Introduction to Databases

CSCI 220: Database Management and Systems Design

Welcome to CSCI 220!

- First 10 minutes:
  - **Sign-in** on the attendance sheet
  - **Create a name card**
  - **Add a profile picture** to your Canvas account
  - Form a group of 3-5 people and **discuss**: What are you hoping to learn in this course? What is a database?

About the Course

- Tour of the course website, Canvas, and Gradescope
  - https://cs.clarku.edu/~cs220/
  - https://canvas.clarku.edu
- Some lectures were adapted from material by Zhenguang Gao, George Kollios, Simon Miner, and John and Tricia Magee

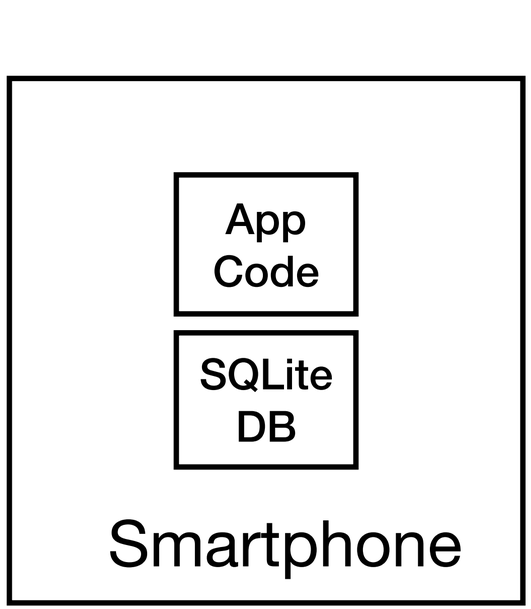
Today you will learn about:

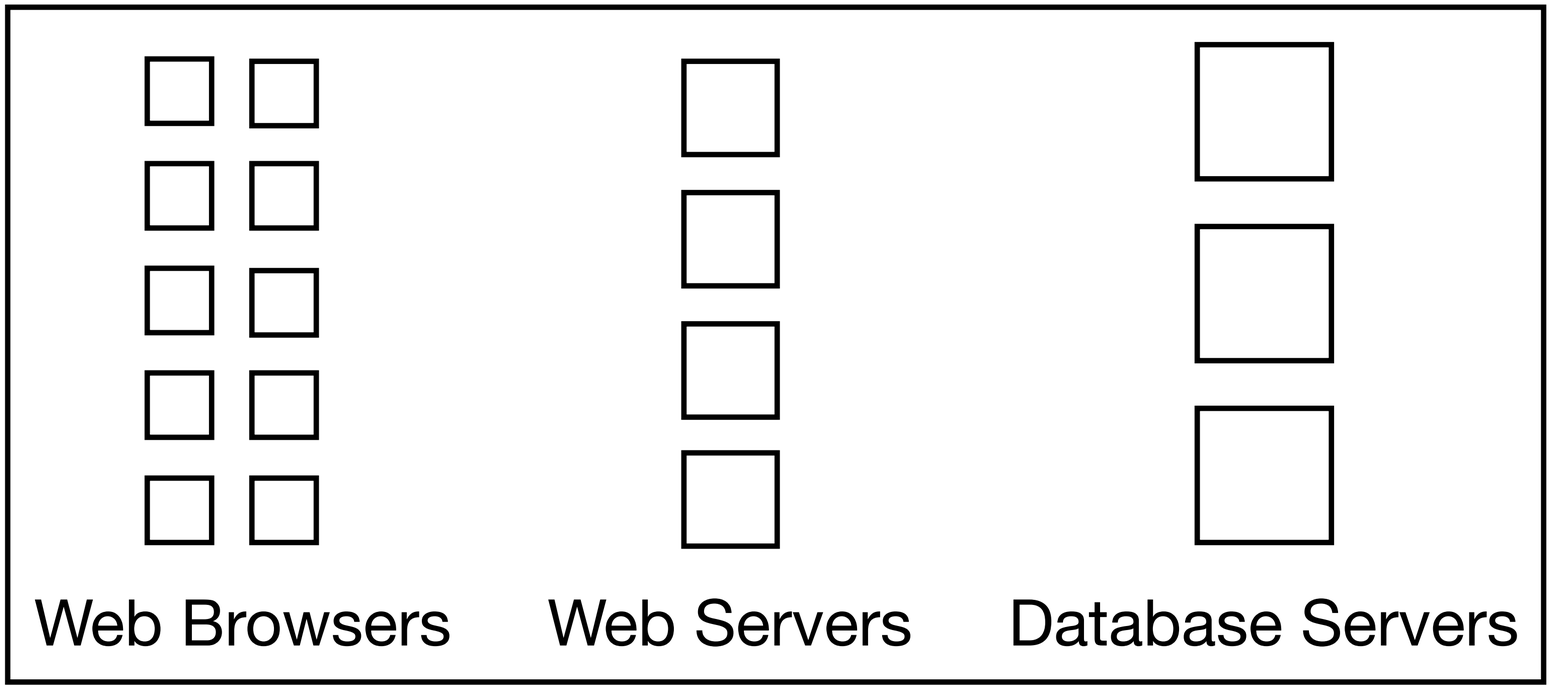
- Why databases are ubiquitous
- The powerful capabilities of database systems
- Database design basics

Why Databases?

- Databases add a layer of abstraction between an application and the physical storage of data

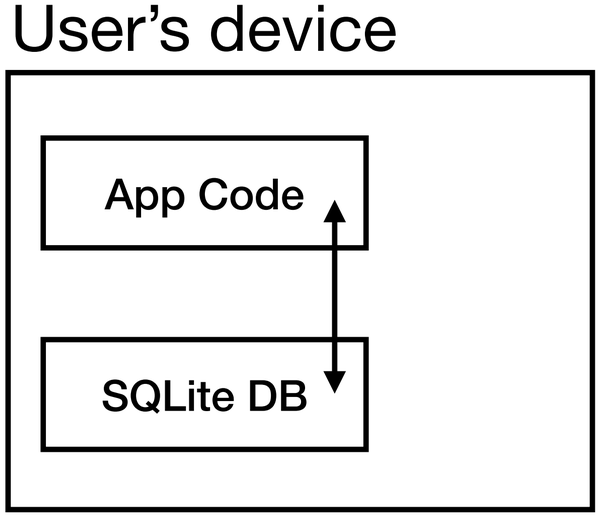
- Databases make it easier to write reliable, high-performance applications





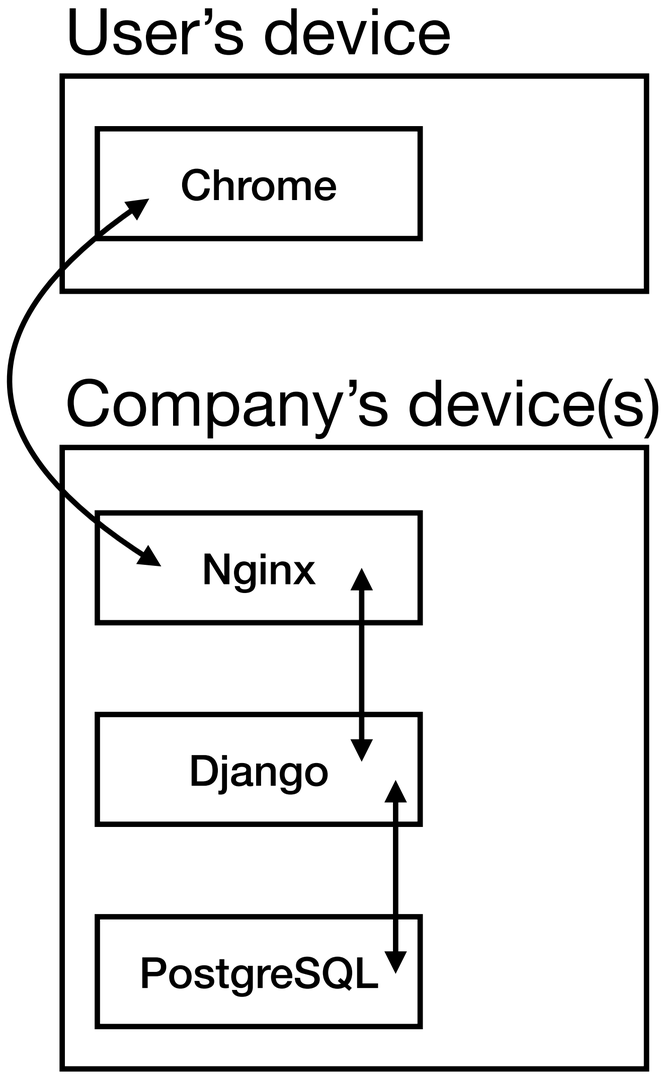
Smartphone App Architecture

- App code renders the app's graphics.
- The app reads data from a SQLite database on disk.



Web App Architecture

- Web browser (e.g., Chrome, Safari) requests pages and renders the application's graphics
- Web server (e.g., nginx, Apache) passes data between the browser and the application code
- Application code (e.g., Django) builds the HTML for dynamic pages, based on data from the database
- The database (e.g., PostgreSQL, MySQL) manages physical storage of the data



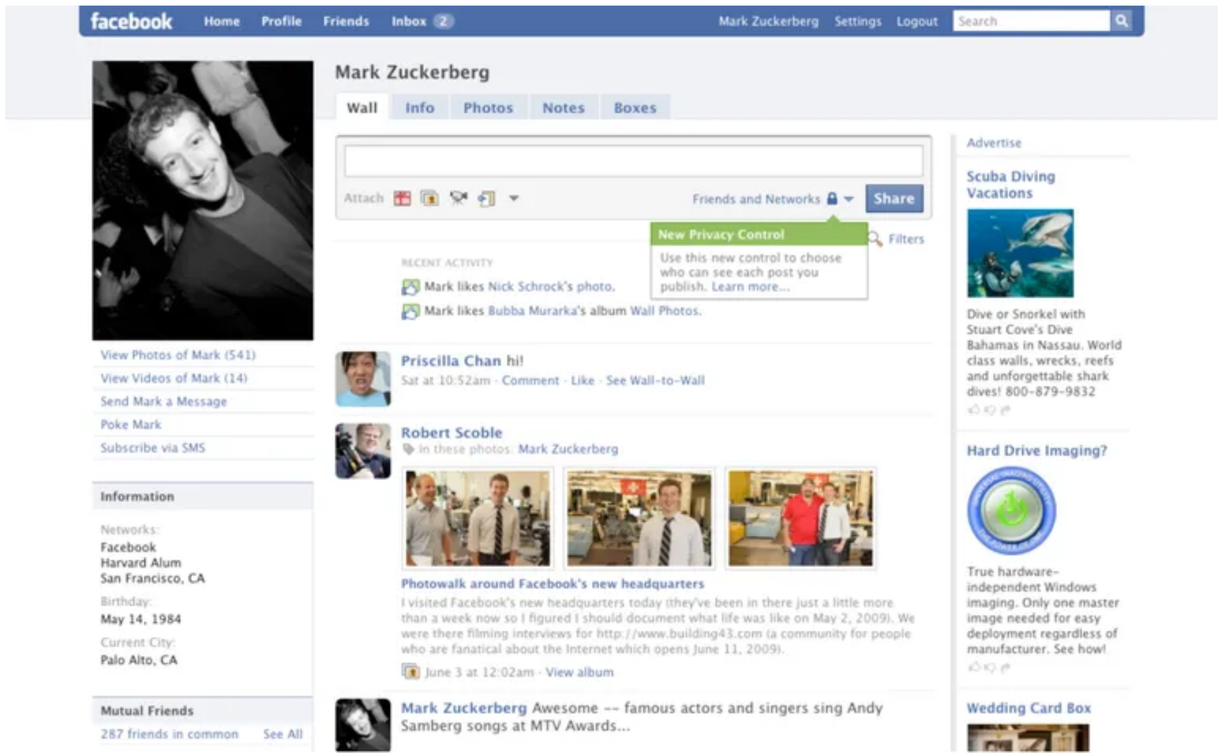
Does your app need a database?

- Is your data updated by multiple users?
  - Databases help **manage concurrency**
- Do you need to enforce properties of your data (e.g., preventing overdrawing from an ATM, ensuring all users have email addresses, etc.)?
  - Databases **enforce constraints**
- Is your data complex (e.g., relationships between entities)?
  - Databases **offer high-performance operations** on complex data
- Should your app recover gracefully from a crash or power failure?
  - Databases **protect data integrity**

Our focus is on "relational databases," which offer all of these features. If some of these features aren't necessary, a NoSQL database *could* be sufficient.

Example: Facebook

- Facebook definitely uses a database
- A simplified list of Facebook's data:
  - Profile info: name, email, password, birthday, …
  - Status updates
  - Friendship



https://www.cnet.com/pictures/facebook-then-and-now-pictures/

Example: Facebook Database

Primary Key

Users

ID	Name	Email	Password
111	Peter Story	PeStory@clarku.edu	*********
112	John Magee	JMagee@clarku.edu	*********
113	Li Han	LHan@clarku.edu	*********

Foreign Key

Status Updates

Timestamp	User ID	Text
2023-11-29 10:57:01	111	The CMACD building is great!
2023-11-29 11:38:17	113	Welcome back students!
2023-11-29 11:46:29	113	Consider declaring your major!

Friendship

User ID 1	User ID 2
111	112
111	113
112	113

Foreign Keys

Primary and Foreign Keys

- A **primary key** uniquely identifies a record (row) in a table
  - It is efficient to retrieve a record if you know its primary key
  - Primary keys often appear in URLs:
    https://www.facebook.com/profile.php?id=111
- A **foreign key** references a primary key in another table
  - This allows relationships between tables

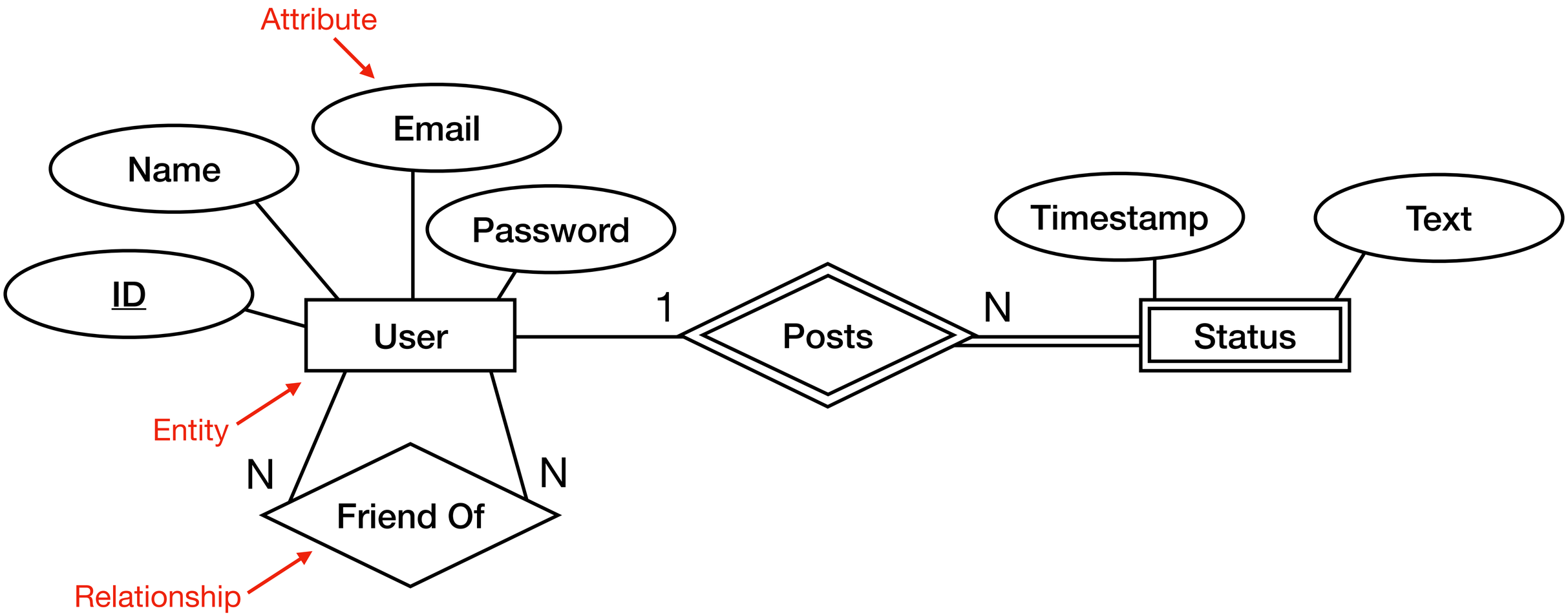
Constraints

- Data types
- Referential integrity constraints
- Uniqueness constraints (AKA key constraints)
- Additional constraints:
  - Email must be formatted properly
  - Status text must be less than a certain length

Data Modeling

- It is essential to understand the data used by an application, the relationships between the data, and the constraints on the data.
    - This is your **database schema**
- We depict the schema using diagrams before we implement the schema in code:
    - An Entity-Relationship diagram shows relationships and constraints
    - A tabular depiction of the database schema (i.e., the relational model) can show example data, and is closer to how data is stored on disk

Example: Facebook ER Model



Example: Facebook Relational Model

Users

ID	Name	Email	Password

Friendship

User ID 1	User ID 2

Status Updates

Timestamp	User ID	Text

Example: Facebook Relational Model with Data

Users

ID	Name	Email	Password
111	Peter Story	PeStory@clarku.edu	*********
112	John Magee	JMagee@clarku.edu	*********
113	Li Han	LHan@clarku.edu	*********

Status Updates

Timestamp	User ID	Text
2023-11-29 10:57:01	111	The CMACD building is great!
2023-11-29 11:38:17	113	Welcome back students!
2023-11-29 11:46:29	113	Consider declaring your major!

Friendship

User ID 1	User ID 2
111	112
111	113
112	113

Course Overview

- 2-3 weeks: Data modeling
- 2-3 weeks: Database queries
- 2-3 weeks: Database programming
- 1 week: Database file structures
- 1 week: Crash recovery
- 1 week: Concurrency control
- 2 weeks: NoSQL databases

Preview of Future Topics

Database Properties (ACID)

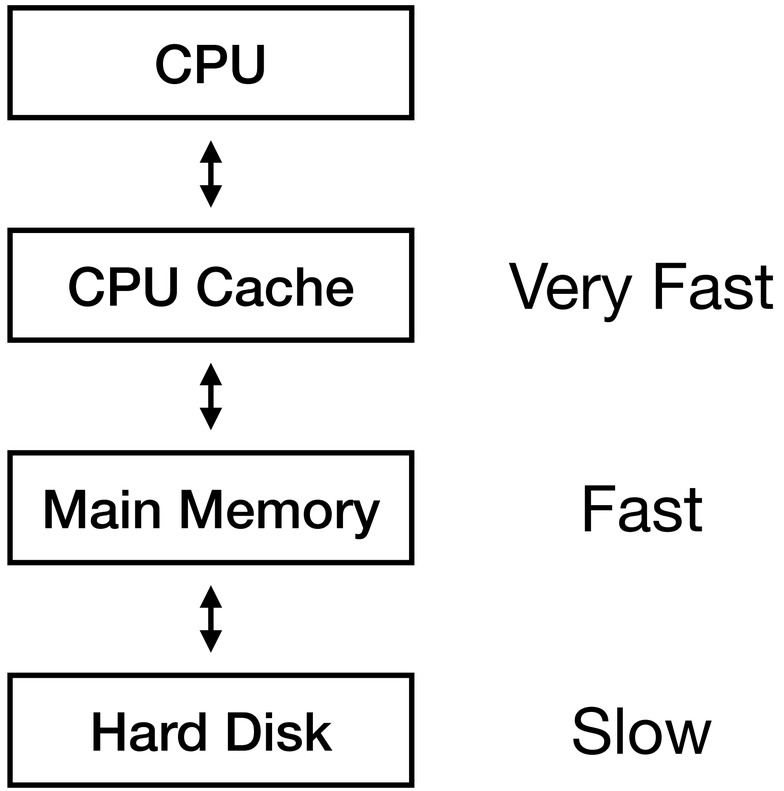
- **Atomicity:** All of a transaction must be completed, or none of it
- **Consistency (Correctness):** Don’t allow the database to enter a corrupted state
- **Isolation:** Concurrently executed transactions must have the same effects as sequentially executed transactions (since databases usually have multiple users)
- **Durability:** After a transaction completes, it should persist even if the power fails, etc.

Database Management Systems (DBMSs)

- SQLite: suitable for single-user apps (doesn’t have a server process)
- PostgreSQL: free open-source (FOSS) DB with many advanced features
- MySQL: widely deployed FOSS DB with fewer features and more quirks
- Proprietary DBs: Oracle, IBM DB2, Microsoft SQL, Microsoft Access, …
- NoSQL DBs: (typically) don’t use a relational data model, and sacrifice ACID properties for performance (Redis, MongoDB, …)

Database Memory Hierarchy

- CPU cache and main memory are fast, but have limited capacity
- The hard disk is slower, but has much more capacity
- Furthermore, memory is volatile, whereas the hard disk is non-volatile
- DMBSs use the memory hierarchy to achieve high performance and the ACID properties



Database Files

SQLite is simple: one file, such as db.sqlite

PostgreSQL and (other multiuser databases) manage 1000s of files:
`ls /var/lib/postgresql/data`

- /var/lib/postgresql/data/postgresql.conf
- /var/lib/postgresql/data/pg_xact/0000
- /var/lib/postgresql/data/pg_subtrans/0000
- /var/lib/postgresql/data/pg_ident.conf
- /var/lib/postgresql/data/postmaster.pid
- /var/lib/postgresql/data/postgresql.auto.conf
- /var/lib/postgresql/data/pg_multixact/offsets/0000
- /var/lib/postgresql/data/pg_multixact/members/0000
- /var/lib/postgresql/data/pg_wal/000000010000000000000001
- /var/lib/postgresql/data/pg_logical/replorigin_checkpoint
- /var/lib/postgresql/data/base/1/826
- /var/lib/postgresql/data/base/1/3599
- ...
- /var/lib/postgresql/data/base/4/2615
- /var/lib/postgresql/data/postmaster.opts
- /var/lib/postgresql/data/pg_hba.conf
- /var/lib/postgresql/data/PG_VERSION
- /var/lib/postgresql/data/global/4176
- /var/lib/postgresql/data/global/6302
- ...

Database Indexes

- How to locate the user with ID 111?
- One approach: scan the user table, check every record, return the one with id=111. Very slow for large tables! Other ideas?
  - Keep records ordered by ID, and use a binary search. But updates will be slow.
  - Use a search tree index. Keep records sorted while allowing insertions, deletions, and updates. The B+-tree (multiway search tree) is common.
  - Use a hash table index. Much faster for exact match queries, but cannot support range queries.
- **Primary and foreign keys should be indexed**

Database Queries

- How to retrieve records from a database?
- Using SQL (Structured Query Language)
- Find the record for the user with ID 111:

```
SELECT *
FROM user
WHERE user.id = 111
```

- Supports sorting, queries across tables, computing averages, etc.

Data Retrieval

- Your SQL query tells the database what you want. The database (usually) retrieves the results as efficiently as possible.
- Often, several plans are considered. For example:
  - Should indices be used?
  - When tracing relationships across tables (joining), which table to start with?
- The choice of plan depends on statistics collected by the database (e.g., table size)

Data Integrity: Transaction Processing

- Suppose John and Jane withdraw $50 and $100 from a common account:

John:	Jane:
get balance	get balance
if balance > $50	if balance > $100
balance = balance - $50	balance = balance - $100
update balance	update balance

- Initial balance $300. Final balance? Depends on whether isolation is enforced!

Data Integrity: Recovery

- Suppose we try to transfer $50 from account A to account B:

get balance for A
if $\text{balance}_A > \$50$
$\text{balance}_A = \text{balance}_A - 50$
update balance_A in database
get balance for B
$\text{balance}_B = \text{balance}_B + 50$
update balance_B in database

Money will be lost if these steps aren't completed!

- Databases can recover from crashes or power outages by rolling back an unfinished transaction. This preserves atomicity.

If Time: Research a DBMS

- Form a group of 3-5 students
- Each student should research a different relational database. Consider researching: SQLite, PostgreSQL, MySQL, Oracle, IBM DB2, Microsoft SQL, Microsoft Access
- For your database, search online to answer these questions:
  - When was the database first released? When was it last updated?
  - How widely deployed is the database (e.g., number of installs)?
  - What makes this database unique?
- ~10 minutes before class ends, discuss your findings with your group

Prepare for Lab