INFO 290T

Human-in-the-loop Data Management
aka Data Engineering

Today's Agenda

- The essentials
- Bird's eye view of the class material
- Getting to know you
- Classical database systems: an introduction

The Essentials I

- Instructor: Aditya Parameswaran
- Assistant Professor, I School and EECS
- Office: 212 South Hall
- Email: adityagp@berkeley.edu
  - **Mention "INFO290" in the title!** (will help ensure a response)
- Meeting slots: M/W 9-10.30am @ 210 South Hall
  - Class slides will be posted after class
- Office hours: W 10.30-12pm @ 212 South Hall
  - Also on demand if need be

The Essentials II

- GSI: Doris Lee
- PhD Student, I School
- Email: dorislee@berkeley.edu
  - **Mention "INFO290" in the title!**
- Office Hours: TBD

- Website: https://info290.github.io

The Essentials: Prerequisites

- "Experience with programming, a basic understanding of computer systems, data structures, and algorithms expected."

The Essentials: Prerequisites

- “Experience with programming, a basic understanding of computer systems, data structures, and algorithms expected.”
- “Experience with programming”
    - Python (plus ideally Java), web programming
    - Unix/shell scripting, installing and using software packages
- “computer systems”
    - At a high level: OS, disk, memory, processor, networks
- “data structures and algorithms”
    - O(n), sorting, graph searching (DFS/BFS), hashing, dynamic programming

The Essentials: Prerequisites

- “Experience with programming, a basic understanding of computer systems, data structures, and algorithms expected.”
- “Experience with programming”
  - Python (plus ideally Java), web programming
  - Unix/shell scripting, installing and using software packages
- “computer systems”
  - At a high level: OS, disk, memory, processor, networks
- “data structures and algorithms”
  - O(n), sorting, graph searching (DFS/BFS), hashing, dynamic programming
- That said, I am happy to have people from different backgrounds
  - Talk to me if you’re not sure
  - Be prepared to read up on concepts outside of class

What sort of course is this?

- A lot of the content is **experimental**
  - Not taught in typical database classes
  - Knowledge and tools you pick up by working on industry data systems
  - Rapidly evolving field, so we will try to focus on the timeless concepts
  - A human-centered perspective: emphasizing usability and expressiveness
- Non-traditional evaluation: 5 assignments and a project
  - So no ”exams”
- Lecture-oriented, but some of the content will be more ”half-baked”

Goals of the Class

- **A high-level overview of data systems for various stages of data science, from data preparation to actionable insight**

- A modern take on a database class that reflects the rich diversity of data systems for "dealing with data at scale" beyond traditional databases

Goals of the Class

- **A high-level overview of data systems for various stages of data science, from data preparation to actionable insight**
- A modern take on a database class that reflects the rich diversity of data systems for "dealing with data at scale" beyond traditional databases
- Takeaways:
  - "Pick the right tool for the job" – when to use which data system.
  - Evaluating data systems: usability, power, scalability
  - The concepts underlying the tools
- This class will **not** teach you how to install or use tool X, Y, or Z
  - Unfortunately, the data systems are constantly evolving, and any such knowledge is likely to be stale in a couple of years
  - Instead, we will emphasize fundamental concepts, which will remain timeless

Relationship to INFO257 (Database Sys.)

- This class will be roughly equivalent to INFO257 (Database Systems)
- If you are a MIMS student, you are free to pick either one to fulfill requirements.

- INFO 257 is a standard, classic, and polished take on database systems;
- INFO 290 is a more modern take, emphasizing user-centered and end-to-end lifecycle aspects
  - but beware of hiccups that come with a new class!

Please Beware of Hiccups!

- Three problems:
  - A new class
  - A class on new non-traditional material
  - The first time I’m teaching at Berkeley!

Please Beware of Hiccups!

- Three problems:
  - A new class
  - A class on new non-traditional material
  - The first time I'm teaching at Berkeley!
- So expect problems to arise! (And please be tolerant!)
  - Bugs, lack of clarity, changing instructions, …

- Please feel free to provide feedback to Doris and/or me at any time
- From our end, we will be as tolerant as we can

Grading Breakdown

- Assignments = 60%
  - 3 Written, each counting for 10%, evaluating roughly a 1/3rd of the class material each
  - 2 Programming-oriented, each counting for 15%, typically evaluating the use of one or more data systems each.
- Project = 35%
  - Implementation or research oriented
- Class participation = 5%

Assignments

- 3 Written, each counting for 10%, evaluating roughly a 1/3rd of the class material each
  - Several questions testing what was covered in class and extending it
  - The primary means of evaluating your understanding of class material
- 2 Programming-oriented, each counting for 15%, typically evaluating the use of one or more data systems each.
  - Hands-on experience
  - To be done in teams of two (but talk to us if you need exceptions)

Late Policy

- Up to **four late days**, to be used in any way that makes sense, without any explanation.
- Any submission after the deadline will be rounded up to the closest day (i.e., 24 hour), so even one hour late will count as the use of a full day.
- After four late days, students will lose 1% of the grade per day late.
- In case of medical issues, please contact the instructors as soon as you are able.

Project

- Build/design/test something new and cool!
- Can be research or development oriented
  - Research-oriented: doing work that might lead to a paper
    - Should be "original"
  - Dev-oriented: pick a real-world application and address it end-to-end using data systems
- Ideally, teams of two
- Midterm report (5%, due 3/20 midnight)
- Final report + presentation (30%, report due: 05/01 midnight; presentation slides due: 4/27 prior to class)

Project Option 1: Research-Oriented

- Goal: have something "publishable" (ish) for a database or HCI venue
- Spectrum of contributions:
  - Mainly algorithmic
  - Mainly system or tool-building oriented
  - Or both
- Important to build on state of the art and extend it
  - Simply reproducing existing work not sufficient
  - Evaluating why it is "better" than prior work is needed
    - Usability, Performance, Accuracy, Expressiveness (or multiple)
  - Getting your hands dirty is a must!

Project Option 1: Research-Oriented

- Rough phases:
  - Identify problem (we can help!)
  - Explore related work/tools
  - Prototype
  - Build & Evaluate
  - Write ”paper”

Project Option 1: Research Oriented

- Build a new scalable graph visualization system
- Design a spreadsheet-computational notebook hybrid
- Develop an extension to databases to support seamless schema changes
- Design a real-time spreadsheet system
- Build an automatic dashboarding tool
- …

Project Option 2: Application-Oriented

- Goal: developing a data systems application
  - Extending one or more "data systems" to a real application
  - Applying techniques from class in a "hands-on" way
- Rough phases:
  - Identify problem (we can help!)
  - Mock-up design and requirements
  - Prototype
  - Build & Evaluate
  - Write "report"

Project Option 2: Application-Oriented

- Building a historical news archive querying database
- Developing a new medical record management database
- Designing the database of MIMS capstone projects
- Develop a course-textbook swap portal
- Design an interface to support retrieval of legal documents
- Design a scalable protein-protein interaction visualizer
- …

Class Participation

- Goals: assess understanding, get feedback, make the class more lively
- Not essential that
  - you ask "good" questions
  - answer questions "correctly"
  - make "intelligent" points
  - you have to attend and be super engaged every class
- Any participation is good participation!

- Hard to come up with a rubric for class participation, but I promise to not be dogmatic about this, nor grade strictly… the goal is to just ensure that you are engaged with the class material

Any Questions?

- Next, an overview of class content

Data is Central to Our Everyday Lives

Data has the potential to inform decisions and improve lives across sectors

- Medicine
- Government
- Science
- Technology
- Commerce
- Manufacturing
- Education
- Journalism
- Social Sciences
- …

Data is Central to Our Everyday Lives: But…

There's a lot of it!

Every minute there's
- 0.5M tweets tweeted
- 4M You Tube videos watched
- 120 new LinkedIn profiles created
- 16M text messages sent
- 600 new page edits to Wikipedia
- 18M new requests to The Weather Channel
- 50K new Uber rides

Data is Central to Our Everyday Lives: As a result…

- "Organizations are drowning in data, but starving for insight."
  [Forrester report "Digital Insights: The New Currency of Business", 2016]
- "An organization with data but no one to analyze it cannot take advantage of it."
  [Hal Varian "AI, Economics, & Industrial Organization", 2018]
- "40% of majors need to be data-savvy, up from 10%."
  [McKinsey Institute report "Age of Analytics: Competing in a Data-driven World", 2017]

The Goal of this Class: Becoming "Data Savvy"!

- This class is meant to teach you the basics of data engineering, i.e., how to store, manage, structure, clean, query, and make sense of data at scale
- A "systems/engineering"-oriented complement to your more algorithmic machine learning classes (which is also essential for data savviness)

Algorithms (ML/Stats)

Systems/Engineering

Another Goal: Emphasizing "*Human-in-the-loop*"

Why the fuss about humans?

- Humans are the ones analyzing data
- Reasoning about them "in the loop" of data science is crucial
- Traditional database research and practice ignores the human aspects!

The Goal of this Class: Becoming "Data Savvy"!

- We start with **classical database systems**
- And then move to non-traditional data engineering
  - Streaming systems
  - Graph processing systems
  - Column stores
  - Visualization systems
  - Dataframe systems
  - …

Algorithms (ML/Stats)

Systems/Engineering

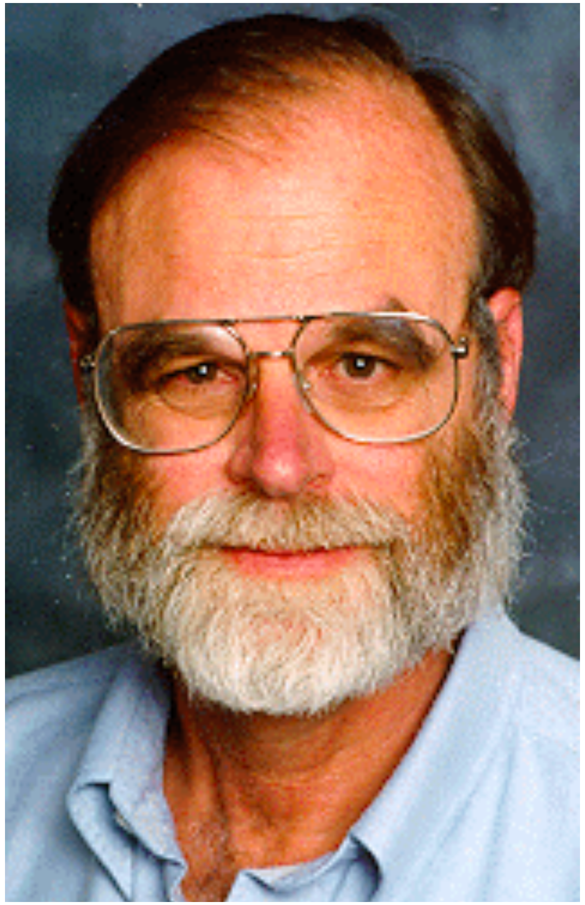
Classical Database Systems? What are those?

- The database systems field began in the 60s…
- Really hit its stride in the 70s thanks to a clean new ”data model” – a way of representing and thinking about data
  - The ”relational” model: essentially a model of relations (or sets)
  - Led to the field of relational databases, with several commercial and open-source database systems
- Led to several software systems “powered” by relational databases
  - Banking systems
  - Flight reservation systems
  - Payment systems
- Relational databases are still at the “core” of the technology stack of most companies today.

Classical Database Systems: A Brief Berkeley Side Note…

The last two Turing awards for databases came from Berkeley!

- Jim Gray (‘98) was the first CS PhD student from Berkeley; pioneer of transaction processing
- Mike Stonebraker (‘14) was faculty for many years at Berkeley; influenced or developed many popular database systems



Classical Database Systems: Are we done?

Most modern data scientists work outside of (classical) relational databases.

Most use a combination of files + scripts + Excel + Python/R

Discussion Question: Why is that?

(Since many of you have experience doing "data science" in industry)

And if you're encountering databases for the first time, don't worry, you'll know this first-hand later on in the class!

Classical Database Systems: Are we done?

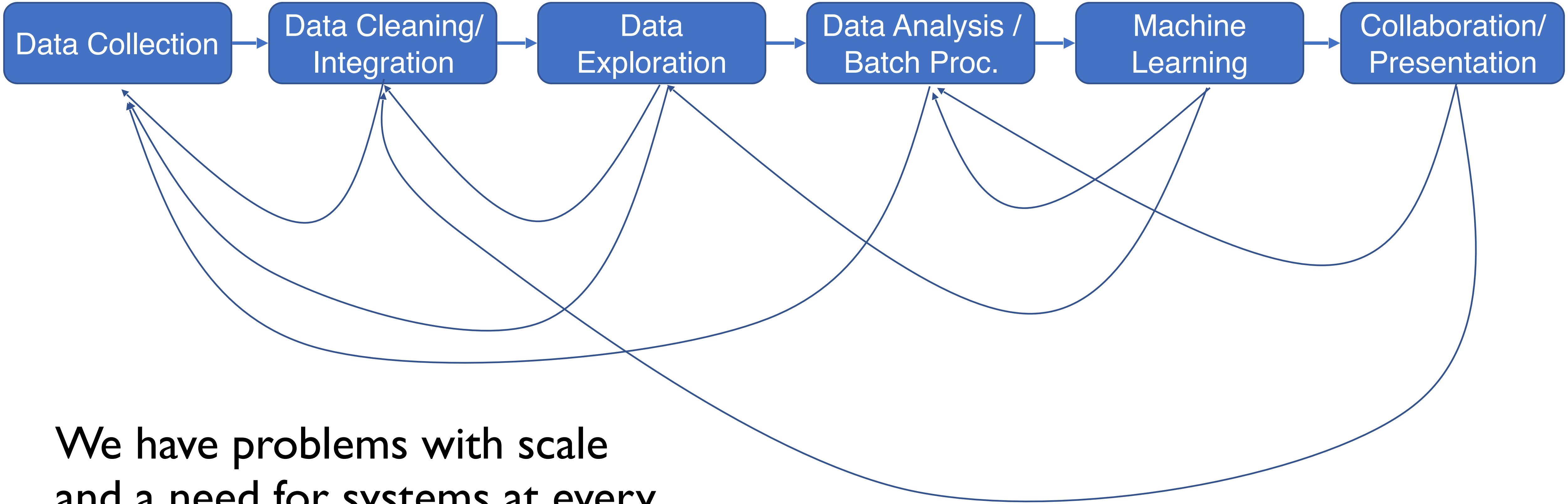
Most modern data scientists work outside of relational databases.

Discussion Question: Why is that?

- Hard to use/learn
- Does not scale
- Not easy to do quick and dirty data analysis
- Does not deal well with ill-formatted, noisy, or unstructured data
- Loading times are high
- Cannot do machine learning or data analysis easily
- Does not do well with special data formats

At a high-level, they are overly rigid, and do not support other stages of the data science lifecycle well.

The Modern Data Science Lifecycle



We have problems with scale and a need for systems at every single stage!

Beyond Classical Database Systems

The second half of the class will cover a range of non-traditional data systems

- Data preparation: dealing with noisy data
  - Semi-structured and unstructured data
  - Data cleaning and integration
- Data exploration: performing "quick-and-dirty" analysis
  - Data frames and Spreadsheets
  - Summarization, OLAP, and visualization
- Batch analysis: doing ad-hoc analysis at scale
  - Map-reduce and parallel data processing
  - Column stores and data compression
- Real-time analysis: dealing with data that is evolving or too large
  - Streaming systems
  - Sketching and approximation
- Specialized analysis requirements
  - Graph processing and machine learning
  - Security and privacy

Recap

- Data is central to our lives
- Data-savviness is the future!
  - Classical (relational) database systems
  - Non-traditional database systems and the data life-cycle
- **Next**: why relational databases

Questions?

- Let's start with Classical (Relational) Databases!

First, what is a Database System?

DBMS (Data Base Management System)
= Database System = Database

System to manage, maintain, query, interact with, transact with data.

More loosely, database systems are used for “data management”

Task: Build a Banking "Data Management" System from Scratch w/o a Database

Goal: Manage customers, accounts, joint accounts, transfers, transactions, interest rates.

Let's say I implement this system using C++/Java/Python, without using a database system

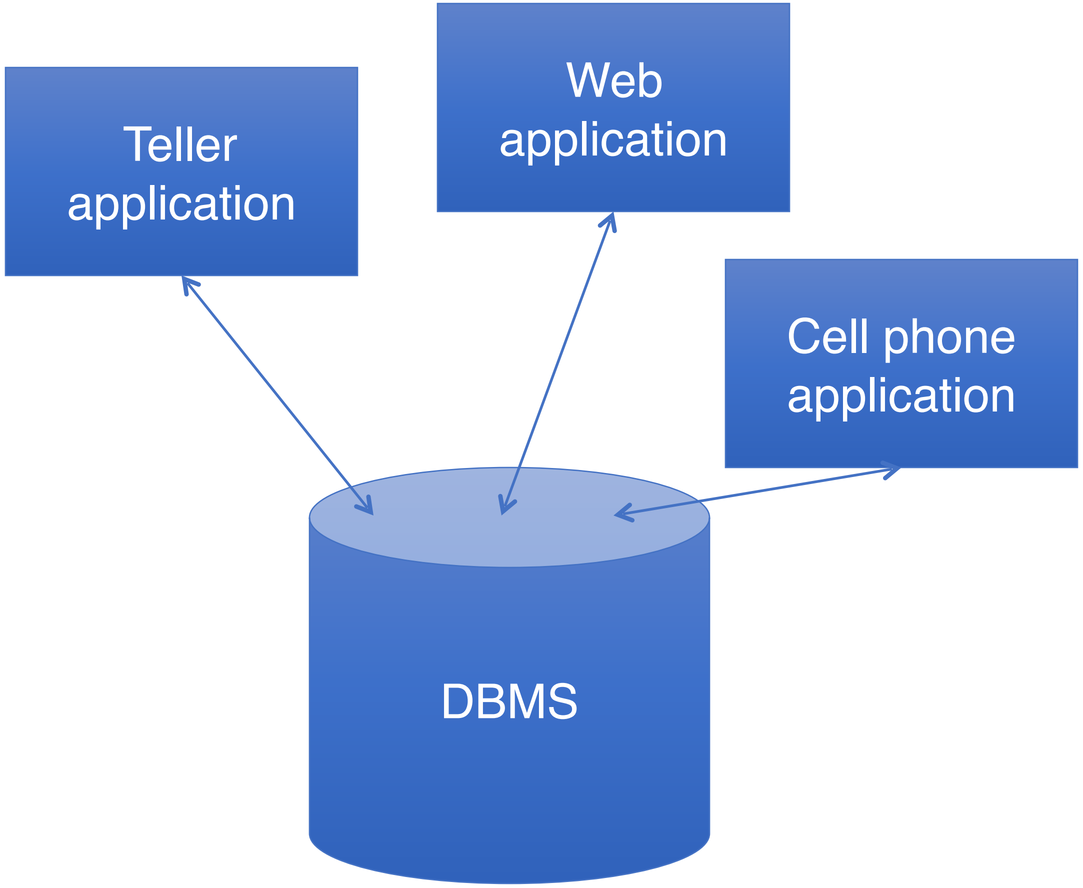
Think like a designer: what aspects do we need to worry about?

Aspects to worry about

- Deal with lots of data
- Be fast
- Don't lose information
- Allow multiple users
- Stay consistent
- Easy to use

The Database Approach: Abstraction

- Abstract out all of the data management functionality into a separate layer
- Many applications can access it
- Turns out this "separate layer" keeps turning up in many many many scenarios
- Makes sense to abstract it out



Database Management System (DBMS)?

One possible definition:

System for providing EFFICIENT, CONVENIENT, and SAFE, MULTI-USER storage of and access to MASSIVE amounts of PERSISTENT data

Database Management System (DBMS)?

System for providing EFFICIENT, CONVENIENT, and SAFE, MULTI-USER storage of and access to MASSIVE amounts of PERSISTENT data

Data: information on accounts, customers, balances, current interest rates, transaction histories, etc.

MASSIVE: many TBs at a minimum for big banks; more if we keep history of all transactions; even more if we keep images of checks!

Database Management System (DBMS)?

System for providing EFFICIENT, CONVENIENT, and SAFE, MULTI-USER storage of and access to MASSIVE amounts of PERSISTENT data

PERSISTENT: data lives on, beyond programs that operate on it, even on system shutdown and power failure.

➔ Can't store data in memory, we have to rely on stable storage (disk, flash)

Database Management System (DBMS)?

System for providing EFFICIENT, CONVENIENT, and SAFE, MULTI-USER storage of and access to MASSIVE amounts of PERSISTENT data

MULTI-USER: many people/programs accessing same database, or even same data, simultaneously
➔ Need controls

- Alice @ ATM1: withdraw $100 from account #002
```
        get balance from database;
                if balance >= 100
                        then balance := balance – 100; // dispense cash
          update balance in database;
```
- Bob @ ATM2: withdraw $50 from account #002
```
                get balance from database;
                if balance >= 50
                        then balance := balance - 50; // dispense cash
          update balance in database;
```
- Initial balance = 100. What's the ideal case? What could go wrong?

Database Management System (DBMS)?

System for providing EFFICIENT, CONVENIENT, and SAFE, MULTI-USER storage of and access to MASSIVE amounts of PERSISTENT data

SAFE:

- from system failures. E.g., money should not disappear or appear from the account, due to a power failure!

  Bob @ ATM2: withdraw $50 from account #002

  ```
  get balance from database;
  if balance >= 50
          then balance := balance - 50; // dispense cash
  update balance in database;
  ```

  Power failure right here

- from malicious users

Database Management System (DBMS)?

System for providing EFFICIENT, CONVENIENT, and SAFE, MULTI-USER storage of and access to MASSIVE amounts of PERSISTENT data

CONVENIENT:

- simple commands to debit account, get balance, write statement, transfer funds, etc.
- also unpredicted queries should be easy
- shouldn't require complex 100s of lines of code

EFFICIENT:

- don't search all files in order to get balance of one account, get all accounts with low balances, get large transactions, etc.

Why Direct Implementation is Super Hard/Won't Work

- Early DBMS evolved from file systems
- Provided storage of MASSIVE amounts of PERSISTENT data, to some extent
- SAFE?
  - when system crashes, no guarantees on how program may behave: we may lose data
- EFFICIENT?
  - Does not intrinsically support fast access to data whose location in file is not known: will need to write custom code

Why Direct Implementation is Super Hard/Won't Work

- Early DBMS evolved from file systems
- Provided storage of MASSIVE amounts of PERSISTENT data, to some extent
- CONVENIENT?
  - need to write a new C++/Java program for every new query
  - small changes to structure entails changing file formats; need to rewrite virtually all applications
- MULTI-USER ACCESS?
  - limited protection
  - need to worry about interfering with other users

This is why Database Systems were Invented!

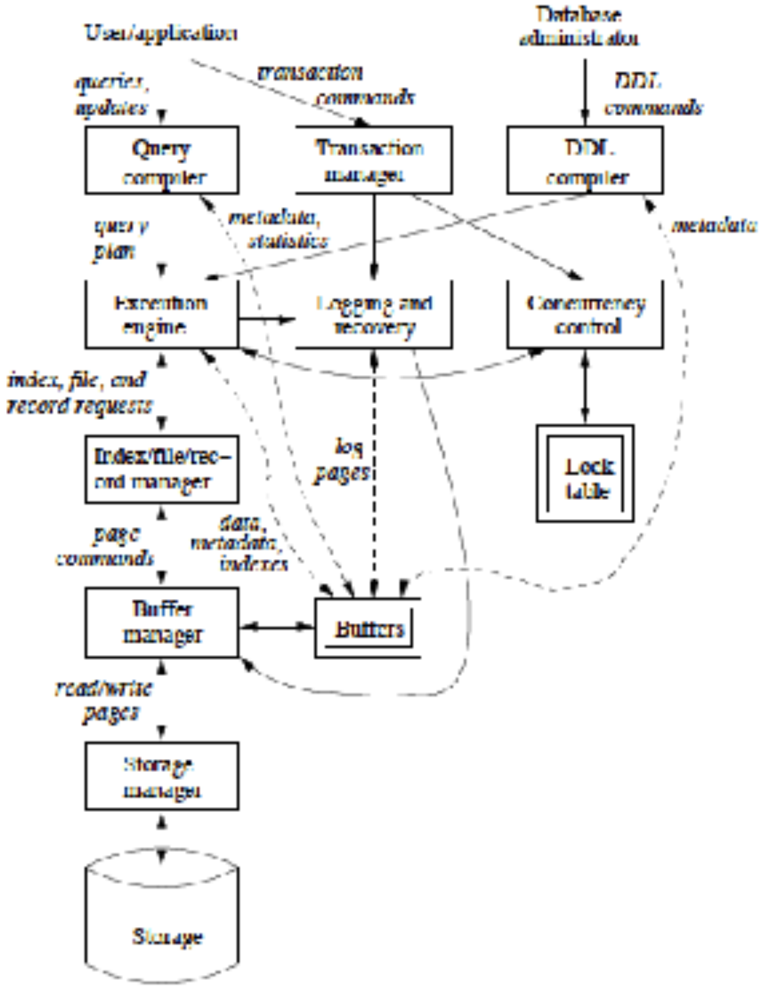
- Buy, install, set up for particular application or applications

- Major relational database vendors:
  - Oracle
  - Microsoft SQL Server
  - IBM DB2
- Open source relational databases:
  - Postgres
  - MySQL
  - Sqlite

A Complex Piece of Software

- We will cover ”bird’s eye view” of database systems, emphasizing the user perspective

- For the “database internals” perspective, consider taking CS186/286 and beyond!



Recap

- Data is central to our lives
- Data-savviness is the future!
  - Classical database systems: today's focus!
  - Non-traditional database systems and the data life-cycle
- Notion of a DBMS
- Requirements for a DBMS
- Example DBMSs, and their complexity
- **Next**: the relational data model

Fundamentals: The Relational Data Model

- A DBMS stores many databases
- Each database has many relations
- A relation = a set of tuples, each w/ a predefined set of attributes

Table/Relation (an instance of)

Attributes/Columns

Tuples/ Rows/ Records

Name	Price	Category	Manufacturer
Gizmo	$19.99	gadgets	GizmoWorks
Power Gizmo	$29.99	gadgets	GizmoWorks
SnapPea Camera	$149.99	photography	Canon
Hi Phone	$700.99	phone	Snapple

Fundamentals: The Relational Data Model

- Each relation is defined by a set of *attributes*
  - Each attribute has a type, called *domain*
  - The domain must be atomic: string, integer, … (but many DBs violate this)
- Each relation contains a *set of tuples or records*
  - Each tuple has values for each attribute

Table/Relation (an instance of)

Attributes/Columns

Tuples/ Rows/ Records

Name	Price	Category	Manufacturer
Gizmo	$19.99	gadgets	GizmoWorks
Power Gizmo	$29.99	gadgets	GizmoWorks
SnapPea Camera	$149.99	photography	Canon
Hi Phone	$700.99	phone	Snapple

Equivalent Representations

- Since a relation has a *set of attributes* and a *set of tuples*, we can reorder both and not change the relation

Name	Price	Category	Manuf.
Hi Phone	$700.99	phone	Snapple
SnapPea Camera	$149.99	photography	Canon
Power Gizmo	$29.99	gadgets	GizmoWorks
Gizmo	$19.99	gadgets	GizmoWorks

Name	Category	Price	Manuf.
Gizmo	gadgets	$19.99	GizmoWorks
SnapPea Camera	photography	$149.99	Canon
Power Gizmo	gadgets	$29.99	GizmoWorks
Hi Phone	phone	$700.99	Snapple

Relational Schema & Instance

- Schema: The structure, format, or scaffolding
  - Schema for a Relation:
    - Relation names plus attribute names & domains
    - Product (Name String, Price Float, Category String, Manuf. String)
  - Schema for a Database:
    - Schemas for many relations
    - Product (…)
    - Vendor (Name String, Address String, Phone Integer)
- Instance: Specific instantiation of the Relation or Database
  - A Relation or Database with "values filled in"

Name	**Price**	**Category**	**Manuf.**
Hi Phone	$700.99	phone	Snapple
SnapPea Camera	$149.99	photography	Canon

Recap: The Relational Data Model

- Data is stored in relations
- Relations are defined by a set of attributes/columns w/ specific domains
  - Together this defines the "schema" of a relation
- At any time, relations comprise of a set of tuples/rows/records
  - This constitutes the "instance" of a relation
- Databases are collections of relations

- Next, we will define the Relational Algebra: the operations that manipulate relations (the operands)
  - Parallels to linear algebra or set algebra

Recap

- Data is central to our lives
- Data-savviness is the future!
  - Classical database systems: today's focus!
  - Non-traditional database systems and the data life-cycle
- Notion of a DBMS
- Requirements for a DBMS
- Example DBMSs, and their complexity
- The relational data model
- **Next**: relational algebra

Set Operations: Union and Difference

- Union & Difference
  - Binary operations from set algebra
  - Recall that relations are sets
- Union
- Notation: $R1 \cup R2$
  - R1 and R2 must have same schema
- Output schema: same as input
- Example: CurrentStudents (Name, Addr) U Alumni (Name, Addr)
- Difference is similar $R1 - R2$: tuples in R1 and not in R2

Unary Op.: Selection (Cross out rows)

- Returns all tuples that satisfy a condition (also called filter)
- Notation $\sigma_c(R)$
  - Condition c can involve attributes in R with =, >, <, AND, OR, etc.
- Output schema: same as input
- Find all employees with salary greater than $40,000 from an Employee relation

Unary Op.: Selection (Cross out rows)

- Returns all tuples that satisfy a condition (also called filter)
- Notation $\sigma_c(R)$
  - Condition c can involve attributes in R with =, >, <, AND, OR, etc.
- Output schema: same as input
- Find all employees with salary greater than $40,000 from an Employee relation

$$\sigma_{Salary>40000}(Employee)$$

Another Selection Example

SSN	Name	DepID	Salary
839930303	Avi	1	100000
182902033	Sarah	2	90000
383930032	Farida	2	120000
123233849	Jacob	1	70000

- Find employees with Salary greater than 100000 and Department ID 2.

Another Selection Example

SSN	Name	DepID	Salary
~~839930303~~	~~Avi~~	~~1~~	~~100000~~
~~182902033~~	~~Sarah~~	~~2~~	~~90000~~
383930032	**Farida**	**2**	**120000**
~~123233849~~	~~Jacob~~	~~1~~	~~70000~~

- Find employees with Salary greater than 100000 and Department ID 2.

$$\sigma_{Salary>100000\ AND\ DepID=2}(Employee)$$

Unary Op.: Projection (Cross Out Columns)

- Notation $\Pi_{A_1, A_2, \ldots, A_n}(R)$
- Where
$$R(B_1, B_2, \ldots, B_m)$$
$$A_1, A_2, \ldots, A_n \subseteq B_1, B_2, \ldots, B_m$$
- That is, you can only sub select from columns that are present in R
- Output Schema $(A_1, A_2, \ldots, A_m)$
- Example: project out only the names from the Employee (SSN, Name, DepID, Salary)

Unary Op.: Projection (Cross Out Columns)

- Notation $\Pi_{A_1, A_2, \ldots, A_n}(R)$
- Where
  $$R(B_1, B_2, \ldots, B_m)$$
  $$A_1, A_2, \ldots, A_n \subseteq B_1, B_2, \ldots, B_m$$
- That is, you can only sub select from columns that are present in R
- Output Schema $(A_1, A_2, \ldots, A_m)$
- Example: project out only the names from the Employee (SSN, Name, DepID, Salary)
  $$\Pi_{Name}(Employee)$$
- Note: **Deletes any duplicate rows to return a set!**

Another Projection (+ Selection) Example

SSN	Name	DepID	Salary
839930303	Avi	1	100000
182902033	Sarah	2	90000
383930032	Farida	2	120000
123233849	Jacob	1	70000

- Find names and salaries of Employees whose Department ID is 2.

Another Projection (+ Selection) Example

SSN	Name	DepID	Salary
839930303	Avi	1	100000
182902033	Sarah	2	90000
383930032	Farida	2	120000
123233849	Jacob	1	70000

Name	Salary
Sarah	90000
Farida	120000

- Find names and salaries of Employees whose Department ID is 2.

$$\Pi_{Name,Salary}(\sigma_{DepID=2}(Employee))$$

Other Operators… Next week!