

CS145: Intro to Databases

Lecture 1: Course Overview

The world is increasingly
driven by data...

This class teaches the basics of
how to use & manage data.

Key Questions We Will Answer

- How can we **collect and store** large amounts of data?
 - By building tools and data structures to efficiently index and serve data
- How can we **efficiently query** data?
 - By compiling high-level declarative queries into efficient low-level plans
- How can we **safely update** data?
 - By managing concurrent access to state as it is read and written
- How do different database systems manage **design trade-offs**?
 - e.g., at scale, in a distributed environment?

When you'll use this material

- Building almost any software application
 - e.g., mobile, cloud, consumer, enterprise, analytics, machine learning
 - Corollary: every application you use uses a database
 - Bonus: every program consumes data (even if only the program text!)
- Performing data analytics
 - Business intelligence, data science, predictive modeling
 - (Even if you're using Pandas, you're using relational algebra!)
- Building data-intensive tools and applications
 - Many core concepts power deep learning frameworks to self-driving cars

Today's Lecture

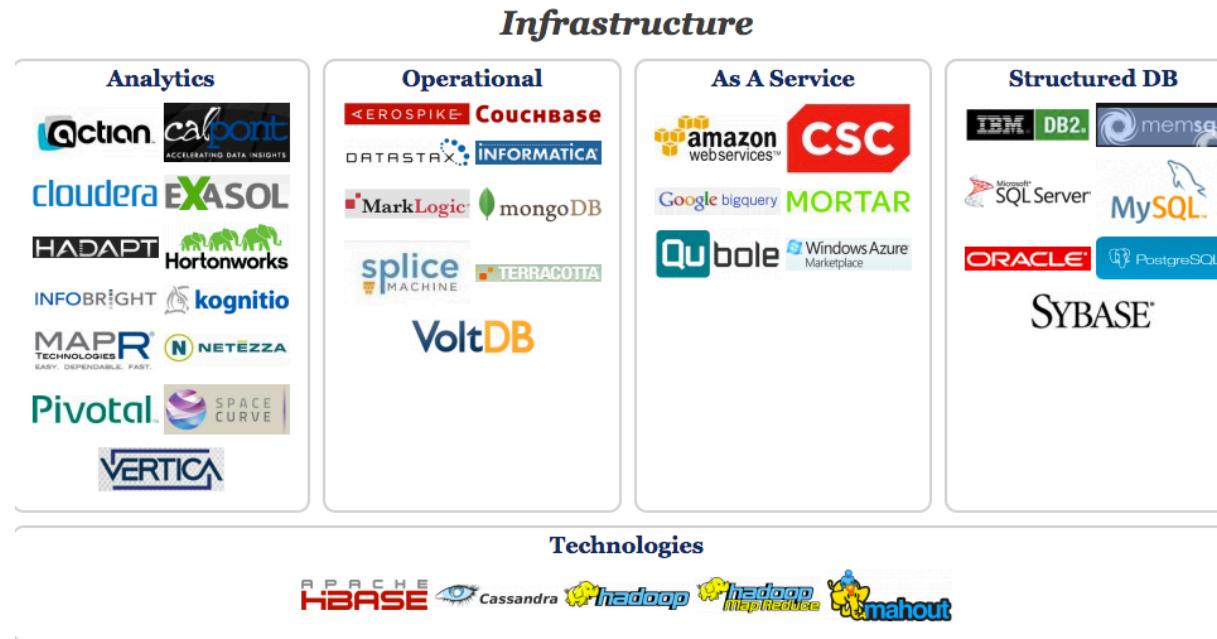
1. Introduction, admin & setup
 - ACTIVITY: Jupyter “Hello World!”
2. Overview of the relational data model
 - ACTIVITY: SQL in Jupyter
3. Overview of DBMS topics: Key concepts & challenges

1. Introduction, admin & setup

What you will learn about in this section

1. Motivation for studying DBs
2. Administrative structure
3. Course logistics
4. Overview of lecture coverage
5. ACTIVITY: Jupyter “Hello World!”

Big Data Landscape... Infrastructure is Changing



New tech. Same Principles.

Why should you study databases?

- **Mercenary- make more \$\$\$:**

- Startups need DB talent right away = low employee #
- Massive industry...



- **Intellectual:**

- Science: data poor to data rich
 - No idea how to handle the data!
- Fundamental ideas to/from all of CS:
 - Systems, theory, AI, logic, stats, analysis....

Many great computer systems ideas started in DB.

What this course is (and is not)

- Discuss **fundamentals of data management**
 - How to design databases, query databases, build applications with them.
 - How to debug them when they go wrong!
 - Not how to be a DBA or how to tune Oracle 12g.
- We'll cover **how database management systems work**
- And some (but not all of) **the principles of how to build** them
 - see 245, 345, and 346.

Who we are...

Instructor (me) Seongjin Lee

- Office hours: Tuesday: 18:00-19:00, 407-314
- Or make an appointment
- Or send an email: insight at gnu dot ac dot kr

Database 2018-02

Class Information

	Class Info
Class	ETA00137 - 데이터베이스
Lecturer	Seongjin Lee
Time and Place	407-507 Tuesday 16:00-18:00 407-507 Thursday 16:00-17:00
Office Hour	Tuesday: 18:00-19:00
Contacts	Office: 407-314
	Email: insight at gnu dot ac dot kr

Course Introduction

Communication w/ Course Staff

- Piazza
- Office hours
- *By appointment!*

Piazza

Discussion

Please provide your info on this [link](#) .

All discussions and assignments are to be submitted in [Piazza](#) . Enroll into the class through the following [link](#) 

<https://piazza.com/class/jkt5x58qttc3u9>

The goal is to get you to answer each other's questions so you can benefit and learn from each other.

CS145 Anonymous Feedback

I value your feedback! Please use this form to provide any constructive anonymous comments, suggestions, and feedback. I am the only staff member who will read this form. Thanks! -- Peter

Please enter your feedback below.

SUBMIT

This content is neither created nor

Feedback

Please provide feedback to Peter and/or any of the course staff [via Piazza](#), or [anonymously via this form](#), which only Peter can access.

Course Website:

open.gnu.ac.kr

Lectures

- Lecture slides cover **essential material**
 - This is your best reference.
 - We are trying to get away from book, but do have pointers
- Try to cover same thing in **many ways**: Lecture, lecture notes, homework, exams (no shock)
 - Attendance makes your life easier...

Attendance

- I dislike mandatory attendance... but in the past we noticed...
 - People who did not attend did worse ☹
 - People who did not attend used more course resources ☹
 - People who did not attend were less happy with the course ☹

Graded Elements

- Attendance (10%)
- Quiz (10%)
- Problem Sets (10%)
- Programming project (10%)
- Midterm (30%)
- Final exam (30%)

Assignments are typically due Tuesday before class, typically 2 weeks to complete

Un-Graded Elements

- Readings provided to help you!
 - Only items in lecture, homework, or project are fair game.
- Activities are again mainly to help / be fun!
 - Will occur during class- not graded, but count as part of lecture material (fair game as well)
- Jupyter Notebooks provided
 - These are optional but hopefully helpful.
 - Redesigned so that you can ‘interactively replay’ parts of lecture

What is expected from you

- **Attend lectures**
 - If you don't, it's at your own peril
- **Be active and think critically**
 - Ask questions, post comments on forums
- **Do programming and homework projects**
 - Start early and be honest
- **Study for tests and exams**

Lectures: 1st half - from a user's perspective

1. Foundations: Relational data models & SQL

- Lectures 2-3
- How to manipulate data with SQL, a declarative language
 - *reduced expressive power but the system can do more for you*

2. Database Design: Design theory and constraints

- Lectures 4-6
- Designing relational schema to keep your data from getting corrupted

3. Transactions: Syntax & supporting systems

- Lectures 7-8
- A programmer's abstraction for data consistency

Lectures: 2nd half - understanding how it works

4. Introduction to database systems

- Lectures 12-16
- Indexing
- External Memory Algorithms (IO model) for sorting, joins, etc.
- Basics of query optimization (Cost Estimates)
- Relational algebra

5. Specialized and New Data Processing Systems

- Lectures 17-19
- Key-Value Stores
- Hadoop and its 10 year anniversary
- SparkSQL. The re-rise of SQL
- Next-gen analytics systems & current intersections with ML & AI

Lectures: A note about format of notes

Take note!!

*These are asides / notes (still
need to know these in general!)*

Definitions in blue with concept being defined bold & underlined

Main point of slide / key takeaway at bottom

Warnings- pay attention here!

Jupyter Notebook “Hello World”

- Jupyter notebooks are interactive shells which **save output in a nice notebook format**
 - They also can display markdown, LaTeX, HTML, js...

FYI: “Jupyter Notebook” are also called iPython notebooks but they handle other languages too.



- You'll use these for
 - in-class activities
 - interactive lecture supplements/recaps
 - homeworks, projects, etc.- if helpful!

Note: you do need to know or learn python for this course!

Jupyter Notebook Setup

1. **HIGHLY RECOMMENDED.** Install on your laptop via the instructions on the next slide / Piazza
2. Other options running via one of the alternative methods:
 1. Ubuntu VM.
 2. Corn
3. Come to our Installation Office Hours after this class and tomorrow!

Please help out your peers by posting issues / solutions on Piazza!

As a general policy in upper-level CS courses, Windows is not officially supported. However we are making a best-effort attempt to provide some solutions here!

Jupyter Notebook Setup

http://open.gnu.ac.kr/mediawiki/index.php?title=Database_2018-02

Ask help for setup & installation

[**DB-WS01a.ipynb**](#)

2. Overview of the relational data model

What you will learn about in this section

1. Definition of DBMS
2. Data models & the relational data model
3. Schemas & data independence
4. ACTIVITY: Jupyter + SQL

What is a DBMS?

- A large, integrated collection of data
- Models a real-world enterprise
 - *Entities* (e.g., Students, Courses)
 - *Relationships* (e.g., Alice is enrolled in 145)

A Database Management System (DBMS) is a piece of software designed to store and manage databases

A Motivating, Running Example

- Consider building a course management system (**CMS**):

- Students
- Courses
- Professors



Entities

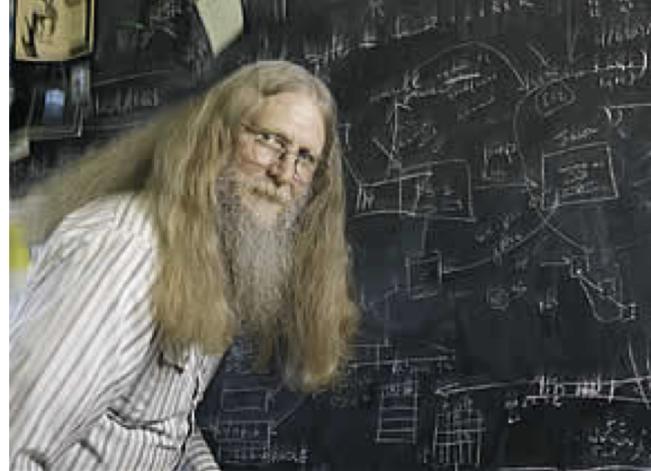
- Who takes what
- Who teaches what



Relationships

Data models

- A **data model** is a collection of concepts for describing data
 - The relational model of data is the most widely used model today
 - Main Concept: the *relation*- essentially, a table
- A **schema** is a description of a particular collection of data, **using the given data model**
 - E.g. every *relation* in a relational data model has a *schema* describing types, etc.



“Relational databases form the bedrock of western civilization”

- Bruce Lindsay, IBM Research
expert in designing database management systems

Modeling the CMS

- *Logical Schema*
 - Students(sid: string, name: string, gpa: float)
 - Courses(cid: string, cname: string, credits: int)
 - Enrolled(sid: string, cid: string, grade: string)

sid	Name	Gpa
101	Bob	3.2
123	Mary	3.8

Relations

cid	cname	credits
564	564-2	4
308	417	2

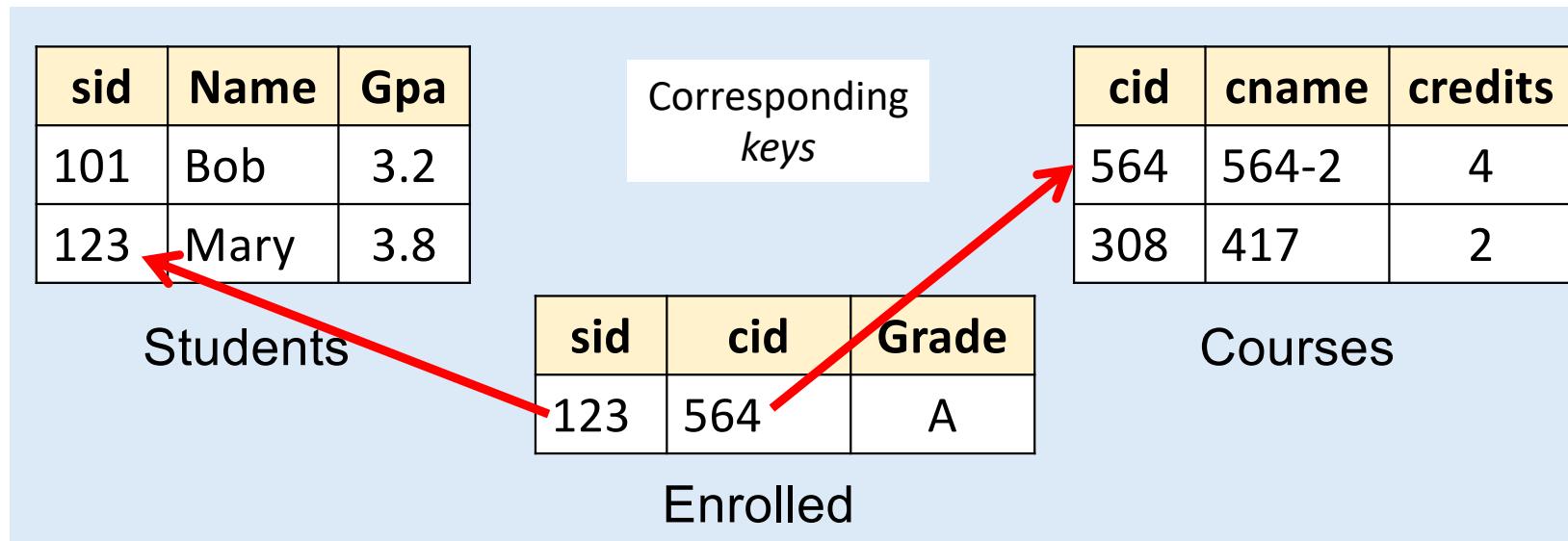
Students Courses

sid	cid	Grade
123	564	A

Enrolled

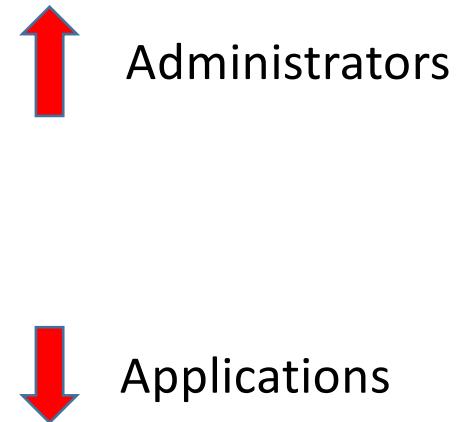
Modeling the CMS

- *Logical Schema*
 - Students(sid: string, name: string, gpa: float)
 - Courses(cid: string, cname: string, credits: int)
 - Enrolled(sid: string, cid: string, grade: string)



Other Schemata...

- *Physical Schema*: describes data layout
 - Relations as unordered files
 - Some data in sorted order (index)
- *Logical Schema*: Previous slide
- *External Schema*: (Views)
 - *Course_info(cid: string, enrollment: integer)*
 - Derived from other tables



Data independence

Concept: Applications do not need to worry about *how the data is structured and stored*

Logical data independence:
protection from changes in the
logical structure of the data

I.e. should not need to ask: can we add a new entity or attribute without rewriting the application?

Physical data independence:
protection from *physical layout changes*

I.e. should not need to ask: which disks are the data stored on? Is the data indexed?

One of the most important reasons to use a DBMS

[**DB-WS01b.ipynb**](#)

3. Overview of DBMS topics

Key concepts & challenges

What you will learn about in this section

1. Transactions
2. Concurrency & locking
3. Atomicity & logging
4. Summary

Challenges with Many Users

- Suppose that our CMS application serves 1000's of users or more - what are some **challenges**?
 - Security: Different users, different roles

We won't look at too much in this course, but is extremely important
 - Performance: Need to provide concurrent access

Disk/SSD access is slow, DBMS hide the latency by doing more CPU work concurrently
 - Consistency: Concurrency can lead to update problems

DBMS allows user to write programs as if they were the **only** user

Transactions

- A key concept is the **transaction (TXN)**: an **atomic** sequence of db actions (reads/writes)

Atomicity: An action either completes *entirely or not at all*

Acct	Balance
a10	20,000
a20	15,000

Transfer \$3k from a10 to a20:

1. Debit \$3k from a10
2. Credit \$3k to a20

Acct	Balance
a10	17,000
a20	18,000

Written naively, in which states is **atomicity** preserved?

- Crash before 1,
- After 1 but before 2,
- After 2.

DB Always preserves atomicity!

Transactions

- A key concept is the **transaction (TXN)**: an **atomic** sequence of db actions (reads/writes)
 - If a user cancels a TXN, it should be as if nothing happened!
- Transactions leave the DB in a **consistent** state
 - Users may write integrity constraints, e.g., ‘each course is assigned to exactly one room’

Atomicity: An action either completes *entirely or not at all*

Consistency: An action results in a state which conforms to all integrity constraints

However, note that the DBMS does not understand the *real* meaning of the constraints— consistency burden is still on the user!

Challenge: Scheduling Concurrent Transactions

- The DBMS ensures that the execution of $\{T_1, \dots, T_n\}$ is equivalent to some **serial** execution
- One way to accomplish this: **Locking**
 - Before reading or writing, transaction requires a lock from DBMS, holds until the end
- **Key Idea:** If T_i wants to write to an item x and T_j wants to read x , then T_i, T_j **conflict**. Solution via locking:
 - only one winner gets the lock
 - loser is blocked (waits) until winner finishes

A set of TXNs is **isolated** if their effect is as if all were executed serially

What if T_i and T_j need X and Y, and T_i asks for X before T_j , and T_j asks for Y before T_i ?
-> *Deadlock!* One is aborted...

All concurrency issues handled by the DBMS...

Ensuring Atomicity & Durability

- DBMS ensures **atomicity** even if a TXN crashes!
- One way to accomplish this: **Write-ahead logging (WAL)**
- **Key Idea:** Keep a log of all the writes done.
 - After a crash, the partially executed TXNs are undone using the log

Write-ahead Logging (WAL): Before any action is finalized, a corresponding log entry is forced to disk

We assume that the log is on “stable” storage

All atomicity issues also handled by the DBMS...

A Well-Designed DBMS makes many people happy!

- End users and DBMS vendors
 - Reduces cost and makes money
- DB application programmers
 - Can handle more users, faster, for cheaper, and with better reliability / security guarantees!
- Database administrators (DBA)
 - Easier time of designing logical/physical schema, handling security/authorization, tuning, crash recovery, and more...

*Must still understand
DB internals*

Summary of DBMS

- DBMS are used to maintain, query, and manage large datasets.
 - Provide concurrency, recovery from crashes, quick application development, integrity, and security
- Key abstractions give **data independence**
- DBMS R&D is one of the broadest, most exciting fields in CS. **Fact!**