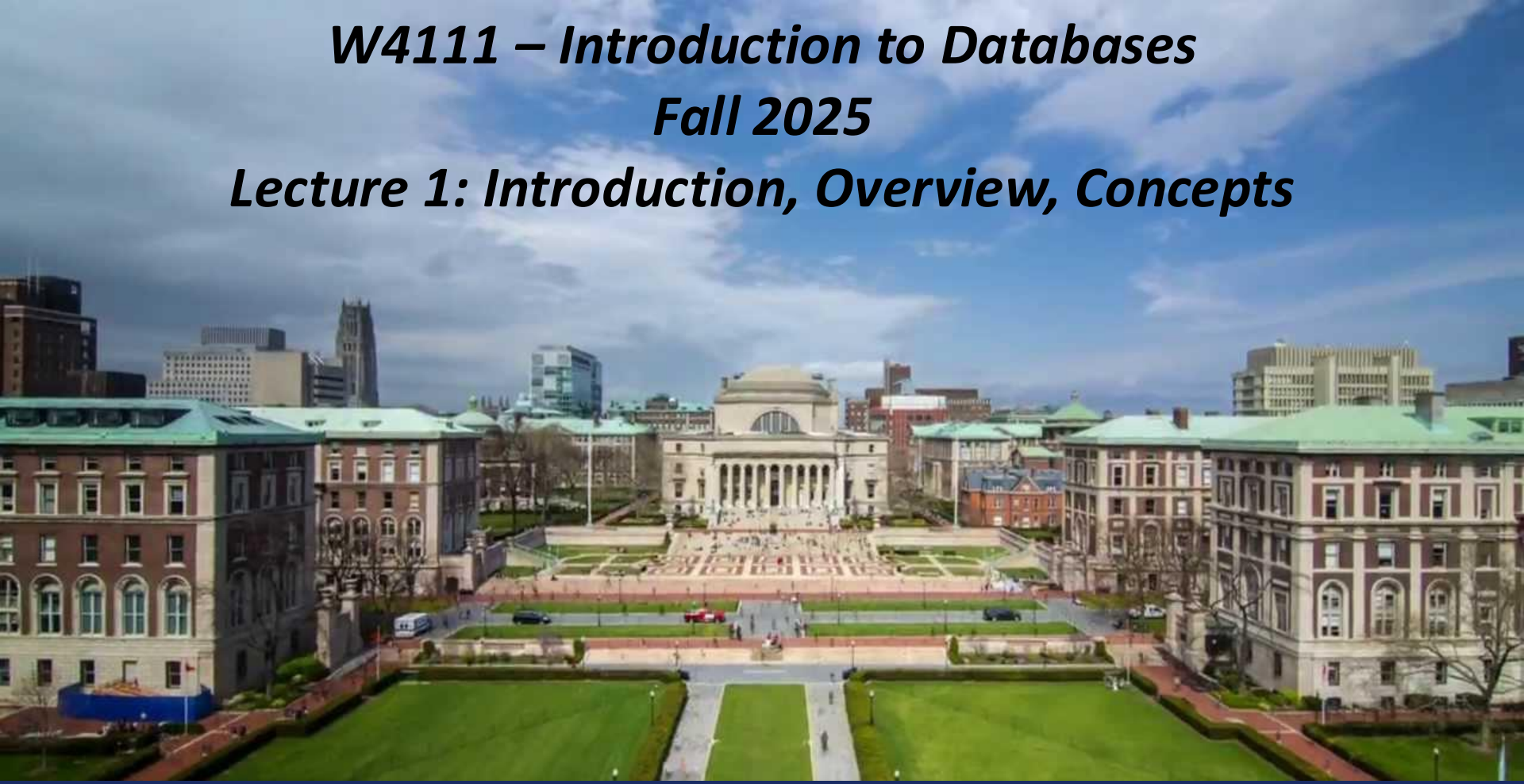
W4111 – Introduction to Databases
Fall 2025
Lecture 1: Introduction, Overview, Concepts



Waitlists and Enrollments

Faculty do not manage waitlists
for some courses, including W4111.

The academic admin staff in the
CS Department manages the waitlist,
priorities and enrollment.

You should contact advising email:

ug-advising@cs.columbia.edu, ms-advising @cs.columbia.edu
or
phd-advising @cs.columbia.edu

Contents

Contents

- Course overview:
  - What you really care about – homework, exams, grading
  - About your instructor and teaching assistants
  - Course objectives and content
  - Approach to lectures and reading material
  - Student resources and environment
- Introduction: Data, Databases, Database Management Systems
  - ER model, ER diagrams, relational model, relational algebra, SQL
  - Database/data-centric applications
- Concepts
  - Entity Relationship Model, Entity Relationship Diagrams
  - Foundational Relational Model and Algebra
  - Relational Database Management Systems and Structured Query Language
- Homework assignments HW0 and HW 1

Course Overview

Homework, Exams, Grading, Attendance

Homework and Grading

- Overview
  - The course grade will be in the range 0 to 100.
  - The mean/median grade will be approximately a B+, using the standard mapping of numerical value to letter grade. That is, 87 to 89 points is a B+.
  - I will “curve up” if necessary. I will not curve down.
- Point value:
  - HW1, HW2, …, HW5 are each worth 5 points.
  - There will be 3 non-cumulative in lecture, written exams. Each is worth 25 points.
  - You MUST take the exams at the specified date and times unless there are very strong, extenuating circumstances.
- Additional information:
  - A homework will have written questions and practical implementation tasks. The tasks incrementally implement a small project.
  - Completing and understanding the homework assignments **without** using ChatGPT, etc. will significantly improve your performance on the exams.

Thoughts on ChatGPT and Generative AI

- I used to only give homework and take-home exams.
  - ChatGPT and generative AI force faculty to fundamentally change the approach to homework and exams.
  - Some students will use generative AI for take home assignments and take-home exams.
    - These students will not learn, internalize and understand how to apply the concepts of this course.
    - Their having ChatGPT do the homework and exams is unfair to students that do the work themselves.
- There is a long, long history of tools to increase productivity.
  - I use generative AI to make me more productive for SW development, presentations, documents, spreadsheets, … …
    I also use generative AI to help me learn a new domain.
  - But, I must understand the material and domain
    - To define the overall project structure, tasks, convert vague requirements to a concrete design, form the prompts, … …
    - To validate and "tinker with" the generated results.
  - Generative AI and ChatGPT is about 97% accurate on homework assignments and exam problems. But, 97% accuracy on
    - Writing an accounting, reporting and compliance application ➔ bankruptcy and/or jail.
    - Computing optimal parameters for jet engine design from data ➔ unscheduled, rapid disassembly.
    - Using a database to maintain patient medication records ➔ dead patients.
- ➔ I must teach you and you must learn the material, or this course is a waste of our time.
- <u>Doing the homework assignments without over reliance on generative AI will significantly, significantly improve your performance on exams.</u>

Lectures and Attendance

- I will stream and record lectures.

- Why do I record lectures?
  - There will be times this semester when I have a compelling reason for not being in-person. If I cannot always be in person, I cannot expect that you will.
  - There will be times when you have a compelling reason.

- Why should you attend?
  - My total commute time to teach in person is approximately 3 hours.
  - The capacity of the classrooms determines the allowed enrollment in W4111. There are disappointed students that cannot enroll because you are in "their chair."

  ➔ Basic decency and respect means you should make the effort to attend.

- If attendance becomes very poor, I will periodically give a pop quiz.

Instructor and TAs

About Your Instructor

- 40 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 (Synopsys) (current):
    - Ansys Fellow, Chief SW Architect;
    - VP/GM, Cloud, AI, Solutions and Developer Enablement BU (CASEBU)
- Academic experience:
  - BA, MS, Ph.D., Computer Science, Columbia University.
  - Approx. 25 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.

I have taught some version of this class 10 times.

Personal:

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

About Your TAs

- All the TAs have either taken databases with me, previously been a TA for databases, or have industry experience. They can relate to your experience in the class and help you be successful.
- I ask them to be your advocates. For example,
  - Based on their feedback last fall,
  - I decreased the length of the final exam by 40%.
- If you are struggling, please contact them and/or me.
- The TAs: I will publish the info.

Course Overview, Objectives and Content

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.

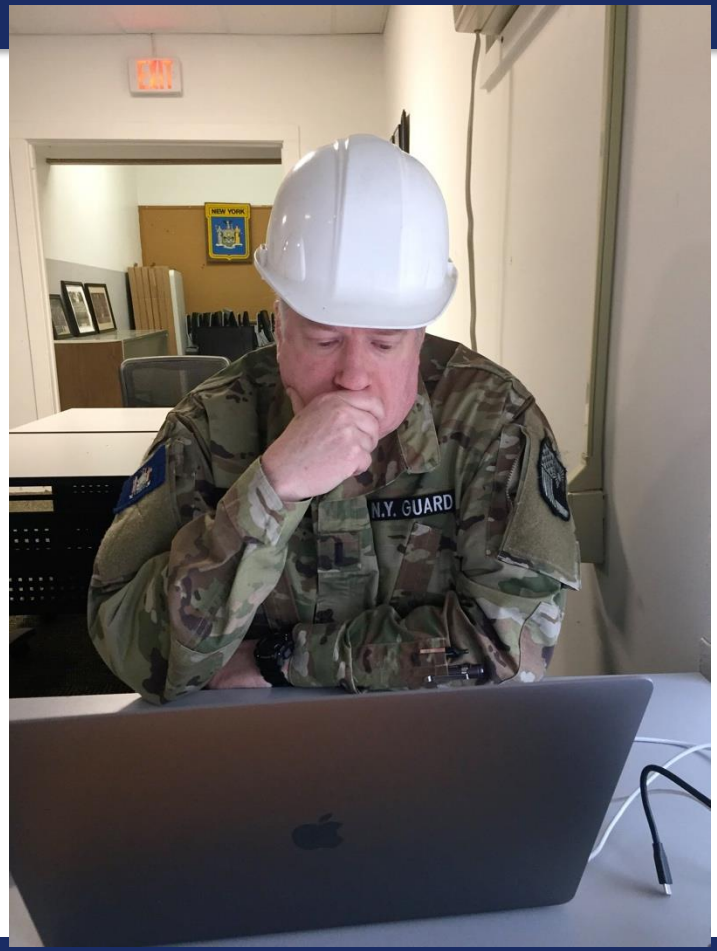
Surprising Example

Example: This is a photo of me 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 course's technology and these skills are surprisingly applicable.

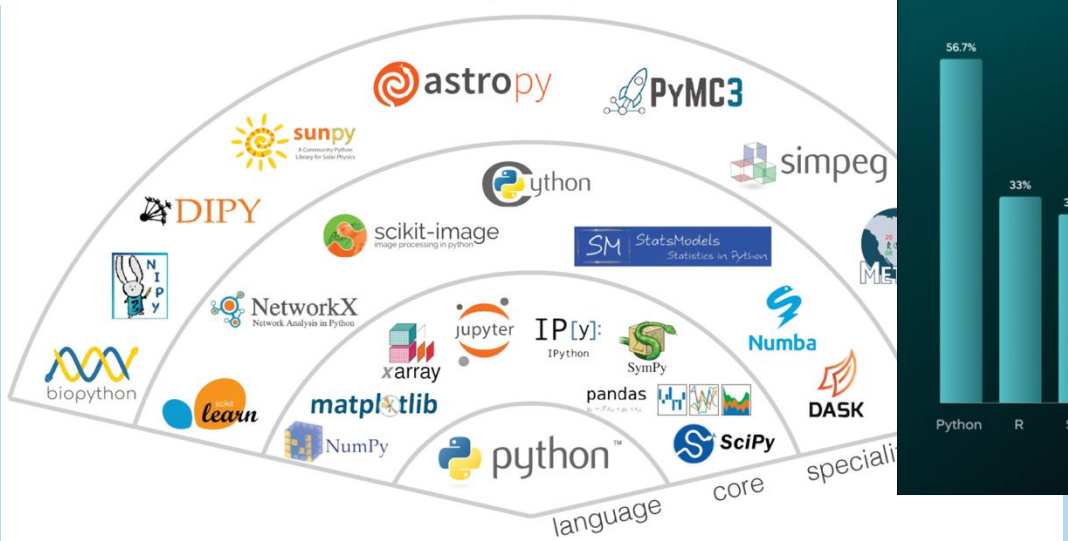


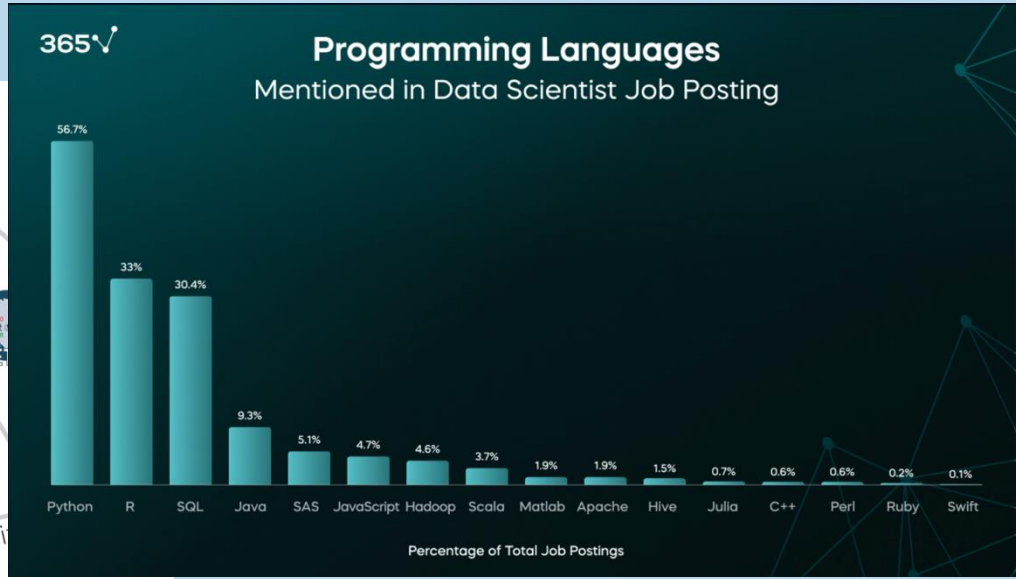
To Program or Not To Program, ...

- From the department's 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 in these jobs 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.
  - 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/data engineering centric project.

Technology Used

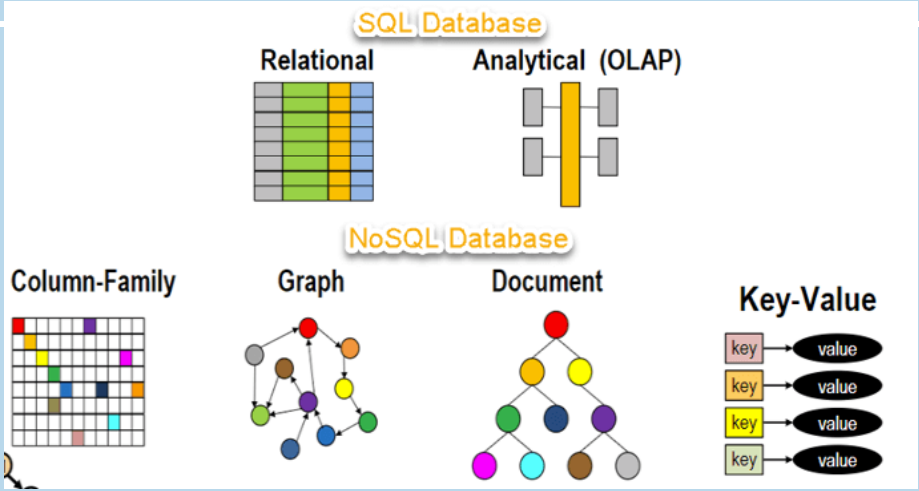
Why Python?

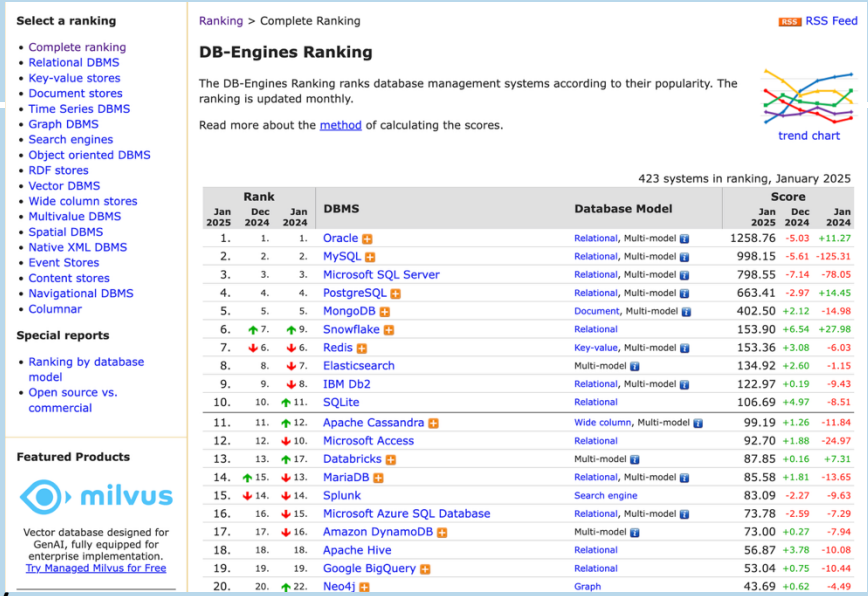




- We use Python and the python ecosystem despite the fact that the bulletin says/used to say Java, and many intro. courses are in Java.
- Most popular language in data engineering/data science, mathematics and DB, data engineering for AI, … …
- Excellent packages, libraries and DB usage examples and tutorials.

Which Databases?





Select a ranking

- Complete ranking
- Relational DBMS
- Key-value stores
- Document stores
- Time Series DBMS
- Graph DBMS
- Search engines
- Object oriented DBMS
- RDF stores
- Vector DBMS
- Wide column stores
- Multivalue DBMS
- Spatial DBMS
- Native XML DBMS
- Event Stores
- Content stores
- Navigational DBMS
- Columnar

Special reports

- Ranking by database model
- Open source vs. commercial

Featured Products

milvus

Vector database designed for GenAI, fully equipped for enterprise implementation.
Try Managed Milvus for Free

Ranking > Complete Ranking

RSS Feed

DB-Engines Ranking

The DB-Engines Ranking ranks database management systems according to their popularity. The ranking is updated monthly.

Read more about the method of calculating the scores.

trend chart

423 systems in ranking, January 2025

Rank Jan 2025	Rank Dec 2024	Rank Jan 2024	DBMS	Database Model	Score Jan 2025	Score Dec 2024	Score Jan 2024
1.	1.	1.	Oracle	Relational, Multi-model	1258.76	-5.03	+11.27
2.	2.	2.	MySQL	Relational, Multi-model	998.15	-5.61	-125.31
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	798.55	-7.14	-78.05
4.	4.	4.	PostgreSQL	Relational, Multi-model	663.41	-2.97	+14.45
5.	5.	5.	MongoDB	Document, Multi-model	402.50	+2.12	-14.98
6.	↑ 7.	↑ 9.	Snowflake	Relational	153.90	+6.54	+27.98
7.	↓ 6.	↓ 6.	Redis	Key-value, Multi-model	153.36	+3.08	-6.03
8.	8.	↓ 7.	Elasticsearch	Multi-model	134.92	+2.60	-1.15
9.	9.	↓ 8.	IBM Db2	Relational, Multi-model	122.97	+0.19	-9.43
10.	10.	↑ 11.	SQLite	Relational	106.69	+4.97	-8.51
11.	11.	↑ 12.	Apache Cassandra	Wide column, Multi-model	99.19	+1.26	-11.84
12.	12.	↓ 10.	Microsoft Access	Relational	92.70	+1.88	-24.97
13.	13.	↑ 17.	Databricks	Multi-model	87.85	+0.16	+7.31
14.	↑ 15.	↓ 13.	MariaDB	Relational, Multi-model	85.58	+1.81	-13.65
15.	↓ 14.	↓ 14.	Splunk	Search engine	83.09	-2.27	-9.63
16.	16.	↓ 15.	Microsoft Azure SQL Database	Relational, Multi-model	73.78	-2.59	-7.29
17.	17.	↓ 16.	Amazon DynamoDB	Multi-model	73.00	+0.27	-7.94
18.	18.	18.	Apache Hive	Relational	56.87	+3.78	-10.08
19.	19.	19.	Google BigQuery	Relational	53.04	+0.75	-10.44
20.	20.	↑ 22.	Neo4j	Graph	43.69	+0.62	-4.49

- We will use 4 specific database management/engineering systems: MySQL, MongoDB, PySpark and Neo4j.
  - My SQL is the most popular “open source, free” relational/SQL DBMS, but PostgreSQL is growing the fastest.
  - MongoDB is the most popular document DBMS.
  - Neo4j is the most popular Graph DBMS.
- There are many technologies for data engineering and big data.
  PySpark is popular, common to many systems, simple to install and use from Python.
- **Note:** Vector databases are growing rapidly because of generative AI/LLMs.

Approach to Lectures and Material

Modules

Each section of W4111 is slightly different based on student interest and professor's focus. There is a common, core syllabus. Professors cover topics in different orders and grouping based on teaching style.

This section of W4111 has four modules:

- **Foundational concepts (50% of semester):** This module covers concepts like data models, relational model, relational databases and applications, schema, normalization, … The module focuses on the relational model and relational databases. The concepts are critical and foundational for all types of databases and data centric applications.
- **Database management system architecture and implementation (10%):** This module covers the software architecture, algorithms and implementation techniques that allow databases management systems to deliver functions. Topics include memory hierarchy, storage systems, caching/buffer pools, indexes, query processing, query optimization, transaction processing, isolation and concurrency control.
- **NoSQL – "Not Only SQL" databases (20%):** This module provides motivation for "NoSQL" data models and databases, and covers examples and use cases. The module also includes cloud databases and databases-as-a-service.
- **Data Enabled Decision Support (20%):** This module covers data warehouses, data import and cleanse, OLAP, Pivot Tables, Star Schema, reporting and visualization, and provides and overview of analysis techniques, e.g. clustering, classification, analysis, mining.

Approach to Lectures; Material

- Material
  - The primary source of information needed for homework, exams and learning is the lecture and supporting slides/presentation, including slides I do not present/skip.
  - There is a recommended textbook, but you can get by with material from the presentations supporting the lectures. (https://www.db-book.com/slides-dir/index.html). I may ask you to read and know material from the book slides. I will identify the material.
- Lectures
  - The format will include a 10-minute break around 11:30.
  - The material I use will be a presentation and a Jupyter Notebook with examples.
  - I am tired of students not coming to lecture, not watching recordings, and then asking questions that I answered or explained in class.
    - I expect you to attend lecture.
    - I may periodically take attendance.
  - There will be a form for pre-explaining why you cannot attend a lecture.
- The standard syllabus presents some material sequentially:
  ER Model → Relational Model → Relational Algebra → SQL
- I incrementally cover all topics in increasing detail in the early lectures

Resources and Environment

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/. The site has:
    - Slides for each chapter.
    - Example data.
  - Textbooks are expensive. You can easily get through the course using website, lecture material, ...
- This is a a GitHub repository, GitHub pages and example project for the course:
  - https://donald-f-ferguson.github.io/W4111-Introduction-to-Databases-New/
  - You should clone this repository,
- There may be individual repositories for some homework assignments.
- HW1 explains how to set up your laptop/PC/Mac for the course.
  - It explains how to install Git, Jupyter Notebook, PyCharm, DataGrip, MySQL Sever Community Edition. It tests the environment. HW0 is submitting the tests to show you set it up.
  - There will be a recorded recitation (This weekend).

An overview of data and database concepts

Basic Concepts and Background

What is a Database

- This course is COMS W4111 – Introduction to Databases.
- Most of us understand what data is.
- But, “What is a database?”

Database Management System (DBMS)

What is a database? In essence a database is nothing more than a collection of information that exists over a long period of time, often many years. In common parlance, the term *database* refers to a collection of data that is managed by a DBMS. The DBMS is expected to:

1. Allow users to create new databases and specify their *schemas* (logical structure of the data), using a specialized *data-definition language*.

Database Systems: The Complete Book (2nd Edition)
by Hector Garcia-Molina (Author), Jeffrey D. Ullman (Author), Jennifer Widom (Author)

Database Management System (DBMS)

2. Give users the ability to *query* the data (a "query" is database lingo for a question about the data) and modify the data, using an appropriate language, often called a *query language* or *data-manipulation language*.

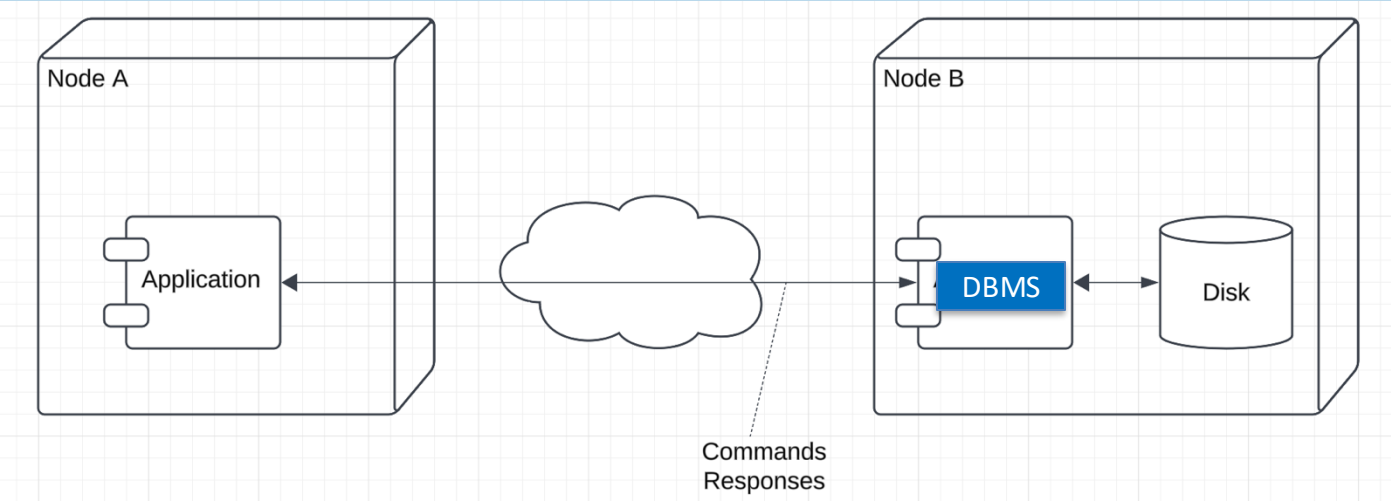
3. Support the storage of very large amounts of data — many terabytes or more — over a long period of time, allowing efficient access to the data for queries and database modifications.

4. Enable *durability*, the recovery of the database in the face of failures, errors of many kinds, or intentional misuse.

5. Control access to data from many users at once, without allowing unexpected interactions among users (called *isolation*) and without actions on the data to be performed partially but not completely (called *atomicity*).

Database Systems: The Complete Book (2nd Edition)
by Hector Garcia-Molina (Author), Jeffrey D. Ullman (Author), Jennifer Widom (Author)

A DBMS is a Special Program



- **Note:** Show: DataGrip, mysqld, /usr/local/mysql, /usr/local/mysql/data
- This is all “local,” i.e. Node A and Node B are the same and my Mac
- We will use DBMS and see examples where the client application and DBMS are on separate nodes.

Outline

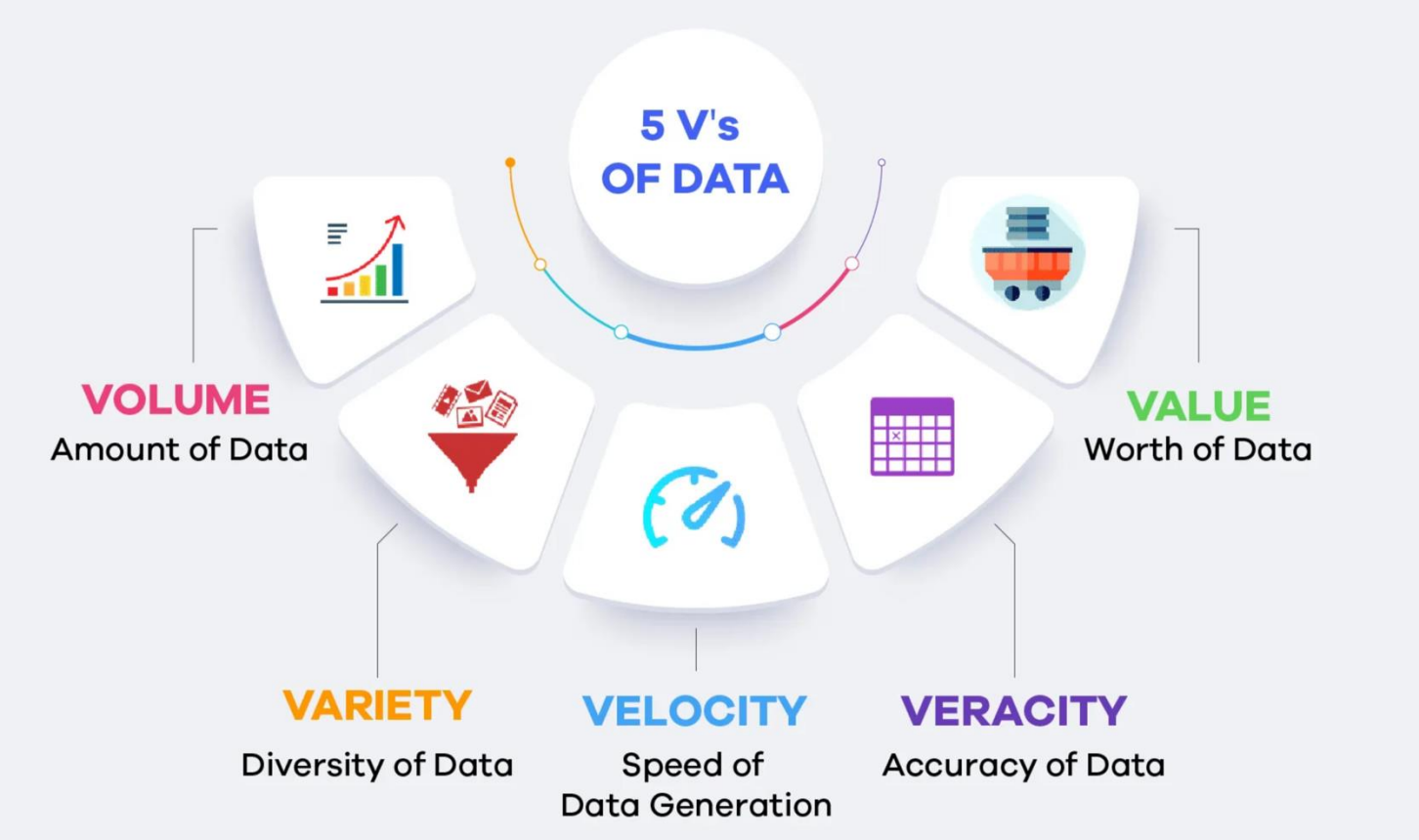
- Database-System Applications
- Purpose of Database Systems
- View of Data
- Database Languages
- Database Design
- Database Engine
- Database Architecture
- Database Users and Administrators
- History of Database Systems

HW 1 requires studying the slides for lecture 1 for the recommended textbook.
https://www.db-book.com/slides-dir/index.html

- When you see a slide formatted this way, it is directly copied from the slides associated with the recommended textbook.
- Typically, my comments and annotations, if I added any, are in read text.
- Cover this basic material, which you can read is not a good use of lecture time.
- I expect you to read the slides and understand the content.
- HW1 will check that you understand the material.
- Future lectures and HW assume you read and understand the material.

Some Observations about Data

5 Vs of Data



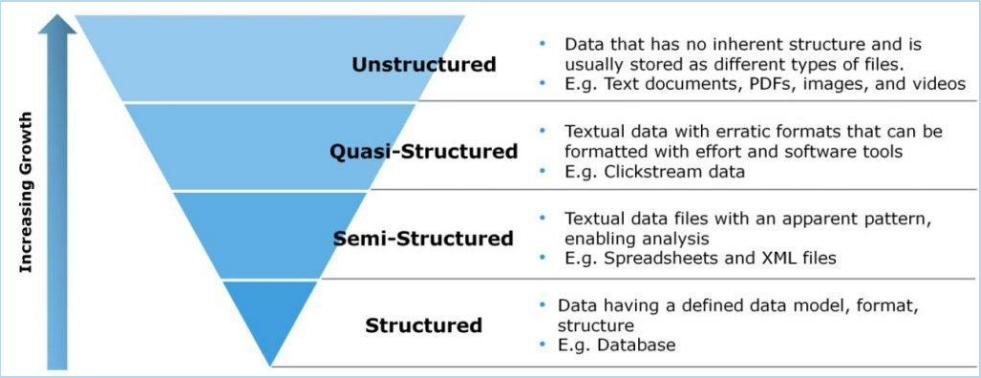
Types of Data

In general, there are three broad-categories/type of data:

- Structured:
  - Every instance of data is from some well-defined “class/category/schema” that defines:
    - Properties, their types, etc.
    - Relationships between data instances, their types and constraints, etc.
  - The canonical models are SQL and the relational model.
- Semi-structured: “You know it when you see it.”
  - There is clearly some structure or pattern to the data,
  - But it is not rigorously applied and constrained.
  - The canonical example is documents, e.g. JSON.
- Unstructured:
  - A data instance is just a bunch of bytes and the internal content is not clear.
  - Examples are photographs, movies, … …

Category of Data

https://www.docsumo.com/blogs/data-extraction/structured-vs-unstructured-vs-semistructured



https://mycloudwiki.com/san/data-and-information-basics/

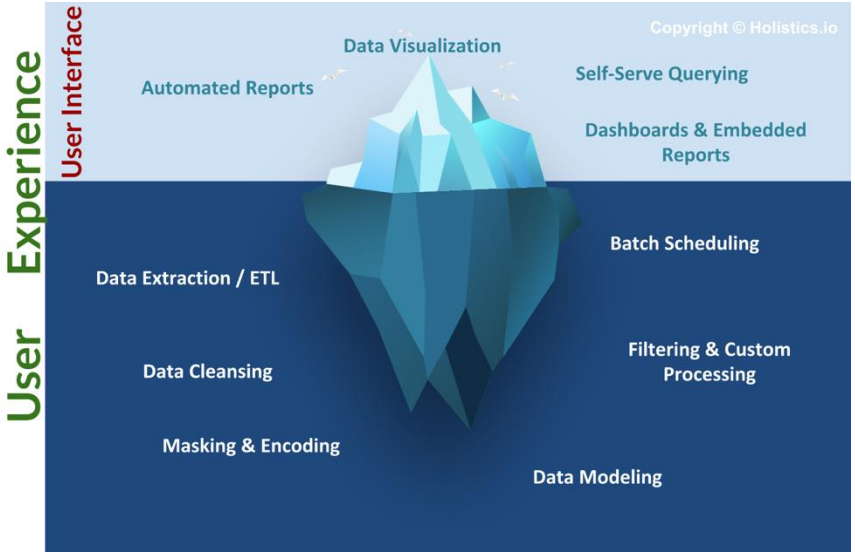
- These are important distinctions, but are not rigorously defined.
- The validity constraints you have on data typically determines the "category," which in turn determines the database product you use.
- All categories additionally have *metadata*, which is usually either structured or semi-structured.

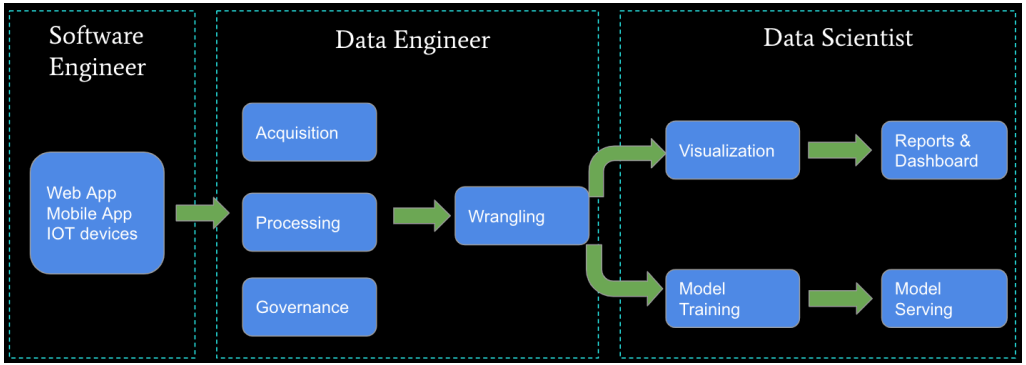
Database Applications

Two Common Database Applications

- Operational/Interactive:
  - Users and roles can create, retrieve, update, search and delete “records.”
  - Examples: SSOL, ATMs, … …
- Business Intelligence, Decision Support, …:
  - Users can perform complex queries and analyze a lot of data to generate a report, make a decision, … …
  - Examples: Build AI/ML training data, dashboards, reports, … …
- Some of our major datasets this semester will be:
  - IMDB: https://developer.imdb.com/non-commercial-datasets/
  - Game of Thrones: https://developer.imdb.com/non-commercial-datasets/
  - Lahman’s Baseball Dataset: http://seanlahman.com/
  - … …
- We will build a simple web application and do some data engineering/science.

Business Intelligence, Insight, Analysis, … …



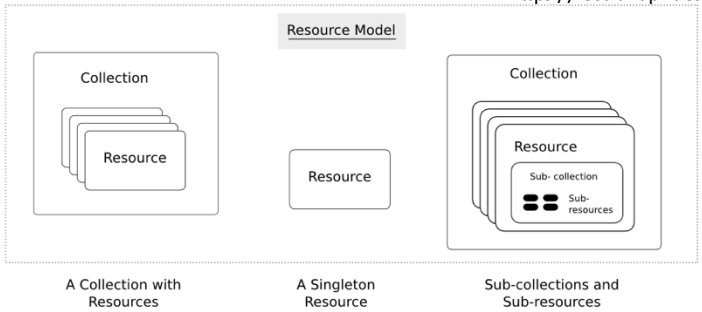


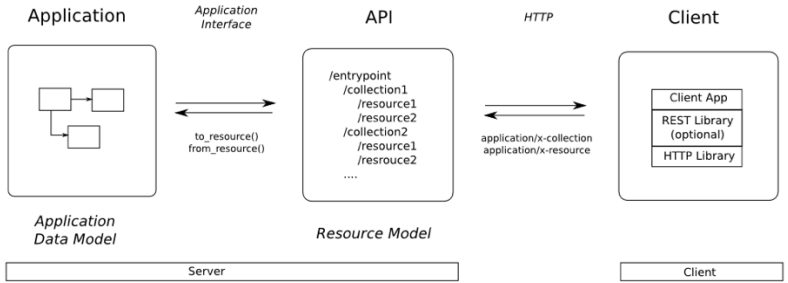
- The “fun” stuff in data science and AI/ML is the “tip of the iceberg.”
- Data engineering is a necessary condition for producing analyzable data. This is often more than 80% of the hard work.
- We will do some small data engineering projects in this course.

Web Application Problem Statement

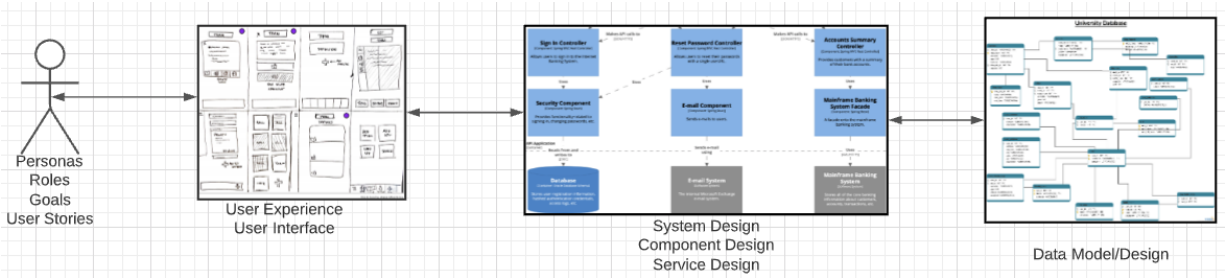
- We must build a system that supports create, retrieve, update and delete for IMDB and Game of Thrones Datasets.
- This requires implementing *create, retrieve, update and delete (CRUD)* for resources.

https://restful-api-design.readthedocs.io/en/latest/resources.html





- We will design, develop, test and deploy the system iteratively and continuously.
- There are four core domains.

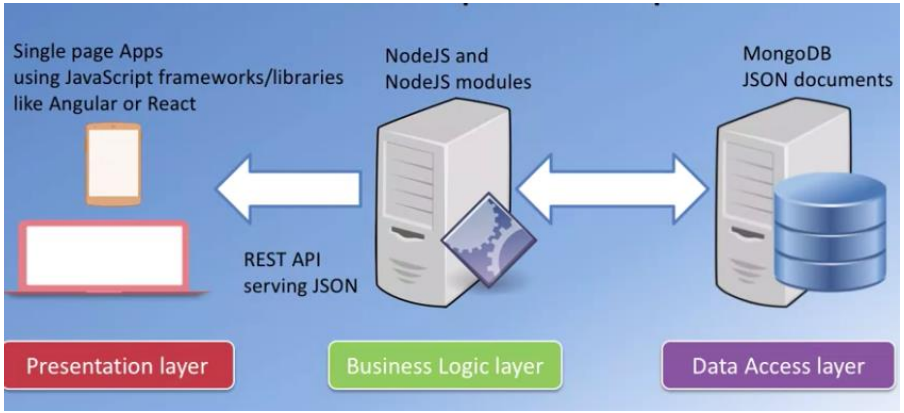


- In this course,
- We focus on the data dimension.
- We will get some insight into the other dimensions.

Interactive/Operational

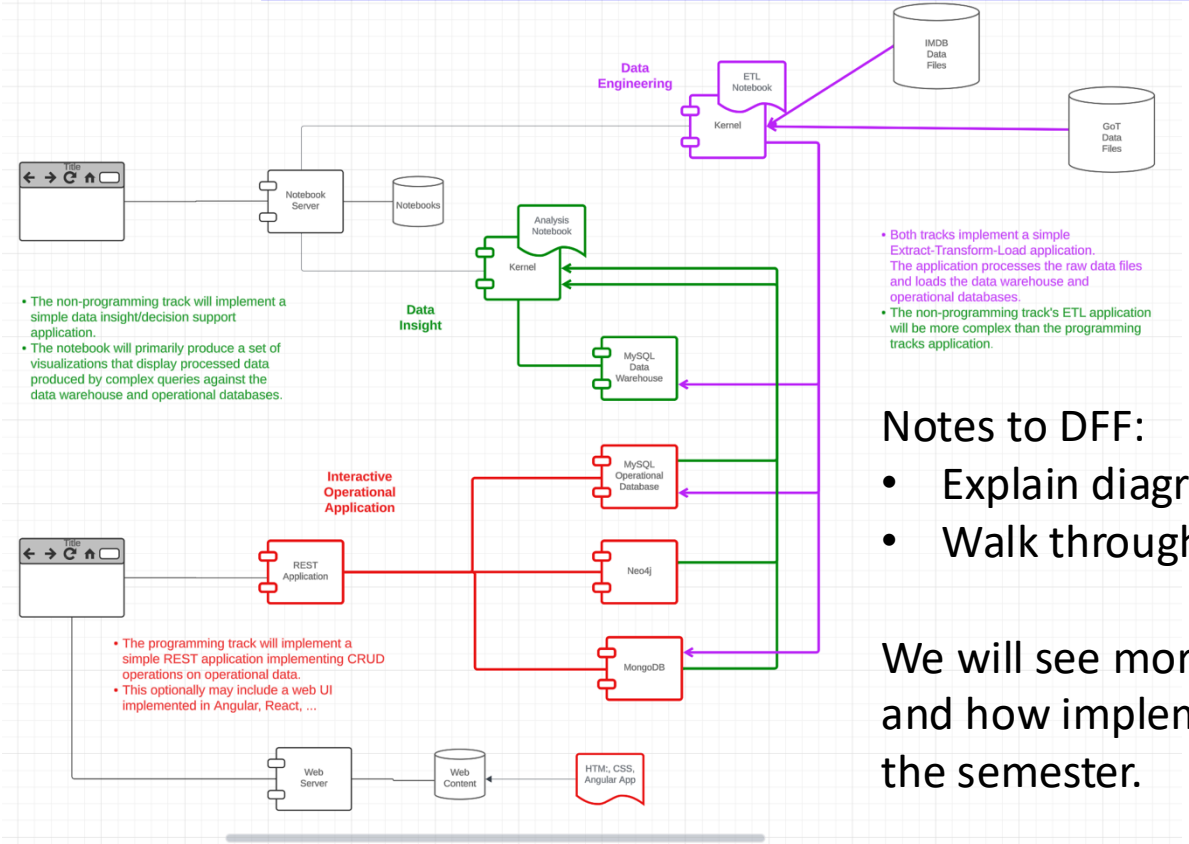
- "A full stack web developer is a person who can develop both client and server software. In addition to mastering HTML and CSS, he/she also knows how to:
  - Program a browser (like using JavaScript, jQuery, Angular, or Vue)
  - Program a server (like using PHP, ASP, Python, or Node)
  - Program a database (like using SQL, SQLite, or MongoDB)"

  https://www.w3schools.com/whatis/whatis_fullstack.asp
- We will do a simple full stack app.
  - Three databases:
    - MySQL
    - MongoDB
    - Neo4j
  - The application tier will be Python and FastAPI.
  - The web UI will be Angular.
  - The primary focus is the data layer and application layer that access it.
  - I will provide a simple UI and template.



Course Project

https://lucid.app/lucidchart/e17c22fd-541e-4d6b-abdb-d5803c342475/edit?viewport_loc=-574%2C-147%2C2411%2C1301%2CBa~Liy5..UW3&invitationId=inv_4ce204b1-0e04-4666-9173-d4e421ab5df9



Notes to DFF:

- Explain diagram.
- Walk through the project template.

We will see more detail on these concepts and how implement them during the semester.

Foundations (1)

ER Model
Database Design Modeling

Data Models

- A collection of tools for describing
  - Data
  - Data relationships
  - Data semantics
  - Data constraints
- Relational model
- Entity-Relationship data model (mainly for database design)
- Object-based data models (Object-oriented and Object-relational)
- Semi-structured data model (XML)
- Other older models:
  - Network model
  - Hierarchical model

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:

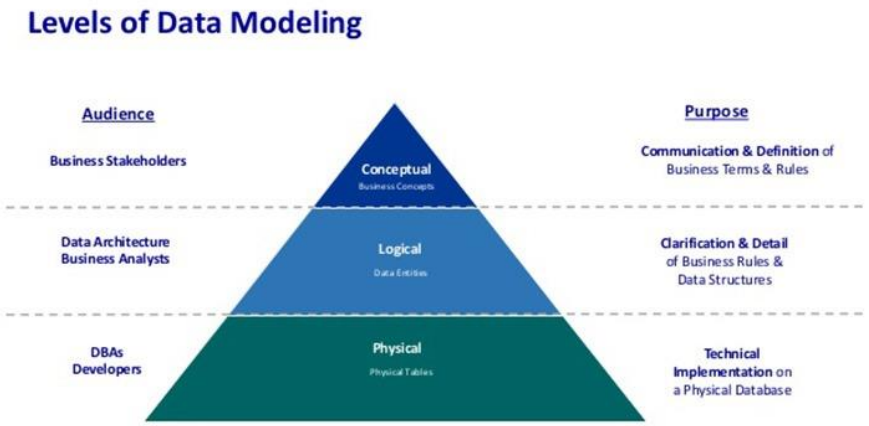
- We you see slides with this formatting, the come directly from the presentations associated with the textbook. (https://www.db-book.com/db7/slides-dir/index.html)
- The number at the bottom is of the form chapter.slide_no.
- I try to put my comments, modifications and annotations in red text, or inside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

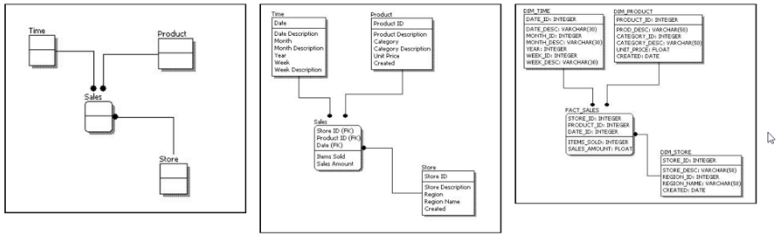
https://ehikioya.com/conceptual-logical-physical-database-modeling/



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



- 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

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

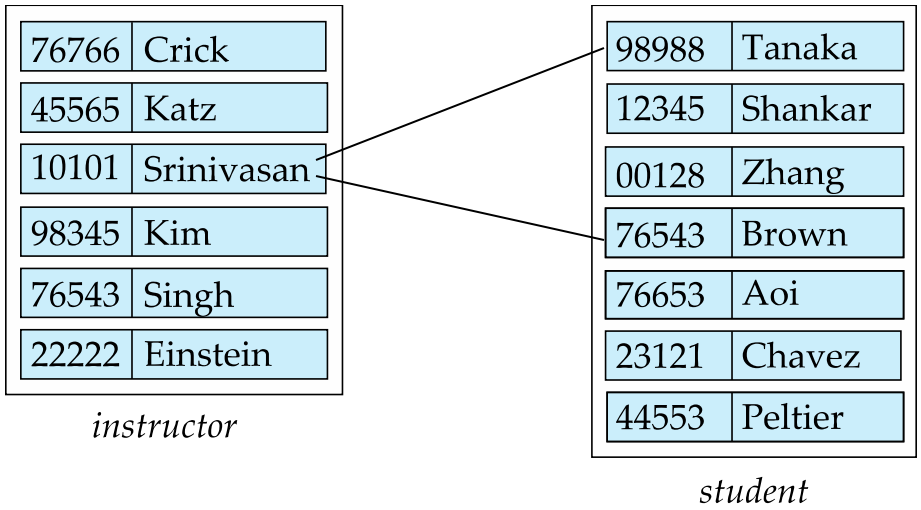
- An **entity** is an object that exists and is distinguishable from other objects.
  - Example: specific person, company, event, plant
- An **entity set** is a set of entities of the same type that share the same properties.
  - Example: set of all persons, companies, trees, holidays
- An entity is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



Relationship Set:

{

(10101, 98988),

(10101,76543)

}

Relationship Sets

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

  Example:

44553 (Peltier)	*advisor*	22222 (Einstein)
student entity	relationship set	*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:

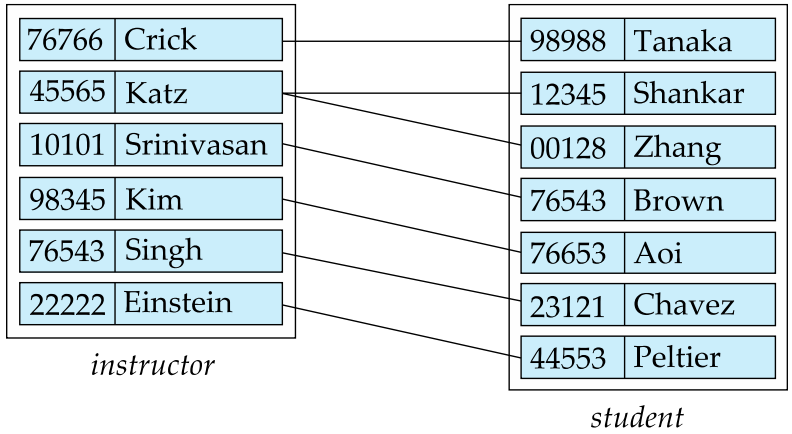
    $(44553,22222) \in advisor$

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 Comment:

- In this diagram, the lines are the relationship set.
- Many DB models use a 3rd entity set to represent complex relationships.

- The relationship set would be:
  (76766, 98988)
  (45565, 12345)
  (45565, 00128)
  (…, …)

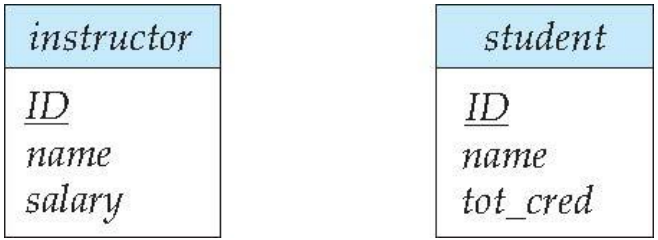
DFF Comments:

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

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

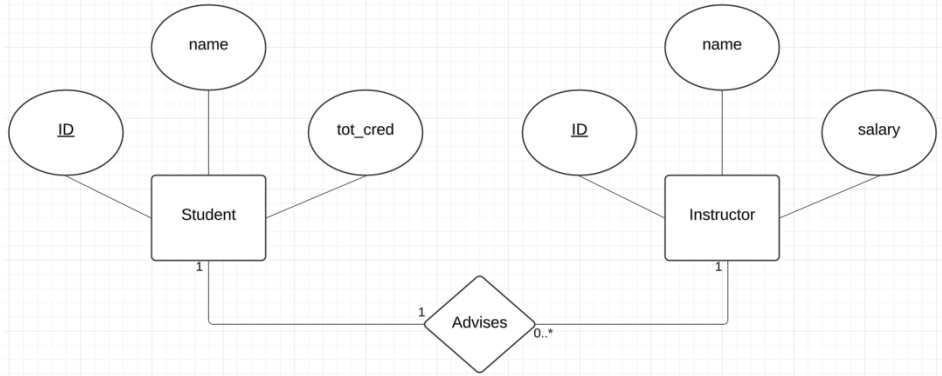
Book Notation



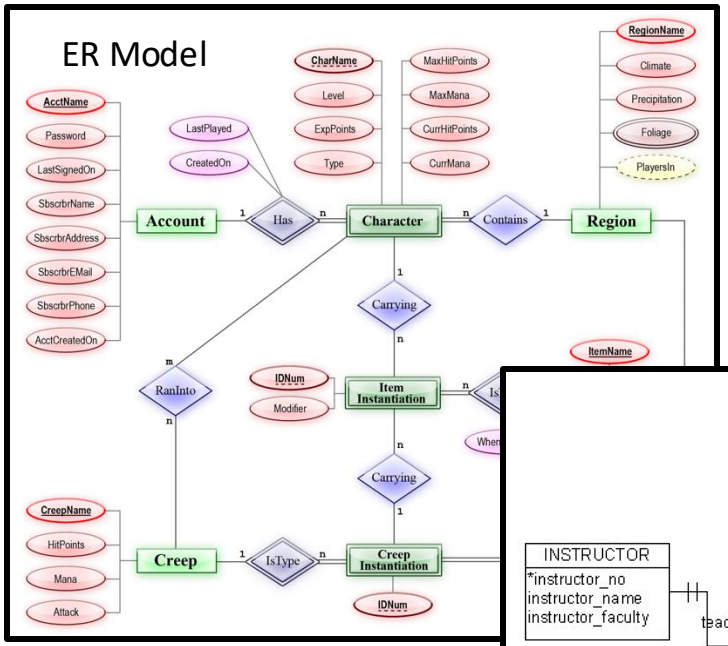
"Visual modeling is the use of semantically rich, graphical and textual design notations to capture software designs. A notation, such as UML, allows the level of abstraction to be raised, while maintaining rigorous syntax and semantics. In this way, it improves communication in the design team, as the design is formed and reviewed, allowing the reader to reason about the design, and it provides an unambiguous basis for implementation."

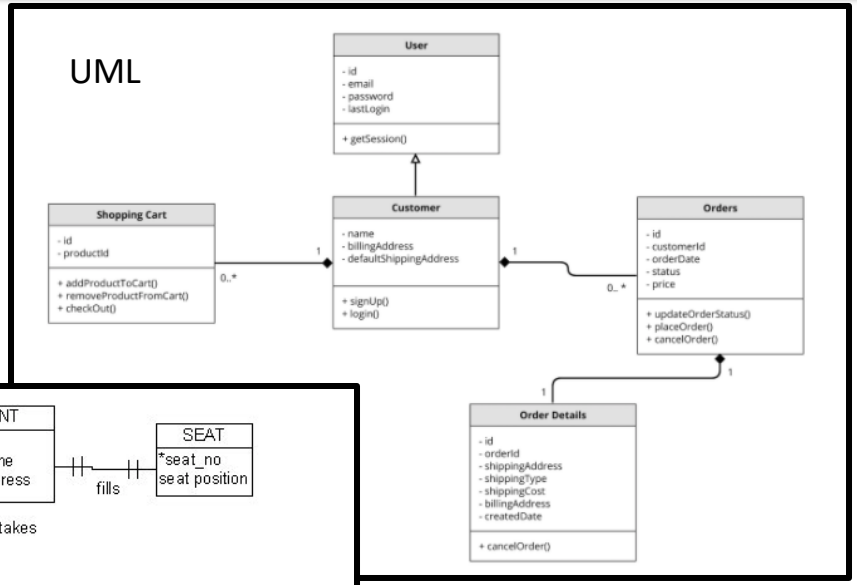
Crow's Foot Notation

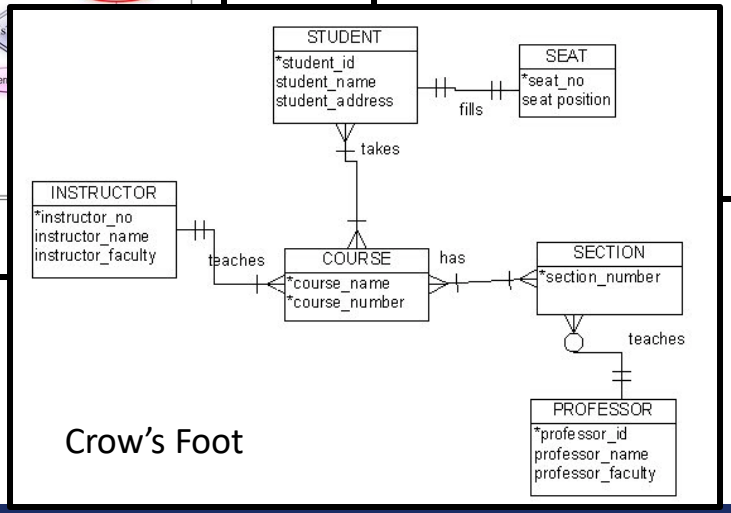
UML ERD Profile



Visual Notation – Many Notations



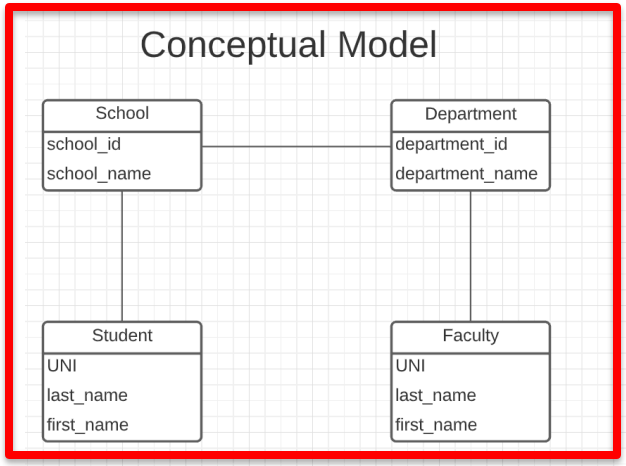


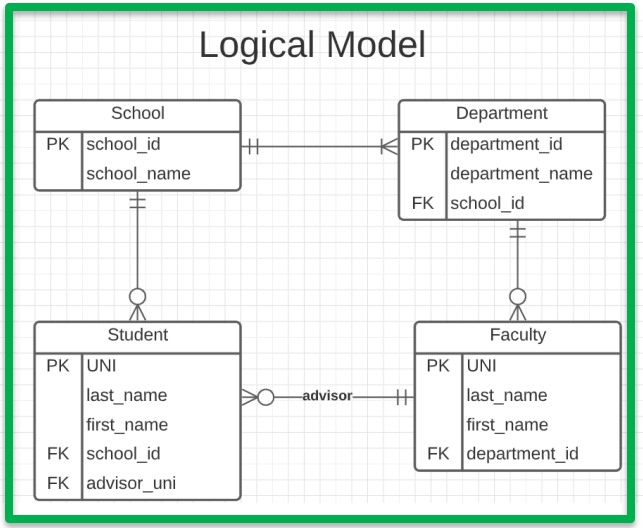


- "Other," i.e. PowerPoint is the most common modeling notation.
- It is easy to get "carried away."
- The trick is to do "just enough modeling."
- I mostly use Crow's Foot
  - It is "just enough"
  - But lacks some capabilities.
- The book uses ER notation.

Do a Slightly More complex University Database ER Model

- The model has:
  - Four entity sets:
    - *School*
    - *Department*
    - *Student*
    - *Faculty*
  - Three relationship sets:
    - *Student-School*
    - *School-Department*
    - *Faculty-Department.*





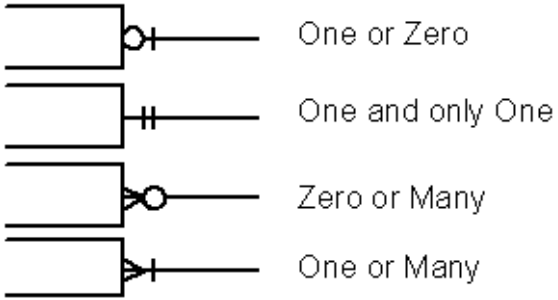
- This is the level of detail I want when I ask for:
  - Conceptual Model diagram.
  - Logical Model diagram.
- Some online tools with "free," constrained usage.
  - Lucidchart (https://www.lucidchart.com/)
  - Vertabelo (https://vertabelo.com/)
  - DrawIO (https://www.drawio.com/)

DFF Note: Demo in Lucidchart

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.

Relational Model
Relational Algebra

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!

Domain, Atomic and NULL

- A *domain* is a data type and rules/constraints on allowed values.
  - *Email* is type *text* and must contain the character "@"
  - *Social Security* number is type text(9) and can only contain '0', '1', … …, '9'
- A domain is *atomic* if it cannot be broken down into a set of more basic domains
  - The domain *name* of the form "Ferguson, Donald, F" is not atomic. It is three domains: *lastname, firstname, middle initial.*
  - The domain *course ID* of the form COMSW4111 is not atomic:
    - COMS is from the domain *department.*
    - W is from the domain *faculty code.*
    - 4111 is from the domain *valid course numbers.*
- *NULL* indicates that a value is not applicable or unknown for the *entity.*
  - ➔ *You cannot use another, ad hoc value like -1 for age or '' for middle* initial.
  - *null == null, lastname != null, null > 5* all evaluate to *NULL,* not TRUE or FALSE.

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

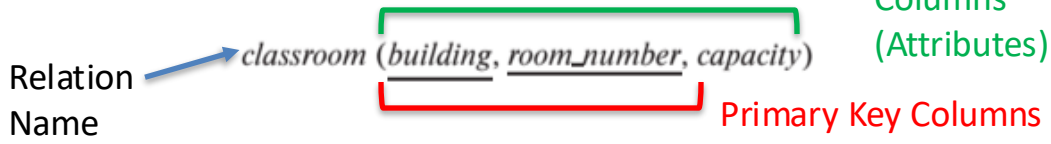
Classroom relation

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.

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.

The Dreaded 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:
  - Has an older version of the data from the recommended textbook. (https://dbis-uibk.github.io/relax/calc/gist/4f7866c17624ca9dfa85ed2482078be8/relax-silberschatz-english.txt/0)
  - You can also upload new data.
- Some queries:
  - σ dept_name='Comp. Sci.' ∨ dept_name='History' (department)
  - π name, dept_name (instructor)
  - π ID, name (
    σ dept_name='Comp. Sci.' (instructor)
)

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.

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

π ID, name (
    σ dept_name='Comp. Sci.' (instructor)
)

=

```
SELECT ID, name FROM instructor
    WHERE
    dept_name='Comp. Sci.'
```

Basic Query Structure

- A typical SQL query has the form:

  select $A_1, A_2, ..., A_n$
  from $r_1, r_2, ..., r_m$
  where P

  - A_i represents an attribute
  - R_i represents a relation
  - P is a predicate.
- The result of an SQL query is a relation.

R1,r2,r3 is an implicit join, which I try to avoid. This basically form the set

R_1 X R_2 X R_3 where X is the cartesian product of the 3 relations, which we will see rarely makes sense.

I make the JOIN explicit.

Note:

- The SELECT ... FROM ... WHERE ... Combines two relational operators, σ and $\prod$.
- Actually, it also combines other operators, e.g. x

The select Clause

- The **select** clause lists the attributes desired in the result of a query
  - corresponds to the projection operation of the relational algebra
- Example: find the names of all instructors:

  select *name*
  from *instructor*
- NOTE: SQL names are case insensitive (i.e., you may use upper- or lower-case letters.)
  - E.g., *Name* ≡ *NAME* ≡ *name*
  - Some people use upper case wherever we use bold font.

The select Clause (Cont.)

- SQL allows duplicates in relations as well as in query results.
- To force the elimination of duplicates, insert the keyword **distinct** after select.
- Find the department names of all instructors, and remove duplicates

```
select distinct dept_name
from instructor
```

- The keyword **all** specifies that duplicates should not be removed.

```
select all dept_name
from instructor
```

The select Clause (Cont.)

- An asterisk in the select clause denotes “all attributes”

  select *
  from *instructor*

- An attribute can be a literal with no **from** clause

  select '437'

  - Results is a table with one column and a single row with value “437”
  - Can give the column a name using:

    select '437' **as** *FOO*

- An attribute can be a literal with **from** clause

  select 'A'
  from *instructor*

  - Result is a table with one column and *N* rows (number of tuples in the *instructors* table), each row with value “A”

The select Clause (Cont.)

- The **select** clause can contain arithmetic expressions involving the operation, +, –, ∗, and /, and operating on constants or attributes of tuples.
  - The query:

    select *ID, name, salary/12*
    from *instructor*

    would return a relation that is the same as the *instructor* relation, except that the value of the attribute *salary* is divided by 12.
  - Can rename "s*alary/12"* using the **as** clause:

    select *ID, name, salary/12* **as** *monthly_salary*

The where Clause

- The **where** clause specifies conditions that the result must satisfy
  - Corresponds to the selection predicate of the relational algebra.
- To find all instructors in Comp. Sci. dept

  select *name*
  from *instructor*
  where *dept_name* = 'Comp. Sci.'
- SQL allows the use of the logical connectives **and, or,** and **not**
- The operands of the logical connectives can be expressions involving the comparison operators <, <=, >, >=, =, and <>.
- Comparisons can be applied to results of arithmetic expressions
- To find all instructors in Comp. Sci. dept with salary > 80000

  select *name*
  from *instructor*
  where *dept_name* = 'Comp. Sci.' **and** *salary* > 80000

The from Clause

- The **from** clause lists the relations involved in the query
  - Corresponds to the Cartesian product operation of the relational algebra.
- Find the Cartesian product *instructor X teaches*

  select ∗
  from *instructor, teaches*

  - generates every possible instructor – teaches pair, with all attributes from both relations.
  - For common attributes (e.g., *ID*), the attributes in the resulting table are renamed using the relation name (e.g., *instructor.ID*)
- Cartesian product not very useful directly, but useful combined with where-clause condition (selection operation in relational algebra).

Examples

- Find the names of all instructors who have taught some course and the course_id
  - **select** *name, course_id*
    from *instructor , teaches*
    where *instructor.ID = teaches.ID*

- Find the names of all instructors in the Art department who have taught some course and the course_id
  - **select** *name, course_id*
    from *instructor , teaches*
    where *instructor.ID = teaches.ID* ***and*** *instructor. dept_name* = 'Art'

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))

Homework 1

Walk Through the Definition

- Ed will contain the details of completion.

- HW 1 has three parts:
  - Demonstrating that you have successfully set up your environment.
  - Written questions demonstrating your knowledge of the book's lecture 1 slides' material.
  - Some simple, "practical" relational algebra and SQL questions.

- HW1 is due on 14-September-2025 at 11:59 PM

- Walkthrough of the homework.