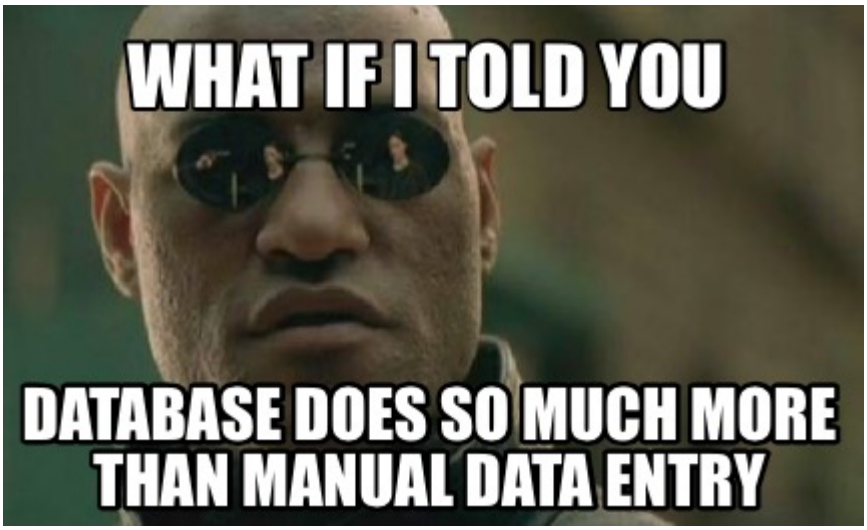
Database and the relational model



Agenda

- What is a database?
- Why do we use DBMS?
- What are the different types of databases?
- What is a relational database?

Key terms

Data

> **Information** or facts and statistics collected together for reference or analysis.

> **Information** that has been translated into a form that is efficient for movement or processing

Examples

- Runs scored by Virat Kohli
- Temperature in Bangalore
- Your food orders from Swiggy
- Students in a class

Database

> an organized collection of **inter-related** data that models some aspect of the real-world

> a set of **related** data and the way it is organized

Examples

- Scorecards of all cricket matches
- Weather conditions of all cities in the world
- All orders placed on Swiggy

- Students, mentors and batches

Database Management System

> software system that enables users to define, create, maintain and control access to the database

> the software that manages a database

Examples

- MySQL
- PostgreSQL
- MongoDB
- Oracle

Relational Database

> An approach to managing data using a structure and language consistent with first-order predicate logic, first described where all data is represented in terms of tuples, grouped into relations. A database organized in terms of the relational model is a relational database.

Non-relational Database

> A non-relational database is a database that does not use the tabular schema of rows and columns found in most traditional database systems. Instead, non-relational databases use a storage model that is optimized for the specific requirements of the type of data being stored

> Non-relational databases (often called NoSQL databases) are different from traditional relational databases in that they store their data in a non-tabular form. Instead, non-relational databases might be based on data structures like documents. A document can be highly detailed while containing a range of different types of information in different formats

Primary Key

> a specific choice of a minimal set of attributes (columns) that uniquely specify a tuple (row) in a relation (table)

Foreign Key

> A foreign key is a set of attributes in a table that refers to the primary key of another table

---

Brute Force - Files

Let us say it is 1960, and you haven't heard of databases. Scaler is early on the scene, and they want to store the following data points

- Student (name, age, address, phone, email, etc.)
- Mentor (name, age, address, phone, email, etc.)
- Batch (name, mentor, start date, type, etc.)

What is the simplest way we can store this data?

Files

- Store each entity as a separate CSV file
    - students.csv
    - mentors.csv
    - batches.csv

Sample students.csv

```
Name,Email,Phone,Age,Address
Tantia Tope,tantia@rani.bai,123456789,20,Jhansi
Kilvish,kil@vi.sh,987654321,21,Andhera
John Watson,i.am@sherlock.ed,123456789,30,221B Baker Street
```

or

Name	Email	Phone	Age	Address
Tantia Tope	tantia@rani.bai	123456789	20	Jhansi
Kilvish	kil@vi.sh	987654321	21	Andhera
John Watson	i.am@sherlock.ed	123456789	30	221B Baker Street

Recap

- Store each entity as a separate CSV file
- The first row of each file is the header
- Header contains the attributes of the entity
- Each row except the header contains the data of the entity
- To read a record, the file needs to be parsed every time

Issues - Can we use files for a real application?

First, let us see how we can interact with a file. The following code will read the file and print name of the students.

```
# Read file and print out name of students
with open(STUDENTS_FILE, "r") as file:
    for line in file:
        name = line.split(",")[0]
        print(name)
```

Can you spot an error in the output or the code?

Skipping the header row.

```
# Read file and print just names of students
with open(STUDENTS_FILE, "r") as file:
    for index, line, in enumerate(file):
        if index == 0:
            continue
        name = line.split(",")[0]
        print(name)
```

How can I search for all the users whose age is less than 25?

- Read the file
- Iterate through each row
- Parse the row and check if the age is less than 25
- Print the name of the user if true

```
# Read file and only print users with age less than 25
with open(STUDENTS_FILE, "r") as file:
    for index, line, in enumerate(file):
        if index == 0:
            continue
        name = line.split(",")[0]
        age = line.split(",")[3]
        if int(age) < 25:
            print(name, age)
```

Problems

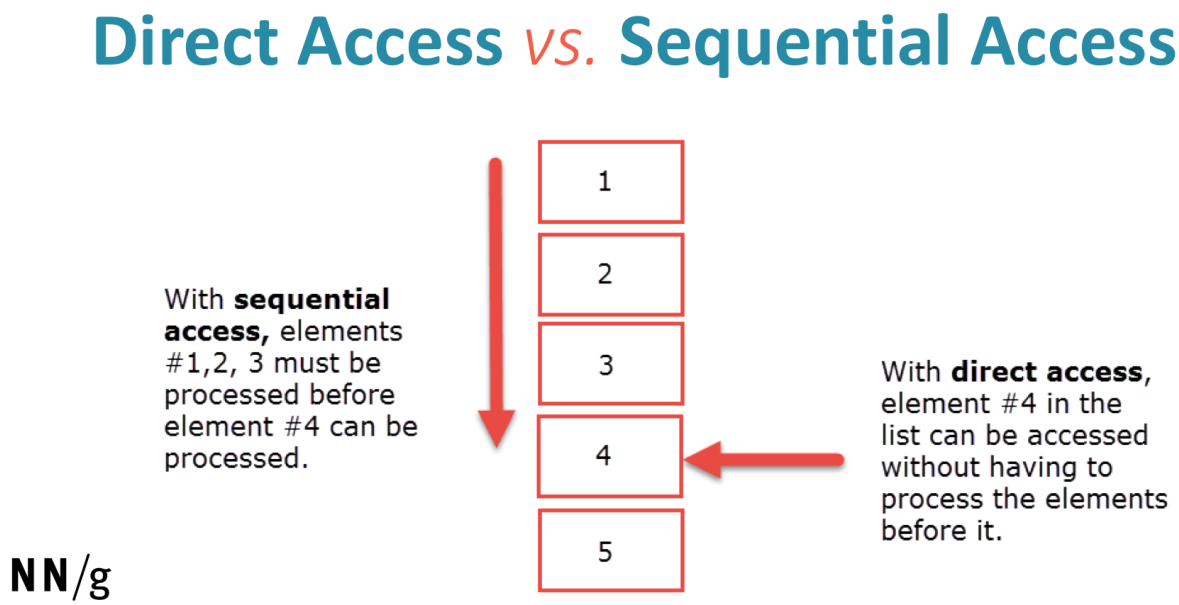
- Not scalable - Inefficient
    - Worst case complexity is O(n)
    - We need to read the file every time we want to search for a user
    - We need to go through the file every time we want to search for a user
- Data integrity
    - What if there are duplicates in the file?
    - What happens if we replace the age of a student with a garbage value?
    - What happens if we delete a mentor that is part of a batch?
- Concurrency
    - What if two users update and save the file at the same time? Which value is saved?
- Security
    - Anyone with access to file system be able to read the file and even update it.
- Fault tolerance
    - What happens if computer crashes while you are updating the file?

Recap

- Files are a simple way to store data
- To read even a single record, we need to parse the file every time

- Files are not scalable, secure or fault-tolerant
- When should we use a file?
    - Static data - Not frequently updated
    - Does not require complex operations like searching, updating, deleting, etc.
    - Small size of data
    - Fewer requests for the data - Less throughput
    - Configurations, log files, mock data, etc.

Sequential vs Random Access



DBMS

A database management systems aims to provide a single interface to a set of database services, that overcome the limitations of storing data in files.

> Codd proposed the following functions and services a fully-fledged general purpose DBMS should provide
>
> - Data storage, retrieval and update
> - User accessible catalog or data dictionary describing the metadata
> - Support for transactions and concurrency
> - Facilities for recovering the database should it become damaged
> - Support for authorization of access and update of data
> - Access support from remote locations
> - Enforcing constraints to ensure data in the database abides by certain rules

Types of DBMS

- Relational
- Non-relational (NoSQL)

- Columnar
  - > stores data tables by column rather than by row for more efficient access to data when only querying a subset of columns
  - MariaDB, InfluxDB
- Graph-based
  - > graph structures with nodes, edges, and properties to represent and store information
  - Neo4j
- Key-value
  - > uses a simple key-value method to store data where a key serves as a unique identifier.
  - DynamoDB, Redis, etc.
- Document-oriented
  - Extension of key-value database that stores data in a more complex structure which allows for optimisations for querying and storing data.
  - MongoDB, CouchDB, etc.
- Time series
  - > to store and retrieve data records that are part of a “time series,” which is a set of data points that are associated with timestamps. The timestamps provide a critical context for each of the data points in how they are related to others.
  - InfluxDB, TimeScaleDB, Prometheus, etc.
