CS 186
Introduction to Database Systems

Alvin Cheung

Aditya Parameswaran

Essential Queries

- **Why** take this class?
- **What** is this class all about?
- **Who** is running this?
- **How** will this class work?

WHY?

Why?

- This class will cover how to develop systems to *efficiently manage, maintain, process, query, transact with, and make sense of data*

Why? Reason #1: Utility

- This class will cover how to develop systems to *efficiently manage, maintain, process, query, transact with, and make sense of data*

- These systems are incredibly useful!
- You're likely using such systems under the hood when you're
  - Booking a hotel, a Lyft, an AirBnB, or a flight
  - Liking a post on Twitter or Facebook
  - Figuring out where to eat from Yelp, GrubHub, or Caviar
  - Posting on Piazza or Slack
  - Transferring money or making a stock trade
  - Making a purchase on Etsy or Amazon
  - <Your next app here>
- Virtually every app is backed by such systems

Why? Reason #1: Utility

- This class will cover how to develop systems to *efficiently manage, maintain, process, query, transact with, and make sense of data*

- Virtually every app is backed by such systems
- These systems are the backbone of modern **science**
  - Genomics, astronomy, medicine, meteorology, …
  - All of which generate massive volumes of data and a need to make sense of it
  - These systems are the key to some of our most pressing societal “grand challenges”: climate change, public health, …

Why? Reason #1: Utility

- This class will cover how to develop systems to *efficiently manage, maintain, process, query, transact with, and make sense of data*

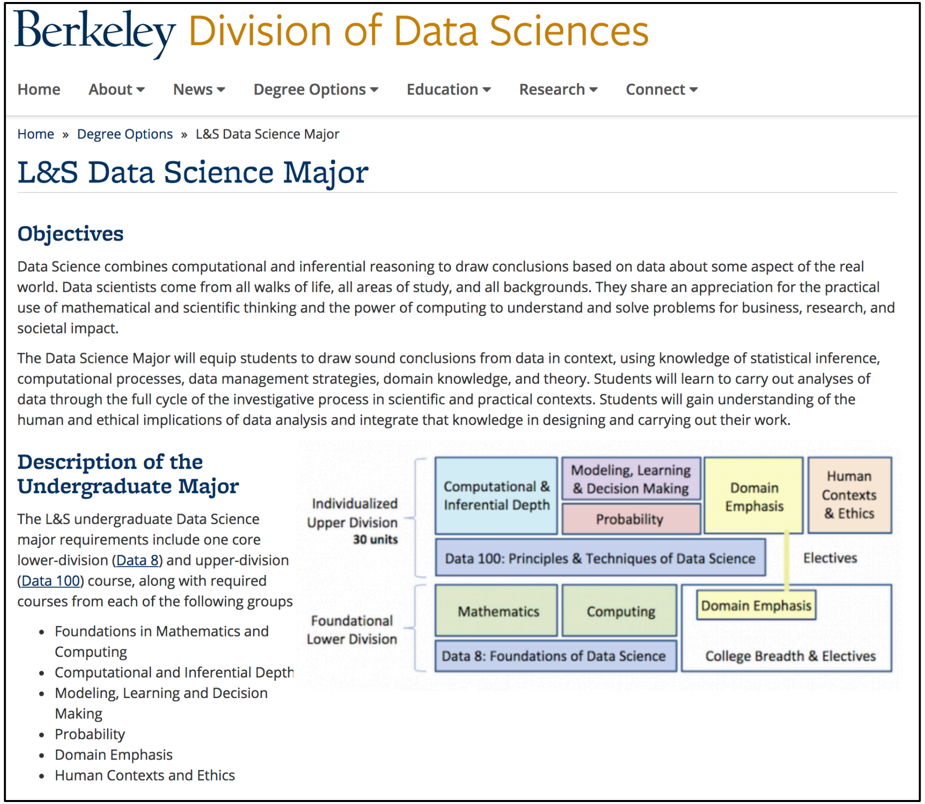
- Virtually every app is backed by such systems
- These systems are the backbone of modern science
- The *principles* taught in this class will play a role in any setting with data at scale [i.e., most settings!]

Why? Reason #2: Centrality

- Data is at the center of modern society
  - Huge promise, but many potential concerns
    - Use and misuse
  - Timely debates about the use of data, privacy, security, ethics, fairness, ….

Berkeley's New(ish) Data Science Major

https://data.berkeley.edu/degrees/data-science-ba



Berkeley Division of Data Sciences

Home | About ▾ | News ▾ | Degree Options ▾ | Education ▾ | Research ▾ | Connect ▾

Home » Degree Options » L&S Data Science Major

L&S Data Science Major

Objectives

Data Science combines computational and inferential reasoning to draw conclusions based on data about some aspect of the real world. Data scientists come from all walks of life, all areas of study, and all backgrounds. They share an appreciation for the practical use of mathematical and scientific thinking and the power of computing to understand and solve problems for business, research, and societal impact.

The Data Science Major will equip students to draw sound conclusions from data in context, using knowledge of statistical inference, computational processes, data management strategies, domain knowledge, and theory. Students will learn to carry out analyses of data through the full cycle of the investigative process in scientific and practical contexts. Students will gain understanding of the human and ethical implications of data analysis and integrate that knowledge in designing and carrying out their work.

Description of the Undergraduate Major

The L&S undergraduate Data Science major requirements include one core lower-division (Data 8) and upper-division (Data 100) course, along with required courses from each of the following groups

- Foundations in Mathematics and Computing
- Computational and Inferential Depth
- Modeling, Learning and Decision Making
- Probability
- Domain Emphasis
- Human Contexts and Ethics

Individualized Upper Division **30 units**

Computational & Inferential Depth

Modeling, Learning & Decision Making

Probability

Domain Emphasis

Human Contexts & Ethics

Data 100: Principles & Techniques of Data Science

Electives

Foundational Lower Division

Mathematics

Computing

Domain Emphasis

Data 8: Foundations of Data Science

College Breadth & Electives

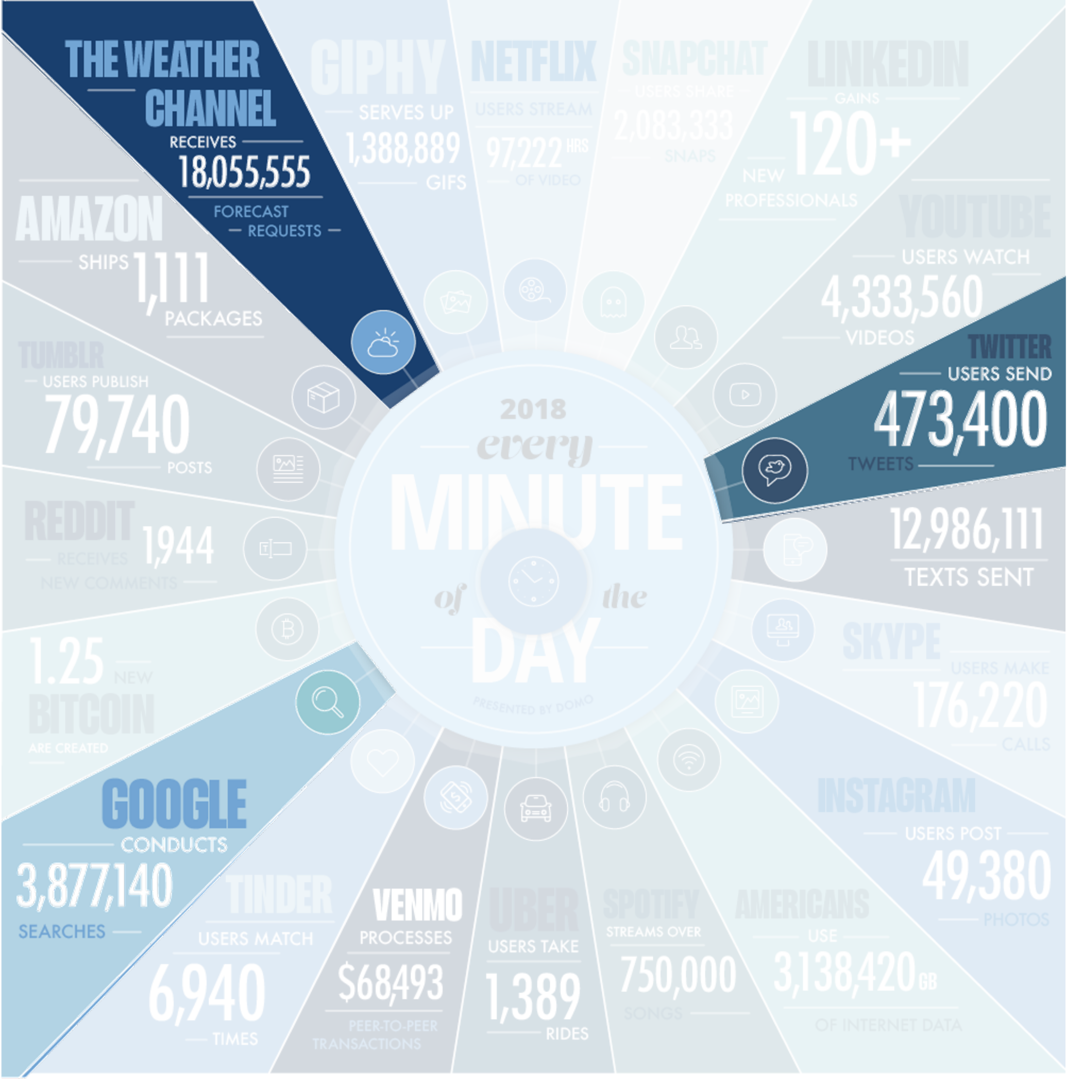
Why? Reason #2: Centrality

- Data is at the center of modern society
  - Huge promise, but many potential concerns
    - Use and misuse
  - Timely debates about the use of data, privacy, security, ethics, fairness, ….

- Data infrastructure (i.e., the systems we will study/develop) determines what's possible and what is feasible
- As data is central, the infrastructure to manage data is just as central

Why #3? The Core of Computing

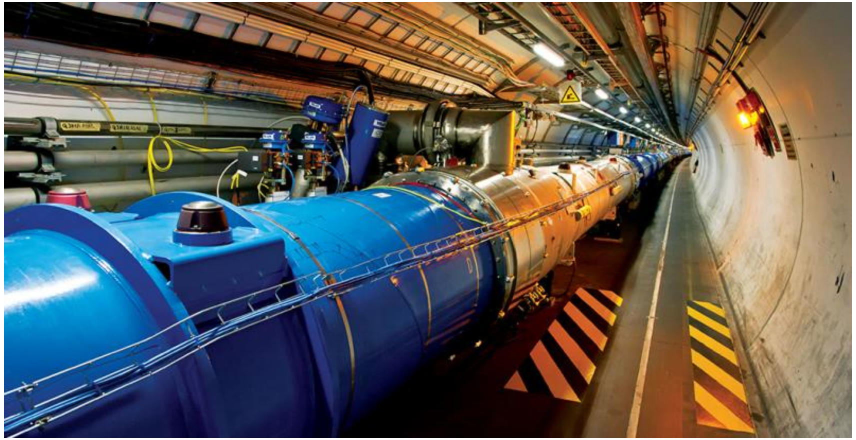
- Data growth will continue to outpace computation
  - Key bottleneck in the future: data processing
- Systems for Data at Scale: the core of modern computing

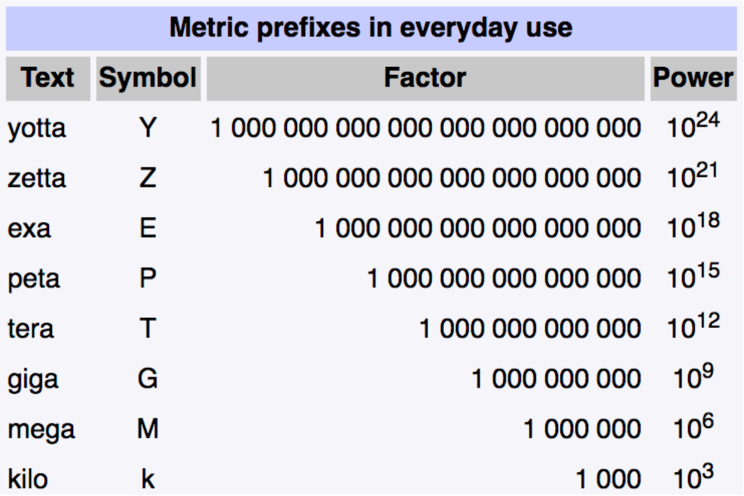


Every Minute!

https://www.domo.com/learn/data-never-sleeps-5

Scale of Scientific Data





Metric prefixes in everyday use			
Text	**Symbol**	**Factor**	**Power**
yotta	Y	1 000 000 000 000 000 000 000 000	10^{24}
zetta	Z	1 000 000 000 000 000 000 000	10^{21}
exa	E	1 000 000 000 000 000 000	10^{18}
peta	P	1 000 000 000 000 000	10^{15}
tera	T	1 000 000 000 000	10^{12}
giga	G	1 000 000 000	10^{9}
mega	M	1 000 000	10^{6}
kilo	k	1 000	10^{3}

Large Hadron Collider, CERN

- Raw data: 1MB/event. 600,000,000 events/sec.
  = 1.9×10^{22} bytes/year = **19 ZettaBytes/year**
- Downsampled: 25GB/sec = 7.88×10^{17} bytes/year = **788 PetaBytes/year**
- Downsampled further: 1050MB/sec = $3.3*10^{16}$/year = **33 PetaBytes/year**

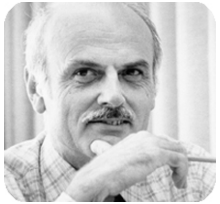
https://home.cern/about/computing/processing-what-record

Why #3? The Core of Computing

- Data growth will continue to outpace computation
  - Key bottleneck in the future: data processing
- Systems for Data at Scale: the core of modern computing
- Techniques you learn in this class underlie many topics in computing
  - Abstraction, representation & modeling, reuse, rapid access, declarativity, …

Why #4? Tons of Opportunities in Academic Research

Charles Bachman, 1973
IDS and CODASYL



Ted Codd, 1981
Relational model

Jim Gray, 1998
Transaction processing

Michael Stonebraker, 2014
INGRES and Postgres

Turing Awards in Data Management

Developing scalable systems for data is one of the most exciting areas of CS research!

Essential Queries, Pt 2

- **Why** take this class?
- **What** is this class all about?
- **Who** is running this?
- **How** will this class work?

What is this class all about?

- Databases?
  - What is a database?

Task: Build a Banking Data Management System from Scratch without 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

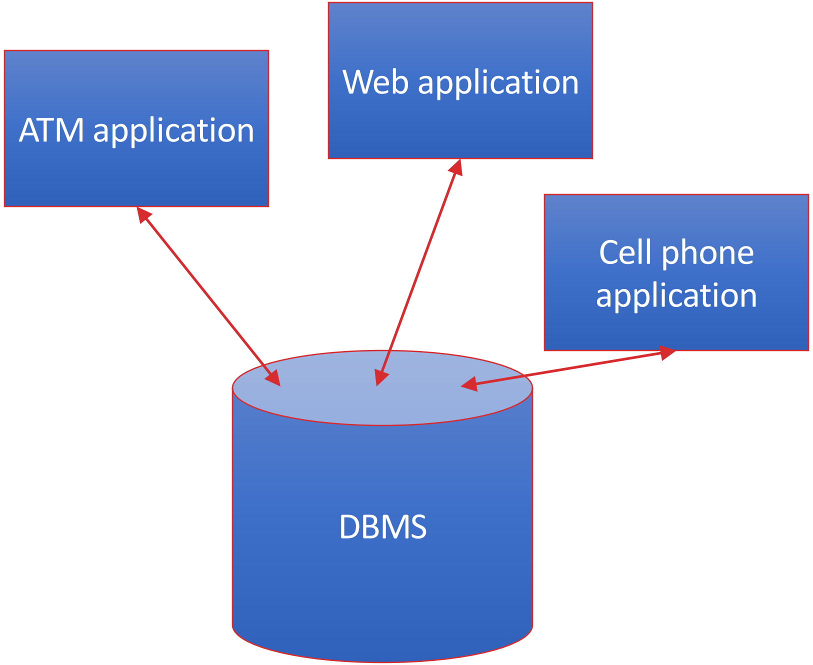
Q: *Think like a designer: what aspects do we need to worry about?*

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 System" 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 System?

One possible (but clunky!) definition:

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

Database System?

System for providing EFFICIENT, CONVENIENT, & 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 System?

System for providing EFFICIENT, CONVENIENT, & 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.

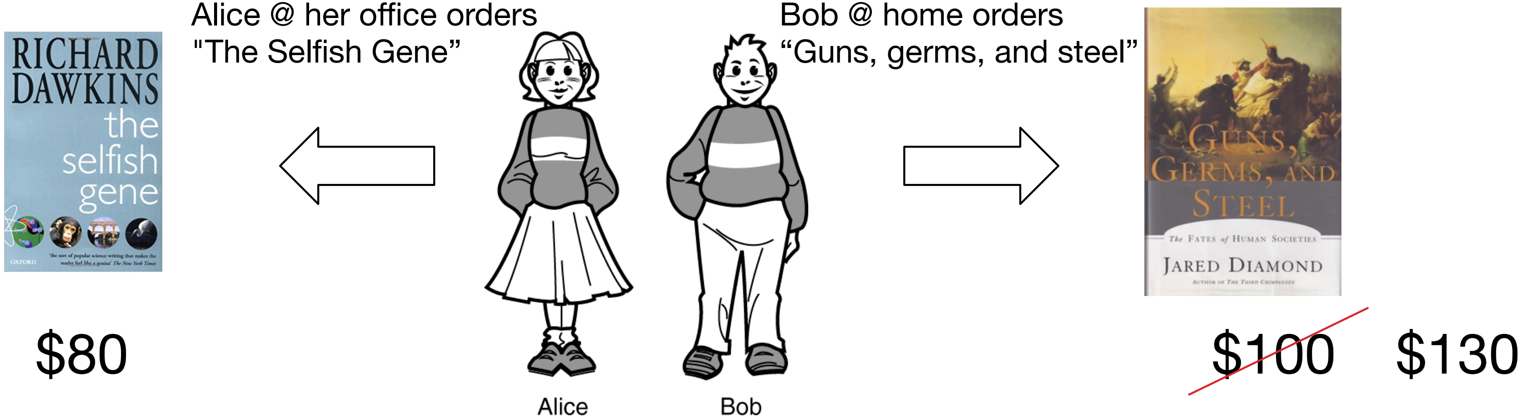
➔ Can’t store data in memory, we have to rely on stable storage (disk, flash)

Database System?

System for providing EFFICIENT, CONVENIENT, & 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 and Bob have $200 in their bank account



Alice @ her office orders "The Selfish Gene”

Bob @ home orders “Guns, germs, and steel”

$80

~~$100~~ $130

Database System?

System for providing EFFICIENT, CONVENIENT, & 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 @ ATM: withdraw $50 from account #002

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

← *Power failure here*

- from malicious users

Database System?

System for providing EFFICIENT, CONVENIENT, & 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 (of tens of millions of accounts) in order to get balance of one account, get all accounts with low balances, get large transactions, etc.

Why Direct Implementation is Hard/Won't Work

- Early database systems 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

Database System?

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

That’s why Database Systems were invented!

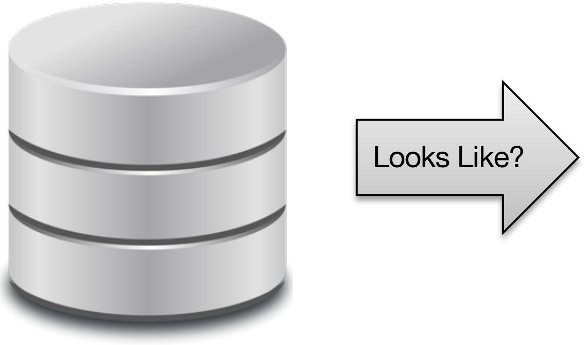
- Describe real-world entities
- Store large datasets persistently
- Query & update efficiently
- Change structure (e.g., add attributes)
- Handle concurrent updates
- Crash recovery
- Security and integrity

Databases, Database Systems and DBMSs

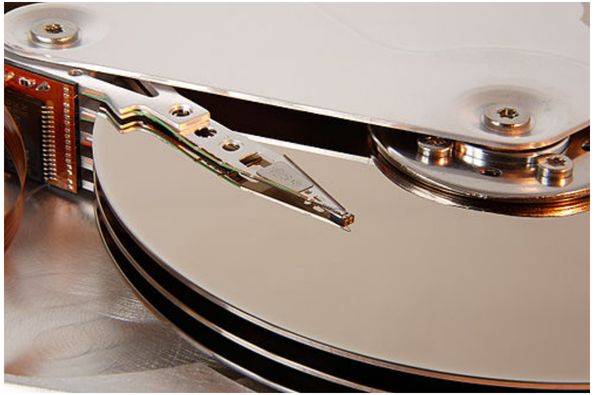
- What we’ve called a “Database System” is also known by its complete name, Database Management System (DBMS)
  - *A DBMS is software that* ***stores, manages,*** *and facilitates* ***access*** *to data.*
- On the other hand, *a database is a large, organized collection of data.*
  - This is what a database system/DBMS manages
  - But sometimes, databases are also used to refer to the database system or DBMS itself
    - Use should be clear from the context

Universal Symbol for a Database or DBMS

Why the Symbol?



Platters on a Disk Drive



What is this class all about?

- Databases?
  - What is a database?
- Database Management Systems?
- Implementation?

Examples of Database Systems

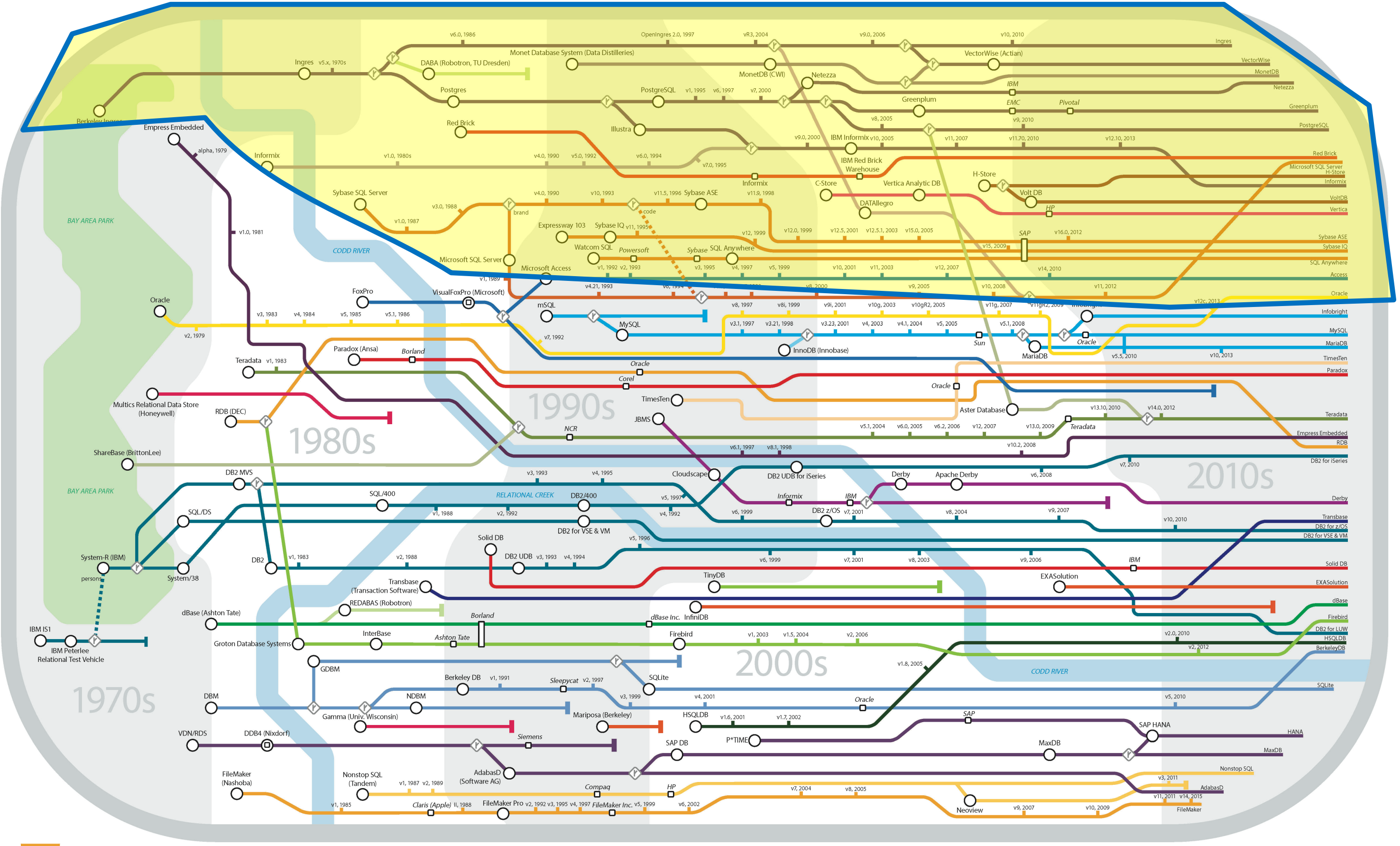
- Traditionally DBMS referred to relational database systems, or RDBMSs, or simply relational databases

ORACLE
TERADATA
Microsoft SQL Server
PostgreSQL
SQLite

- Many other non-relational database systems exist:
  - Graphs
  - Document stores
  - Key value stores
  - What else?

- We will discuss what “relational” means

Genealogy of Relational Database Management Systems

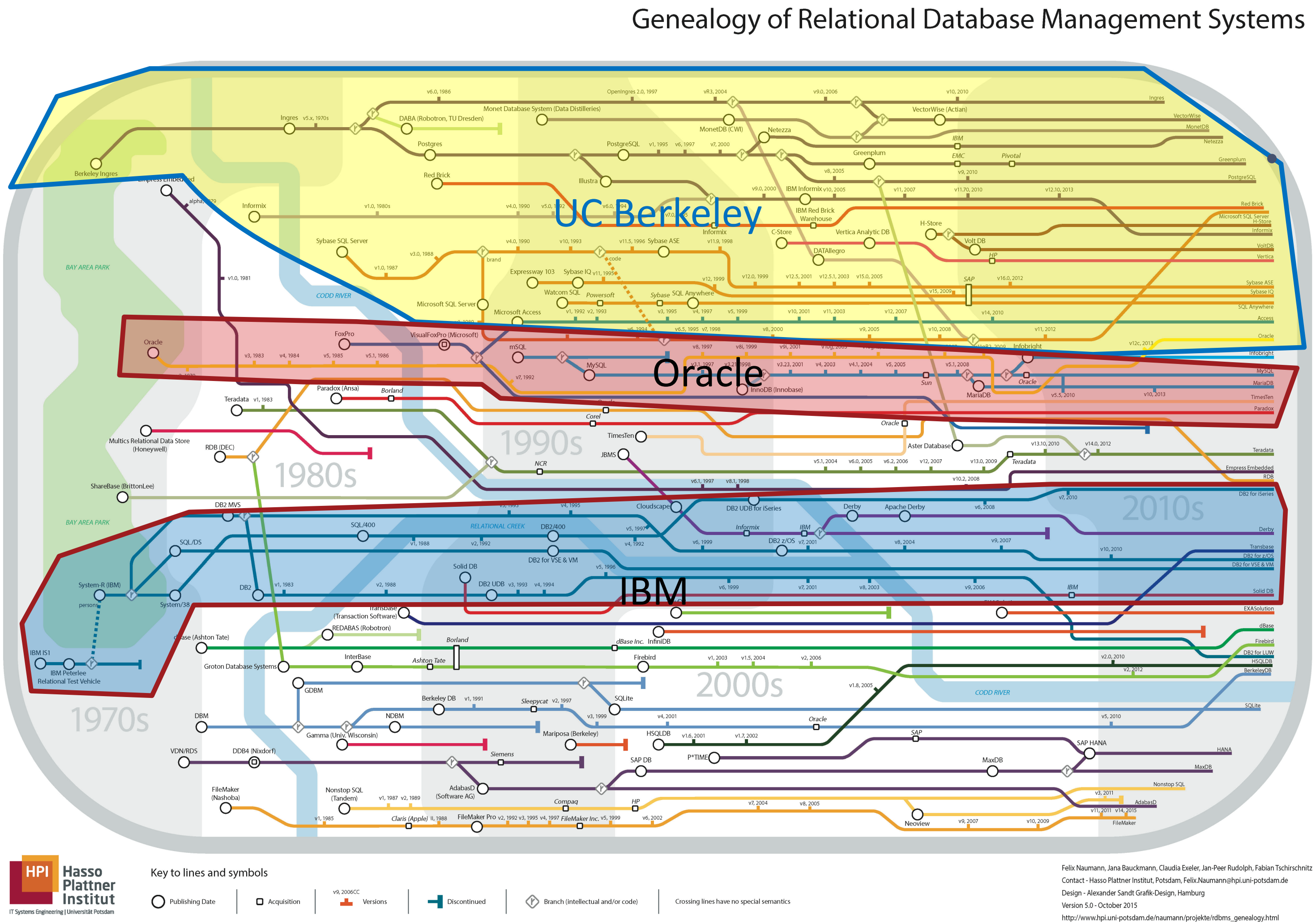


HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Felix Naumann, Jana Bauckmann, Claudia Exeler, Jan-Peer Rudolph, Fabian Tschirschnitz
Contact - Hasso Plattner Institut, Potsdam, Felix.Naumann@hpi.uni-potsdam.de
Design - Alexander Sandt Grafik-Design, Hamburg
Version 5.0 - October 2015
http://www.hpi.uni-potsdam.de/naumann/projekte/rdbms_genealogy.html

Berkeley Roots!

- Ingres/Postgres
- Sybase
- Informix



Berkeley Roots!

- Ingres/Postgres
- Sybase
- Informix

Will focus mostly on rel. database systems

- Why? Isn’t this old stuff?
  - In fact, our main textbook is rather out of date (2003!)
- But… we will focus on **Foundational System Principles** that transcends different types of DBMSs
  - Reusable ideas and components
  - Compositional approach
- Goal:
  - You will be able to **use** existing & **build new** DBMS technologies!

You will learn...

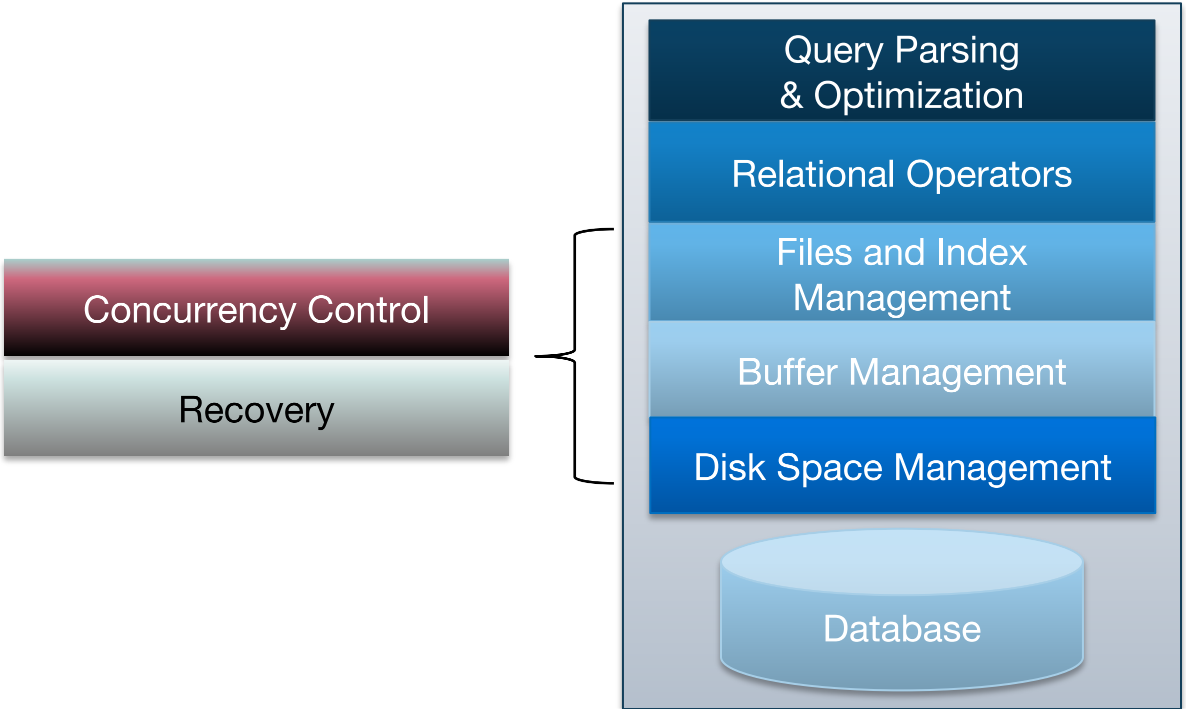
- Data Oriented Programming with SQL
- Foundations of Data System Design
  - Storage, indexing
  - Query processing and optimization
- Transactions
  - Concurrency, Consistency, Recovery
- Data Modeling
  - Application-level representations of data

Principles

- Data Independence
- Declarative Programming
- Rendezvous in Time and Space
- Isolation and consistency
- Data representations

Systems

We will examine various levels of a DBMS



What is this class all about?, cont

- Databases?
    - What is a database?
- Database Management Systems?
- Implementation?
- Big Ideas in Database Management Systems
    - Principles and Algorithms
    - System Designs
    - *What makes computer science a “science”*

Essential Queries, Pt 3

- **Why** take this class?
- **What** is this class all about?
- **Who** is running this?
- **How** will this class work?

Who Are We?

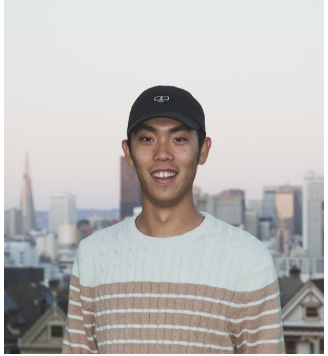
Alvin Cheung
PhD from MIT
Databases-meets-PL
Asst Prof @ Berkeley CS

Aditya Parameswaran
PhD from Stanford
Databases-meets-HCI
Asst Prof @ Berkeley CS and I School

Your Amazing Head TAs

Saurav Chhatrapati

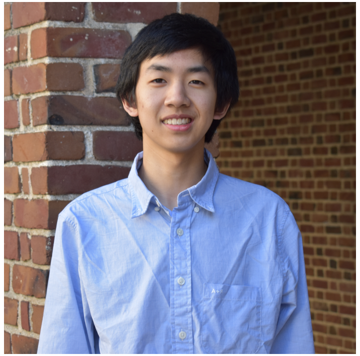
Ethan Shang



Jerry Song

Chris Wong

Your Amazing TAs


Justin
Cheng

Samy
Cherfaoui

Gabe
Fierro

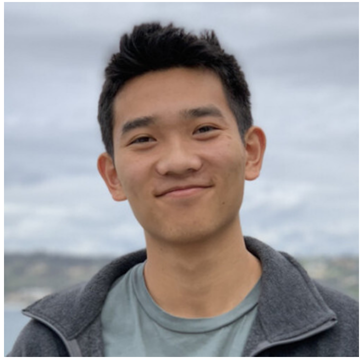
Amy
Hung

Your Amazing TAs

Kayli
Jiang

Su Min
Kim

Shreyas
Krishnaswamy



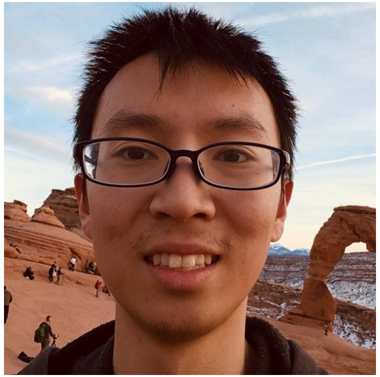
Noah
Kuo

Your Amazing TAs

Kaitlyn
Lee

Mantej
Panesar

Aditya
Ramkumar



Allen
Shen

Your Amazing TAs

Berkeley cs186

Dylan
Tran

Jennifer
Tsui

Sabrina
Zhao

You!

- This class is in your hands.
    - Use Piazza as a resource to connect with others like you
- Everything is doable, with steady work.
    - Our goal is to really help you learn the material, not stress you out
- We will help pace you
    - Weekly section worksheets, vitamins keep you on schedule
    - Weekly sections and office hours
    - Multi-week programming projects to help you hone in your skills
    - (More on all of this later)

Essential Queries, Part 4

- **Why** take this class?
- **What** is this class all about?
- **Who** is running this?
- **How** will this class work?

What is different about CS 186 this semester?

- Everything now moved online ☹
  - Lectures, sections, OHs
- Poll for students for time zones is out – just so that we are aware of where you are and make the class as convenient as we possibly can (given our resources)
- We will play by ear as we go along
  - If there are issues due to COVID-19, please feel free to raise them on Piazza

What is different about CS 186 this semester? (Part 2)

- CS186 has been taught entirely MOOC-style with videos recorded in 2018 courtesy Prof. Joe Hellerstein
- We (Alvin & Aditya) will be teaching it synchronously
  - We will cover a similar (but possibly not identical) set of concepts
  - Those videos are still available if you'd like a different perspective
- Since it's our first time teaching this class (and a class of this size!) bear with us as we figure things out. There will be hiccups.

Main Points of Information

- Newly Revamped(!) Course Website: cs186berkeley.net
  - Syllabus
  - Calendar: sections and OH
  - Lecture slides
  - HW
- Piazza discussion group
- **All this info linked on website.**

Workload: Lectures

- Two lectures per week
  - Synchronous on zoom, also recorded
  - Please try to attend if you can!
- Strongly suggest turning on your video to make the experience less dull for everyone
  - If you do so, please don't do anything you won't do in an ordinary class
  - e.g., take calls, cook meals, take a shower, …
- Please keep your audio off if you're not speaking
- If you need to ask questions, please use the "raise hand" feature: we will monitor and get to you!
- Please engage with us
  - We love questions! We love answers!

Workload

- OH start this week [but will start in full force next week]
- Vitamins: simple weekly online quizzes
  - You can drop 2
  - You need to complete exercises to submit the vitamin

- 5-6 programming projects (next slide)
- 2 midterm exams
- 1 final exam
  - Exam format TBD [Waiting for guidance from campus administration]
  - Exam has been moved to a different group (group 3)

- Tentative schedule for programming projects and exams on the website.

Programming Projects

- Real-world focus
  - SQL querying: basics and algorithmics
  - Building pieces of a DBMS
    - B+-tree indexes
    - Join Algorithms
    - Dynamic Programming Query Optimizer
    - Concurrency (2PL) and Recovery (ARIES)
    - [A fun extension]
- Project 1 goes out next week!!
- For the first time, latter projects will be in teams of 2 [Details TBD]

Deadlines and Slip Time

- You have up to 5 days (up from 3 last sem) of slip time
    - Can be used for projects (unless otherwise noted)
    - Counted at the granularity of days
        - gradescope idiosyncracies

- Slip time is a safety net, not convenience
    - You should not plan on using them
    - If you use all 5 days you are doing it wrong

Academic Integrity

- We trust that you will do your own work
- Zero Tolerance. It is uncool. Don’t.
  - We have the technology to find out.
- Most cheating happens due to stress
  - Plan ahead and stay on schedule to minimize stress
  - You have built-in safety valves
    - Dropped vitamins
    - Slip days on homeworks: save for when you **need** them
    - Midterms weighted to the higher grade [exact policy TBD]
    - Keep an eye on the course drop date. Don’t take too many courses!
  - Feeling stressed? Reach out!
    - Campus resources
    - Course staff is here for you
    - Incompletes are appropriate for health issues of any kind
- Staff perspective
  - We want you to learn and to succeed
  - We want things to be fair, so need to stick to rules

Bottom line

Please don’t cheat!!

Staying in touch

- All class communication via Piazza
  - tiny.cc/cs186-fall20-piazza
  - We are already live
- Announcements and discussion
  - read it regularly
  - post all questions/comments there
  - answer each other's questions!
- Direct email to Prof or TAs is not a good idea
  - And will likely not get answered unless sensitive
  - Private posts on piazza to instructors is a much better bet

Now onto the real stuff…