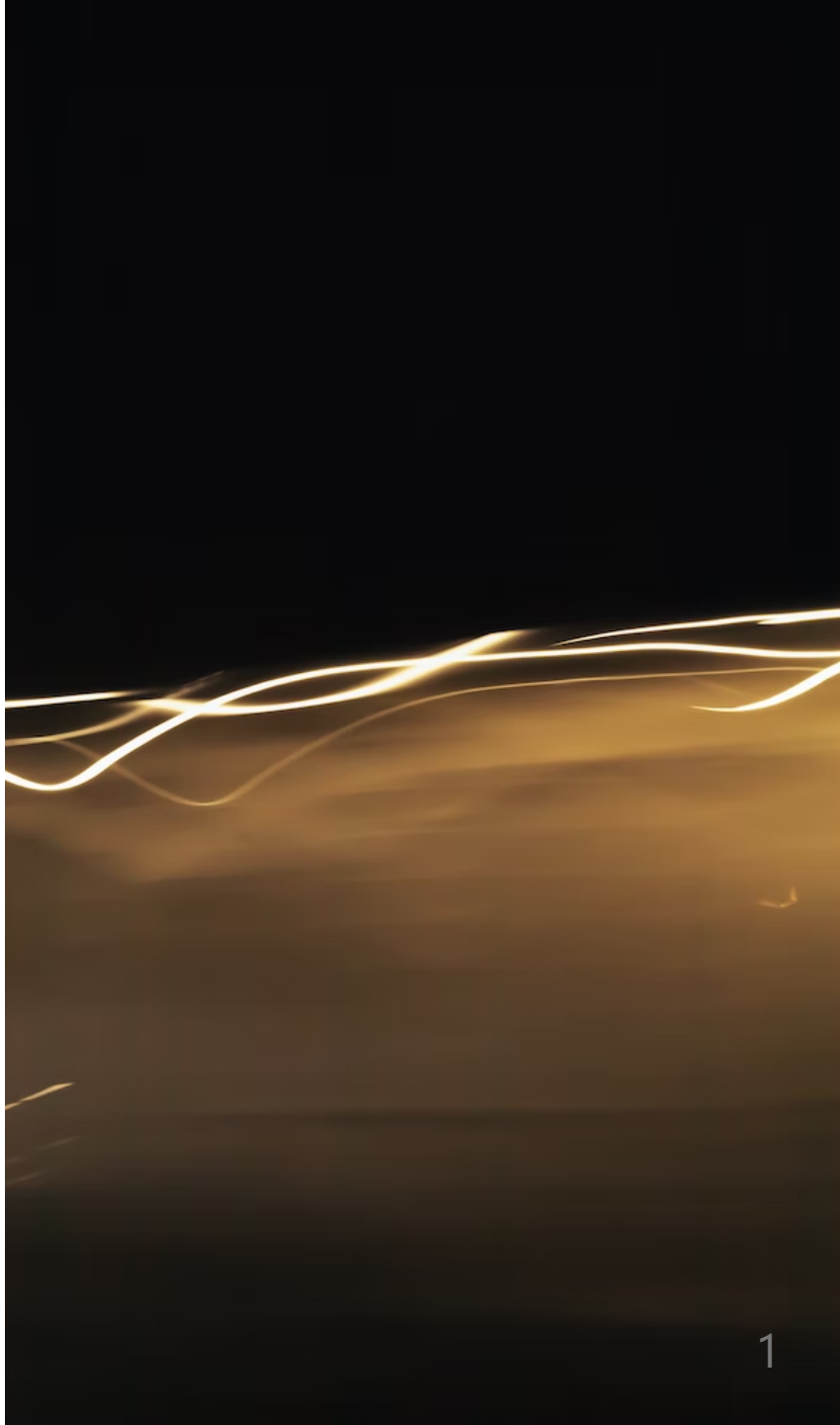
Introduction to Deep Learning

Textbook recap

Student group:

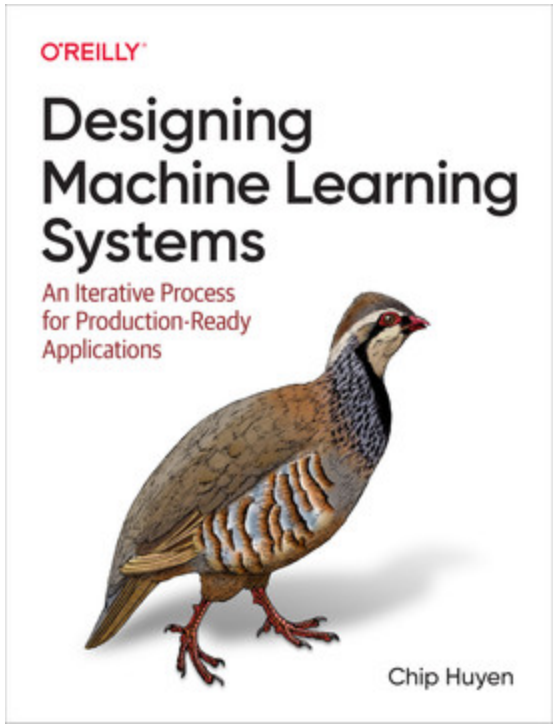
- 520K0127 - Do Pham Quang Hung
- 520K0343 - Le Phuoc Thinh



CHAPTER 3: Data Engineering Fundamentals

Designing Machine Learning Systems

An Iterative Process for Production-Ready Applications



Outlines

1. Data Sources
2. Data Formats
3. Data Models
4. Data Storage Engines and Processing
5. Modes of Dataflow
6. Batch Processing Versus Stream Processing
7. Summary

1. Data Sources

An ML system can work with data from many different sources. They have different characteristics, can be used for different purposes, and require different processing methods. Understanding the sources your data comes from can help you use your data more efficiently.

1. Data Sources [1]

- **user input data**: data explicitly input by users. 👩‍🦰👱‍♂️✍️

> User input can be text, images, videos, uploaded files, etc. But if it's even remotely possible for users to input wrong data, they are going to do it.

> User input data can be ***easily malformatted***.

> User input data requires more ***heavy-duty*** checking and processing.

> *** User input data tends to require ***fast*** processing. 🥲

1. Data Sources [2]

- **system-generated data**: This is the data generated by different components of your systems 💻, which include various types of logs and system outputs such as model predictions.

  > Logs 📝 can record the state and significant events of the system, such as memory usage, number of instances, services called, packages used, etc.

  > these data are **less likely to be malformatted** the way user input data is. Overall, logs don't need to be processed as soon as they arrive.

1. Data Sources [3]

- There are also **internal databases**: generated by various services and enterprise applications in a company.

> These databases manage their assets such as inventory, customer relationship, users, and more. This kind of data can be used by ML models directly or by various components of an ML system

> ***First-party data*** is the data that your company already collects about your users or customers. ***Second-party data*** is the data collected by another company on their own customers that they make available to you, though you'll probably have to pay for it. ***Third-party data*** companies collect data on the public who aren't their direct customers.

2. Data Formats

Since your data comes from multiple sources with different access patterns, storing your data isn't always straightforward and, for some cases, can be costly. It's important to think about how the data will be used in the future so that the format you use will make sense.

2. Data Formats [1]

Format	Binary/Text	Human-readable	Example use cases
JSON	Text	Yes	Everywhere
CSV	Text	Yes	Everywhere
Parquet	*Binary*	*No*	*Hadoop, Amazon Redshift*
Avro	Binary primary	No	Hadoop
Protobuf	Binary primary	No	Google
Pickle	Binary	No	Python

2. Data Formats [2]

JSON (JavaScript Object Notation)

```
{
    "firstName": "Boatie",
    "lastName": "McBoatFace",
    "isVibing": true,
    "age": 12,
    "address": {
        "streetAddress": "12 Ocean Drive",
        "city": "Port Royal",
        "postalCode": "10021-3100"
    }
}
```

The same data can also be stored in an unstructured blob of text like the following:

```
{
    "text": "Boatie McBoatFace, aged 12, is vibing, at 12 Ocean Drive, Port Royal, 10021-3100"
}
```

2. Data Formats [3]

JSON (JavaScript Object Notation)

JSON is ***ubiquitous***, the pain it causes can also be felt everywhere. Once you've committed the data in your JSON files to a schema, it's pretty painful to retrospectively go back to change the schema. JSON files are text files, which means they take up a lot of space, as we'll see in the section "Text Versus Binary Format"

2. Data Formats [4]

Row-Major Versus Column-Major Format

The two formats that are common and represent two distinct paradigms are **CSV** and **Parquet**.

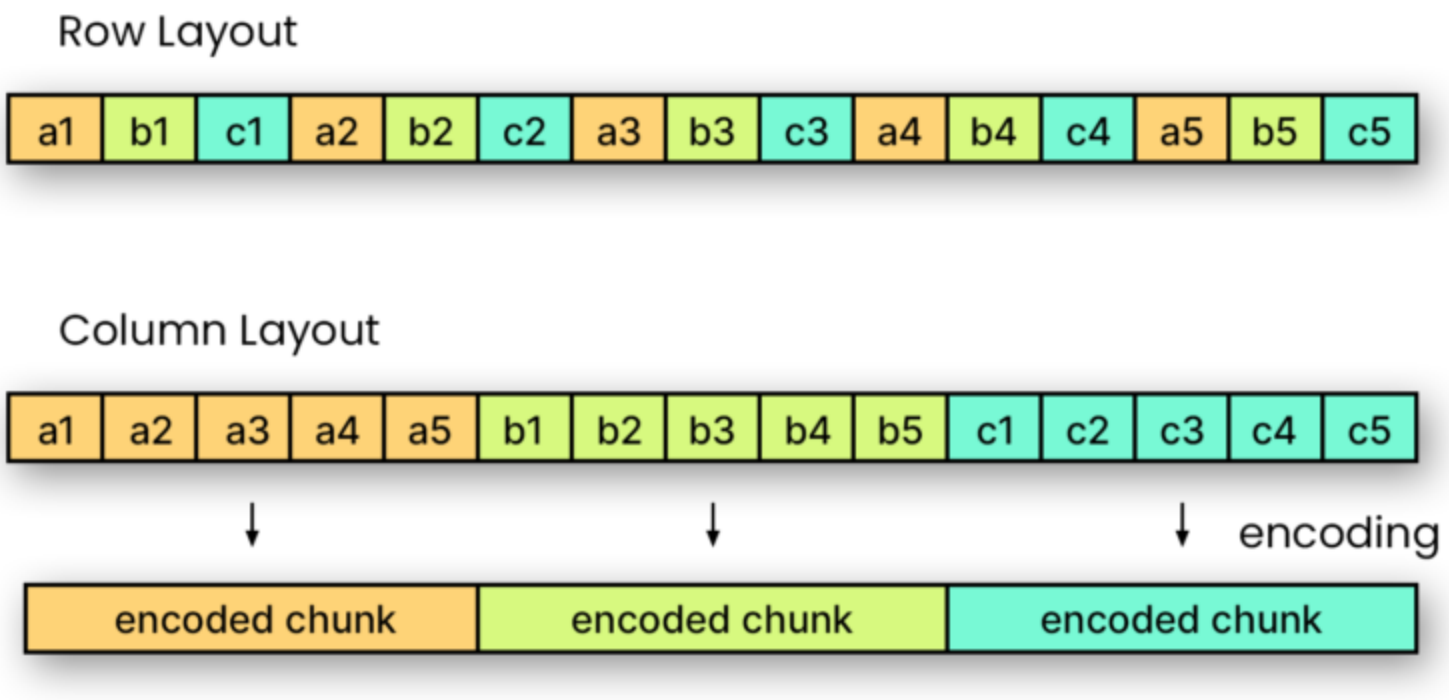
- **CSV (comma-separated values)** is row-major, which means consecutive elements in a row are stored next to each other in memory.
- **Parquet** is column-major, which means consecutive elements in a column are stored next to each other.

2. Data Formats [5]

Row-Major Versus Column-Major Format

Logical table representation

a	b	c
a1	b1	c1
a2	b2	c2
a3	b3	c3
a4	b4	c4
a5	b5	c5



2. Data Formats [6]

Row-major:

- Data is stored and retrieved row by row
- Good for accessing samples

Column-major:

- Data is stored and retrieved column by column
- Good for accessing features

	Column 1	Column 2	Column 3
Example 1	...	...	...
Example 2	...	...	...
Example 3	...	...	...

2. Data Formats [7]

Column-major formats allow ***flexible column-based reads***, especially if your data is large with thousands, if not millions, of features

Row-major formats allow faster data writes.

> Overall, row-major formats are better when you have to do a lot of writes, whereas column-major ones are better when you have to do a lot of column-based reads.

2. Data Formats [8]

```
# Iterating pandas DataFrame by column
start = time.time()
for col in df.columns:
    for item in df[col]:
        pass
print(time.time() - start, "seconds")
```

```
0.06656503677368164 seconds
```

```
# Iterating pandas DataFrame by row
n_rows = len(df)
start = time.time()
for i in range(n_rows):
    for item in df.iloc[i]:
        pass
print(time.time() - start, "seconds")
```

```
2.4123919010162354 seconds
```

```
df_np = df.to_numpy()
n_rows, n_cols = df_np.shape
```

```
# Iterating NumPy ndarray by column
start = time.time()
for j in range(n_cols):
    for item in df_np[:, j]:
        pass
print(time.time() - start, "seconds")
```

```
0.005830049514770508 seconds
```

```
# Iterating NumPy ndarray by row
start = time.time()
for i in range(n_rows):
    for item in df_np[i]:
        pass
print(time.time() - start, "seconds")
```

```
0.019572019577026367 seconds
```

(Left) Iterating a pandas DataFrame by column takes 0.07 seconds but iterating the same DataFrame by row takes 2.41 seconds. (Right) When you convert the same DataFrame into a NumPy ndarray, accessing its rows becomes much faster.

2. Data Formats [9]

Text Versus Binary Format

- `CSV` and `JSON` are *text files*, whereas `Parquet` files are *binary files*
- Binary files are more compact
- AWS recommends using the Parquet format because “the Parquet format is up to **2x faster to unload and consumes up to 6x less storage** in Amazon S3, compared to text formats.”

3. Data Models

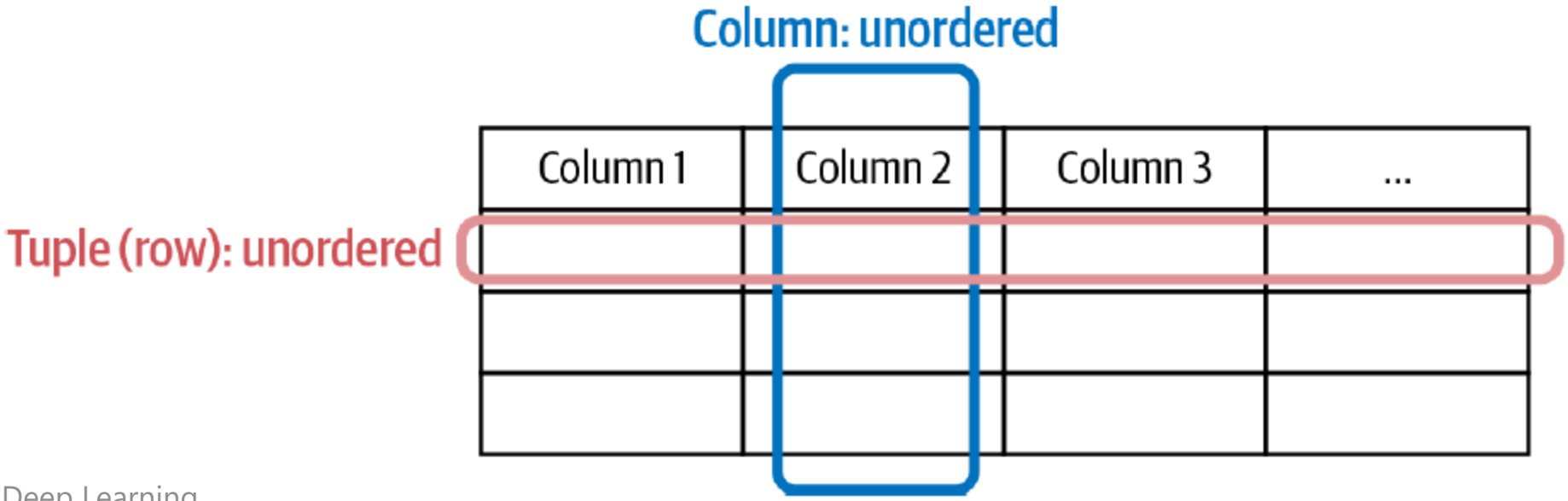
Data models describe **how data is represented**. Consider cars in the real world. In a database, a car can be described using its make, its model, its year, its color, and its price

3. Data Models [1]

Relational Model

Relational models are among the most persistent ideas in computer science. Invented by Edgar F. Codd in 1970,9 the relational model is still going strong today, even getting more popular.

> data is organized into relations; each relation is a set of tuples. A table is an accepted visual representation of a relation, and each row of a table makes up a tuple



3. Data Models [2]

Relational Model

It’s often desirable for relations to be normalized. Data normalization can follow normal forms such as the first normal form (1NF), second normal form (2NF), etc.,

3. Data Models [x1]

No SQL

3. Data Models [x2]

Structured Versus Unstructured Data

4. Data Storage Engines and Processing

Data formats and data models specify the interface for how users can store and retrieve data. Storage engines, also known as databases, are the implementation of how data is stored and retrieved on machines. It’s useful to understand different types of databases as your team or your adjacent team might need to select a database appropriate for your application.

4. Data Storage Engines and Processing [1]

Typically, there are two types of workloads that databases are optimized for, **transactional processing** and **analytical processing**, and there's a big difference between them, which we'll cover in this section. We will then cover the basics of the **ETL** *(extract, transform, load)* process that you will inevitably encounter when building an ML system in production.

4. Data Storage Engines and Processing [2]

Transactional and Analytical Processing

> **Traditionally**, a transaction refers to the action of buying or selling something.

> In the **digital world**, a transaction refers to any kind of action: tweeting, ordering a ride through a ride-sharing service, uploading a new model, watching a YouTube video, and so on

4. Data Storage Engines and Processing [3]

Transactional and Analytical Processing

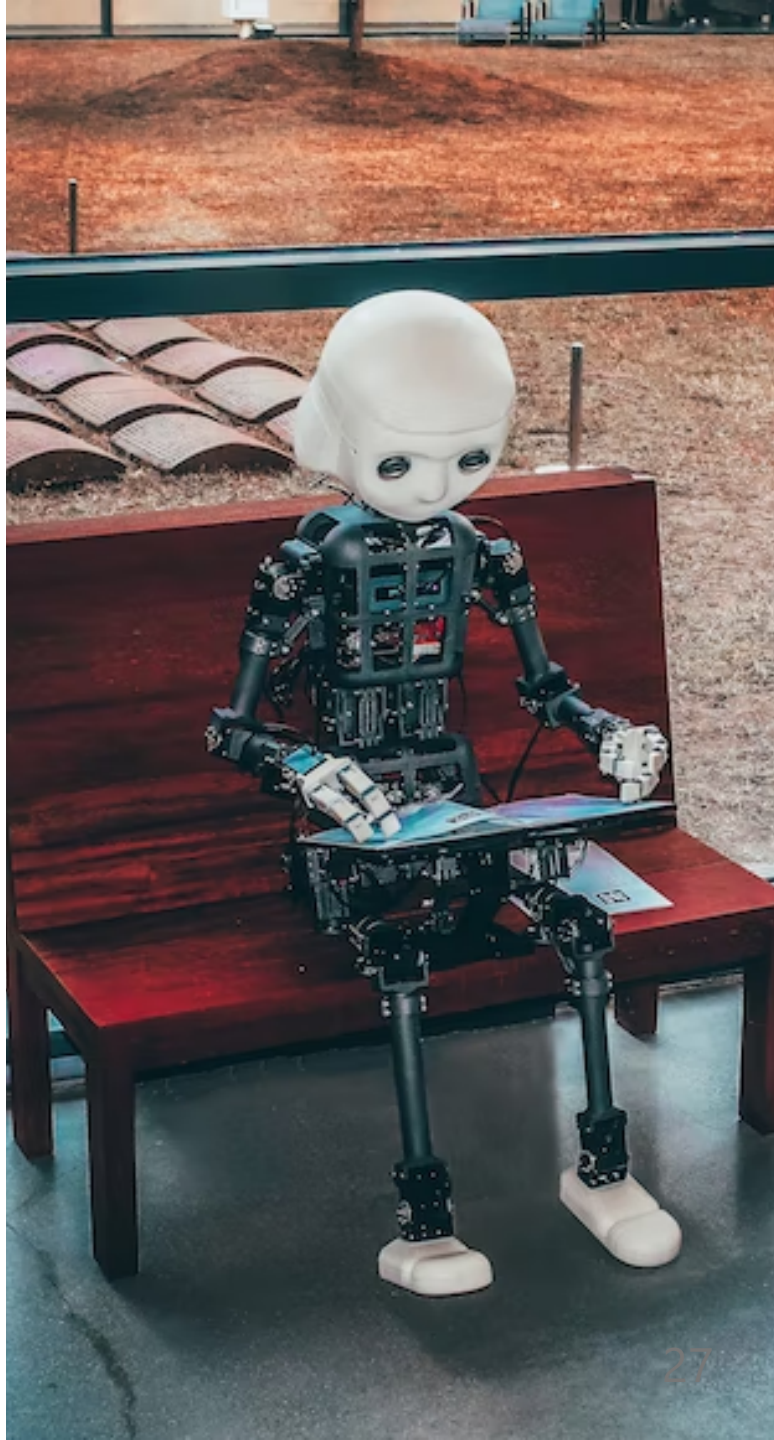
- The transactions are inserted as they are generated, and occasionally updated when something changes, or deleted when they are no longer needed. This type of processing is known as *online transaction processing* (OLTP).

> These transactions need to be processed **fast (low latency)** and the processing method needs to have **high availability**. Besides, if your system **can't process** a transaction, that transaction **won't go through**.

4. Data Storage Engines and Processing [4]

Transactional and Analytical Processing

Transactional databases are designed to process online transactions and satisfy the low latency, high availability requirements. When people hear transactional databases, they usually think of ACID (atomicity, consistency, isolation, durability).



4. Data Storage Engines and Processing [4]

Transactional and Analytical Processing [ACID]

Atomicity

> To guarantee that all the steps in a transaction are completed successfully as a group. If any step in the transaction fails, all other steps must fail also. For example, if a user's payment fails, you don't want to still assign a driver to that user.

Consistency

> To guarantee that all the transactions coming through must follow predefined rules. For example, a transaction must be made by a valid user.

4. Data Storage Engines and Processing [5]

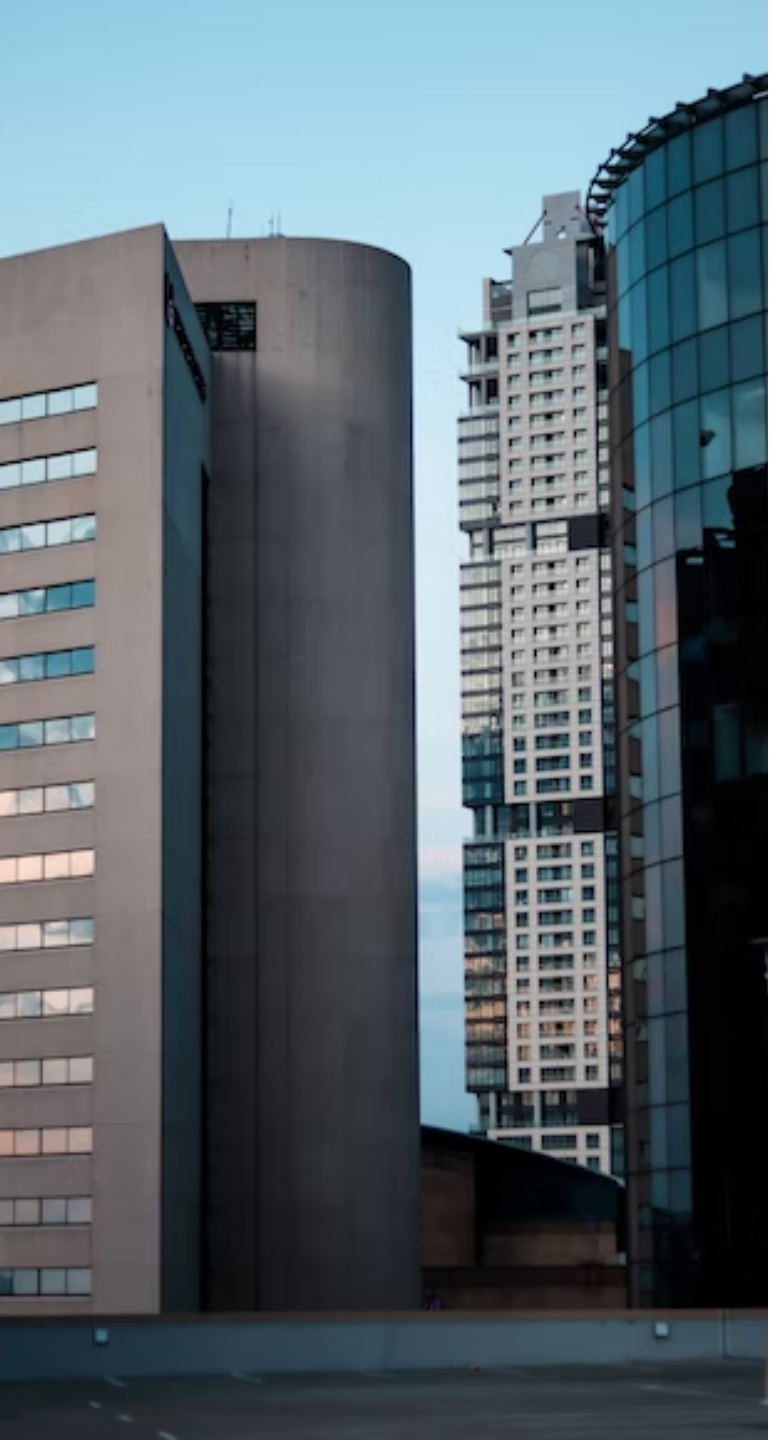
Transactional and Analytical Processing [ACID]

Isolation

> To guarantee that two transactions happen at the same time as if they were isolated. Two users accessing the same data won't change it at the same time. For example, you don't want two users to book the same driver at the same time.

Durability

> To guarantee that once a transaction has been committed, it will remain committed even in the case of a system failure. For example, after you've ordered a ride and your phone dies, you still want your ride to come.



4. Data Storage Engines and Processing [6]

Transactional and Analytical Processing

However, transactional databases don’t necessarily need to be ACID, and some developers find ACID to be too restrictive. According to Martin Kleppmann, “systems that do not meet the ACID criteria are sometimes called **BASE**, which stands for *Basically Available, Soft state, and Eventual consistency*. This is even more vague than the definition of ACID

4. Data Storage Engines and Processing [7]

ETL: Extract, Transform, and Load

In the early days of the relational data model, data was mostly structured. When data is *extracted* from different sources, it's first *transformed* into the desired format before being *loaded* into the target destination such as a database or a data warehouse. This process is called *ETL*, which stands for extract, transform, and load.

4. Data Storage Engines and Processing [8]

ETL: Extract, Transform, and Load

Extract is extracting the data you want from all your data sources. Some of them will be corrupted or malformatted. In the extracting phase, you need to validate your data and reject the data that doesn’t meet your requirements.

5. Modes of Dataflow

In this chapter, we've been discussing data formats, data models, data storage, and processing for data used within the context of a single process. Most of the time, in production, you don't have a single process but multiple. A question arises: how do we pass data between different processes that don't share memory?

5. Modes of Dataflow [1]

When data is passed from one process to another, we say that the data flows from one process to another, which gives us a dataflow. There are three main modes of dataflow:

- Data passing through databases
- Data passing through services using requests such as the requests provided by REST and RPC APIs (e.g., POST/GET requests)
- Data passing through a real-time transport like Apache Kafka and Amazon Kinesis

5. Modes of Dataflow [2]

[1] Data Passing Through Databases

- The easiest way to pass data between two processes is through databases. For example, to pass data from process A to process B, process A can write that data into a database, and process B simply reads from that database.
- However, doesn't always work because of two reasons.

> First, it requires that both processes must be able to access the same database. This might be infeasible, especially if the two processes are run by two different companies.

> Second, it requires both processes to access data from databases, and read/write from databases can be slow, making it unsuitable.

5. Modes of Dataflow [3]

Data Passing Through Services

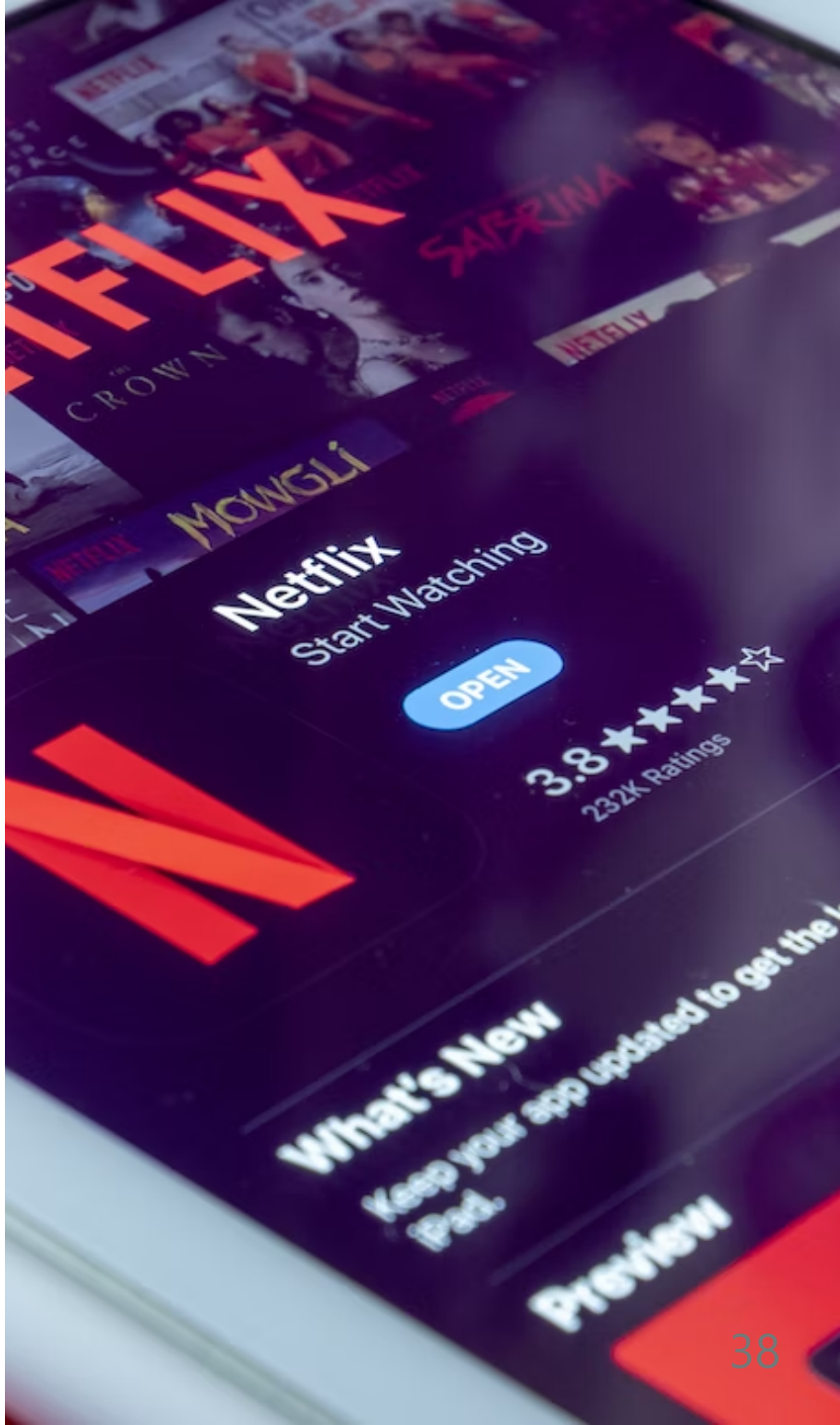
5. Modes of Dataflow [x]

Data Passing Through Real-Time Transport

6. Batch Processing Versus Stream Processing

Once your data arrives in data storage engines like databases, data lakes, or data warehouses, it becomes **historical data**. This is opposed to **streaming data** (data that is still streaming in). Historical data is often processed in batch jobs—jobs that are kicked off *periodically*.

> Read more at Introduction to streaming for data scientists | Chip Huyen Blog



6. Batch Processing Versus Stream Processing [1]

- When data is processed in batch jobs, we refer to it as **batch processing**. Batch processing has been a research subject for many decades, and companies have come up with distributed systems like MapReduce and Spark to process batch data efficiently.
- When you have data in real-time transports like *Apache Kafka* and *Amazon Kinesis*, we say that you have streaming data. **Stream processing** refers to doing computation on streaming data. Computation on streaming data can also be kicked off periodically, but the periods are usually much shorter than the periods for batch jobs (e.g., every five minutes instead of every day).

7. Summary

In this chapter, we learned it’s important to choose the right format to store our data to make it easier to use the data in the future. We discussed different data formats and the pros and cons of row-major versus column-major formats as well as text versus binary formats.

7. Summary

Thank you for your attention