Lecture 21: Databases

CS109B, STAT 121B, AC209B, CSCI E109B

David Wihl

April 18, 2018

Are you a database user?

What’s wrong with files, such as CSVs?

Does your data ever get updated?

- By more than one person at a time?
- How do you enforce integrity?
- Update two CSVs together (e.g. bank transaction)?

Ever needed to search for something?

Does your data have any hierarchical relationships?

Does your data fit into memory?

Enter the Database Management System (DBMS)

Problem with files and CSVs	Solution by DBMS
Does your data ever get updated? By more than one person at a time? How do you enforce integrity?	**Concurrent updates, locking, integrity, constraints**
Two linked updates?	**Transactions, “ACID”**
Ever needed to search for something?	**Indexing**
Does your data have any hierarchical relationships?	**Multiple tables, Keys, Joins**
Does your data fit into memory?	**Massive scaling, cursors**

Databases

- Massive (terabytes)
- Persistent
- Safe - built in redundancy
- Multi-user - concurrency control
- Convenient -
  - physical independence, high level query language, **declarative**
- Efficient - thousands of queries / updates per second
- Reliable - >=5 9’s

CRUD - the heart of all data applications

C: Create

R: Read

U: Update

D: Delete

The Relational Model

Set of tables

- Schema -> structural description (“class definition”)
- Instance -> actual contents

A relational database:

- set of tables with indices
- relationships, ie. keys
- administration: security, operations, implementation

SQL - Structured Query Language

Declarative Language: “What I want”, not “How to find it”

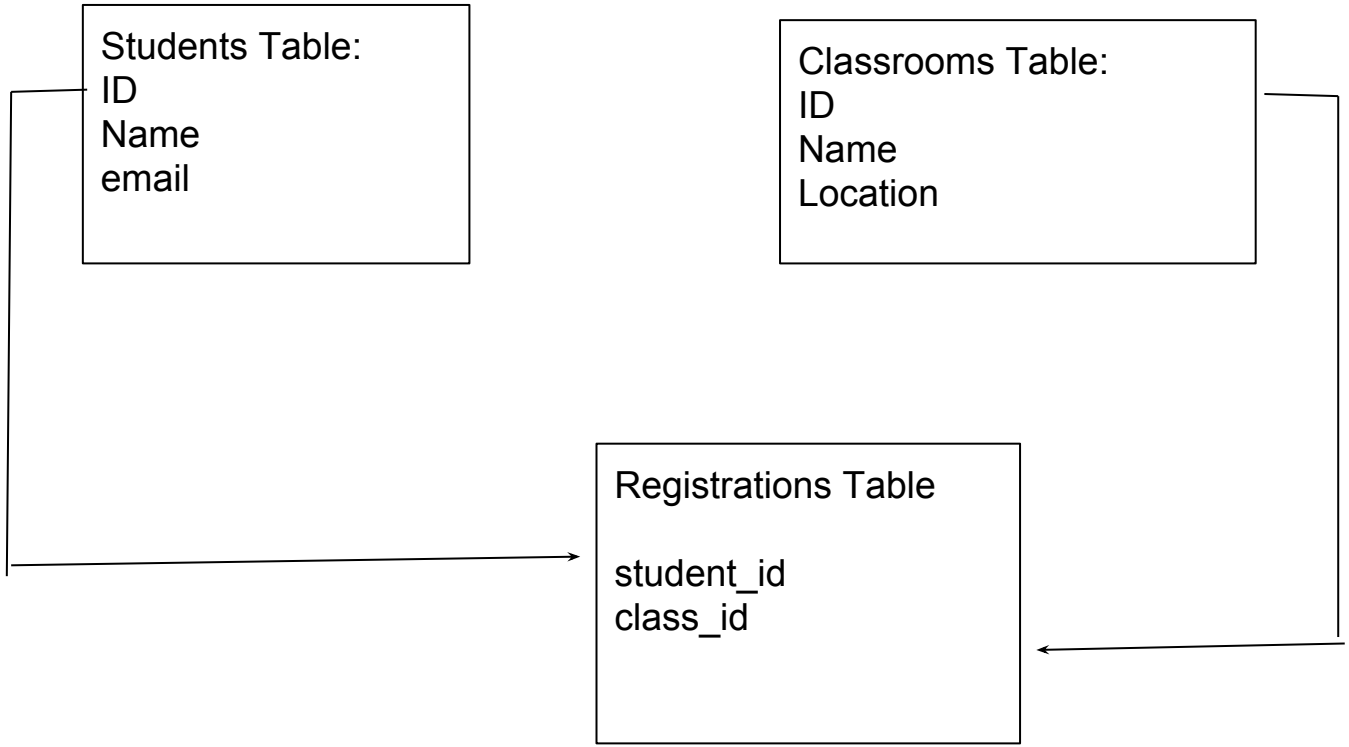
Independent of Data Representation, Server implementation

Compact, easy to understand and debug language

Usable by non-programmers

Inconsistent across implementations, lots of dialects

An Example: Modeling a Classroom



Indexing

Speed up common search criteria

Can span columns

Can enforce uniqueness

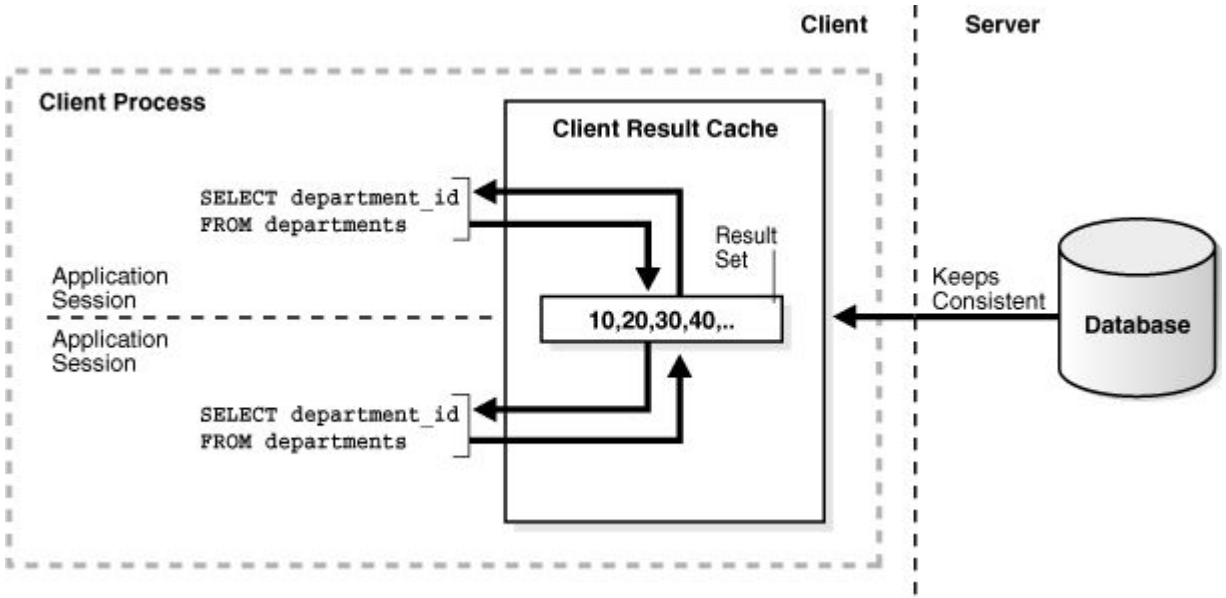
Downsides:

- creation and maintenance, avoid excessive indices

Aggregating

Grouping / subtotals / sorting

slice close to the data



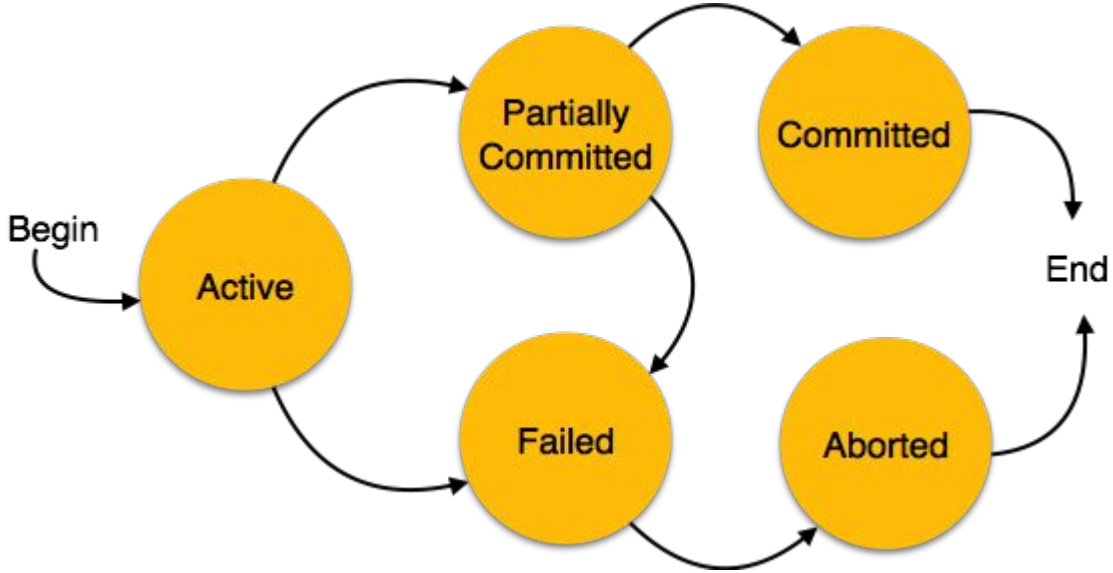
Transactions

A: Atomic

C: Consistent

I: Isolated

D: Durable



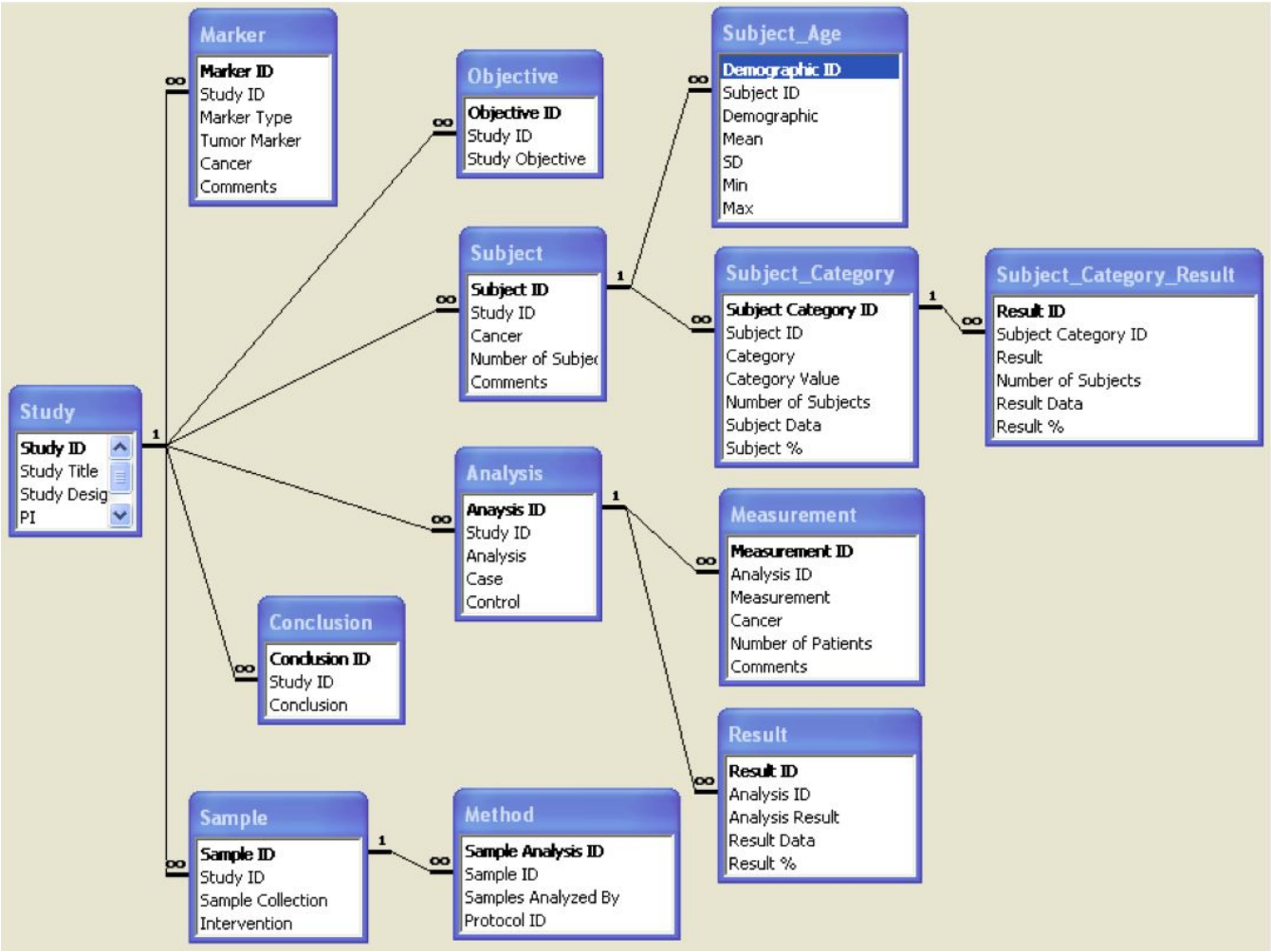
Guarantee that data stays consistent in multiple locations

General programming model?

Schemas

Fixed vs Flexible

“Normal forms”



https://cancer.jpl.nasa.gov/documents/applications/biomarker-database/images/BMDB%20Database%20Schema%20%28obsolete%29.png

NoSQL Databases

Flexible schema

Relaxed consistency -> fewer guarantees

Massive Scalability

No declarative SQL -> more programming

Flavors of NoSQL

MapReduce - OLAP

Key-value stores - OLTP

Document stores - JSON

Graph databases

MapReduce

No data model - data stored in files

DBMS is “glue” for fault tolerance and scalability

Useful for large scale querying, large data stores

e.g. web log files, unstructured information

Key-Value Stores

Persistent, scalable, shared Python dictionary

Replication “eventually consistent”

High speed lookup for small items

Consistency not critical

Document Stores

CouchDB, MongoDB

“Persistent collection of JSON files”

Flexible schema - great for prototyping

Terrible writing and indexing performance

- Only recently multi-write

Now incorporated into classic relational databases like Postgres

Graph Database

Neo4J, FlockDB, Pregel

Persistent storage of nodes, edges

Flexible relationship traversal

Recursive search without custom joins

Future

You’ll likely rarely use CSV files in real-world applications

Understanding databases is key to building scalable applications, especially for changing data

Questions?

https://www.linkedin.com/in/davidwihl/