“Data is the Future”

- My cab driver in Pittsburg
The world is increasingly driven by data...

This class teaches **the basics** of how to use & manage data.
Today’s Lecture

1. Introduction, admin & setup
   • ACTIVITY: IPython “Hello World!”

2. Overview of the relational data model
   • ACTIVITY: SQL in IPython

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: IPython “Hello World!”
Big Data Landscape...
Infrastructure is Changing

New tech. Same Principles.
Lectures: 1\textsuperscript{st} 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 4-5, 7
   - Designing relational schema to keep your data from getting corrupted

3. **Transactions**: Syntax & supporting systems
   - Lectures 8-9
   - A programmer’s abstraction for data consistency
Lectures: 2\textsuperscript{nd} 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
   • “Dark data” systems & current intersections with ML & AI
Lectures: A note about format of notes

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!
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: IPython + 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

  \[ \text{Entities} \]

  • Who takes what
  • Who teaches what

  \[ \text{Relationships} \]
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
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)
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

**Physical data independence:**
protection from *physical layout* changes

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

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
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

  • **Performance:** Need to provide concurrent access

  • **Consistency:** Concurrency can lead to update problems

  > We won’t look at too much in this course, but is extremely important

  > Disk/SSD access is slow, DBMS hide the latency by doing more CPU work concurrently

  > 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)
  • 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!
Scheduling Concurrent Transactions

• The DBMS ensures that the execution of \{T_1, ..., 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!