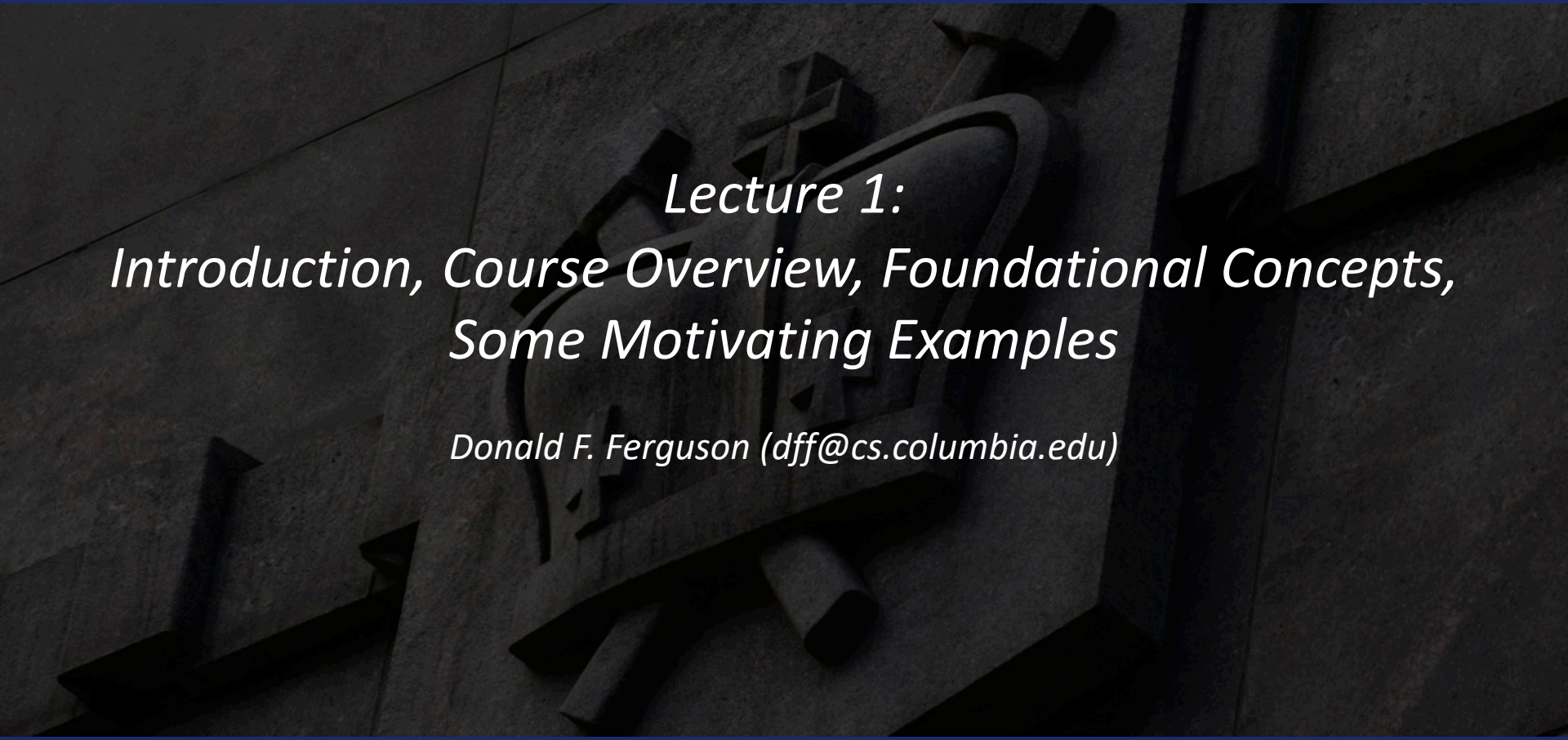
W4111 – Introduction to Databases
Section 002, Spring 2021

We will start in a couple of minutes



Lecture 1:
Introduction, Course Overview, Foundational Concepts, Some Motivating Examples

Donald F. Ferguson (dff@cs.columbia.edu)

We will start in a couple of minutes

Contents

Contents

- Introduction
  - Logistics, about your instructor, OHs, TAs.
  - Homework, exams.
  - Target application/solution/project.
- Core concepts
  - Motivating examples (Gives you a feel for programming and non-programming projects).
  - ~~Introductory concepts: data, databases, database management systems.~~
  - ~~Application, application architectures, roles.~~
- Database design, Entity-Relationship Model (Part 1)
  - Database design process.
  - ER-Model and diagrams.
- The theory: The *Relational Model* (Part 1)
  - Relational model, schema, keys, schema diagrams.
  - Basics of relational algebra.
- The realization: Structure Query Language (SQL) (Part 1)
  - Basics of Data Definition Language.
  - Basics of Data Manipulation Language (Query).
- Homework 0 – Definition and discussion.

- Read chapter 1 slides.
- Will cover in recitation.

Note:
- Make sure I do enough hands-on examples to support HW 1.
- Will also cover in recitation and lecture 2.

Introduction

Logistics

Exams, Homework

Target Application Scenario

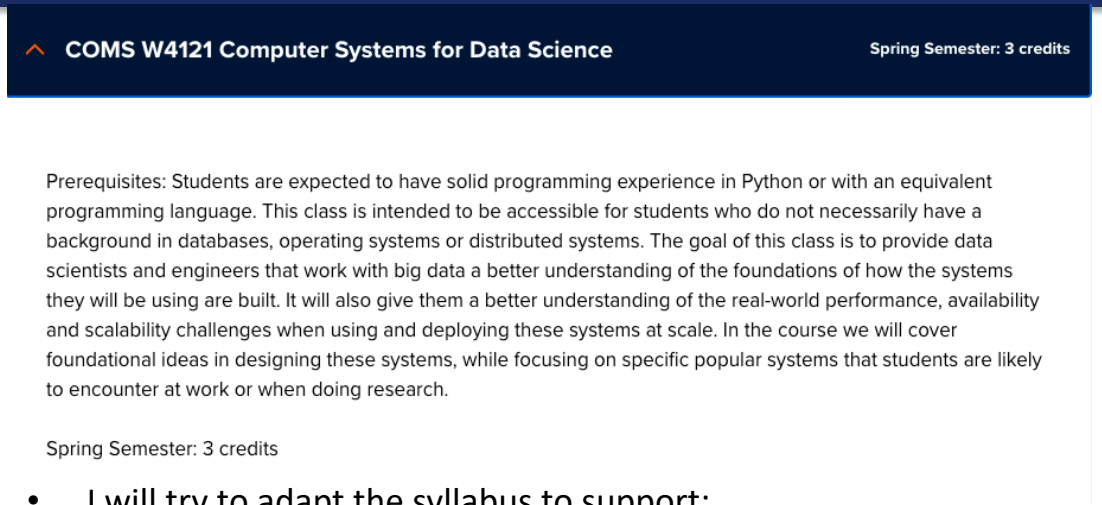
Logistics

Waitlist

- The administrative staff of the Dept. of Computer Science **directly** manages enrollment in the W4111 sections and the wait lists. This is true of several courses.
  - COMS W4111 is **ALWAYS** oversubscribed and has huge wait lists.
  - The course is a requirement for several majors and tracks.
  - There are **complex rules** about prioritizing students for course enrollment, and also which students can take the hybrid section.
- Section 002, Spring 2021 is like all sections of W4111, but has some special emphasis on supporting (MS) students in Data Science Institute.
  - In place of required COMS 4121, which is not taught this year.
  - DSI students have first enrollment priority, followed by CS Dept. prioritization for enrollment.
- Faculty follow department policies and do not manage waitlists, despite the list that identifies the waitlist as "Instructor Managed."
- Contacts: You must work with administrative leaders on enrollment:
  - Computer Science: Cynthia Meekins, cynthia@cs.columbia.edu)
  - Data Science Institute: Brianne Cortese (bc2640@columbia.edu)

A Comment about this Section

- The department has trouble offering W4121
- W4121 is a requirement for M.S. in Data Science.
- The DSI and CS Dept. have been substituting
  - W4111 for
  - W4121
- The CS Dept. asked me to prioritize my section for M.S. in Data Science students.



- I will try to adapt the syllabus to support:
  - CS students.
  - Non-CS students.
  - DSI students.
- If you are a DSI student, there will be content that I tailor to have W4111 better meet your needs.
- If you are not a DSI student, you will get the same benefits from this section as previous versions of W4111 that I taught.

Lecture Format, Recitation, Office Hours

- Lecture: Friday, 10:10 to 12:40 – This is a hybrid section, like many classes this fall.
  - The in-person lectures are optional. *I encourage you to attend in-person if you are on campus.*
  - I will hold as many lectures in-person as possible but may hold some lectures online only.
  - I will announce in advance if I plan to hold a lecture online only.
  - Lectures are streamed and recorded.
  - Students can attend in-person, watch the stream or watch the recording. These options are available to all students.
  - There are limitations on how many students can attend in person due to social distancing. I will provide a sign-up method to reserve seats if necessary.
- Recitation: Optional recitation on Saturday, 10:00 to 11:30.
  - Either in-person/online or just online. Will announce in advance.
  - Will hold when there is student interest, homework due, exams, etc.
- Office hours: (Friday: 0830 to 1000, 1300-1400; and by-request as needed)
  - I will hold on-campus and online offices hours. There will be online office hours that accommodate all time zones.
  - I will be on campus when possible for students that want to speak with me or need more focused help.
  - I will publish in-person OH when I plan them.
  - I will provide a sign-up mechanism for in person office hours if necessary.
- **Note: Your health, safety and well-being are ALWAYS my primary concern. These are bizarre, stressful times. Speak to me if you need special considerations and I will do the best that I can.**

Additional Basic Course Logistics

- Lecture: Friday, 10:10 AM-12:40 PM, Room TBD
- Instructor: Donald F. Ferguson (dff@cs.columbia.edu)
- Office Hours:
  - Fridays: 8:30 AM - 10:00 PM, 1:00 - 2:00 PM (Location TBD; in-person and online)
  - By appointment, as needed, as available.
  - I hold a lot of extra office hours, usually on weekends.
    - I typically post on Piazza when I will have extra availability.
    - The extra OHs may be "recitations," in which I go over exam questions, homework, etc.
    - Extra OH and recitations are often on the weekend or in the evening.
  - I will be flexible about time zones for extra office hours to accommodate students.
- **Recitation Section R02:** This is an optional recitation/office hours. It will be in person (when possible) and on-line. You do not need to register for it.
- Collaboration/Contact/Content:
  - The class is on Piazza for general questions, feedback, exam/homework clarification, ...
  - The course lectures, sample code, etc. will be on GitHub.
    - GitHub home page
    - Repository – contains lectures, examples, etc.
  - There is a Slack Team/Group for my courses. Please join the channel w4111-f2w.

Observations from Last Semester

- Student evaluation:
  - I pay a lot of attention to students' evaluations and comments.
  - Fall 2020 evaluations:
    - Course overall quality was 3.49 (out of 5). My past average has consistently been over 4.1.
    - Instructor overall quality was 3.66. My past average has consistently been over 4.1.
  - Some recurring (negative) themes:
    - "This is the most disorganized course I've taken."
    - HW assignments are vague and unclear.
    - Too much work.
- We will try to make improvements this semester.
  - Much, much better adherence to a HW and exam cadence/schedule.
  - We will make the HW assignments definitions clearer, but not as clear as you want.
  - Relying very heavily on the head TAs for homework/exam definition, explanation, ...
  - Will have periodic checkpoint polls to get feedback and change suggestions.

About Your Instructor

About your Instructor

- 35 years in computer science industry:
  - IBM Fellow
  - Microsoft Technical Fellow.
  - Chief Technology Officer, CA technologies.
  - Dell Senior Technical Fellow.
  - CTO, Co-Founder, Seeka.tv.
  - Ansys Fellow
- Academic experience:
  - BA, MS, Ph.D., Computer Science, Columbia University.
  - Approx. 15 semesters as an Adjunct Professor.
  - Professor of Professional Practice in CS (2018)
  - Courses:
    - E1006: Intro. to Computing
    - W4111: Intro. to Databases
    - E6998, E6156: Advanced Topics in SW Engineering (Cloud Computing)
- Approx. 65 technical publications; Approx. 12 patents.

Personal:

- Two children:
  - College freshman (Ithaca College).
  - 2019 Barnard Graduate.
- Hobbies:
  - Krav Maga, Black Belt in Kenpo Karate.
  - 1LT, New York Guard.
  - Bicycling.
  - Astronomy.
  - Languages:
    - Proficient in Spanish.
    - Learning Arabic.

About the Course

The Course

- From the new/pending Columbia University Bulletin

  "Prerequisites: COMS W3134, COMS W3136, or COMS W3137; or the instructor's permission.

  The course covers what a database system is, how to design databases effectively and in a principled manner, how to query databases, and how to develop applications using databases: entity-relationship modeling, logical design of relational databases, relational algebra, SQL, database application development, database security, and an overview of query optimization and transaction processing. Additional topics generally include NoSQL, graph, object-relational, and cloud databases, as well as data preparation and cleaning of real-world data. The course offers both programming and non-programming paths for homework and projects, to accommodate students with different programming skills and backgrounds."

- Prerequisites:
  - COMS W3134, COMS W3136, or COMS W3137 are data structures classes. All of these courses require extensive programming, in Java.
  - A course in data structures is helpful for this section of W4111 but not essential. I waive the requirement.
  - We will help you with any data structures knowledge you lack.

- Programming in/for this class:
  - There will be a "non-programming" option, which we will discuss below.
  - For students who want to take the programming track, we will use Python. I will provide motivation for choosing Python below.

Course Objectives

- Have fun, learn a lot and come to appreciate and enjoy some amazing technology. Data and databases have and will change the world.
- Provide a foundation that allows you to succeed in future courses. This is an *introduction* to databases. The technology is crucial for future courses
  - Big data, data analysis, data science
  - Advanced database classes
  - Machine learning
  - Numerical and data analytics in operations research, engineering, economics, finance, life sciences, financial engineering, medicine, etc.
- Enable you to successfully apply the technology in your work and profession.

Have cool stuff to talk about on interviews and in your resumes.

The Course – Value and my Perspective

- This course is foundational, and will teach you the core concepts in
  - Data modeling
  - Data model implementation; Data manipulation.
  - Different database models and database management systems.
  - Implementation and architecture of data centric applications and database management systems.
- ANY non-trivial application
  - Requires a well-designed data model.
  - Implements a data model and manipulates data.
  - Uses a database management system.
- Understanding databases and database management is core to the “hottest fields” in computer science, e.g.
  - Data science
  - Machine learning
  - Intelligent (Autonomous) systems
  - Internet-of-Things
  - Cybersecurity
  - Cloud Computing
- Database, database application, etc. skills are increasingly central to many disciplines, including:
  - Economics.
  - IEOR, financial engineering.
  - Mechanical and electrical engineering.
  - Medicine, pharmaceuticals
  - ...
- University courses on databases sometimes focus on theory and abstract concepts. This course will cover theory, but there will be an increased emphasis on:
  - Practical, hands-on applications of databases.
  - Patterns and best practices.
  - Developing and understanding database centric applications.
  - Using databases and various tools for data analysis, visualization and insight.
- **Personal perspective**
  - A large percent of my career has been spent figuring out or leading teams that figured out how to model, implement and manipulate data.
  - I have used the information in this class more than anything else I have learned.
  - This will likely be true for you.

Surprising Example

Example: This is a photo of mean using a database during COVID-19 mobilization.

- Match service members
- To JTF-HQ requests for personnel
- Based on assignment needs
- And service member
  - Skills
  - Availability
  - Tracked in a DBMS.
- The hardhat is because DB usage can be very dangerous 😱.

- No joking: I had to build an application that used:
  - Relational DBMS.
  - Python, Jupyter Notebook.
  - Google Sheets with Apps Script.
  - Google Forms.

- This technology and these skills are surprisingly applicable.

Environment

Course Resources and Development Environment

- Recommended textbook:
  - *Database System Concepts, Seventh Edition* (ISBN 9780078022159)
  - –
- S
- I
- D
  - Students have had problems with MySQL Workbench. We will try DataGrip. Please install.
  - Installing PyCharm is recommended for all and required for the programming track.

- See HW 0 for environment.
- See website for additional information on environment, resources, etc. (https://donald-f-ferguson.github.io/W4111S21/reference-material.html)

Course Resources and Development Environment

- Recommended textbook:
  - *Database System Concepts. Seventh Edition*. (ISBN 9780078022159)
  - There is a website associated with the textbook: https://www.db-book.com/db7/
  - Textbooks are expensive. You can easily get through the course using website, lecture material, ...

- Signup for an AWS Account. We will start with creating an instance of Relational Data Service.
  - There are instructions on the course website. (Under HW0)
  - Using cloud systems reduces complex SW setup on your personal machines.
    Having used cloud systems is a useful skill and looks good on resumes

- Install a new, most recent, isolated/single user instance of Anaconda environment.
  - You must install the most recent version for Python 3.
  - You can isolate the new instance from other instances to avoid conflicts, or set up custom environments.

- Development environments: Students. Are entitled to a free, annual JetBrains professional license.
  - Students have had problems with MySQL Workbench. We will try DataGrip. Please install.
  - Installing PyCharm is recommended for all and required for the programming track.

Homework
Exams
Grading

Assignments, Exams, Grading

- Point value of assignments and exams
  - 50%: Homework assignments:
    - 4 HWs, approximately one HW every two-three weeks.
    - Each is worth either 10 or 15 points based on difficulty.
  - Homework format:
    - Common to all tracks:
      - Questions requiring written answers and diagrams.
      - Implement/execute/test of various database operations required to solve a use case/question.
      - Format will be an iPython/Jupyter Notebook.
    - Track specific: Incremental development of a project:
      - Programming track – web application.
      - Non-programming track – data analysis/visualization.

All tracks will learn core concepts and get experience with other tracks' focus topics.

- Exams: Take home, the same for all tracks.
  - 20% of grade is midterm exam score.
  - 30% is final exam score.
- 97-100 points is **an A.** There are opportunities to get extra-credit.

To Program or Not To Program, ...

- From the department's new guidance for the course:
  - "To accommodate the diverse backgrounds of the students who take this class, all sections of the class should include a non-programming option for projects and assignments. We (NOTE: Does not include me) have successfully offered such an option in some sections of the class for many years: students can either program a web application to interface with a DB of their own design, or alternatively follow the non-programming option and come up with a quantifiably more detailed DB design/data.
- What does not constitute programming?
  - Writing queries for a relational, graph, document, or other data management system does not constitute programming and can be expected of all students.
- What constitutes programming?
  - Reading more than a handful (1-5) lines of code, Writing Python code that is not directly required to write a query."
- Why I previous only did a programming track:
  - Any non-trivial use of data and databases in any field requires programming. This is not Star Trek. You cannot shout, "Computer! Analyze ..."
  - You might think, "But, I do not want to program. I want to go into financial services, private equity, medicine, ... I can just use tools like Pandas or Tableau. You will do some programming for complex scenarios. Get over it.
- Tracks: Programming and non-programming.
  - There is a common core that involves understanding concepts, modeling data, using databases, etc.
  - Exams are the same for both tracks. Mix of written questions and practical exercises. Completely take home with several days to complete.
  - Homework assignments: There is a common core on all assignments for both tracks. Additionally,
    - Programming track will incrementally build a database centric, cloud-based web application.
    - Non-programming track will do a more complex, database centric project.

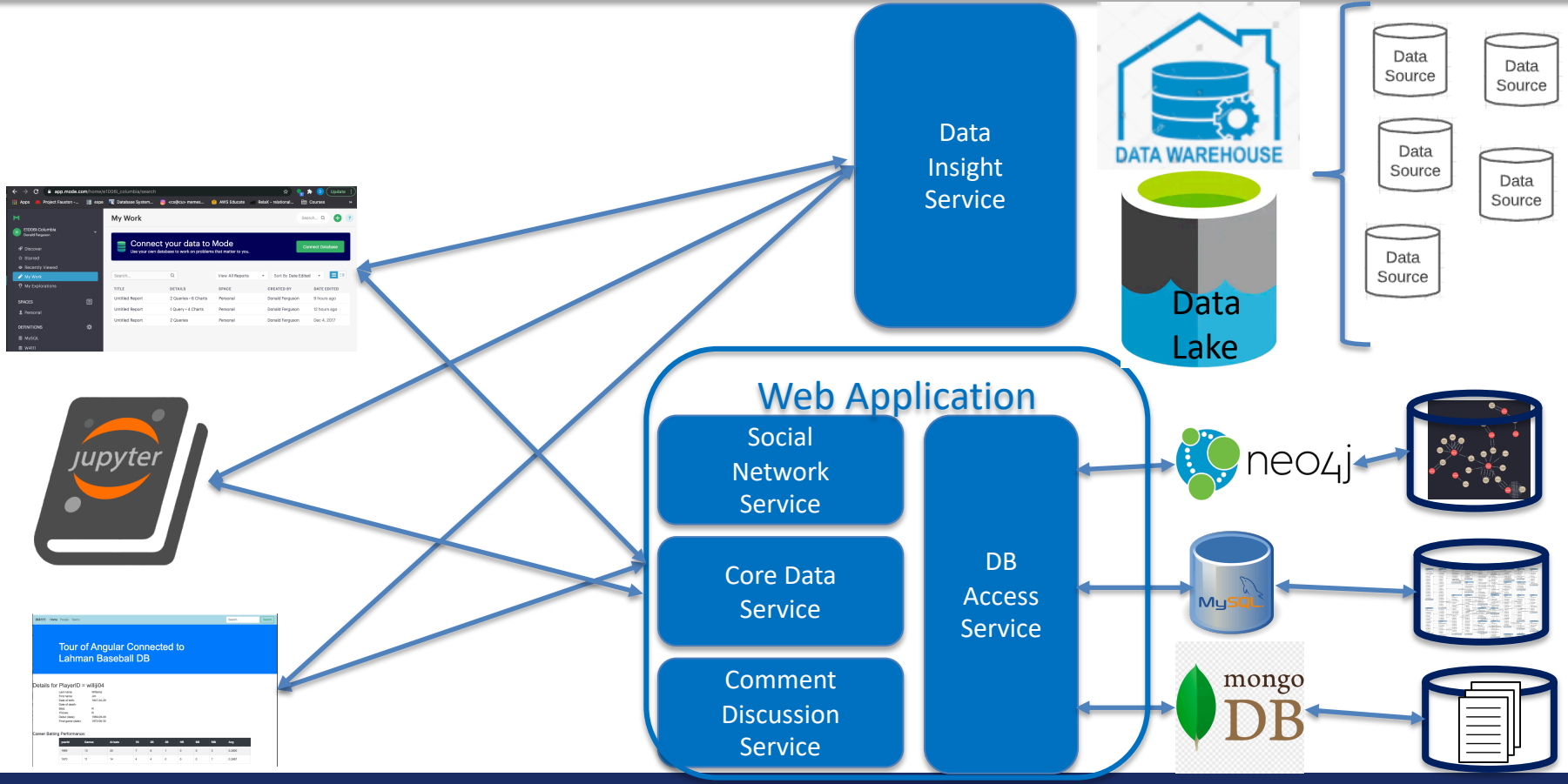
Homework Assignments

- My homework assignments are:
  - Open ended.
  - Vaguely specified.
- You will complain. I will listen sympathetically.
  - You will savage me on the instructor reviews and CULPA.
  - My professional colleagues and I will laugh at you behind your back.
- Management, clients, partners, etc.
  - Do not understand technology as well as we do.
  - You will get requirements like, "I want it colored mauve because that has more RAM."
- Converting vague requests into a useful, meaningful project that we can implement is what we do. Get over it.
- BUT, I WILL SIGNIFICANTLY IMPROVE CLARITY AND DETAILS.

> **You are senior people at a top school. You need to define concrete solutions and approaches from ambiguity. No one showed Thomas Edison a light bulb and then said, "Build this."**
>
> **Donald Ferguson** on 2019-01-29

Homework and Projects

Target Application/Project(s)



Target Application/Project

- That diagram was pretty confusing.
- Basically, what it comes down to is that there will be major subsystems:
  1. Interactive web application for viewing, navigating and updating data.
     The updates have to preserve *semantic constraints* and correctness.
  2. A decision support warehouse/lake that allows us to explore data and get insights.
- Programming and non-programming tracks will get experience with both, but
  - Non-programming track focuses on data engineering needed to produce (2).
  - Programming track will focus on (1).
- The example that I will use is *fantasy baseball* because it covers both aspects, and there is a lot of interesting data available.
- You will have some freedom on the data models and scenarios you choose.

Core Concepts

COVID-19 and Population Data

First Source of Information – NY Times

- NY Times Dataset (https://github.com/nytimes/covid-19-data)
  - Date
  - County
  - State
  - FIPS
  - Cases (to date)
  - Deaths (to date)

100% 52:1

Result Grid Filter Rows: Search Export: Fetch rows:

date	county	state	fips	cases	deaths
date	county	state	fips	cases	deaths
2020-01-21	Snohomish	Washington	53061	1	0
2020-01-22	Snohomish	Washington	53061	1	0
2020-01-23	Snohomish	Washington	53061	1	0
2020-01-24	Cook	Illinois	17031	1	0
2020-01-24	Snohomish	Washington	53061	1	0
2020-01-25	Orange	California	06059	1	0
2020-01-25	Cook	Illinois	17031	1	0
2020-01-25	Snohomish	Washington	53061	1	0
2020-01-26	Maricopa	Arizona	04013	1	0
2020-01-26	Los Angeles	California	06037	1	0
2020-01-26	Orange	California	06059	1	0
2020-01-26	Cook	Illinois	17031	1	0
2020-01-26	Snohomish	Washington	53061	1	0
2020-01-27	Maricopa	Arizona	04013	1	0
2020-01-27	Los Angeles	California	06037	1	0
2020-01-27	Orange	California	06059	1	0
2020-01-27	Cook	Illinois	17031	1	0
2020-01-27	Snohomish	Washington	53061	1	0
2020-01-28	Maricopa	Arizona	04013	1	0
2020-01-28	Los Angeles	California	06037	1	0

- I did not pick this data for any reason other than it is simple for a demo.

Second Source of Information

- United States Census Bureau – City and Town Population Totals: 2010-2019 (https://www.census.gov/newsroom/press-kits/2019/national-state-estimates.html)
- A subset of the fields.
- A lot of online data comes with a PDF or txt explaining the fields, values, sources, etc.

VARIABLE	DESCRIPTION
SUMLEV	Geographic summary level
STATE	State FIPS code
COUNTY	County FIPS code
PLACE	Place FIPS code
COUSUB	Minor Civil Division FIPS code
CONCIT	Consolidated city FIPS code
PRIMGEO_FLAG	Primitive Geography Flag (1=Yes; 0=No)
FUNCSTAT	Functional Status Code
NAME	Area name
STNAME	State name
CENSUS2010POP	4/1/2010 resident Census 2010 population
ESTIMATESBASE2010	4/1/2010 resident population estimates base
POPESTIMATE2010	7/1/2010 resident population estimate
POPESTIMATE2011	7/1/2011 resident population estimate
POPESTIMATE2012	7/1/2012 resident population estimate
POPESTIMATE2013	7/1/2013 resident population estimate
POPESTIMATE2014	7/1/2014 resident population estimate
POPESTIMATE2015	7/1/2015 resident population estimate
POPESTIMATE2016	7/1/2016 resident population estimate
POPESTIMATE2017	7/1/2017 resident population estimate
POPESTIMATE2018	7/1/2018 resident population estimate
POPESTIMATE2019	7/1/2019 resident population estimate

NST-EST2019-alldata

Annual Population Estimates, Estimated Components of Resident Population Change, and Rates of the Components of Resident Population Change for the United States, States, and Puerto Rico: April 1, 2010 to July 1, 2019 [<1.0 MB]

File Layout [<1.0 MB]

Second Source of Information (Sample)

100% 1:1

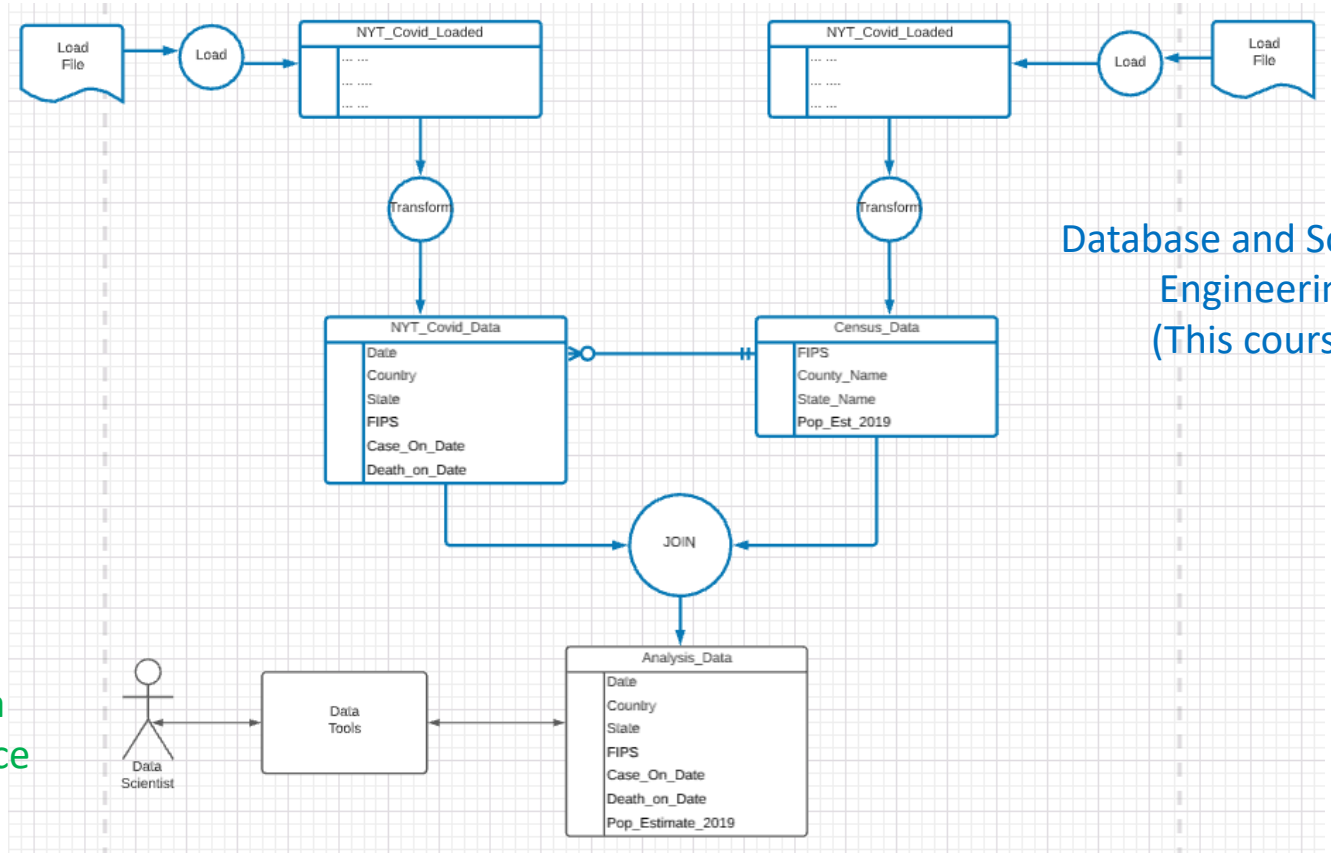
Result Grid Filter Rows: Search Export: Fetch rows:

SUMLEV	STATE	COUNTY	PLACE	COUSUB	CONCIT	PRIMGEO_FLAG	FUNCSTAT	NAME	STNAME	CENSUS2010POP	ESTIMATESBASE2010	POPESTI
040	01	000	00000	00000	00000	0	A	Alabama	Alabama	4779736	4780125	4785437
162	01	000	00124	00000	00000	0	A	Abbeville city	Alabama	2688	2705	2699
162	01	000	00460	00000	00000	0	A	Adamsville city	Alabama	4522	4506	4500
162	01	000	00484	00000	00000	0	A	Addison town	Alabama	758	754	751
162	01	000	00676	00000	00000	0	A	Akron town	Alabama	356	356	355
162	01	000	00820	00000	00000	0	A	Alabaster city	Alabama	30352	31112	31209
162	01	000	00988	00000	00000	0	A	Albertville city	Alabama	21160	21209	21196
162	01	000	01132	00000	00000	0	A	Alexander City city	Alabama	14875	14984	14929
162	01	000	01228	00000	00000	0	A	Aliceville city	Alabama	2486	2481	2480
162	01	000	01396	00000	00000	0	A	Allgood town	Alabama	622	622	622
162	01	000	01660	00000	00000	0	A	Altoona town	Alabama	933	937	937
162	01	000	01708	00000	00000	0	A	Andalusia city	Alabama	9015	9015	9025
162	01	000	01756	00000	00000	0	A	Anderson town	Alabama	282	281	281
162	01	000	01852	00000	00000	0	A	Anniston city	Alabama	23106	22987	22932
162	01	000	02116	00000	00000	0	A	Arab city	Alabama	8050	8080	8095
162	01	000	02260	00000	00000	0	A	Ardmore town	Alabama	1194	1194	1203
162	01	000	02320	00000	00000	0	A	Argo town	Alabama	4071	4085	4093
162	01	000	02428	00000	00000	0	A	Ariton town	Alabama	764	762	763
162	01	000	02500	00000	00000	0	A	Arley town	Alabama	357	357	356
162	01	000	02836	00000	00000	0	A	Ashford town	Alabama	2148	2149	2151
162	01	000	02860	00000	00000	0	A	Ashland town	Alabama	2037	2039	2034

Result Grid | Form Editor | Field Types | Query Stats | Execution Plan

- Why do I need a census data?
- Total death/sickness numbers are meaningless without considering population.

Preparing the Data for Data Science



Database and Software Engineering (This course)

Data Science

Problem 1 – Join on FIPS Codes

- There are two sets of data linked by values in columns:
  - NY Times data FIPS column.
  - Census Data Place concatenation of state and county codes.
- No. of rows: NY Times Data – 870K, Census Data: 81K
- Producing a new "table" that contains matching rows can be tricky.
  - I could write a program, or
  - There is a complicated set of instructions I can follow to accomplish in Excel.
- Even if I could figure it out, it would execute slowly. There are a lot of possible matches.
- *It would seem that solving this problem of efficiently "joining" rows from multiple, large tables based on column values is a common problem.*
  - It would be nice if there were something that could implement the operation in a general way for arbitrary tables.
  - Oh. And the Census data is by town, and I need to roll up by county. This just got harder.

Data Engineering Problems

- Problems:
  - Basic problems
    1. Get and load the data in some form that I can handle.
    2. Perform two transforms:
       1. NYT Data: Convert the cumulative cases and deaths to daily cases and deaths.
       2. US Census Data:
          1. Drop columns for population estimates except for 2019.
          2. Merge some columns to produce a FIPS.
          3. Aggregate/Sum individual town/place values to produce county values.
  - Moderately complex problem: Join the data to correlate population data and deaths/cases.
  - Complex problem: Resolve inconsistencies and mismatches.

- There are several ways to solve these problems.
  - Write programs, but they would be specific to this problem.
  - Do a lot of editing in spreadsheets, but this can be tricky and will not work for enormous datasets.

- These operations seem like operations that are common to many scenarios.
  If we had a thing that solved the problems in a general way, it would be awesome.

Go to Notebook

Data Engineering Problems

- Problems:
  - Basic problems
    1. Get and load the data in some form that I can handle.
    2. Perform two transforms:
       1. NYT Data: Convert the cumulative cases and deaths to daily cases and deaths.
       2. US Census Data:
          1. Drop columns for population estimates except for 2019.
          2. Merge some columns to pr
          3. Aggregate/Sum individual t
  - Moderately complex problem: Join the
  - Complex problem: Resolve inconsistenc

Remember this aspiration/question?

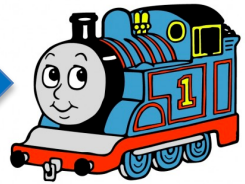
- There are several ways to solve these problems:
  - Write programs, but they would be specific to this
  - Do a lot of editing in spreadsheets, but this can be and will not work for enormous datasets.

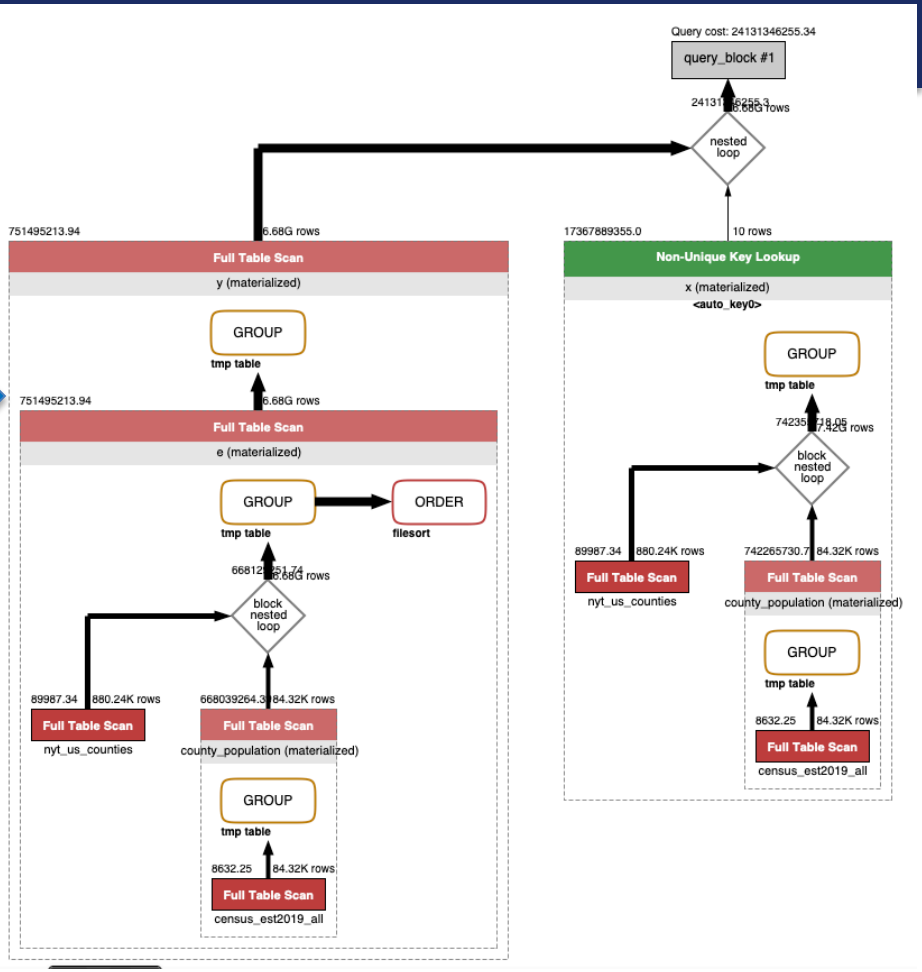
- These operations seem like operations that are common to many scenarios.
  It would be awesome if we had a thing that solved the problems in a general way.

Query Engines

```
select c_year as `Year`, c_month as `Month`, state, cast(x.total_deaths as unsigned) as nyc_cumulative_deaths,
  cast(y.total_deaths as unsigned) as non_nyc_cumulative_deaths
from
  (select
    year(date) as c_year, month(date) as c_month, county, state, max(deaths) as total_deaths
  from
  (select * from
    (SELECT
      concat(state, county) as fips,  sum(popestimate2019) as estimated_2020_county_population
    FROM aaaaS21Examples.census_est2019_all
    group by fips) as county_population
  right join
    nyt_us_counties
  using(fips)) as sub_result
  where county='New York City'
  group by c_year, c_month, county, state) as x
join
  (select c_year, c_month, non_nyc, state, sum(total_deaths) as total_deaths from
    (select
      year(date) as c_year, month(date) as c_month, 'Non NYC' as non_nyc, state, max(deaths) as total_deaths
    from
    (select * from
      (SELECT
        concat(state, county) as fips,  sum(popestimate2019) as estimated_2020_county_population
      FROM aaaaS21Examples.census_est2019_all
      group by fips) as county_population
    right join
      nyt_us_counties
    using(fips)) as sub_result
    where county !='New York City' and state='New York'
    group by c_year, c_month, county, state
    order by county, c_year, c_month) as e
  group by
    c_year, c_month, state) as y
using(c_year, c_month, state);
```

Database Engine

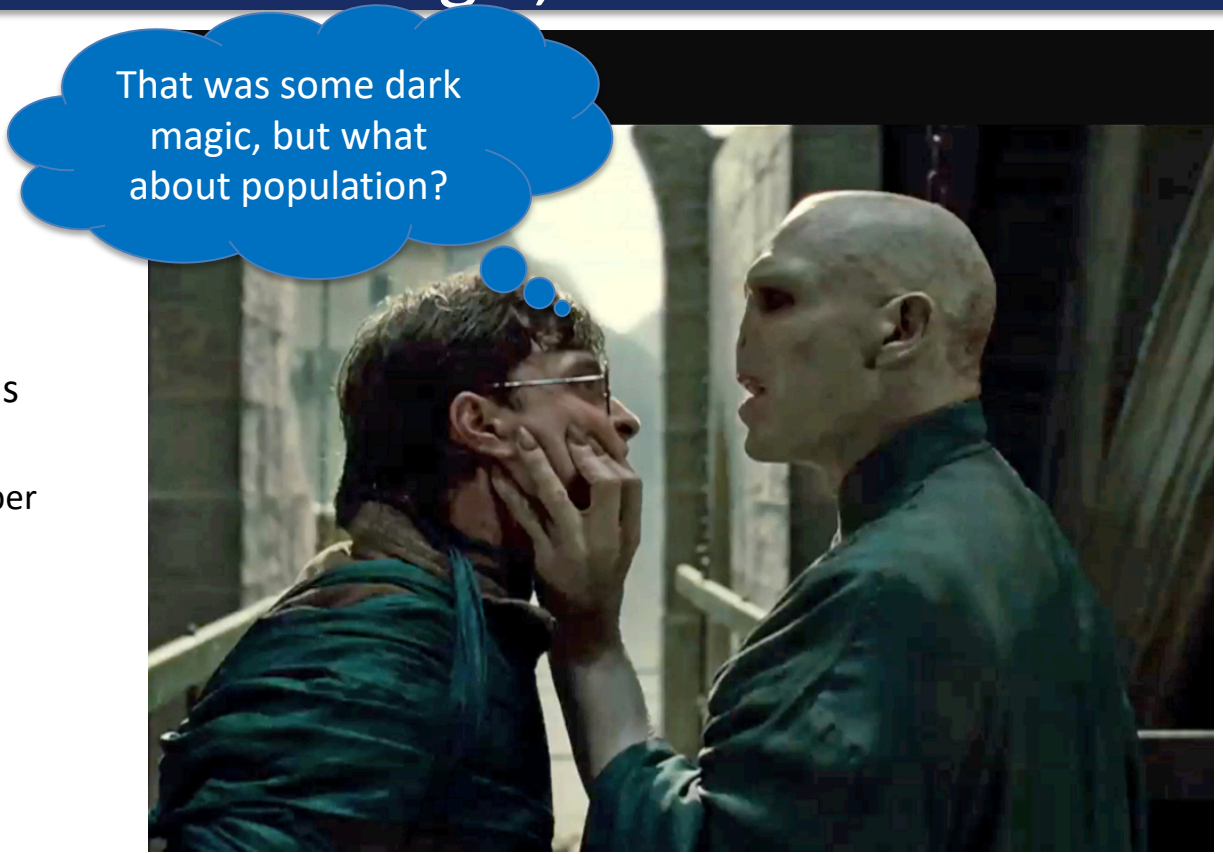




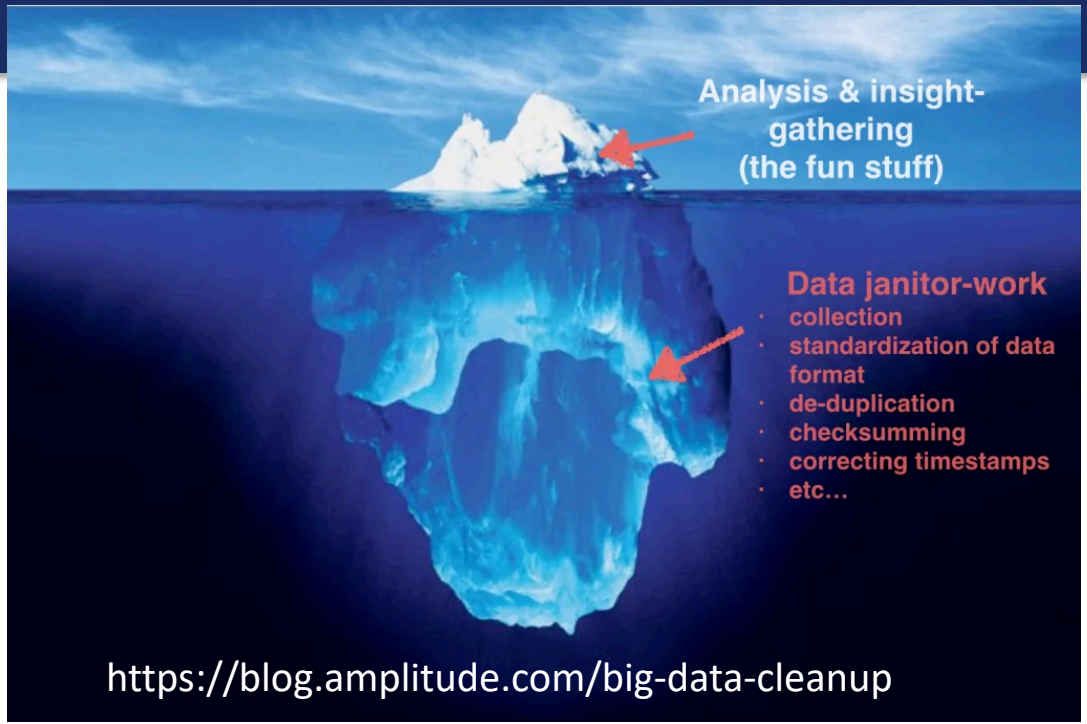
- SQL seems complex. Once you learn SQL, it is more productive than hand writing code.
- We will also see that the engine performs sophisticated performance optimizations.

Hmm. That was some dark magic, but ... ...

- I started this example with the complexity of "JOINing" data from two tables.
  - COVID data.
  - Census population data.
- I observed that I could not due the final step because ..., it turns out the data is incomplete.
  - NYT COVID data has one number for NYC.
  - But, ... NYC is five counties.
  - I could not do the JOIN.
- This requires me to perform *Data Cleansing.*



Data Cleansing



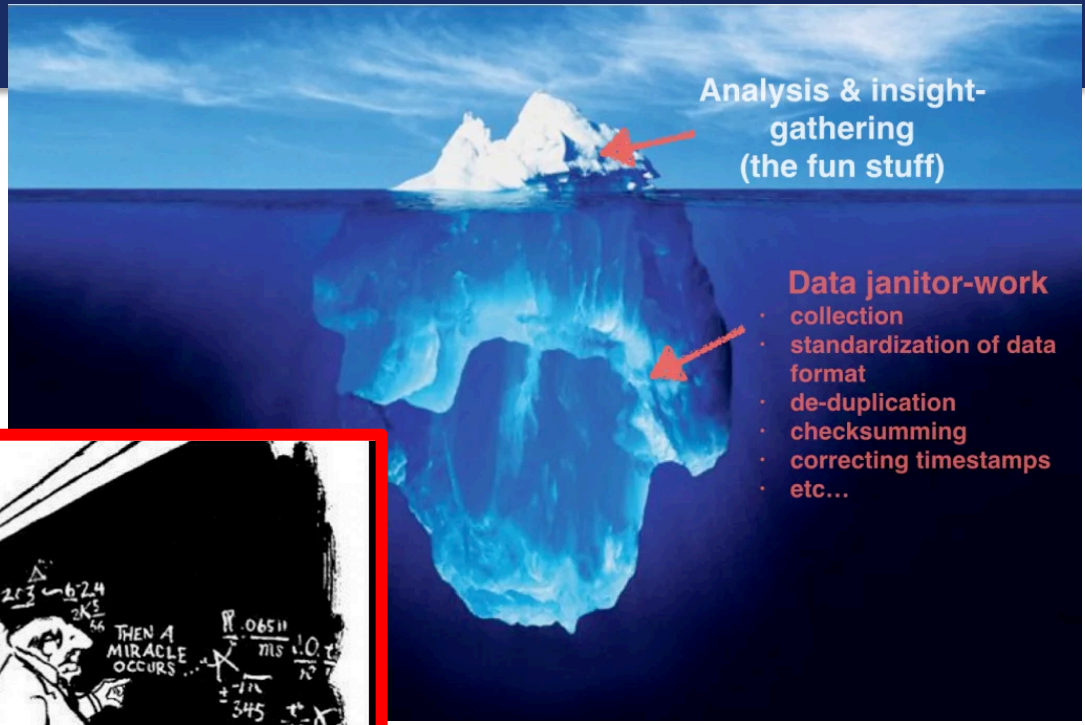
Database and data science classes **love to teach the fun stuff**
queries, data modeling, machine learning, spooky math, algorithms, ...

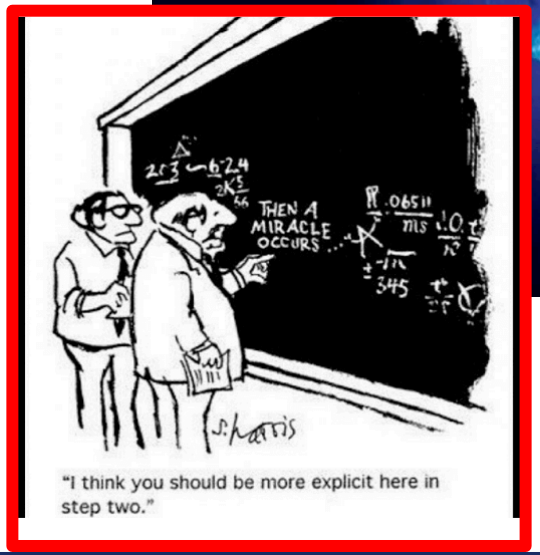
Data Cleansing

Syllabus Topics

- Relational Foundations
  - Overview (1 lecture)
  - ER Model (2 lectures)
  - Relational Model (4 lectures)
  - Relational Algebra (2 lectures)
  - SQL (5 lectures)
  - Application Programming and Database APIs (1 lecture)
  - Security (2 lectures)
  - Normalization (2 lectures)
  - Overview of Storage and Indexes (1 lecture)
  - Overview of Query Optimization (1 lecture)
  - Overview of Transaction Processing (1 lecture)
- Beyond Relational Foundations
  - NoSQL (1 lecture)
  - Data Preparation and Cleaning (1 lecture)
  - Graphs (1 lecture)
  - Object-Relational Databases (2 lectures)
  - Cloud Databases (1 lecture)

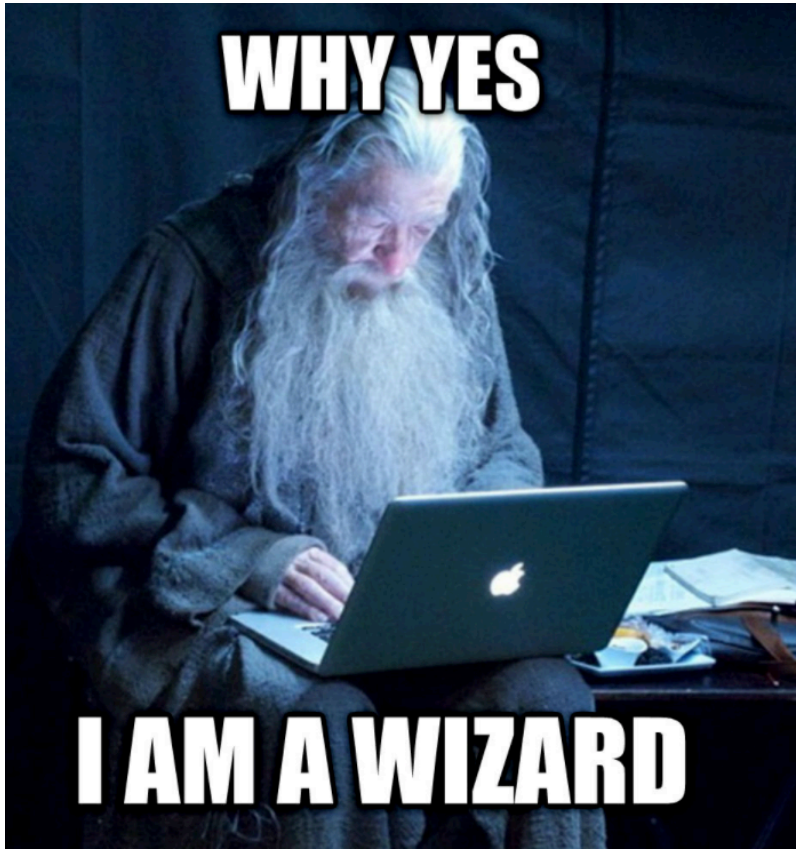
Recommended Syllabus





I place more emphasis on data cleansing and refactoring than other sections of W4111.

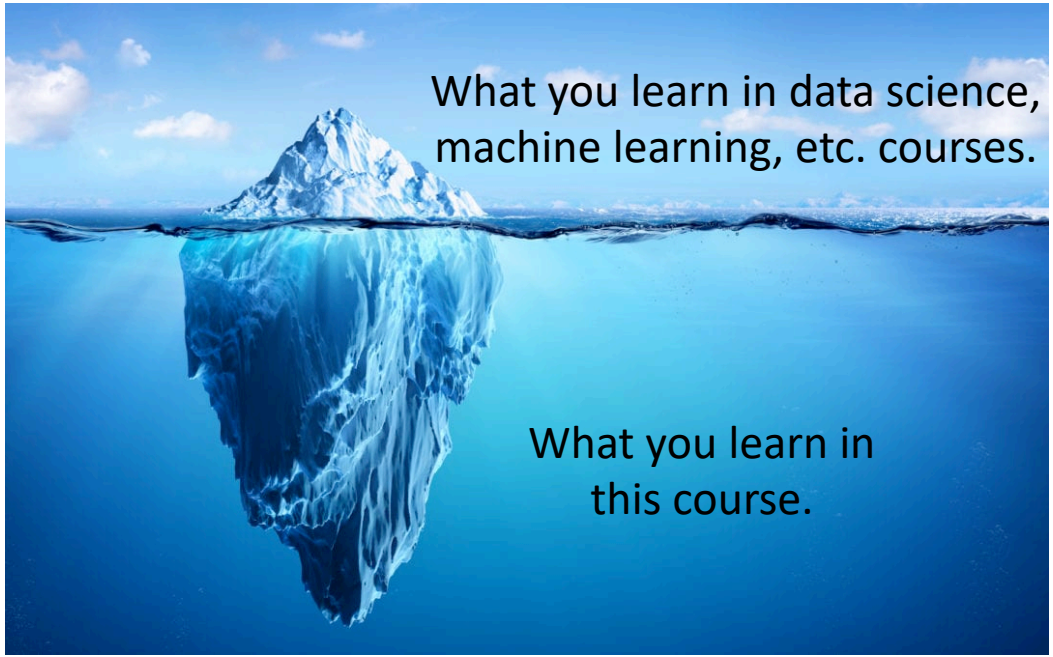
After a Bunch of Wizard Stuff ...



- I finally got the data in a format that I can analyze and explore.
- I am not ready to be a “data scientist.”

	c_year	c_month	county	state	cfips	cases	deaths	pop_estimate	death_rate	case_rate
0	2020	3	Albany	New York	36001	226	1	305506	0.000	0.001
1	2020	4	Albany	New York	36001	1165	53	305506	0.000	0.004
2	2020	5	Albany	New York	36001	1860	142	305506	0.000	0.006
3	2020	6	Albany	New York	36001	2102	159	305506	0.001	0.007
4	2020	7	Albany	New York	36001	2503	169	305506	0.001	0.008
...	...	...	...	...	...	...	...	...	...	...
574	2020	8	Yates	New York	36123	62	6	24913	0.000	0.002
575	2020	9	Yates	New York	36123	64	6	24913	0.000	0.003
576	2020	10	Yates	New York	36123	132	6	24913	0.000	0.005
577	2020	11	Yates	New York	36123	265	7	24913	0.000	0.011
578	2020	12	Yates	New York	36123	473	10	24913	0.000	0.019

Why Did You Put Us Through That?



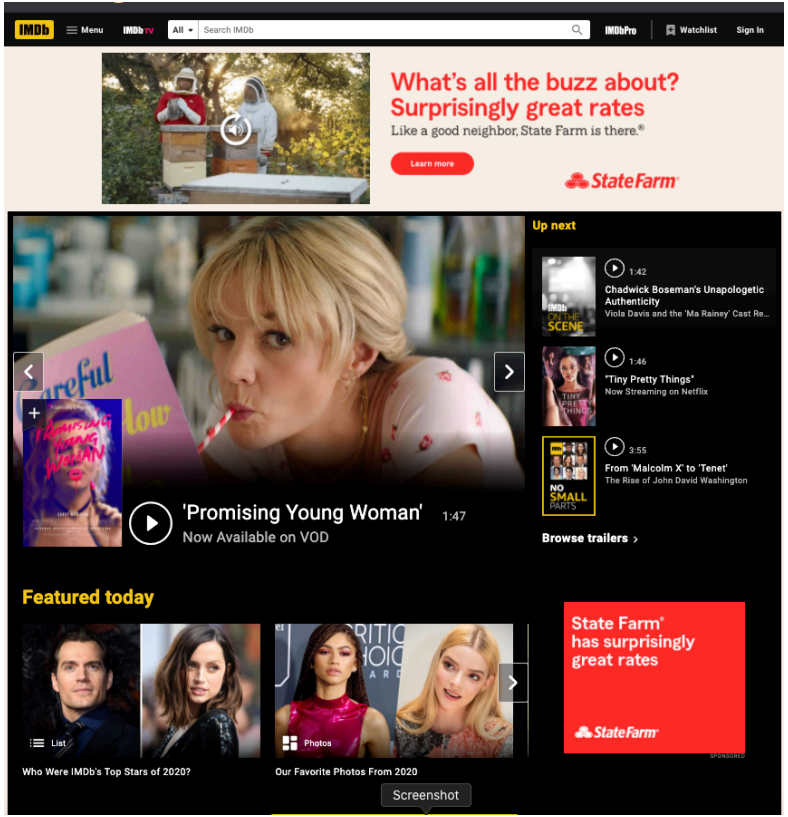
- You got a sense of the essence of database and data.
  - Query languages.
  - Schema.
  - Operations on data.
  - Scale and performance.

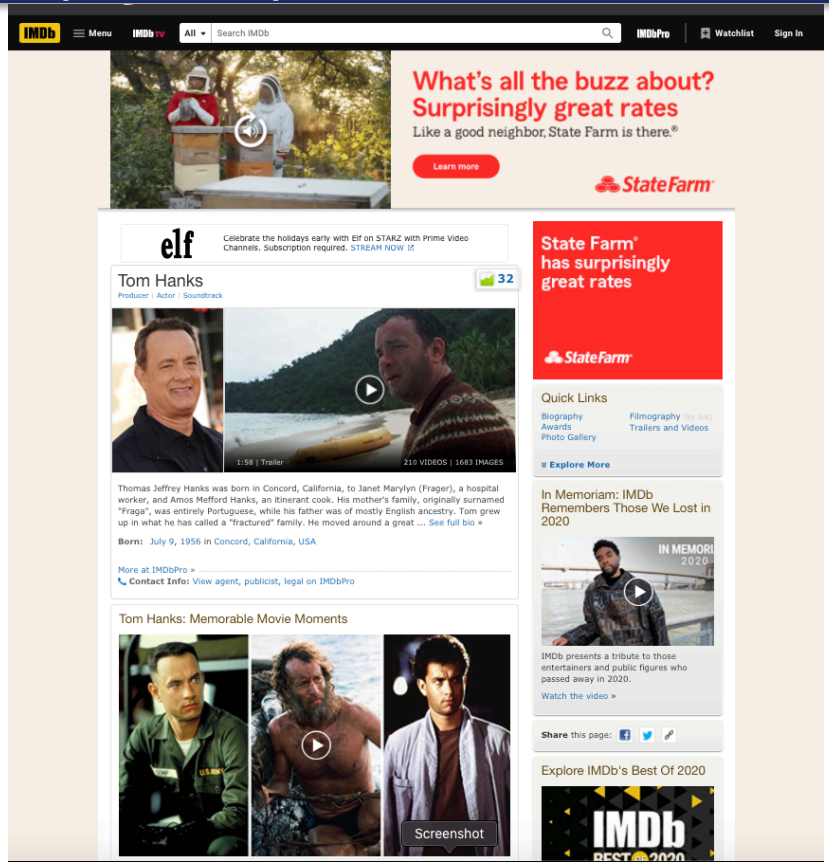
And ...

- Both programming and non-programming tracks will do work like what I just did.
- The non-programming track does a lot more of it.
- I know you are now afraid to ask, "But, what does the programming track have to do?"

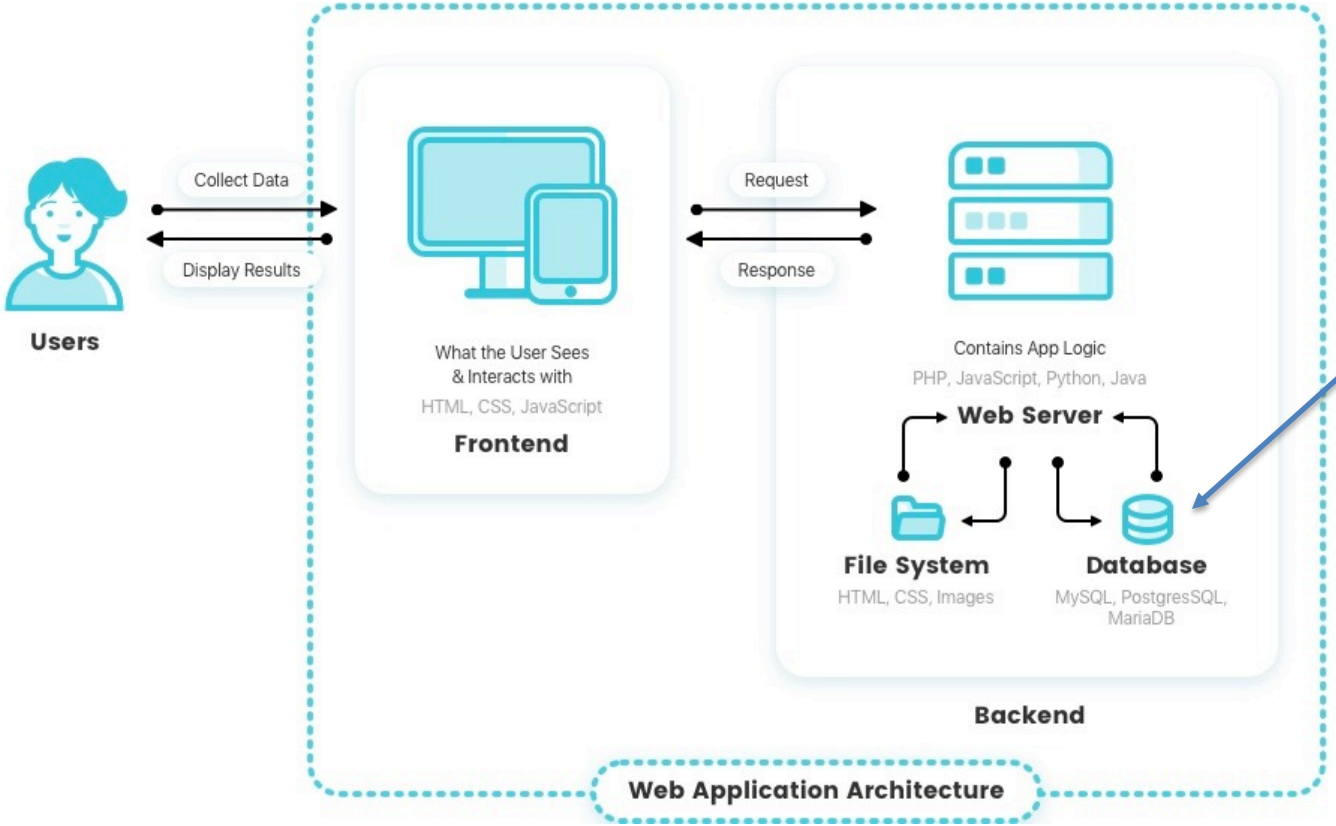
IMDB

Internet Movie Database (IMDB)





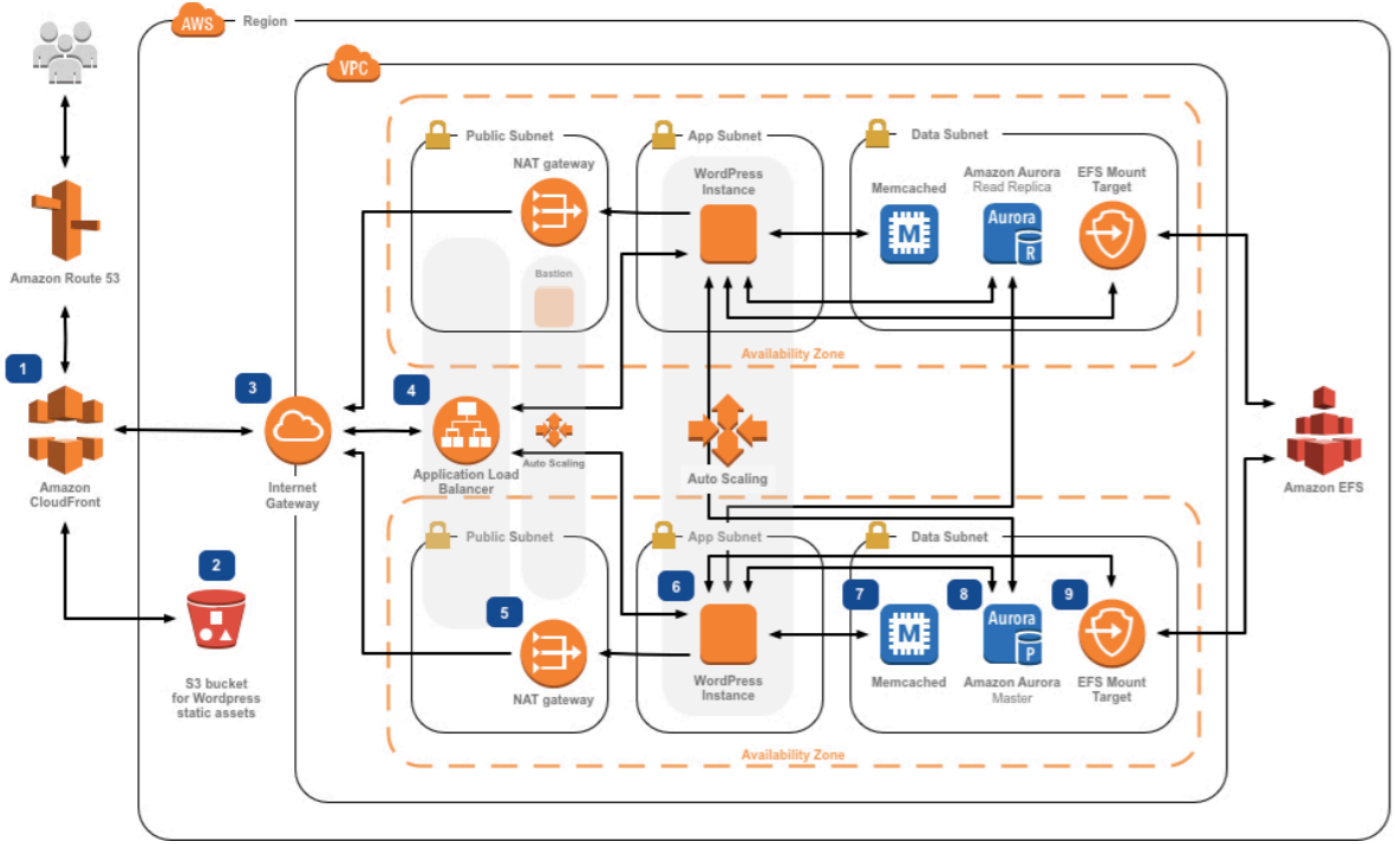
Web Application



IMDB Database

It would be “cool” if we could look at the data directly!

An Aside, Web Application are Complex



This is:

- A trivial example.
- From a tutorial.

Real solutions are staggering complex.

Well, we can download some of it!

IMDb Dataset Details

Each dataset is contained in a gzipped, tab-separated-values (TSV) formatted file in the UTF-8 character set. The first line in each file contains headers that describe what is in each column. A '\N' is used to denote that a particular field is missing or null for that title/name. The available datasets are as follows:

title.akas.tsv.gz - Contains the following information for titles:

- titleId (string) - a tconst, an alphanumeric unique identifier of the title
- ordering (integer) – a number to uniquely identify rows for a given titleId
- title (string) – the localized title
- region (string) - the region for this version of the title
- language (string) - the language of the title
- types (array) - Enumerated set of attributes for this alternative title. One or more of the following: "alternative", "dvd", "festival", "tv", "video", "working", "original", "imdbDisplay". New values may be added in the future without warning
- attributes (array) - Additional terms to describe this alternative title, not enumerated
- isOriginalTitle (boolean) – 0: not original title; 1: original title

title.basics.tsv.gz - Contains the following information for titles:

- tconst (string) - alphanumeric unique identifier of the title
- titleType (string) – the type/format of the title (e.g. movie, short, tvseries, tvepisode, video, etc)
- primaryTitle (string) – the more popular title / the title used by the filmmakers on promotional materials at the point of release
- originalTitle (string) - original title, in the original language
- isAdult (boolean) - 0: non-adult title; 1: adult title
- startYear (YYYY) – represents the release year of a title. In the case of TV Series, it is the series start year
- endYear (YYYY) – TV Series end year. '\N' for all other title types
- runtimeMinutes – primary runtime of the title, in minutes
- genres (string array) – includes up to three genres associated with the title

title.episode.tsv.gz – Contains the tv episode information. Fields include:

- tconst (string) - alphanumeric identifier of episode
- parentTconst (string) - alphanumeric identifier of the parent TV Series
- seasonNumber (integer) – season number the episode belongs to
- episodeNumber (integer) – episode number of the tconst in the TV series

title.principals.tsv.gz – Contains the principal cast/crew for titles

- tconst (string) - alphanumeric unique identifier of the title
- ordering (integer) – a number to uniquely identify rows for a given titleId
- nconst (string) - alphanumeric unique identifier of the name/person
- category (string) - the category of job that person was in
- job (string) - the specific job title if applicable, else '\N'
- characters (string) - the name of the character played if applicable, else '\N'

title.ratings.tsv.gz – Contains the IMDb rating and votes information for titles

- tconst (string) - alphanumeric unique identifier of the title
- averageRating – weighted average of all the individual user ratings
- numVotes - number of votes the title has received

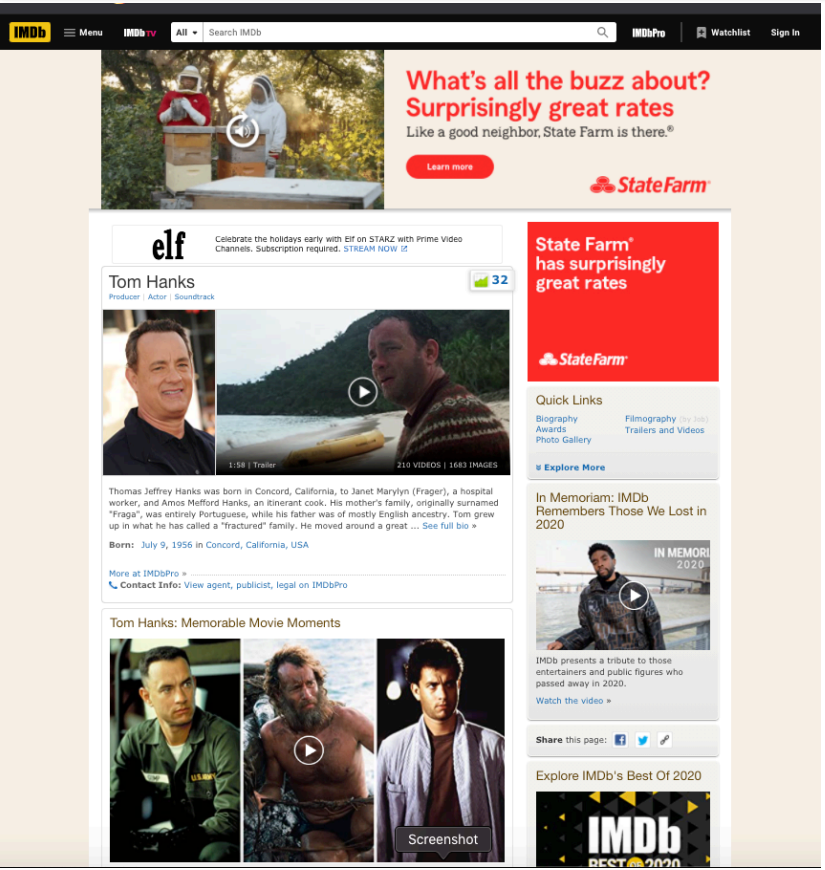
name.basics.tsv.gz – Contains the following information for names:

- nconst (string) - alphanumeric unique identifier of the name/person
- primaryName (string)– name by which the person is most often credited
- birthYear – in YYYY format
- deathYear – in YYYY format if applicable, else '\N'
- primaryProfession (array of strings)– the top-3 professions of the person
- knownForTitles (array of tconsts) – titles the person is known for

But, ... The files are pretty big.

- The uncompressed file size is about 5 GB.
- Some of the files have a lot of rows:
  - name.basics approx. 10 million.
  - title.principals approx. 37 million.
- There is no way I can
  - Open these in Excel
  - Let alone do my column-based joins.
- And by “real world” DB standards, these files and tables are tiny.
- DBs specialize in managing staggering large datasets and querying efficiently.
- Applications enable users to interact with the data in meaningful ways.

How Does a Page Like this Work?

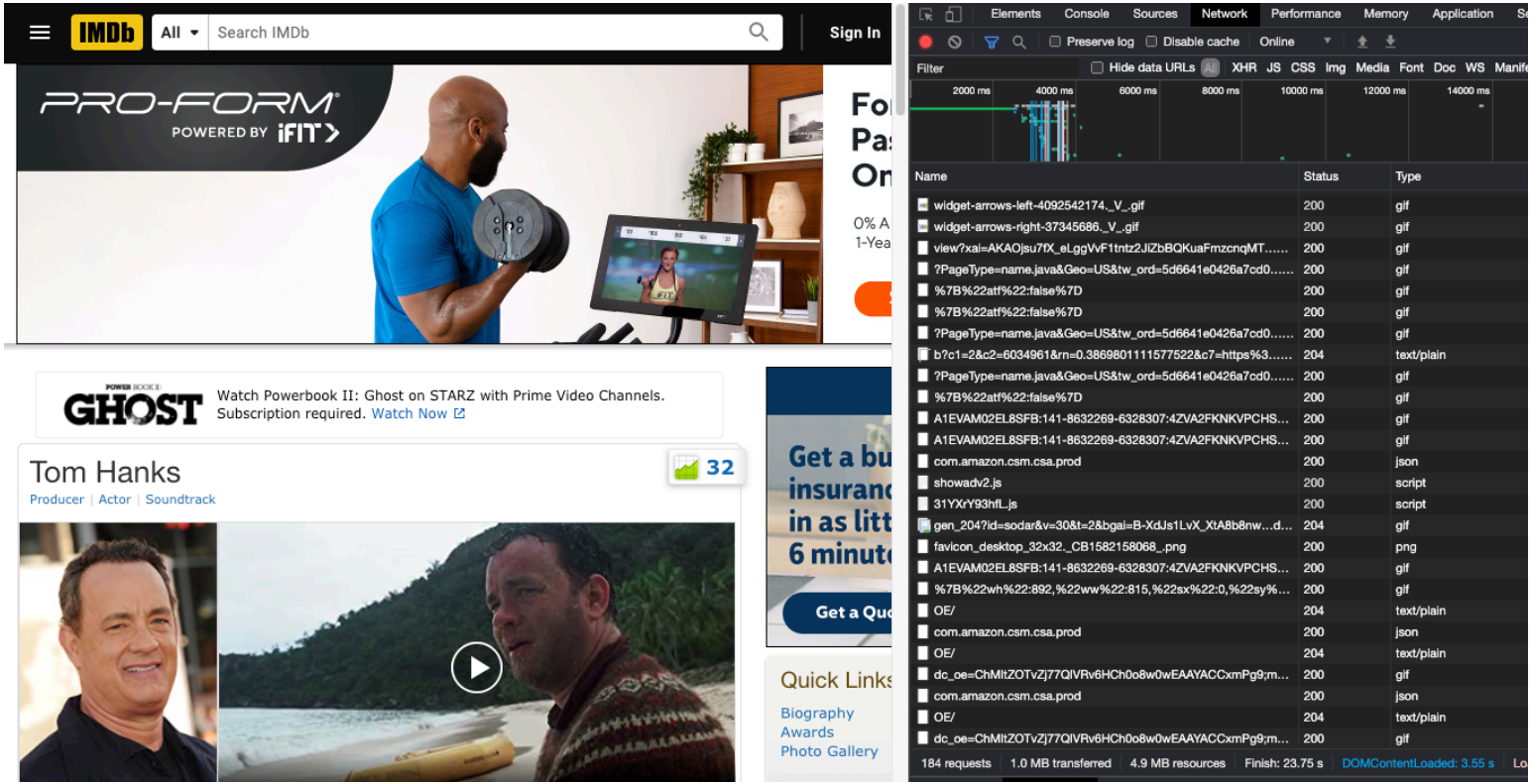


Well, overly simplistically ...

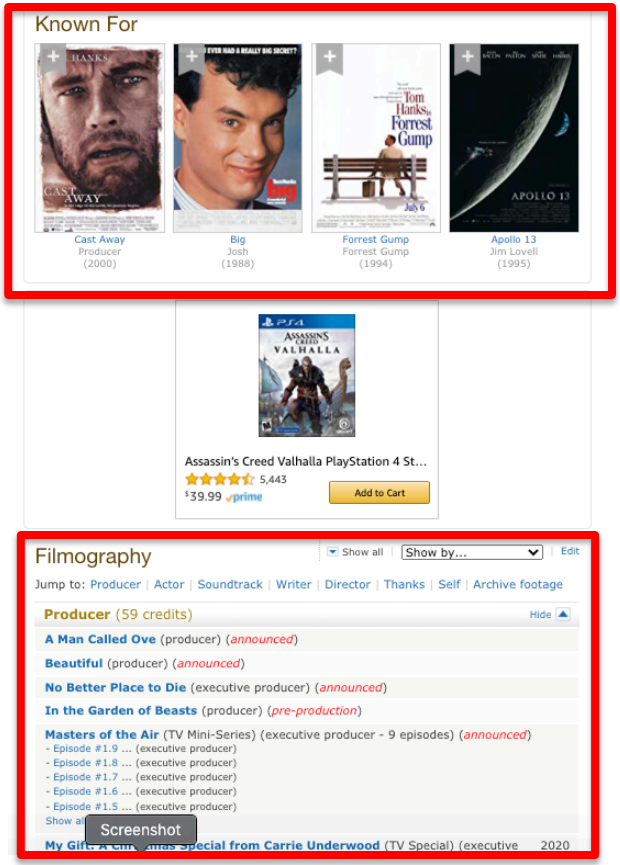
- The web browser may a query https://www.imdb.com/find?q=%22tom+hanks
- This produced the search results.
- You clicked a link, and the browser made another query https://www.imdb.com/find?q=%22tom+hanks%22
- The browser gets back to sets of "things."
  - A page template with a bunch of slots.
  - Queries that determine what to put in the slots.
  - Based on
    - Context: previous queries, connections in data, ...
    - Policies: promotions, ad campaigns, ...
    - User info: prior queries, location, language, ...
    - ... ...

How Does a Page Like this Work?

[One look up https://www.imdb.com/name/nm0000158](https://www.imdb.com/name/nm0000158) ==> 183 other look ups.

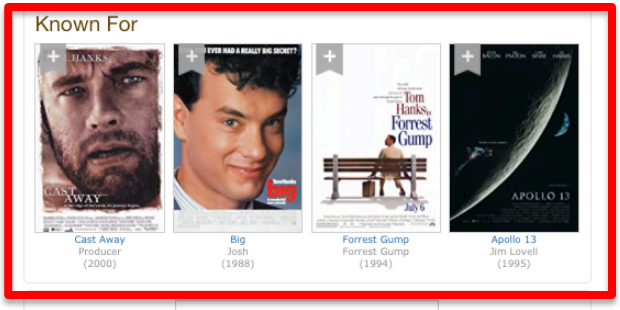


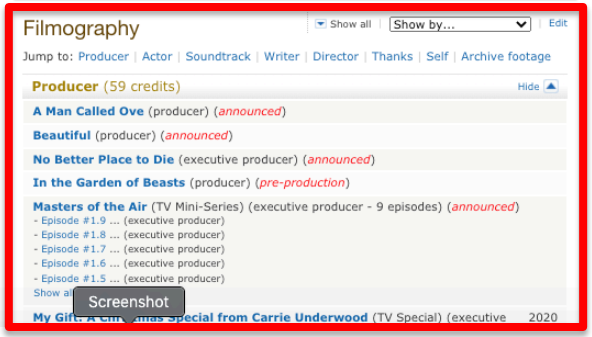
How did I Know this Information?

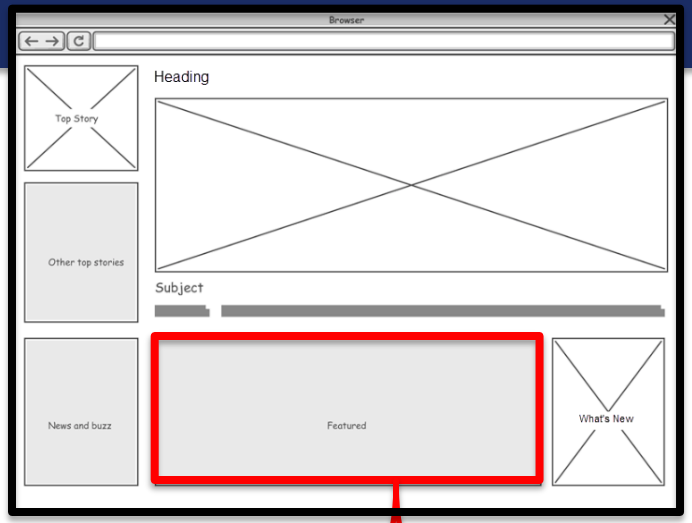


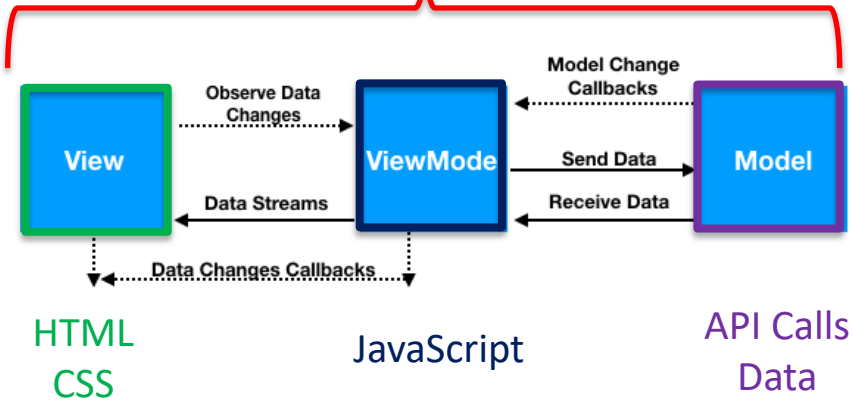
- The core is answering 4 questions.
  - "Tom Hanks'" ID in my system is nm0000158
  - nm0000158 is known for:
    - tt0094737
    - tt0109830
    - tt0162222
  - nm0000158 had some kind of role in
    - tt0080202
    - tt0084314
    - tt0086927
    - tt0088161
    - tt0089543
    - tt0090274
    - tt0091019
    - tt0091541
    - tt0091653
    - … …
  - The ability to turn ttxxxxxxx into useful information.

How this all Work?







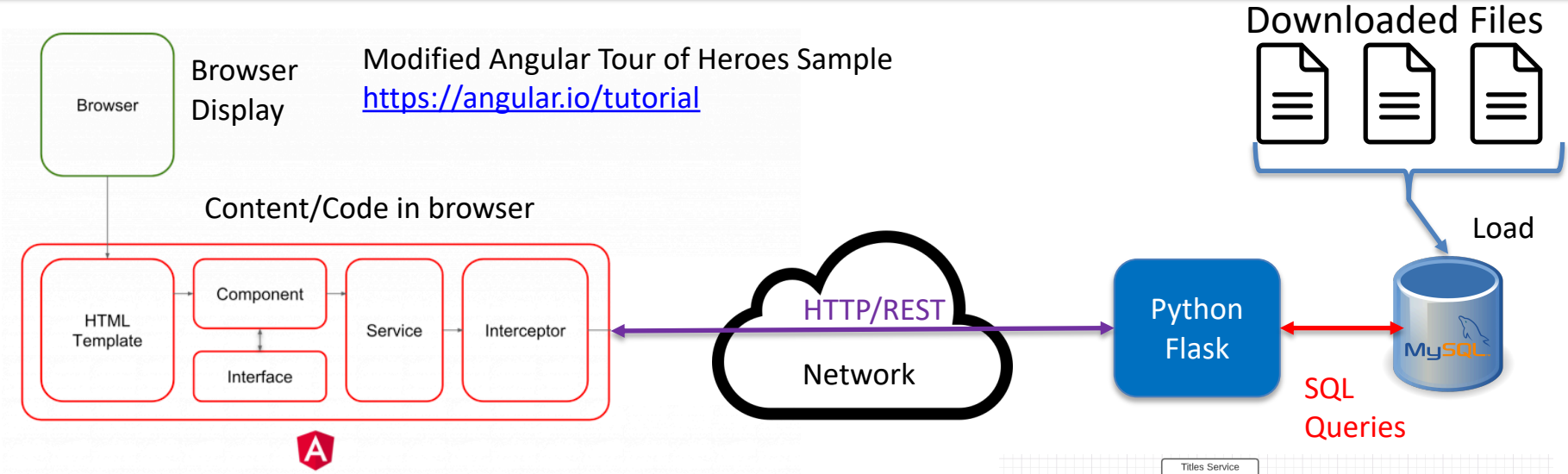


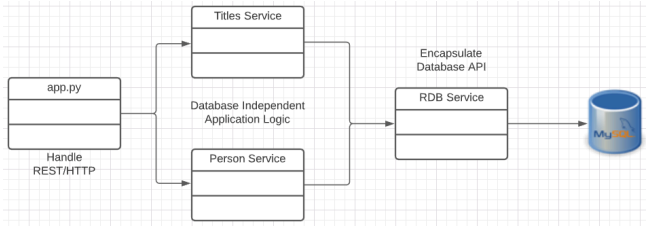
There is a page template with slots for "components"

W4170 teaches:
- Page design
- View
- ViewModel
- User experience
- ...

W4111 teaches
- Model
- Efficiency, design
- ... ...

A Simple IMDB-Like Web Application





Three components

1. Main application
2. People Dashboard
3. People Detail

One service: People

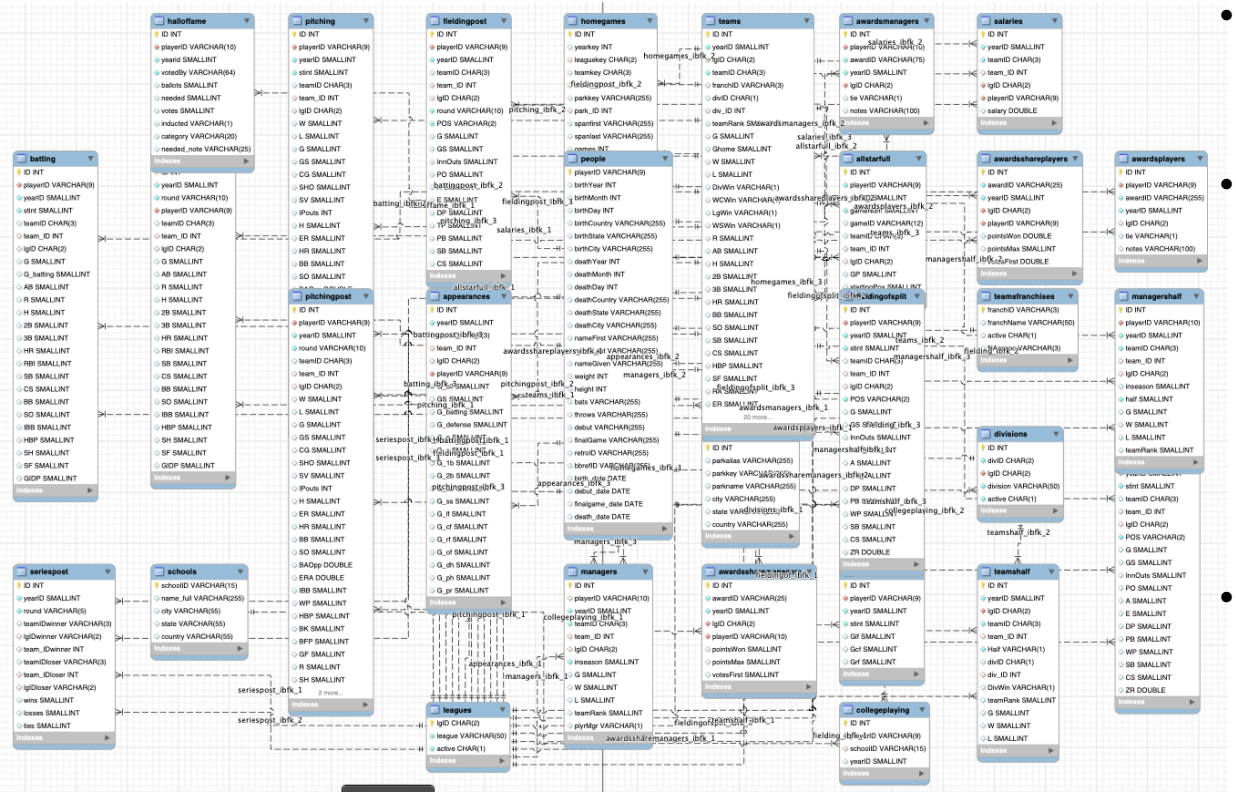
Very simple Python, Flask web application

https://flask.palletsprojects.com/en/1.1.x/

Let’s take a look ... ...

- Walkthrough, ...
  - Browser.
  - Code walkthrough.
  - Some simple tests.
- But, ...
  - I am going to use baseball data.
  - Because,
    - Over time we can do interesting numerical analysis and operations.
    - The database is a lot more complex.

The Dreaded Lahman 2019 Baseball DB



- This is:
  - An Entity-Relationship diagram.
  - In Crow's Foot Notation.
- The first few lectures will focus on:
  - Entity-Relationship Models
  - Notation and diagrams.
  - Realizations of the model:
    - *Relational (Algebra) Model.*
    - *SQL*
  - Understand how to use SQL to solve problems.
- Incrementally make progress on a project in the space of web applications and/or data engineering.

Core Concepts:

- Data
- Databases
- Database management systems.
- Applications, application architectures.
- Roles.
- History of databases.
- etc.

Covering these topics is not a good use of lecture time:

- Read the slides for chapter 1 from the recommended textbook. (https://www.db-book.com/db7/slides-dir/PPTX-dir/ch1.pptx)
- We will cover the material through questions on the reading, recitation, Piazza questions, ...

Approach for First Few Lectures

Lectures and Topics

- ▶ Chapter 1 Introduction
- ▼ PART ONE RELATIONAL LANGUAGES
  - ▶ Chapter 2 Introduction to the Relational M...
  - ▶ Chapter 3 Introduction to SQL
  - ▶ Chapter 4 Intermediate SQL
  - ▶ Chapter 5 Advanced SQL
- ▼ PART TWO DATABASE DESIGN
  - ▶ Chapter 6 Database Design Using the E-R ...
  - ▶ Chapter 7 Relational Database Design
- ▼ PART THREE APPLICATION DESIGN AND DEV...
  - ▶ Chapter 8 Complex Data Types

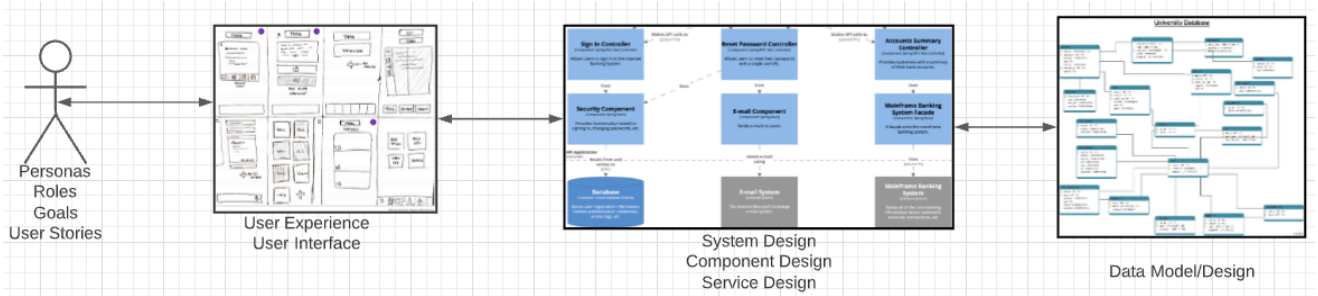
- Chapter 1:
  - Is interesting, but something easily learned from reading slides, docs, etc.
  - I can cover in OHs, recitation, Q&A, Piazza, ...
- Books and standard syllabuses tend to be sequential.
  Cover and complete topics one at a time.
- My view is that there is a core conceptual model with three realizations:
  - Entity, relationships, ...; the concepts.
  - ER design and modeling.
  - Implementation:
    - Relational model/algebra.
    - SQL.
    - Resource/REST oriented.
- My approach is to incrementally and iteratively:
  - Introduce concepts.
  - Explain the realizations.
  - Because a implementing a real system requires the approach of concept, design, implement in several layers.

Database design, Entity-Relationship Model (Part 1)

- Entity, relationship, attribute.
- Database design process.
- ER-Model and diagrams.

Problem Statement

- We must build a system that supports academic operations in a university.
- There are two major domains:
  - Interactive operations, e.g. register, choose class, assign grade, ...
  - Insight and reporting, e.g. enrollment trends, overloaded resources, ...
- We will design, develop, test and deploy the system iteratively and continuously.
- There are four core domains.



- In this course,
- We focus on the data dimension.
- We will get some insight into the other dimensions.

- The processes are iterative, with continuous extension and details.

Design Phases

- Initial phase -- characterize fully the data needs of the prospective database users.
- Second phase -- choosing a data model
  - Applying the concepts of the chosen data model
  - Translating these requirements into a conceptual schema of the database.
  - A fully developed conceptual schema indicates the functional requirements of the enterprise.
    - Describe the kinds of operations (or transactions) that will be performed on the data.

DFF Comments:

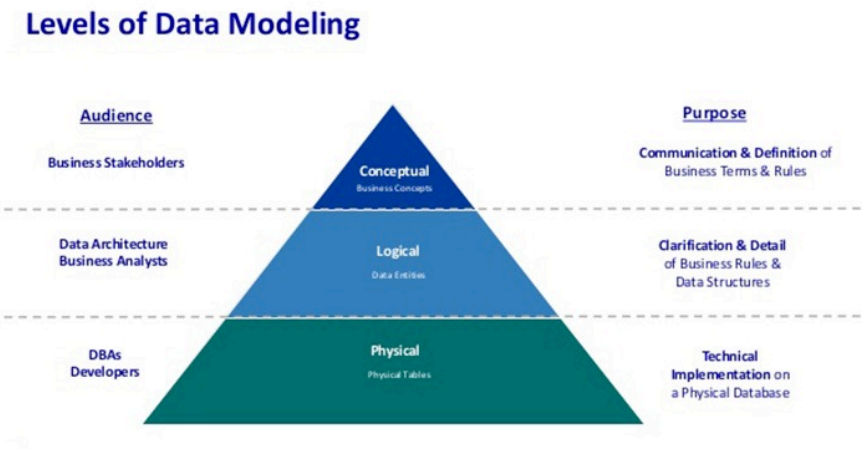
- We you see slides with this formatting, the come directly from the presentations associated with the textbook. (https://www.db-book.com/db7/slides-dir/index.html)
- The number at the bottom is of the form chapter.slide_no.
- I try to put my comments, modifications and annotations in red text, or inside a red rectangle/callout.

Design Phases (Cont.)

- Final Phase -- Moving from an abstract data model to the implementation of the database
  - Logical Design – Deciding on the database schema. Database design requires that we find a “good” collection of relation schemas.
    - Business decision – What attributes should we record in the database?
    - Computer Science decision – What relation schemas should we have and how should the attributes be distributed among the various relation schemas?
  - Physical Design – Deciding on the physical layout of the database

A Common and my Approach: Conceptual → Logical → Physical

https://ehikioya.com/conceptual-logical-physical-database-modeling/

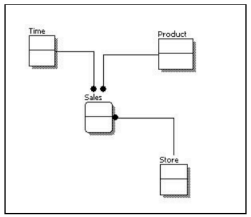


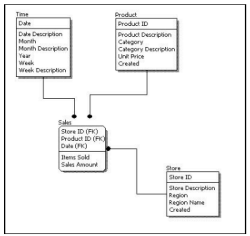
- It is easy to get carried away with modeling. You can spend all your time modeling and not actually build the schema.
- We will use the approaches in class.
- Mostly to understand concepts and patterns.

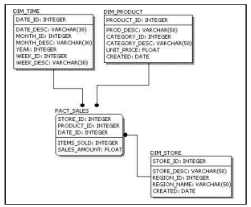
https://www.1keydata.com/datawarehousing/data-modeling-levels.html

Feature	Conceptual	Logical	Physical
Entity Names	✓	✓	
Entity Relationships	✓	✓	
Attributes		✓	
Primary Keys		✓	✓
Foreign Keys		✓	✓
Table Names			✓
Column Names			✓
Column Data Types			✓

Conceptual Model Design | Logical Model Design | Physical Model Design







https://www.1keydata.com/datawarehousing/data-modeling-levels.html

ER model -- Database Modeling

- The ER data mode was developed to facilitate database design by allowing specification of an **enterprise schema** that represents the overall logical structure of a database.
- The ER data model employs three basic concepts:
  - entity sets,
  - relationship sets,
  - attributes.
- The ER model also has an associated diagrammatic representation, the **ER diagram**, which can express the overall logical structure of a database graphically.

Entity Sets

- An **entity** is an object that exists and is distinguishable from other objects.
  - Example: specific person, company, event, plant
- An **entity set** is a set of entities of the same type that share the same properties.
  - Example: set of all persons, companies, trees, holidays
- An entity is represented by a set of attributes; i.e., descriptive properties possessed by all members of an entity set.
  - Example:

    instructor = (*ID, name, salary*)
    course= (*course_id, title, credits*)
- A subset of the attributes form a **primary key** of the entity set; i.e., uniquely identifying each member of the set.

DFF Comments:

- Some of these statements apply primarily to OO systems and the relational/SQL models.
- A motivation for "No SQL" is to relax the constraints.

Entity Sets -- *instructor* and *student*

76766	Crick
45565	Katz
10101	Srinivasan
98345	Kim
76543	Singh
22222	Einstein

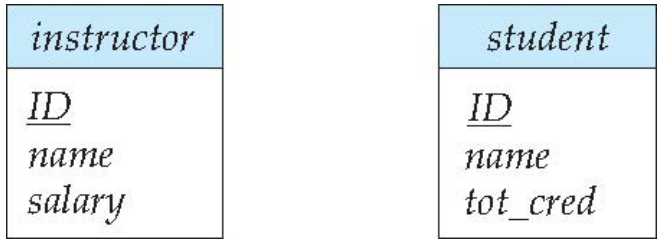
instructor

98988	Tanaka
12345	Shankar
00128	Zhang
76543	Brown
76653	Aoi
23121	Chavez
44553	Peltier

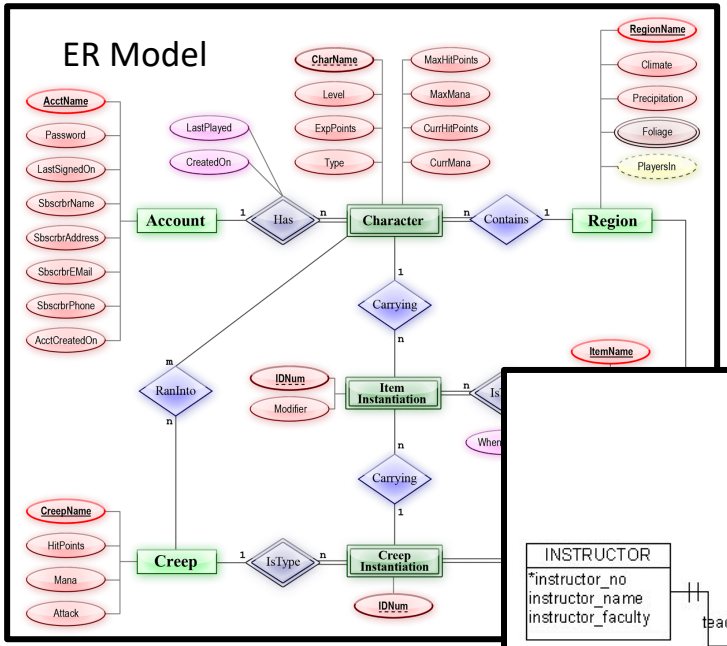
student

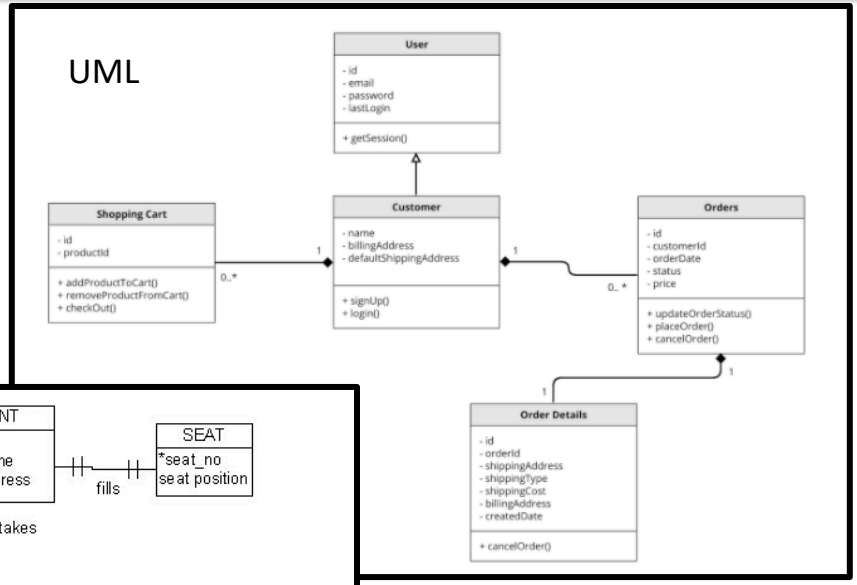
Representing Entity sets in ER Diagram

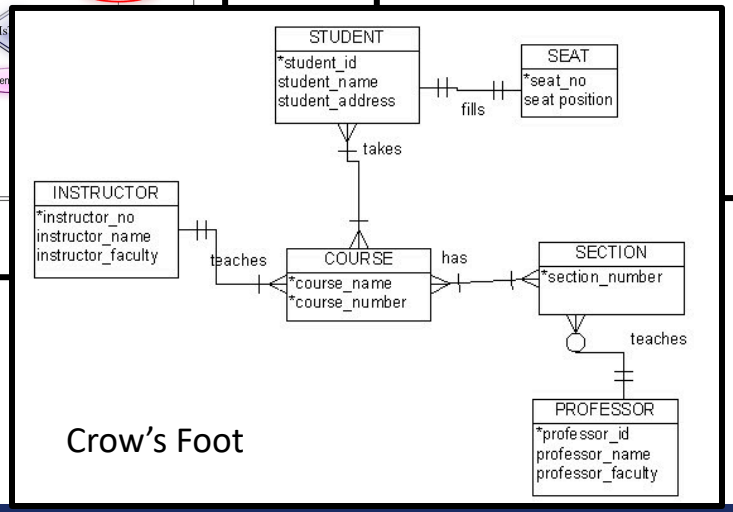
- Entity sets can be represented graphically as follows:
  - Rectangles represent entity sets.
  - Attributes listed inside entity rectangle
  - Underline indicates primary key attributes



Visual Notation – Many Notations



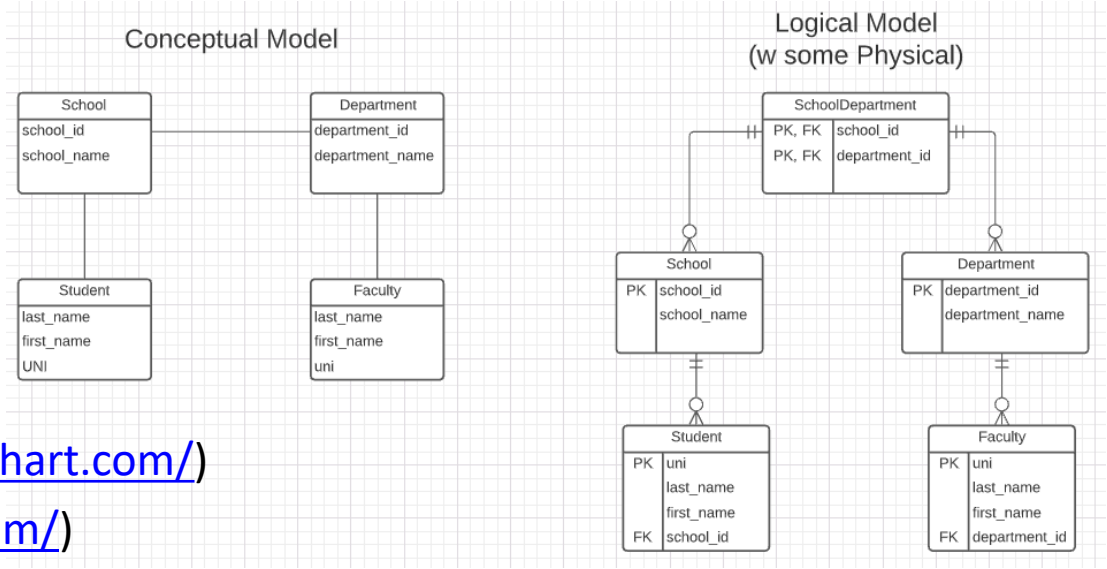




- "Other," i.e. PowerPoint is the most common modeling notation.
- It is easy to get "carried away."
- The trick is to do "just enough modeling."
- I mostly use Crow's Foot
  - It is "just enough"
  - But lacks some capabilities.
- The book uses ER notation.

First Iteration/Step for University Data Model

- This is the level of detail I want when I ask for:
  - Conceptual Model diagram.
  - Logical Model diagram.
- Some online tools with "free," constrained usage.
  - Lucidchart (https://www.lucidchart.com/)
  - Vertabelo (https://vertabelo.com/)
- The model has:
  - Four entity sets: *School, Department, Student, Faculty*
  - Three relationship sets: *Student-School, School-Department, Faculty-Department.*



Entity Sets -- *instructor* and *student*

76766	Crick
45565	Katz
10101	Srinivasan
98345	Kim
76543	Singh
22222	Einstein

instructor

98988	Tanaka
12345	Shankar
00128	Zhang
76543	Brown
76653	Aoi
23121	Chavez
44553	Peltier

student

DFF Comments: Just a reminder of a previous slide.

Relationship Sets

- A **relationship** is an association among several entities

  Example:

  44553 (Peltier) — *student* entity; *advisor* — relationship set; 22222 (Einstein) — *instructor* entity

- A **relationship set** is a mathematical relation among $n \geq 2$ entities, each taken from entity sets

  $$\{(e_1, e_2, \ldots e_n) \mid e_1 \in E_1, e_2 \in E_2, \ldots, e_n \in E_n\}$$

  where $(e_1, e_2, \ldots, e_n)$ is a relationship

  - Example:

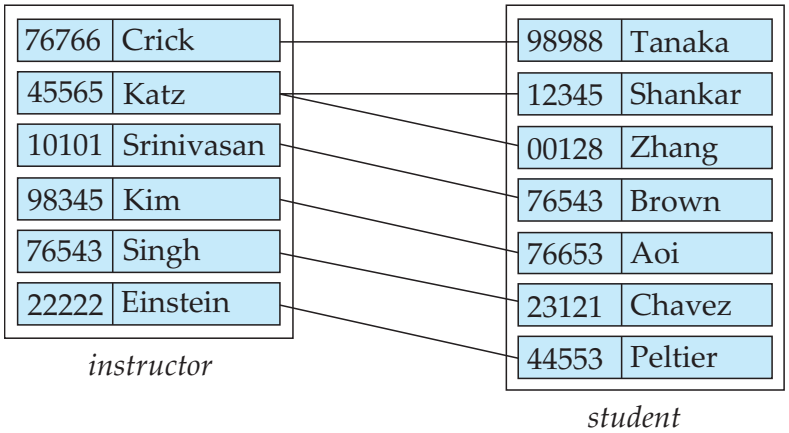
    $(44553,22222) \in advisor$

DFF Comments:

- Nobody thinks about relationships this way.
- There is no idea so simple that a DB professor cannot make it confusing, usually by using math.

Relationship Sets (Cont.)

- Example: we define the relationship set *advisor* to denote the associations between students and the instructors who act as their advisors.
- Pictorially, we draw a line between related entities.



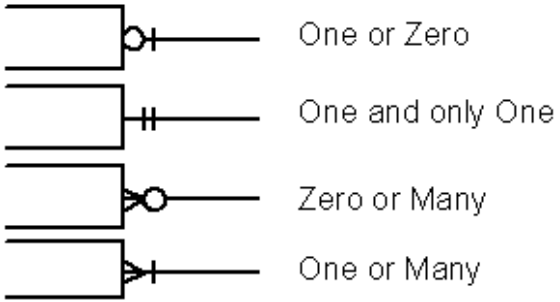
DFF Comments:

- Nobody draws the diagrams this way, but ...
- Sometimes thinking this way helps understand other ways to depict the concept.

Notation has Precise Meaning

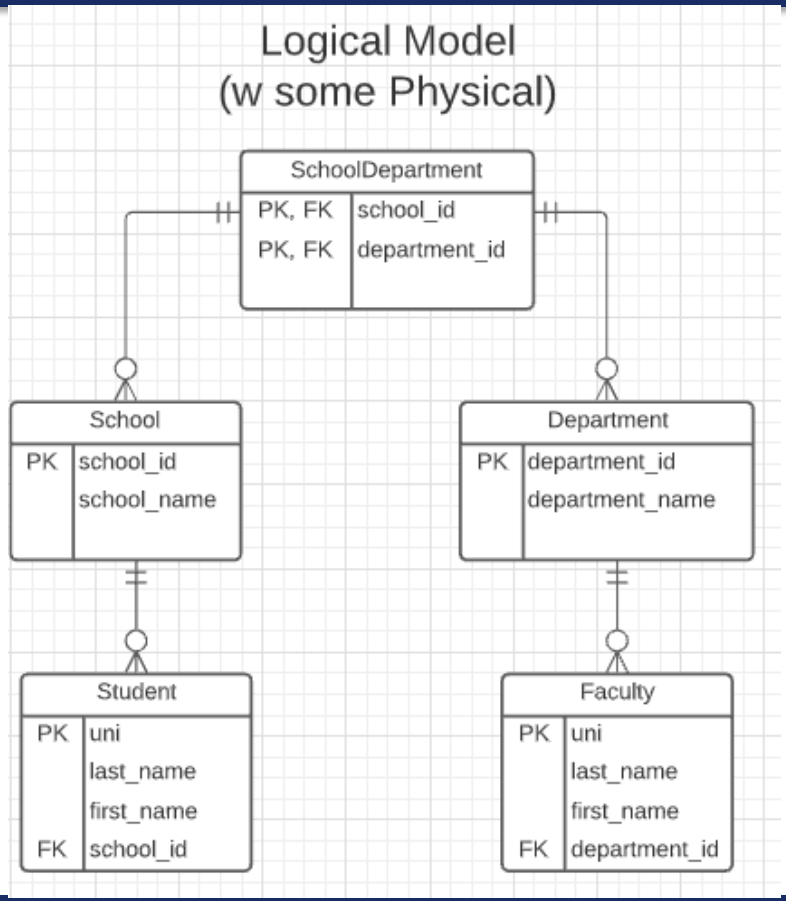
- Attribute annotations:
  - PK = Primary Key
  - FK = Foreign Key
- Line annotations:

Summary of Crow's Foot Notation



- We will spend a lot of time discussing keys.
- We will start in a couple of slides.

We will learn over time and there are good tutorials (https://www.lucidchart.com/pages/er-diagrams) to help study and refresh.



What Does this Mean? Let's Get Started

Primary Key means that the value occurs **at most once.**

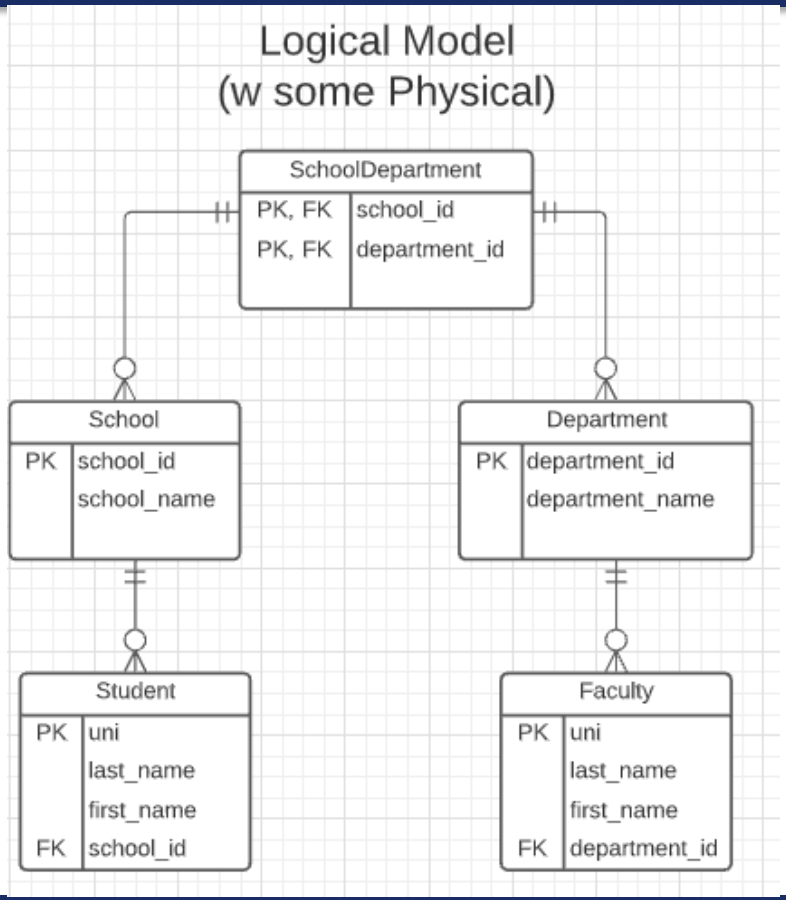
School Code	School Name
CC	Columbia College
SEAS	Fu Foundation School of Engineering and Applied Science
GSAS	Graduate School of Arts and Sciences
GS	General Studies
… …	… …

Foreign Key means that if a value occurs in school_id for any row, there must be a row in School with that key.

UNI	Last name	First name	school_id
dff9	Ferguson	Donald	CC
js11	Smith	John	GS
jp9	Public	James	CC
bb101	Baggins	Bilbo	CC
… …	… …	… …	… …

The line notations mean:

- A student is related to EXACTLY ONE school.
- A School may be related to 0, 1 or many students.



ER model -- Database Modeling

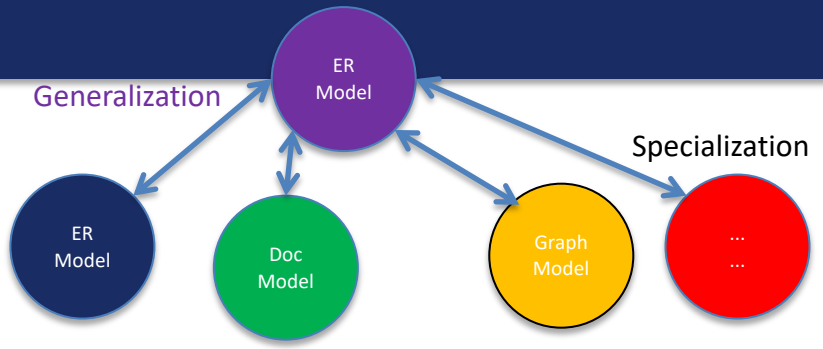
- The ER data mode was developed to facilitate database design by allowing specification of an **enterprise schema** that represents the overall logical structure of a database.
- The ER data model employs three basic concepts:
  - entity sets,
  - relationship sets,
  - attributes.
- The ER model also has an associated diagrammatic representation, the **ER diagram**, which can express the overall logical structure of a database graphically.

DFF Comments:

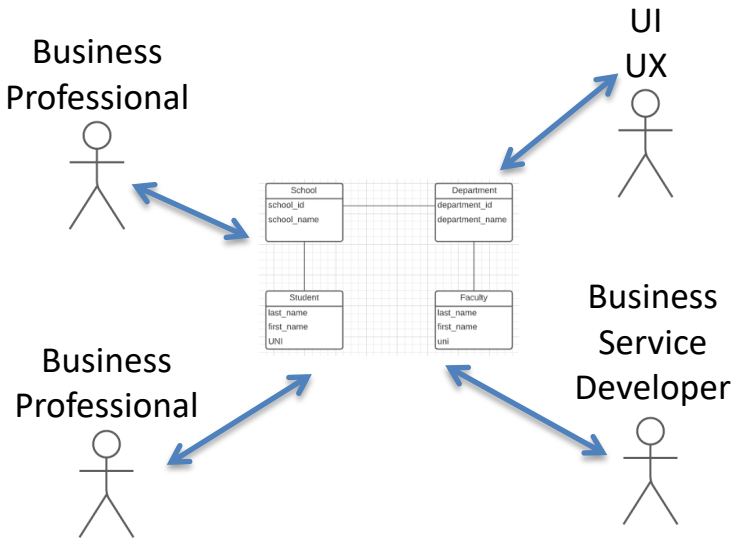
- The book and slides do not do a great job of motivating the ER model or ER diagrams.
- Why do people and teams think about or use the ER model and modeling?

ER Model and ER Modeling

- ER Model: Agility, Separation of Concerns
  - ER model is a generalization that most DB models implement in some form.
  - Using the ER model enables:
    - Thinking about and collaborating on design with getting bogged down in details.
    - Enable flexible choices about how to realize/Implement data.



- ER Diagrams: Communication, Quality, Precision
  - With a little experience, everyone can understand and ER diagram.
  - Easier to discuss and collaborate on application's data than showing SQL table definitions, JSON, ...
  - People think visually. That is why we have whiteboards. ER diagrams are precise and unambiguous.
  - Guides you to think about relationships, keys, ... And prevents "re-dos" later in the process. It is easier to fix a diagram than a database schema.



ER Modeling – Reasonably Good Summary

Advantages of ER Model

Conceptually it is very simple: ER model is very simple because if we know relationship between entities and attributes, then we can easily draw an ER diagram.

Better visual representation: ER model is a diagrammatic representation of any logical structure of database. By seeing ER diagram, we can easily understand relationship among entities and relationship.

Effective communication tool: It is an effective communication tool for database designer.

Highly integrated with relational model: ER model can be easily converted into relational model by simply converting ER model into tables.

Easy conversion to any data model: ER model can be easily converted into another data model like hierarchical data model, network data model and so on.

Disadvantages of ER Model

Limited constraints and specification

Loss of information content: Some information be lost or hidden in ER model

Limited relationship representation: ER model represents limited relationship as compared to another data models like relational model etc.

No representation of data manipulation: It is difficult to show data manipulation in ER model.

Popular for high level design: ER model is very popular for designing high level design

No industry standard for notation

https://pctechnicalpro.blogspot.com/2017/04/advantages-disadvantages-er-model-dbms.html

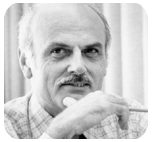
Note:

- If you get to use Google to help with take home exams, HW, etc.
- I get to use Google to help with slides.

The Relational Model

Relational Model

- All the data is stored in various tables.
- Example of tabular data in the relational model



Ted Codd
Turing Award 1981

Columns

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

Rows

(a) The *instructor* table

- The “relation” is the “table.”
  - In my big space of pieces of data. *ID, name, dept_name, salary* are somehow related.
  - This causes confusion, because the ER and other models use “relation” to mean something else.
- Core concepts:
  - Relation
  - Tuple (Row)
  - Column (Attribute)

Example of a *Instructor* Relation

attributes
(or columns)

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

tuples
(or rows)

Attribute

- The set of allowed values for each attribute is called the **domain** of the attribute
- Attribute values are (normally) required to be **atomic**; that is, indivisible
- The special value ***null*** is a member of every domain. Indicated that the value is “unknown”
- The null value causes complications in the definition of many operations

DFF Comments:

- I will explain the importance of atomic attributes and null in examples.
- Atomic and use of Null is important?

Relations are Unordered

- Order of tuples is irrelevant (tuples may be stored in an arbitrary order)
- Example: *instructor* relation with unordered tuples

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

Database Schema

- Database schema -- is the logical structure of the database.
- Database instance -- is a snapshot of the data in the database at a given instant in time.
- Example:
  - schema: i*nstructor* (*ID, name, dept_name, salary*)
  - Instance:

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

Keys

- Let K $\subseteq$ R
- *K* is a **superkey** of *R* if values for *K* are sufficient to identify a unique tuple of each possible relation *r(R)*
  - Example: {*ID*} and {ID,name} are both superkeys of *instructor.*
- Superkey *K* is a **candidate key** if *K* is minimal
  Example: {*ID*} is a candidate key for *Instructor*
- One of the candidate keys is selected to be the **primary key**.
  - which one?
- **Foreign key** constraint: Value in one relation must appear in another
  - **Referencing** relation
  - **Referenced** relation
  - Example: *dept_name* in i*nstructor* is a foreign key from *instructor* referencing *department*

Notation

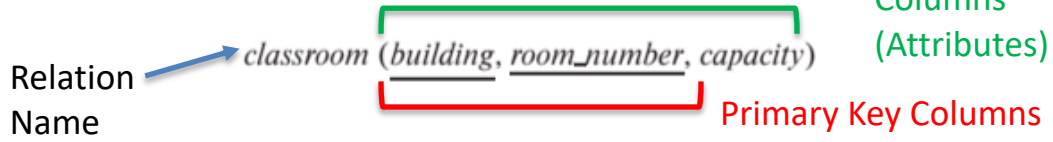
Classroom relation

building	room_number	capacity
Packard	101	500
Painter	100	125
Painter	514	10
Taylor	3128	70
Watson	100	30
Watson	120	50

classroom schema

It is customary to list the primary key attributes of a relation schema before the other attributes; for example, the *dept_name* attribute of *department* is listed first, since it is the primary key. Primary key attributes are also underlined.

Consider the *classroom* relation:



- The primary key is a *composite key.* Neither column is a key (unique) by itself.
- Keys are statements about all possible, valid tuples and not just the ones in the relation.
  - Capacity is unique in this specific data, but clearly not unique for all possible data.
  - In this domain, there cannot be two classrooms with the same building and room number.
- Relation schema:
  - Underline indicates a primary key column. There is no standard way to indicate other types of key.
  - We will use **bold** to indicate foreign keys.
  - You will sometimes see things like *classroom(building:string, room_number:number, capacity:number)*

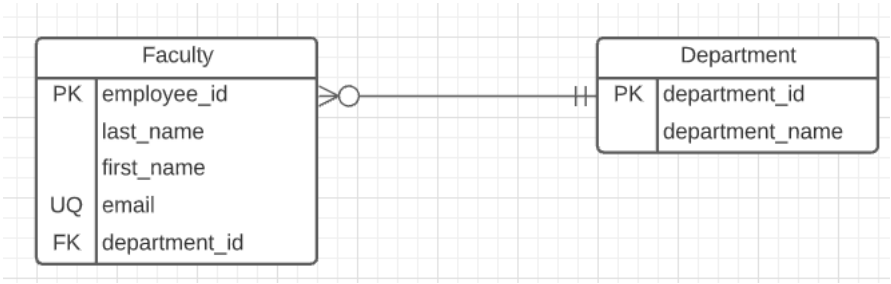
Observations

- Keys:
  - Will be baffling. It takes time and experience to understand/appreciate.
  - There are many, many types of keys with formal definitions.
  - I explain the formality but focus on the concepts and applications.

- No one uses the formal, relational model. So, why do we study it?
  - Is very helpful when understanding concepts that we cover later in the course, especially query optimization and processing.
  - There are many realizations of the model and algebra, and understanding the foundation helps with understanding language/engine capabilities.
  - The model has helped with innovating new approaches, and you may innovate in data and query models in your future.

Start Building our Datamodel

- First two relations:
  - Departments(title, id)
  - Faculty(employee_id, first_name, last_name, email, **department_id**)
- Note:
  - *email* is almost certainly a *candidate key.*
  - A primary key is chosen arbitrarily. Why did I use *employee_id?*
    - Heuristic: People may change emails. I may lose "identity" in the data over time. Same person but two different emails.
    - employee_id is *immutable.* I cannot change.
- ER Diagram:
  - Note: "UQ" is my non-standard convention.
  - Relationship:
    - A faculty "has" exactly one department.
    - A department as 0, 1 or many faculty.

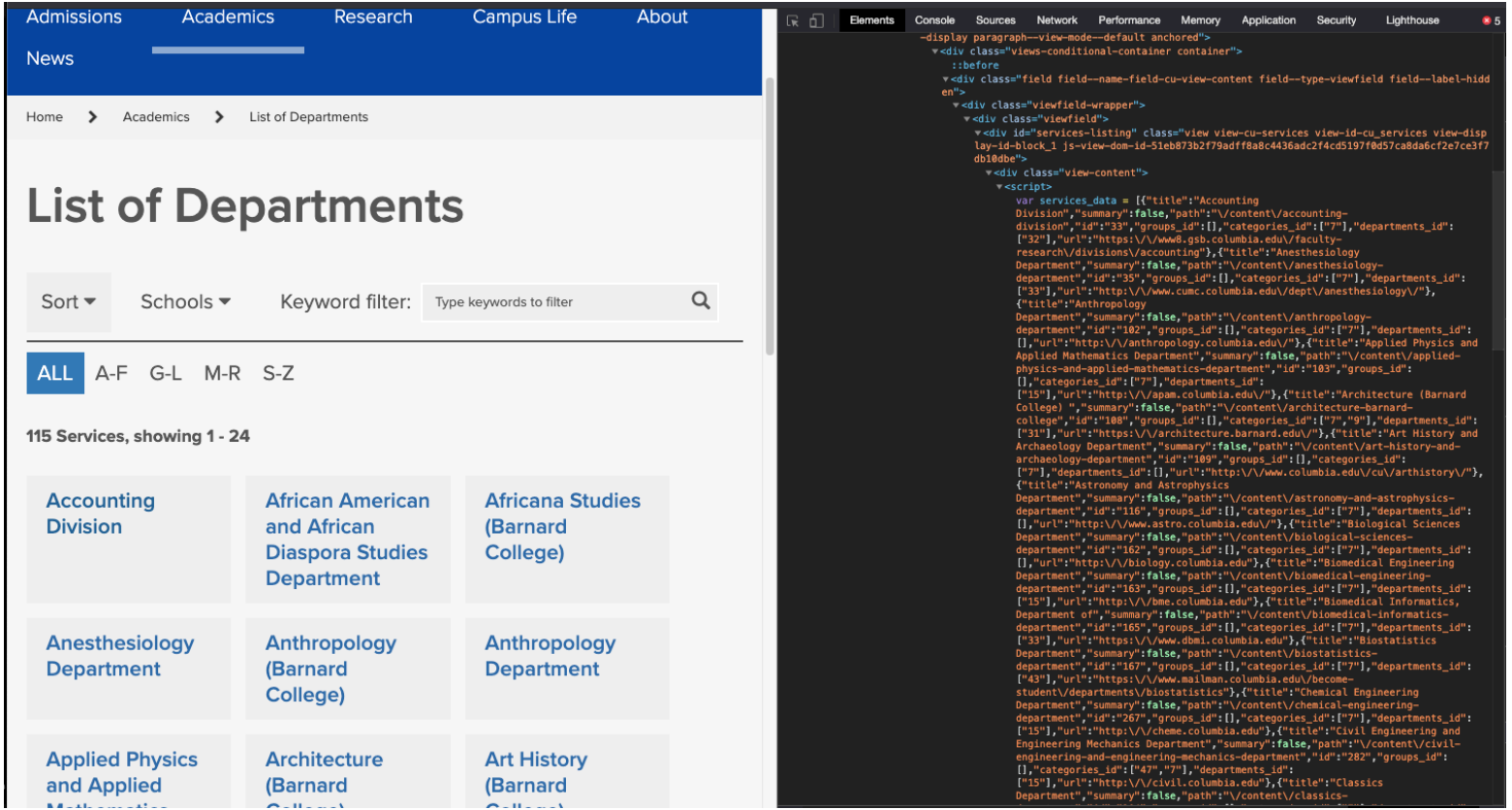


Start Building our Datamodel

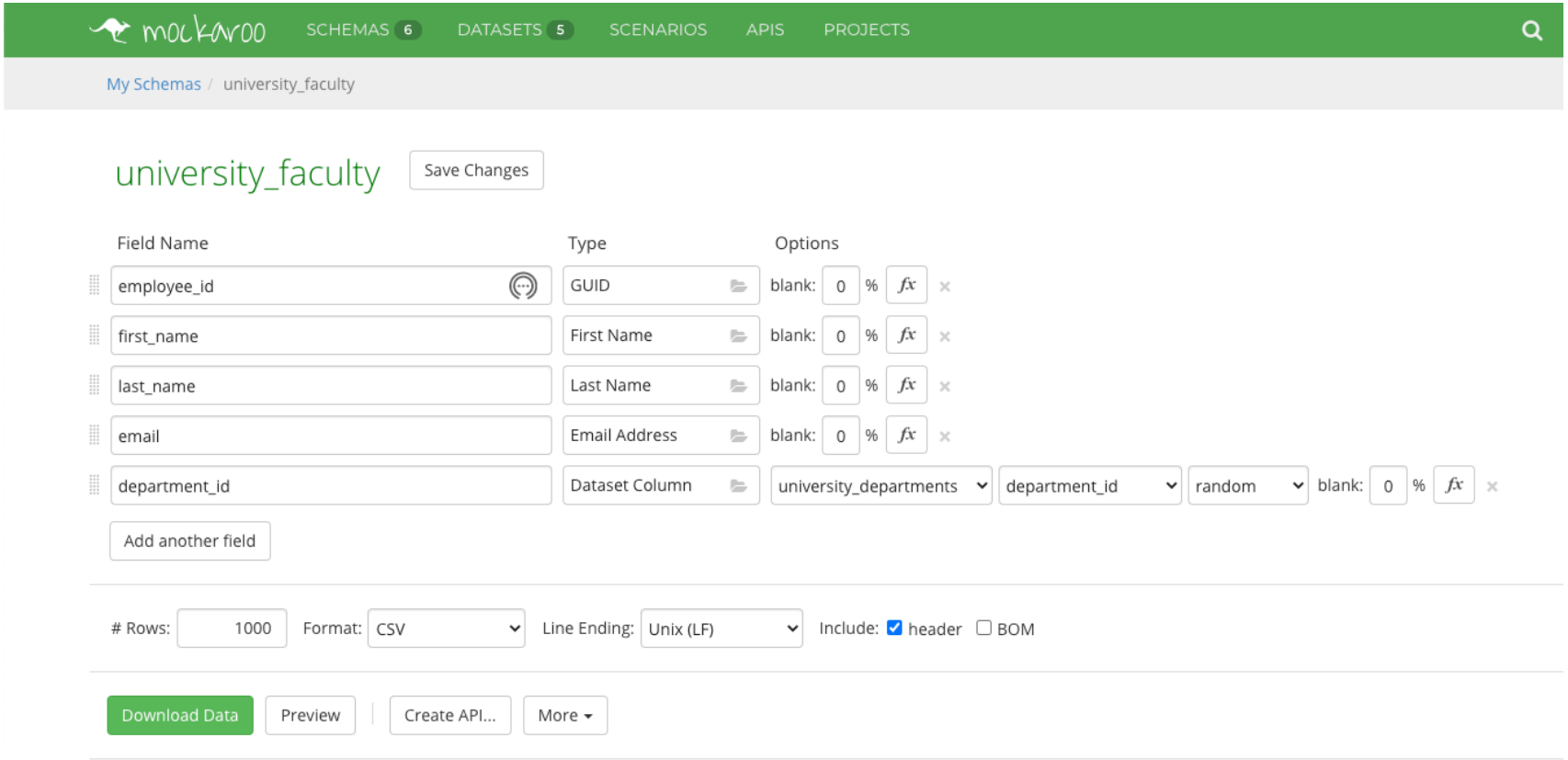
OK, that is interesting, but ... ...

1. Where does the data come from?
    1. Users enter and update data through an application.
    2. You can *import* pre-existing data.
    3. You can generate test data.
2. How do you "manipulate" the data (CRUD)
    1. Create.
    2. Retrieve.
    3. Update.
    4. Delete.

Import the Data – I screen-scraped the Website



Generate Test Data – I Used to Tool



Relational Query Languages

- Procedural versus non-procedural, or declarative
- "Pure" languages:
  - Relational algebra
  - Tuple relational calculus
  - Domain relational calculus
- The above 3 pure languages are equivalent in computing power
- We will concentrate in this chapter on relational algebra
  - Not turning-machine equivalent
  - Consists of 6 basic operations

DFF Comments:

- You will sometimes see other operator, e.g. ← Assignment.
- Relational algebra focuses on *retrieve.* You can sort of do Create, Update, Delete.
- The SQL Language, which we will see, extends relational algebra.

Relational Algebra

- A procedural language consisting of a set of operations that take one or two relations as input and produce a new relation as their result.
- Six basic operators
  - select: σ
  - project: Π
  - union: $\cup$
  - set difference: –
  - Cartesian product: x
  - rename: ρ

Select Operation

- The **select** operation selects tuples that satisfy a given predicate.
- Notation: $\sigma_p(r)$
- p is called the **selection predicate**
- Example: select those tuples of the *instructor* relation where the instructor is in the "Physics" department.
  - Query

$$\sigma_{dept\_name="Physics"}(instructor)$$

  - Result

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
33456	Gold	Physics	87000

Select Operation (Cont.)

- We allow comparisons using

  $=, \neq, >, \geq, <, \leq$

  in the selection predicate.
- We can combine several predicates into a larger predicate by using the connectives:

  $\wedge$ (**and**), $\vee$ (**or**), $\neg$ (**not**)
- Example: Find the instructors in Physics with a salary greater $90,000, we write:

  $\sigma_{dept\_name=\text{“Physics”} \wedge salary > 90,000}(instructor)$
- Then select predicate may include comparisons between two attributes.
  - Example, find all departments whose name is the same as their building name:
  - $\sigma_{dept\_name=building}(department)$

Project Operation

- A unary operation that returns its argument relation, with certain attributes left out.
- Notation:

$$\Pi_{A_1, A_2, A_3 \ldots A_k}(r)$$

  where A_1, A_2 are attribute names and r is a relation name.
- The result is defined as the relation of k columns obtained by erasing the columns that are not listed
- Duplicate rows removed from result, since relations are sets

Project Operation (Cont.)

- Example: eliminate the *dept_name* attribute of *instructor*
- Query*:*

$$\Pi_{ID,\ name,\ salary}\ (instructor)$$

- Result:

ID	*name*	*salary*
10101	Srinivasan	65000
12121	Wu	90000
15151	Mozart	40000
22222	Einstein	95000
32343	El Said	60000
33456	Gold	87000
45565	Katz	75000
58583	Califieri	62000
76543	Singh	80000
76766	Crick	72000
83821	Brandt	92000
98345	Kim	80000

Composition of Relational Operations

- The result of a relational-algebra operation is relation and therefore of relational-algebra operations can be composed together into a **relational-algebra expression**.
- Consider the query -- Find the names of all instructors in the Physics department.

$$\Pi_{name}(\sigma_{dept\_name = \text{“Physics”}}(instructor))$$

- Instead of giving the name of a relation as the argument of the projection operation, we give an expression that evaluates to a relation.

Relax Calculator

- Let's look at an online tool that you will use.
- RelaX (https://dbis-uibk.github.io/relax/calc/local/uibk/local/0)
- The calculator:
  - Does have the sample data from the textbook.
  - You can also upload new data. (Show how and example)
- Some queries:
  - σ title='Computer Science Department' ∨ title='Accounting Division' (departments)
  - π last_name, email (faculty)
  - π last_name, email (
    σ department_id=165 ∨ department_id=1010 (faculty)
)

SQL

History

- IBM Sequel language developed as part of System R project at the IBM San Jose Research Laboratory
- Renamed Structured Query Language (SQL)
- ANSI and ISO standard SQL:
  - SQL-86
  - SQL-89
  - SQL-92
  - SQL:1999 (language name became Y2K compliant!)
  - SQL:2003
- Commercial systems offer most, if not all, SQL-92 features, plus varying feature sets from later standards and special proprietary features.
  - Not all examples here may work on your particular system.

SQL Parts

- DML -- provides the ability to query information from the database and to insert tuples into, delete tuples from, and modify tuples in the database.
- integrity – the DDL includes commands for specifying integrity constraints.
- View definition -- The DDL includes commands for defining views.
- Transaction control –includes commands for specifying the beginning and ending of transactions.
- Embedded SQL and dynamic SQL -- define how SQL statements can be embedded within general-purpose programming languages.
- Authorization – includes commands for specifying access rights to relations and views.

SQL Language Statements

The core SQL language statements are:

- SELECT: Implements both σ, π
- INSERT
- UPDATE
- DELETE
- CREATE TABLE
- ALTER TABLE
- JOIN, which is an operator within SELECT.

π last_name, email (
    σ department_id=165 ∨ department_id=1010 (faculty)
)

=

SELECT last_name, email FROM faculty
WHERE department_id=165 or department_id=1019

- Many, if not most, SQL statements:
  - Require multiple relational algebra expressions.
  - Cannot easily (or at all) be represented in relational algebra.

Data Definition Language

The SQL data-definition language (DDL) allows the specification of information about relations, including:

- The schema for each relation.
- The type of values associated with each attribute.
- The Integrity constraints
- The set of indices to be maintained for each relation.
- Security and authorization information for each relation.
- The physical storage structure of each relation on disk.

Create Table Construct

- An SQL relation is defined using the **create table** command:

  create table *r*

  (A_1 D_1, A_2 D_2, ..., A_n D_n,
  (integrity-constraint$_1$),
  ...,
  (integrity-constraint$_k$))

  - *r* is the name of the relation
  - each A_i is an attribute name in the schema of relation *r*
  - D_i is the data type of values in the domain of attribute A_i

- Example:

  create table *instructor* (
  ID **char**(5),
  name **varchar**(20),
  dept_name **varchar**(20),
  salary **numeric**(8,2))

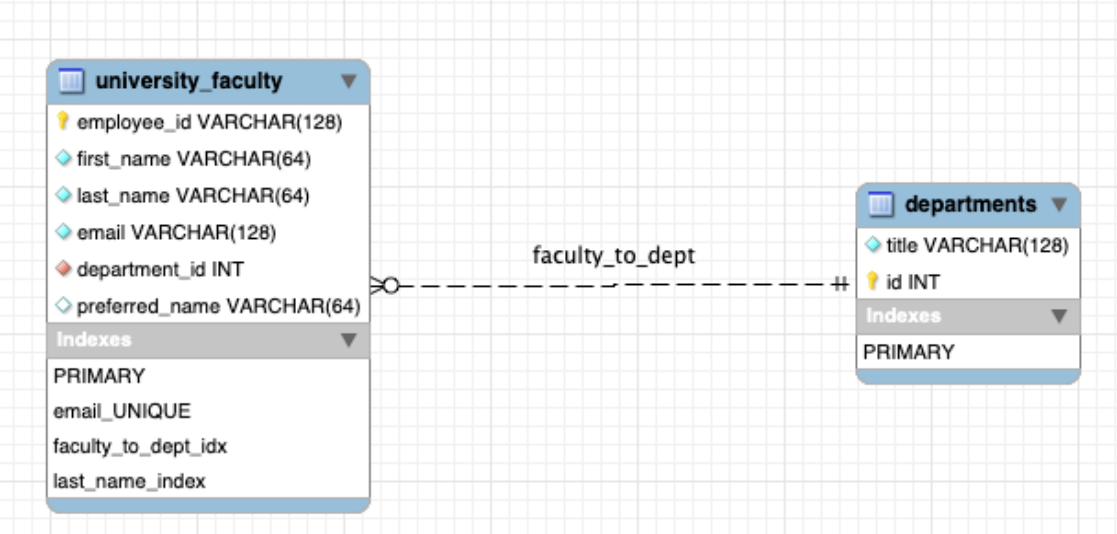
SQL Create Table for our Datamodel

```
CREATE TABLE `departments` (
  `title` varchar(128) NOT NULL,
  `id` int NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB
      DEFAULT CHARSET=utf8mb4
      COLLATE=utf8mb4_0900_ai_ci;

CREATE TABLE `university_faculty` (
  `employee_id` varchar(128) NOT NULL,
  `first_name` varchar(64) NOT NULL,
  `last_name` varchar(64) NOT NULL,
  `email` varchar(128) NOT NULL,
  `department_id` int NOT NULL,
  `preferred_name` varchar(64) DEFAULT NULL,
  PRIMARY KEY (`employee_id`),
  UNIQUE KEY `email_UNIQUE` (`email`),
  KEY `faculty_to_dept_idx` (`department_id`),
  KEY `last_name_index` (`last_name`),
  CONSTRAINT `faculty_to_dept` FOREIGN KEY (`department_id`) REFERENCES `departments` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

- The ` (back tick) is optional in most cases.
- You can look up the various column types.
  - INT is an integer.
  - VARCHAR(N) is a variable length string with maximum length 128.
- DEFAULT CHARSET, ENGINE, COLLATE are MySQL specific and you can usually ignore.
- NOT NULL is a Constraint. Constraints are core parts of SQL and we will cover in detail.
- Keys/Constraints/Index
  - You can see PRIMARY KEY definition.
  - UNIQUE KEY is similar to Candidate Key. UNIQUE KEY and the columns NOT NULL is a Candidate Key.
  - Foreign key defines source columns and reference columns.
  - “Key” sometimes means “Index” and is just used for performance.
- Some databases conflate the concepts of Key and Index. We will cover both keys and indexes in detail.

Crow's Foot Like Physical Diagram



- It is possible to draw a physical model in an ER Diagram tool and generate the CREATE TABLE statements.
  - I never to do.
  - I find it more trouble than it is worth.
- I will *reverse engineer* an existing database to understand its schema.

Switch to Jupyter Notebook and DataGrip

- Show some queries.