Introduction

Spring 2025

Agenda

- Why databases?
- Structure of the course
- Go

Why do we need databases?

- Applications need to manage data
  - internal data structures for storing data
  - arrays, hashmaps, dataframes, ...
- BUT: how do you persist the data structures?
  - so that they survive an application stopping and restarting
- BUT: how do you share the data structures?
  - so that more than one user can interact with the data at the same time
- BUT: how do you manage data that's larger than memory?
  - historical data
  - system logs
  - large machine learning training sets

Databases

Pull data out of the application and persist to disk

- DIY in the early days
- tends to be slow and error-prone to implement by hand

Create "generic" tool to store data on disk — a database

- Provide a convenient interface to the data
- Efficient storage and retrieval mechanism
- Run as a server accessed by other applications
- Support concurrent access and replication

Databases support a specific **data model** (how data is conceptualized)

Relational databases

Data conceptualized as rows of values in **tables**

- Think Excel sheets
- Original data model
- **Convergence** to a unified way of presenting that model in the 70s
  - Oracle, Postgres, MySQL, Microsoft SQL Server, …
  - Standardized query language: **SQL**
- Requires pre-defined data schemas (think static typing)
- ACID properties: atomicity, consistency, isolation, durability
  - basically each user has the illusion of being the only user of the database

Non-relational databases

Also known as NoSQL databases — any database that is not relational:

- Document databases: MongoDB, CouchDB, Firebase Realtime…
- Key-value stores: Redis, DynamoDB, Cassandra, HBase, ...
- Graph databases: Neo4j, ...

They relax the structural and consistency restrictions of relational databases

- No need for schema definitions (think dynamic typing)
- More resilient replication
- What's the catch?

Looks complicated...

As a casual programmer, beyond figuring out whether you'll be storing arrays (relational) or storing JSON objects (non-relational), **it mostly doesn't matter** which database you use for your projects

> ***They're all pretty much equivalent for casual use***

There are big differences, but they don't really appear until you have millions of data points, high traffic, or extensive database replication to reduce latency

- Most likely, an architect at your company will have chosen for you

… and sometimes it is complicated

A project/startup starting small, growing huge fast

- The database choices made early often won't scale
- Need for database migration
- Made easier by abstracting the interface to the DB as much as possible
  - DAL = data access layer, which interacts with the DB directly
  - interact with the DAL via IPC / REST calls
  - migrating databases requires changing the DAL, not the client
- There are other approaches (ORMs, etc)

Structure of the course

We're going to study the various classes of databases that exist

- relational vs non-relational databases
- learn how to interact with them
- learn the key features of each class
  - data structuring
  - query languages
  - transactions, concurrency
  - replication, consistency
- learn a little bit how they work inside
  - this is not a database theory course
  - learn enough to understand what is happening
  - (learn enough to know you don't want to roll-your-own except for small projects)

Structure of the course

- Lecture-based — I like talking
  - I also like to demo code
  - Textbook *Principles of Database Management*
- Homeworks
  - Go code to build up an understanding of the concepts we study
  - some hands-on experience with databases: install them, populate them, query them
  - work in pairs (choose your team)
- Final project
  - choose project mid-semester
  - can be a deeper exploration of a topic we talked about, or an introduction to a topic we skipped
  - final presentations during events week
- Meet on Mondays, office hours before class
  - No CA, but I'm available for questions, and we can find tutors if you need one
  - Speak up, reach out — as usual, the more interactive it is the more fun it is

Introduction to Go

(online demo)