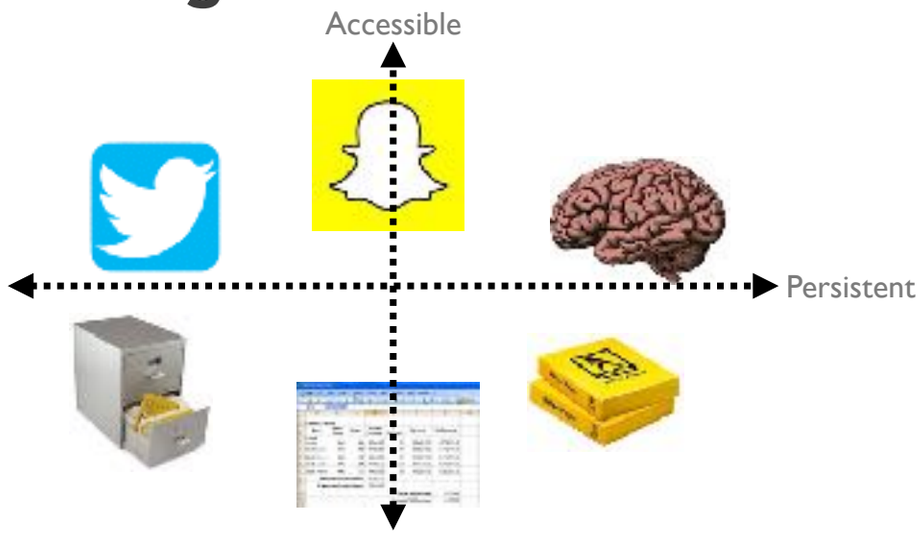
Intro to Databases

SQL

What is a database?

Things that hold info



A database **persists** information and is **accessible** via code

organized **queryable** **manageable**

Organized: Standard Storage Formatting

- DBs are a collection of Tables (or *relations*)
- Tables have Columns (*attributes* / *fields*) that describe Rows (*instances* / *tuples*)
- Duplicate rows are not allowed
- Rows often have a primary key (unique identifier)

Imagine an excel spreadsheet.
natural way to express information.
Objects often nicely map to tables

Table / Relation

Column / Attribute / Field | Column / Attribute / Field | Column / Attribute / Field

	ID	Name	Type
Row / Tuple / Instance	1	Pikachu	lightning
Row / Tuple / Instance	2	Squirtle	water
Row / Tuple / Instance	3	Charmander	fire
Row / Tuple / Instance	4	Bulbasaur	grass

Queryable: via a Standard Language

- A simple, structured query language: SQL
- Declarative (vs. imperative)
- No more hand-rolled algorithms / data structures
- DBMS picks an efficient execution strategy based on indexes, data, workload etc.

relational algebra

SQL

```
-- Pikachu, I choose you!
SELECT id, name
  FROM pokemon
  WHERE type = 'lightning'
  LIMIT 1
```

Manageable: Easy, Safe, Performant

- Offloads work and requisite understanding of programming
- Knowledge is portable
- Abstraction
- Transfer data between systems
- DBMS can make certain guarantees
  - prevent unsafe operations
  - built-in redundancies
  - handle multiple users, threads

ACID Guarantees

- Atomicity
- Consistency
- Isolation
- Durability

Atomicity - A set of database operations that must occur together. All or nothing. A transaction cannot partially finish, it must either fail or complete.
Consistency - If a process has a writer, no other process can read from it, and no other process can write to it, which gives us consistent information at all times.
Isolation - Multiple clients can make queries to read and update without the risk of deadlock or starvation.
Durability - Store information without power (flash drive / hard drive, for example doesn’t need power and still stores data)

Atomic Transactions

- **atomic transaction: A set of database operations that must occur together**
  - i.e. A debit to one bank account, and a credit to another
- **A transaction must either succeed or fail; it cannot partially complete.**
- **Every database query is represented by a transaction**

We might also say that a database is atomic or a database has the property of atomicity.

We can define where transactions start and end. You may need to as a programmer

Consistency

- **Specify rules that columns need to follow**
  - Gender column can only contain M, F, or U.
  - Savings account must start with S or checking with C
  - Column cannot be null
- **Protect the database from inconsistencies and simplify software logic**
  - Allows software to make assumptions about underlying data

Resource Management

- Processes can be readers and writers
- Files can have many readers
- If a process has a writer, no other process can read from it, and no other process can write to it

Proposed File Scheme

- Suppose that we have decided not to use a database and instead store our data in a series of files.
- How might our setup fail to serve queries from multiple users?

Deadlock

- Three files: A, B, and C
- Process 1 needs files A then B for writing
- Process 2 needs files B then C for writing
- Process 3 needs files C then A for writing
- What happens if all three processes start a database request at once?

Databases give us concurrency (Isolation)

- Multiple clients can make queries to read and update without the risk of deadlock or starvation.

Persistence/Durability

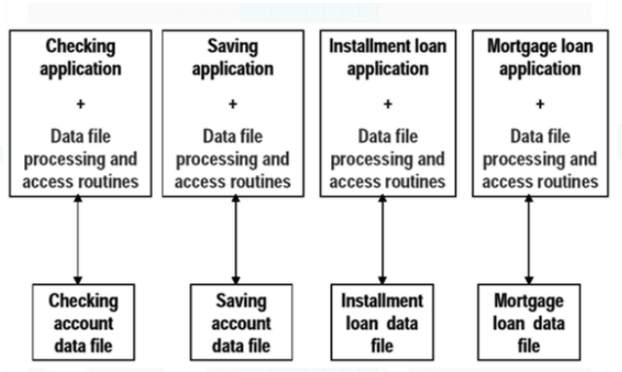
- Files are also persistence (store information without power)

- What happens when two people try to access the same file at the same time?
- the second process waits and retries until the first process finishes?

How Did We End Up Here?

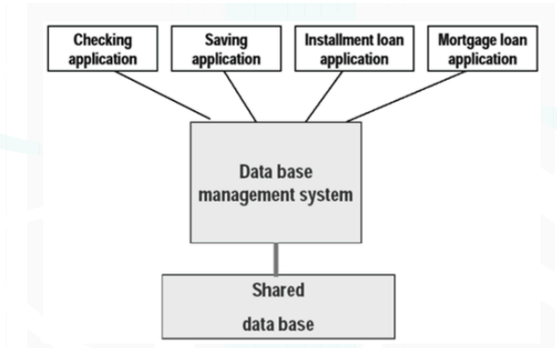
Before Relational DBs (ca. < 1970s)

- Data stored in custom “data files”
- Queried via application-specific code
- Advantages
  - Middle layer not needed
  - Solutions customized for each application
- Disadvantages
  - Hard to change the system
  - Knowledge not compounding
  - Data-transfer is difficult



- Separate file for every topic. Routines and functions for dealing with each particular file.
- But every company formatted their files differently based on developer whims and needed their own accessor functions.
- Making changes was really hard seeing as formats were inflexible and etched in stone.

Database Management Systems (DBMS)



- One layer and language to store and access data
- Sold as a way for “non-technical people” to manage data

Ubiquitous
Standardized
Don’t need to be a programmer
common way of thinking about data. And a common language for querying it.

"Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation)."

– E. F. CODD,
A RELATIONAL MODEL OF DATA FOR
LARGE SHARED DATA BANKS

Relational Databases & Logic

- **1969: Edgar Frank "Ted" Codd outlines *relational model* of data**
- **Wrote Alpha (never implemented) as a *query language***
- **IBM slow to adopt his ideas**
  - Competitors started to do so
  - IBM team formed without Codd, created **S**tructured **E**nglish **Qu**ery **L**ang
- **SEQUEL way better than what came before**
  - 1979: copied by Larry Ellison (from pre-launch papers / talks!) as "SQL"
- **SQL became the standard (ANSI 1986, ISO 1987)**
  - Codd continued to fault SQL compared to his theoretical model
  - The Third Manifesto: solve the *object-relational impedance mismatch*

Appreciating Databases

- **Ubiquitous**
- **Standardized**
- **Complex / deep**
- **Powerful: database admins are**
  - Feared by developers
  - …but also taken for granted until things break
  - Befriended by business people
  - Contacted by the government for secret data (e.g. NSA)

Progression of Databases

- **Navigational (< 1970s)**
  - More common during tape era; entries had references to next entries.
- **Relational (> 1970s)**
  - Based on relational (table-based) logic, see E.F. Codd.
- **NoSQL (> 2000s)**
  - "Not only SQL" — document storage, for example.

Navigational: common during tape era, entries had references to the next entries.
Relational: based on relational (table-based) logic, EF Codd.
NoSQL ("not only SQL"): document storage.

RDBMS vs NoSQL

- **A DBMS doesn't *have* to be relational**
  - Remember, DBMS is just an application that intelligently stores data and can answer requests to manage that data
- **Lately, many "NoSQL" or non-relational DBMSs have been gaining popularity**
  - Graph databases (e.g. Neo4J)
  - Document databases (e.g. MongoDB)
  - Hybrids (e.g. PostgreSQL)
- **RDBMSs still remain the #1 DB option for now**

Some well-known rDBMSs


PostgreSQL

Why PostgreSQL?

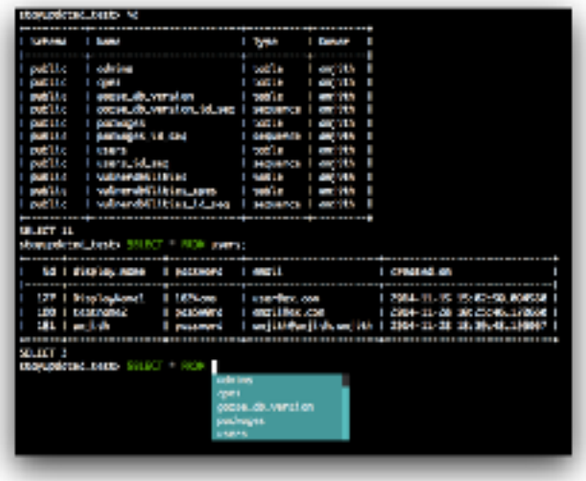
- Advanced, powerful, and popular
- Rapid open source development
- Highly extensible (stored procedures)
- Deep SQL standards compliance
- NoSQL ("Not Only SQL"), objective support
- Excellent transactions / ACID reliability; focus on integrity
- Multi-user management / administration

History of PostgreSQL

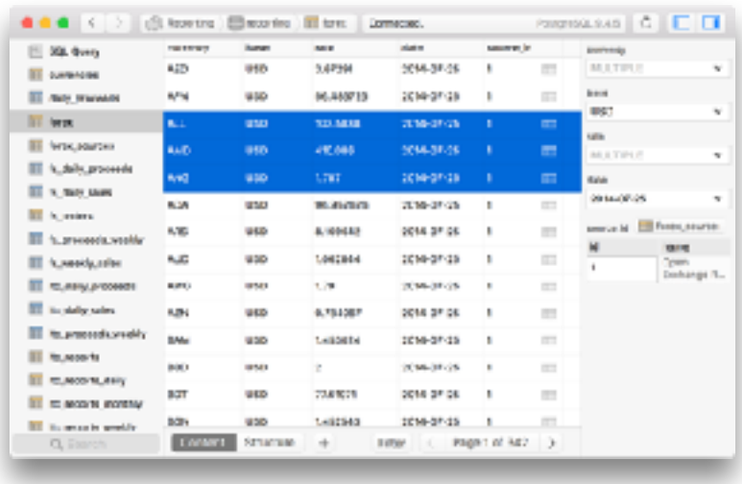
- 1970s at UC Berkeley:
  INteractive **G**raphics **RE**trieval **S**ystem (INGRES)
- 1980s: POSTGRES ("Post-Ingres")
- 1995: POSTQUEL and Postgres95.
  - `monitor -> psql`
- 1996: Adopted by the open source community
  - Ongoing: stability, testing, documentation, new features
  - PostgreSQL

psql

pgcli



Postico



LAB