CMSC 724: Database Management Systems
Introduction/Background

Instructor: Amol Deshpande
amol@cs.umd.edu

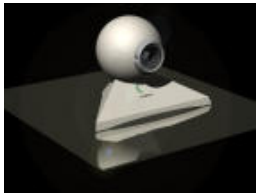
Today

- Motivation: Why study databases ?
- Background: 424 Summary
- Administrivia
  - Workload etc.
- No laptop use allowed in the class !!

Motivation: Data Explosion

- There is a *HUGE* amount of data in this world
- Everywhere you see…
- Personal
  - Emails, data on your computer
- Enterprise
  - The original primary motivation
  - Banks, supermarkets, universities, airlines, phone call data etc.
- Scientific
  - Biological, astronomical
- World wide web
  - Social networks etc…

Motivation: Data Explosion



Much more is produced every day

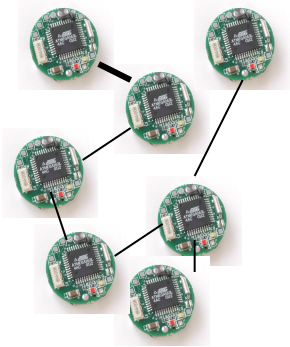
Wal-mart: 583 terabytes of sales and inventory data

Adds a billion rows every day

"we know how many 2.4 ounces of tubes of toothpastes sold yesterday and what was sold with them"

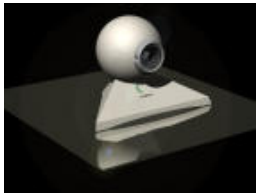
"Wellcome Trust Sanger Institute's World Trace Archive database of DNA sequences hit one billion entries.." [[Since defunct]]

Stores all sequence data produced the world scientific community 22 Tbytes and doubling every 10 months



A single astrophysics simulation of galaxy formation can generate several PB of data, most of it thrown away

Motivation: Data Explosion

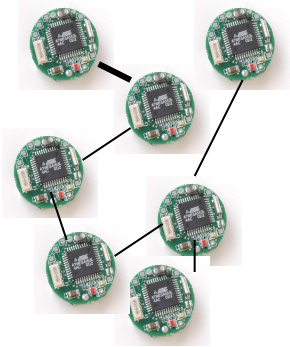


Machine-generated data

Sensor devices/networks, microphones/camera, Web server logs, Network Monitoring, etc.

Online services (2 year old data)

- Twitter: 177M tweets sent on 3/1/2011 (nothing special about the date), 572,000 accounts added on 3/12/2011
- Dropbox: 1M files saved every 15 mins
- Facebook: 135+ Billion Messages a Month
- Reddit: 270 Million Page Views a Month in May 2010



Much of this data not stored in traditional RDBMS

Motivation: Data Explosion

A major challenge to manage this data, answer queries over it, glean interesting and useful insights from it

"Big Data"

- Everyone is either doing big data, or wants to…
- No one seems to agree on what it really means ☺
  - Not just about scale/volume of data
  - Many of the datasets are not that large

"Data Scientist"

- Goal: Extract meaning from data and creating data products
- Need a broad range of skills: programming, statistics, math, …

The FOUR V's of Big Data

From traffic patterns and music downloads to web history and medical records, data is recorded, stored, and analyzed to enable the technology and services that the world relies on every day. But what exactly is big data, and how can these massive amounts of data be used?

As a leader in the sector, IBM data scientists break big data into four dimensions: **Volume, Velocity, Variety and Veracity**

Depending on the industry and organization, big data encompasses information from multiple internal and external sources such as transactions, social media, enterprise content, sensors and mobile devices. Companies can leverage data to adapt their products and services to better meet customer needs, optimize operations and infrastructure, and find new sources of revenue.

By 2015
4.4 MILLION IT JOBS
will be created globally to support big data, with 1.9 million in the United States

Volume
SCALE OF DATA

40 ZETTABYTES
[43 TRILLION GIGABYTES]
of data will be created by 2020, an increase of 300 times from 2005

2005
2020

It's estimated that
2.5 QUINTILLION BYTES
[2.3 TRILLION GIGABYTES]
of data are created each day

6 BILLION PEOPLE
have cell phones

WORLD POPULATION: 7 BILLION

Most companies in the U.S. have at least
100 TERABYTES
[100,000 GIGABYTES]
of data stored

Variety
DIFFERENT FORMS OF DATA

As of 2011, the global size of data in healthcare was estimated to be
150 EXABYTES
[161 BILLION GIGABYTES]

By 2014, it's anticipated there will be
420 MILLION WEARABLE, WIRELESS HEALTH MONITORS

4 BILLION+ HOURS OF VIDEO
are watched on YouTube each month

30 BILLION PIECES OF CONTENT
are shared on Facebook every month

400 MILLION TWEETS
are sent per day by about 200 million monthly active users

Velocity
ANALYSIS OF STREAMING DATA

The New York Stock Exchange captures
1 TB OF TRADE INFORMATION
during each trading session

Modern cars have close to
100 SENSORS
that monitor items such as fuel level and tire pressure

By 2016, it is projected there will be
18.9 BILLION NETWORK CONNECTIONS
– almost 2.5 connections per person on earth

Veracity
UNCERTAINTY OF DATA

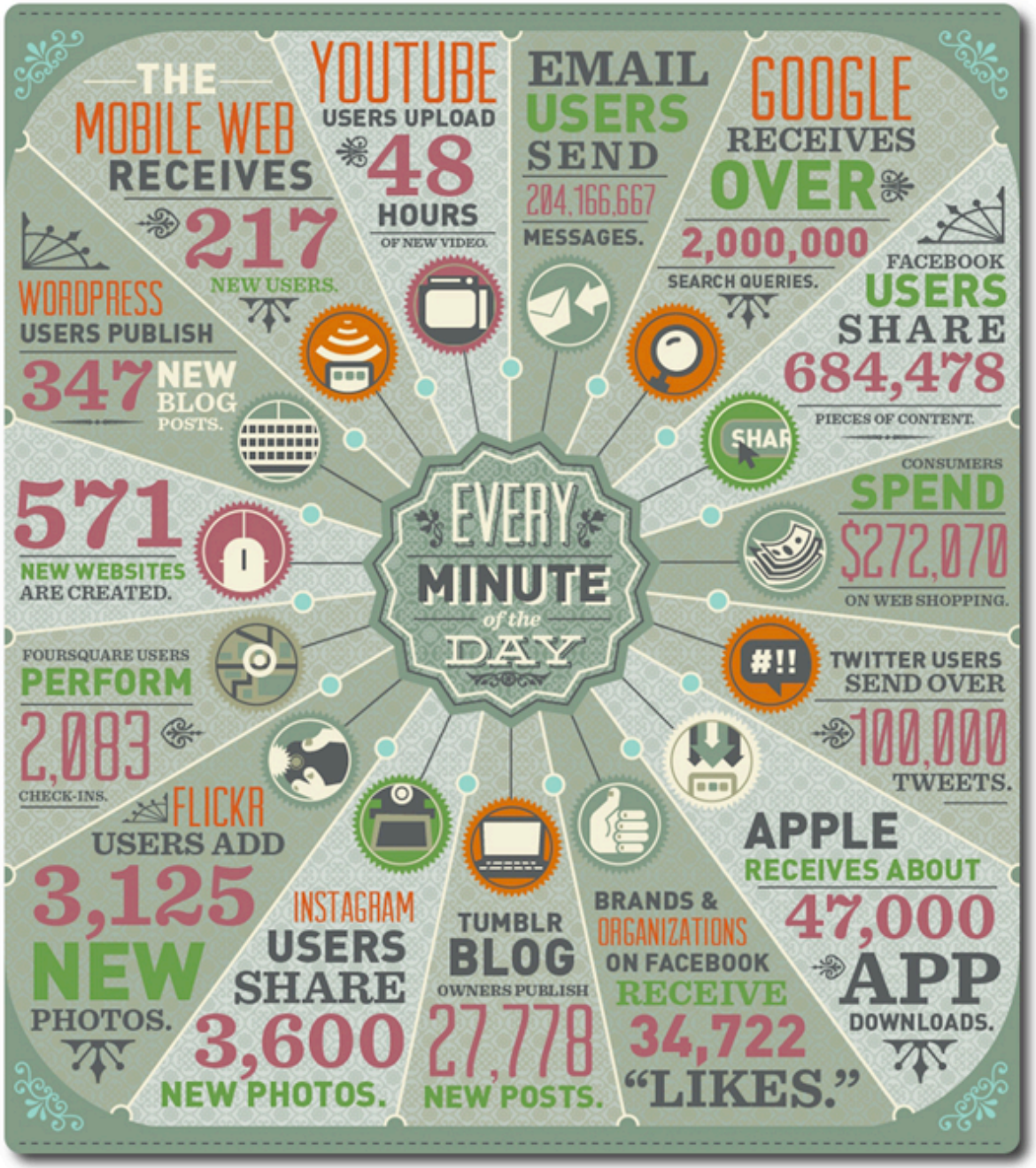
1 IN 3 BUSINESS LEADERS
don't trust the information they use to make decisions

27% OF RESPONDENTS
in one survey were unsure of how much of their data was inaccurate

Poor data quality costs the US economy around
$3.1 TRILLION A YEAR

Sources: McKinsey Global Institute, Twitter, Cisco, Gartner, EMC, SAS, IBM, MEPTEC, QAS

IBM



http://www.thebigdatainsightgroup.com/site/article/big-data-infographic

Data Management

A large fraction of the data still in traditional DBMS systems

Still open and active research areas about improving performance, energy efficiency, new functionalities, changing hardware spectrum (SSDs) and so on…

Much of the data not stored in traditional database systems

For a variety of fairly valid reasons

- Stream processing systems (focusing on *streaming* data)
- Special-purpose data warehousing systems (most start from some RDBMS)
- Batch analysis frameworks (like Hadoop, Pregel)
  - Typically data stored in distributed file systems
- Key-value stores (like HBase, Cassandra, Redis, MongoDB, …)
  - Basically persistent distributed hash tables
- Semi-structured data stores (for XML query processing)
- Graph databases (somewhat new)
- Scientific data management (somewhat new)

However, many lessons to be learned from database research

- We see much reinvention of the wheel and similar mistakes being made as early on

What we will cover

A large fraction of the data still in traditional DBMS systems

- A deeper study of traditional RDBMS solutions (compared to 424)
- New functionalities/features
- Revisit some of the old design decisions (e.g., lay out data column-by-column instead of row-by-row, fully in-memory processing, etc)

Much of the data not stored in traditional database systems

- **Basic ideas behind, and why different from RDBMS:**
  - Stream processing systems
  - Special-purpose data warehousing systems
  - Batch analysis frameworks (specifically MapReduce)
  - Key-value stores (focus on the consistency issues)
- **If time permits:**
  - Semi-structured data stores (we will cover XML model)
  - Graph databases
  - Scientific databases

Learning Goals

- Intended to prepare you for data management research, broadly defined
  - Includes better understanding of data management issues in other fields
- Some specific goals:
  - You should be able to read, understand, and hopefully critique a data management paper
  - Given a new application domain, you should be able to:
    - ask the right questions to understand the key data management issues, and design/suggest appropriate solutions.
    - identify flaws (if any) with a proposed design or solution.
    - devise and reason about abstraction (independence) layers and their applicability to the application domain.
  - You should also have enough familiarity with how big data systems are built to be able to easily start using any of them, and reason about the observed performance of a deployed system, if only superficially.

Differences from Previous Years

- Somewhat major changes to the topics covered
  - More query processing and optimization, less transactions and consistency
  - Fewer papers overall
  - Rough sketch:
    - Background (2 weeks)
    - Large-scale analysis systems – Parallel, MR (2 weeks)
    - Query processing and optimization (3.5 weeks)
    - Survey of Newer DBMS Architectures – OLTP, XML, Graphs, etc (2 weeks)
    - Data streams/Dataflow Engines (2 weeks)
    - Complex/Interactive analytics (2.5 weeks)

- CMSC 8xx Course in Fall 2017 by Daniel Abadi
  - Focus on systems issues, in particular transactions
  - Can get credit for both

Course Overview

- We will cover:
  - A blend of classic papers + ongoing research
  - Textbook:
    - Readings in Database Systems, 5th edition. Mike Stonebraker and Joe Hellerstein, Peter Bailis.
  - Almost all papers are available online
  - Book contains some very nice overview chapters though – all available online at the book website (http://redbook.io)
- Prerequisite: CMSC 424
  - Class notes off of my webpage

Course Overview: Grading

- 3-4 Programming Assignments (15%)
  - First one already online
  - Increased programming component
- Paper readings + Class Participation (20%)
  - 1-2 Papers per class (starting the week after next)
  - Two students responsible for putting up a summary + discussion points for every class, and others responsible for replying to those
  - Will set up a sign-up sheet later
- 3 Written Assignments/Homeworks (15%)
  - First one on background readings to be posted later today and due soon
- Research project + Presentation (35%)
  - More on that in the next class
  - Presentations in the second half on the "problem definition+background"
- Final (15%)
  - Basically a slightly longer written assignment

Current Industry Outlook

- Relational DBMSs
  - Oracle, IBM DB2, Microsoft SQL Server, Sybase
- Open source alternatives
  - MySQL, PostgreSQL, Apache Derby, BerkeleyDB (mainly a storage engine – no SQL), neo4j (graph data) …
- Data Warehousing Solutions
  - Geared towards very large volumes of data and on analyzing them
  - Long list: Teradata, Oracle Exadata, Netezza (based on FPGAs), Aster Data (founded 2005), Vertica (column-based), Kickfire, Xtremedata (released 2009), Sybase IQ, Greenplum (eBay, Fox Networks use them)
  - Usually sell package/services and charge per TB of managed data
  - Many (especially recent ones) start with MySQL or PostgreSQL and make them parallel/faster etc..

Web Scale Data Management, Analysis

- Ongoing debate/issue
  - Cloud computing seems to eschew DBMSs in favor of homegrown solutions
  - E.g. Google, Facebook, Amazon etc…
- MapReduce: A paradigm for large-scale data analysis
  - Hadoop: An open source implementation
- Why ?
  - DBMSs can't scale to the needs, not fault-tolerant enough
    - These apps don't need things like transactions, that complicate DBMSs (???)
  - Mapreduce favors Unix-style programming, doesn't require SQL
    - Try writing SVMs or decision trees in SQL
  - Cost
    - Companies like Teradata may charge $100,000 per TB of data managed

Current Industry Outlook

- Bigtable-like
  - Called “key-value stores”
  - Think highly distributed hash tables
  - Allow some transactional capabilities – still evolving area
  - PNUTS (Yahoo), Cassandra (Facebook), Dynamo (Amazon)

- Mapreduce-like
  - Hadoop (open source), Pig (@Yahoo), Dryad (@Microsoft), Spark (@Berkeley)
  - Amazon EC2 Framework
  - Not really a database – but increasing declarative SQL-like capabilities are being added (e.g. HIVE at Facebook)

My Current Research Interests

- Managing and querying large graph-structured datasets
- Data management for cloud
  - Programming frameworks; transaction management
- Scalable analytics and statistical modeling
- Managing and reasoning about uncertainty in data

- Suggestions for class projects will skew in those directions

- Older work:
  - Data streams, Adaptive query processing, Sensor network data management

Databases: Major Conferences

- ACM SIGMOD (Originally SIGFIDET)
- VLDB (very large databases)
- IEEE ICDE (intl. conf. data engineering)
- EDBT (European database technology)
- PODS, ICDT
  - Theory focused
- CIDR
  - Tends to have vision/overview papers
  - I recommend browsing through 2017 proceedings for ideas on class projects

Today

- Motivation: Why study databases ?
- Background: 424 Summary
- No laptop use allowed in the class !!

Why not use file systems ?

- Drawbacks of using file systems to store data:
  - Data redundancy and inconsistency
    - Multiple file formats, duplication of information in different files
  - Difficulty in accessing data
    - Need to write a new program to carry out each new task
  - Data isolation — multiple files and formats
  - Integrity problems
    - Integrity constraints (e.g., account balance > 0) become "buried" in program code rather than being stated explicitly
    - Hard to add new constraints or change existing ones

Why not use file systems ?

- Drawbacks of using file systems to store data:
  - Atomicity of updates
    - Failures may leave database in an inconsistent state with partial updates carried out
    - Example: Transfer of funds from one account to another should either complete or not happen at all
  - Concurrent access by multiple users
    - Concurrent access needed for performance
    - Uncontrolled concurrent accesses can lead to inconsistencies
      - Example: Two people reading a balance (say 100) and updating it by withdrawing money (say 50 each) at the same time
  - Security problems
    - Hard to provide user access to some, but not all, data

DBMSs to the Rescue

- Provide a systematic way to answer many of these questions…
- Aim is to allow easy management of high volumes of data
  - Storing , Updating, Querying, Analyzing ….

- <u>What is a Database ?</u>
  - A large, integrated collection of (mostly *structured*) data
  - Typically models and captures information about a real-world ***enterprise***
    - Entities *(e.g. courses, students)*
    - Relationships *(e.g.* ***John*** *is taking* ***CMSC 424****)*
    - Usually also contains:
      - Knowledge of constraints on the data *(e.g. course capacities)*
      - Business logic *(e.g. pre-requisite rules)*
      - Encoded as part of the data model (preferable) or through external programs

DBMSs to the Rescue

- Massively successful for *highly structured data*
  - Why ? Structure in the data (if any) can be exploited for ease of use and efficiency
    - If there is no structure in the data, hard to do much
    - Contrast managing emails vs managing photos
  - Much of the data we need to deal with is highly structured
  - Some data is *semi-structured*
    - E.g.: Resumes, Webpages, Blogs etc.
  - Some has complicated structure
    - E.g.: Social networks
  - Some has no structure
    - E.g.: Text data, Video/Image data etc.

Structured vs Unstructured Data

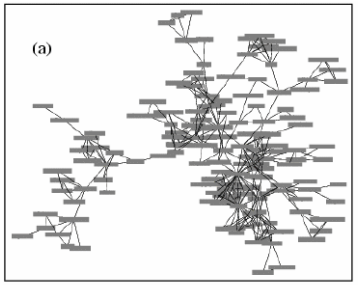
- A lot of the data we encounter is structured
  - Some have very simple structures
    - E.g. Data that can be represented in tabular forms (i.e., as **relations)**
  - Significantly easier to deal with

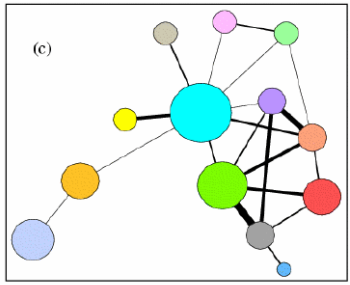
Account		
bname	**acct_no**	**balance**
Downtown	A-101	500
Mianus	A-215	700
Perry	A-102	400
R.H	A-305	350

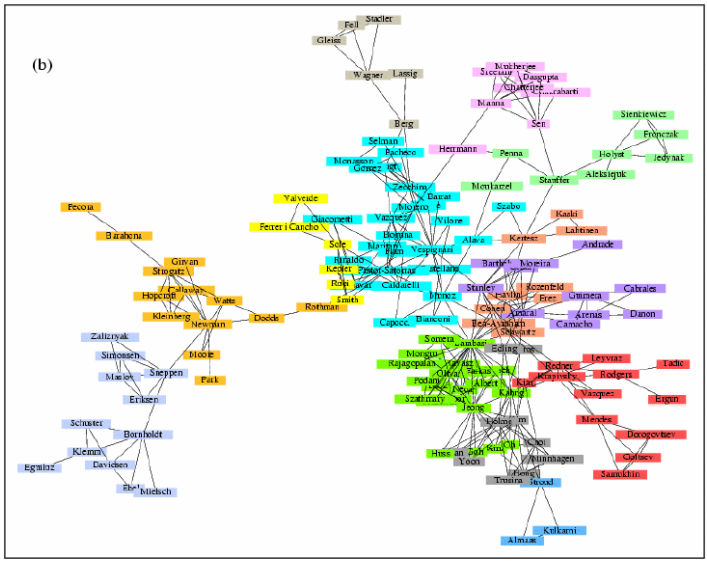
Customer		
cname	**cstreet**	**ccity**
Jones	Main	Harrison
Smith	North	Rye
Hayes	Main	Harrison
Curry	North	Rye
Lindsay	Park	Pittsfield

Structured vs Unstructured Data

- Some data has a little more complicated structure
  - E.g graph structures
    - Map data, social networks data, the web link structure etc
  - Can convert to tabular forms for storage, but may not be optimal
  - Queries often reason about graph structure
    - *Find my "Erdos number"*
    - *Suggest friends based on current friends*
  - Growing importance in recent years in a variety of domains: Biological, social networks, web…



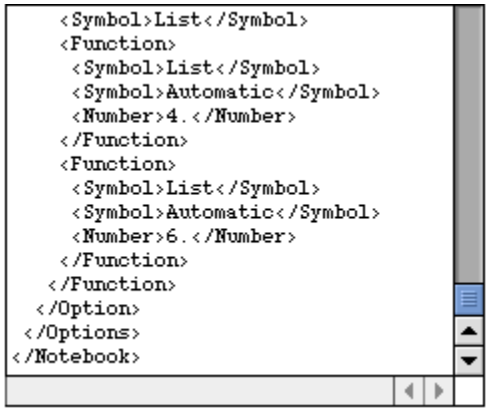




  - **A major research focus for me and others here**

Structured vs Unstructured Data

- Increasing amount of data in a semi-structured format
  - XML – Self-describing tags (HTML ?)
  - Complicates a lot of things
  - We will discuss this toward the end



- A huge amount of data is unfortunately unstructured
  - Books, WWW
  - Amenable to pretty much only text search… so far
    - Information Retreival research deals with this topic
  - What about Google search ?
    - Google search is mainly successful because it uses the structure (in its original incarnation)

- Video ? Music ?
  - Can represent in DBMS's, but can't really operate on them

DBMSs to the Rescue

- Massively successful for *highly structured data*
  - Why ? Structure in the data (if any) can be exploited for ease of use and efficiency
  - How ?
  - Two Key Concepts:
    - Data Modeling: Allows reasoning about the data at a high level
      - e.g. “emails” have “sender”, “receiver”, “…”
      - Once we can describe the data, we can start “querying” it
    - Data Abstraction/Independence:
      - Layer the system so that the users/applications are insulated from the low-level details

DBMSs to the Rescue: Data Modeling

- Data modeling
  - **Data model**: A collection of concepts that describes how data is represented and accessed
  - **Schema:** A description of a specific collection of data, using a given data model

  - Some examples of data models that we will see
    - Relational, Entity-relationship model, XML…
    - Object-oriented, object-relational, semantic data model, RDF…

  - Why so many models ?
    - Tension between descriptive power and ease of use/efficiency
    - More powerful models → more data can be represented
    - More powerful models → harder to use, to query, and less efficient

DBMSs to the Rescue: Data Abstraction

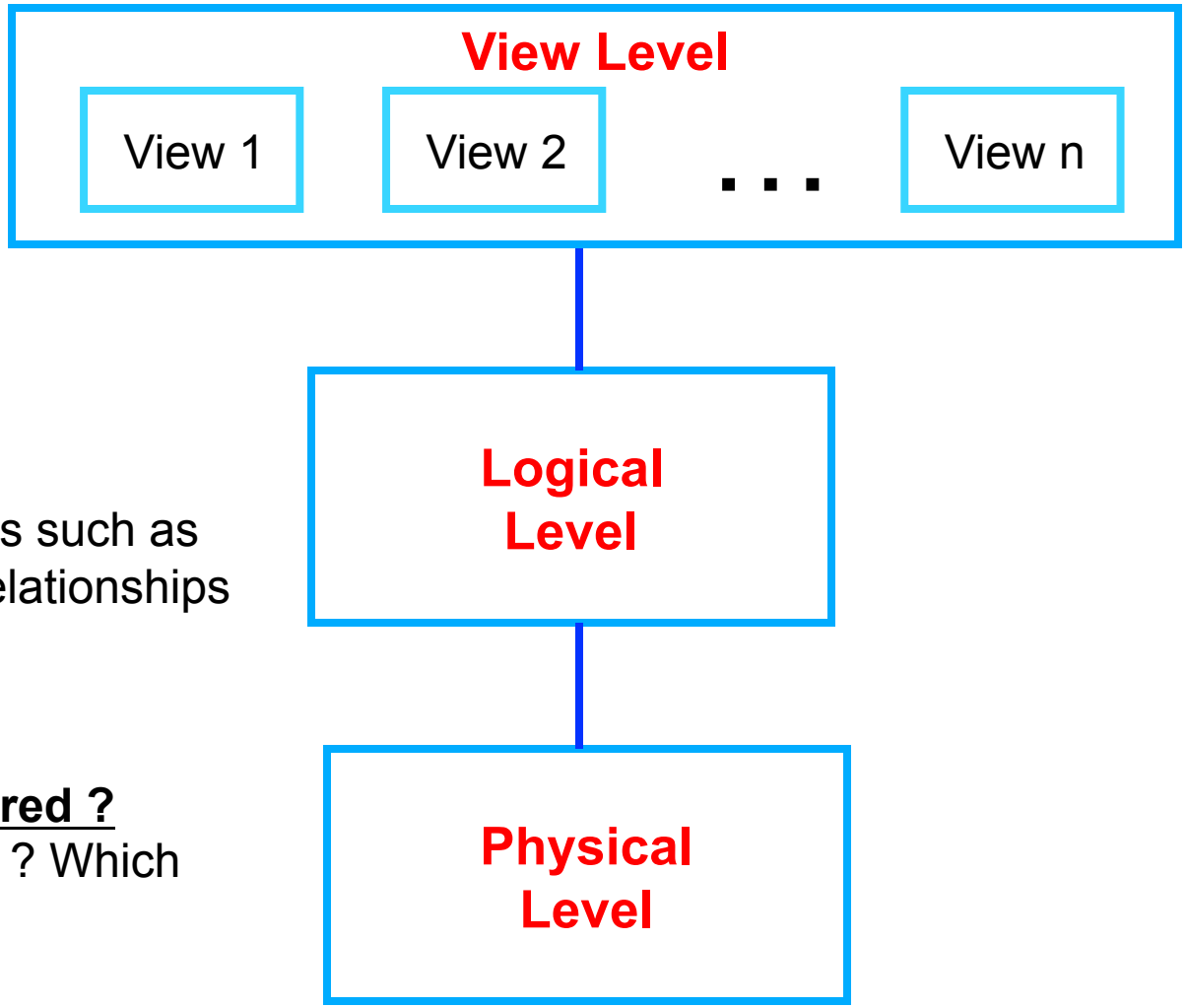
- Also called "Data Independence"

- Probably *the* most important purpose of a DBMS
- Goal: Hiding *low-level details* from the users of the system
  - Alternatively: the principle that
    - *applications and users should be insulated from how data is structured and stored*

- Through use of *logical abstractions*

Data Abstraction

What data users and application programs see ?



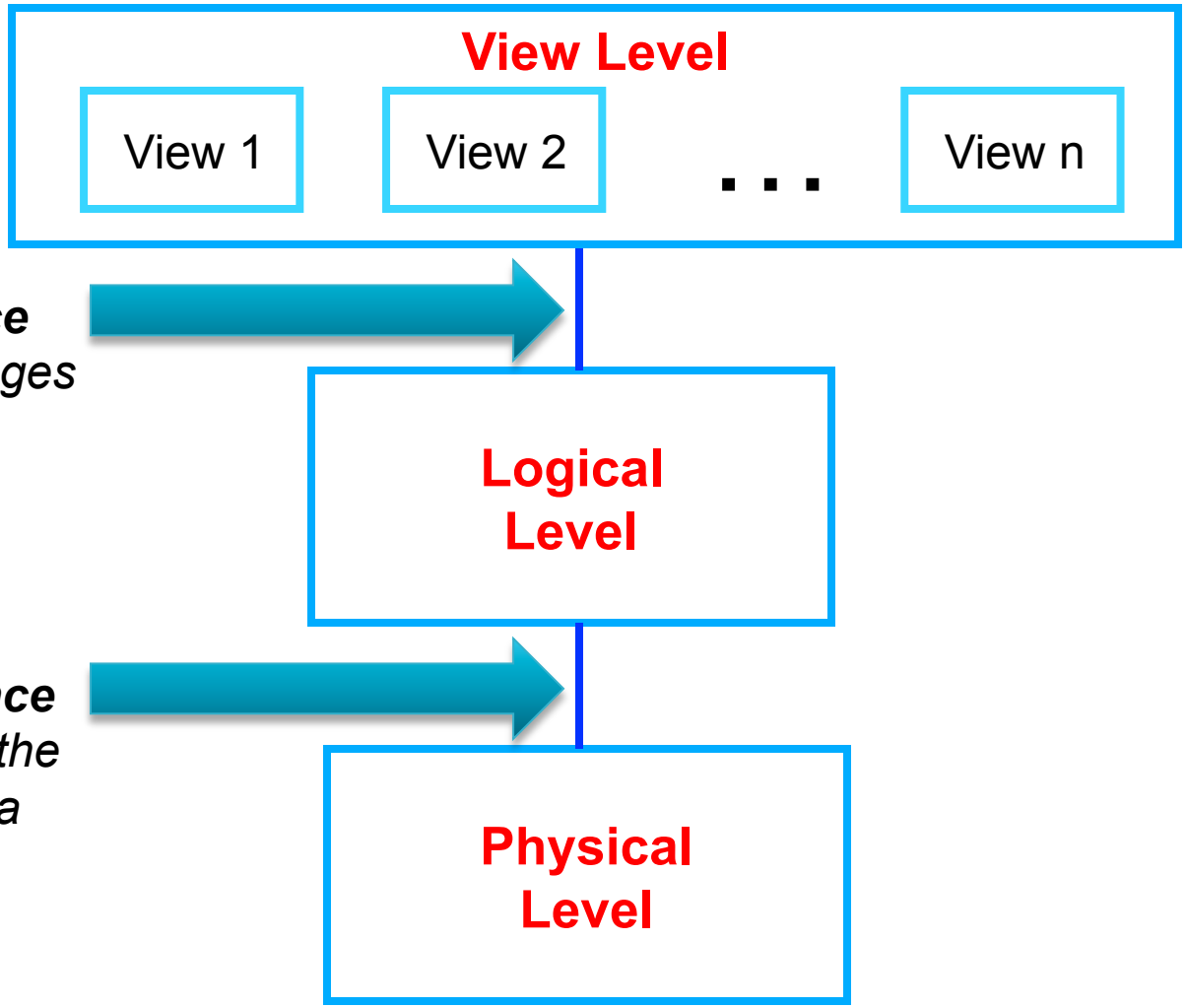
What data is stored ?

describe data properties such as data semantics, data relationships

How data is actually stored ?

e.g. are we using disks ? Which file system ?

Data Abstraction



Logical Data Independence
Protection from logical changes to the schema

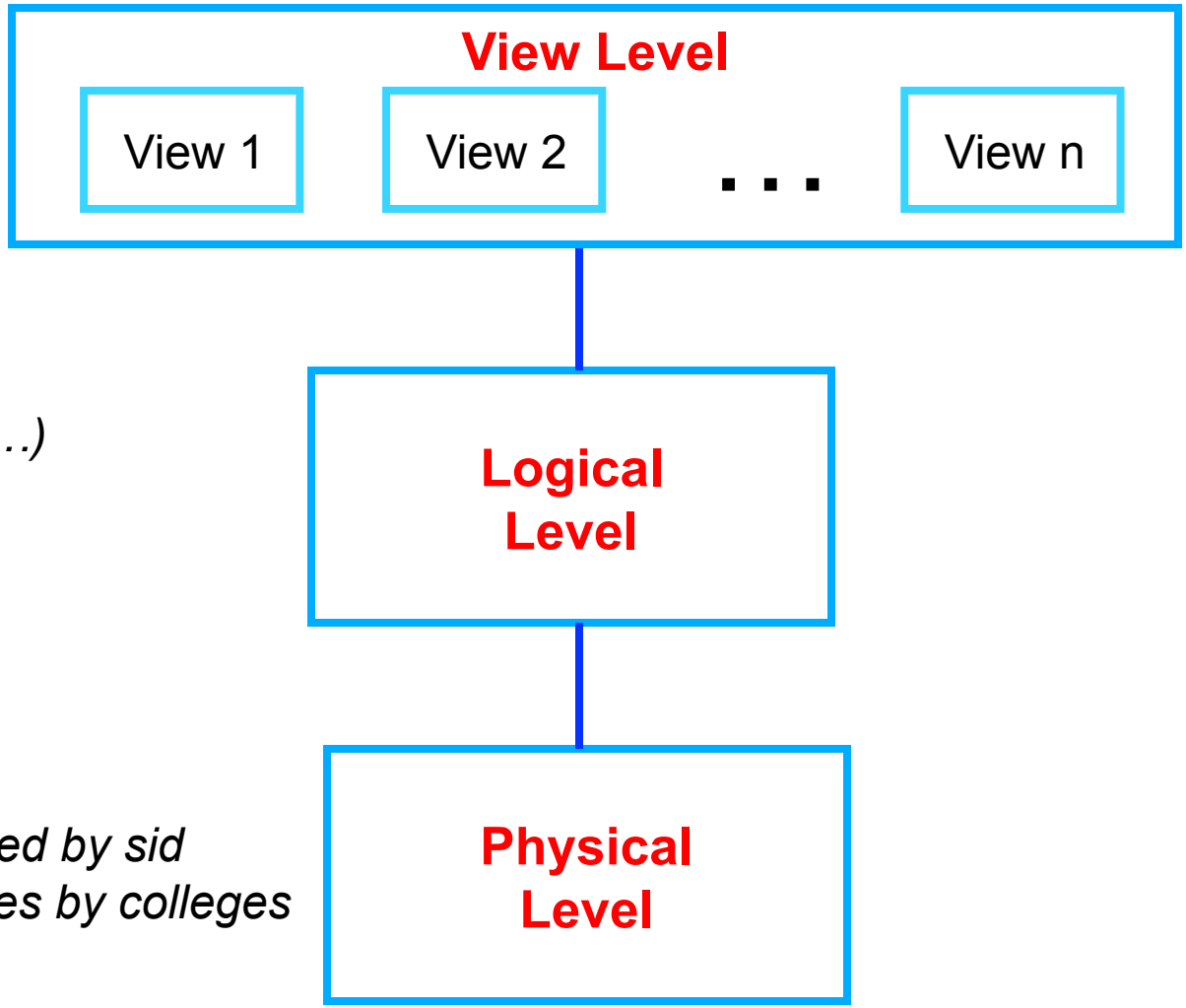
Physical Data Independence
Protection from changes to the physical structure of the data

Data Abstractions: Example

A View Schema
course_info(#registered,…)

Logical Schema
students(sid, name, major, …)
courses(cid, name, …)
enrolled(sid, cid, …)

Physical Schema
all students in one file ordered by sid
courses split into multiple files by colleges



What about a Database System ?

- A DBMS is a software system designed to store, manage, facilitate access to databases

- Provides:
  - Data Definition Language (DDL)
    - For defining and modifying the schemas
  - Data Manipulation Language (DML)
    - For retrieving, modifying, analyzing the data itself
  - Guarantees about correctness in presence of failures and concurrency, data semantics etc.

- Common use patterns
  - Handling transactions (e.g. ATM Transactions, flight reservations)
  - Archival (storing historical data)
  - Analytics (e.g. identifying trends, **Data Mining**)

Relational DBMS: SQL

- **SQL** (sequel): Structured Query Language

- **Data definition (DDL)**
  - **create table** *instructor* (

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

- **Data manipulation (DML)**
  - Example: Find the name of the instructor with ID 22222

    select *name*
    from *instructor*
    where *instructor.ID* = '22222'

Basic topics covered in 424

- representing information
  - data modeling
  - semantic constraints
- languages and systems for querying data
  - complex queries & query semantics
  - over massive data sets
- concurrency control for data manipulation
  - ensuring transactional semantics
- reliable data storage
  - maintain data semantics even if you pull the plug
  - fault tolerance

Basic topics covered in 424

- representing information
  - data modeling: *relational models, E/R models*
  - semantic constraints: *integrity constraints, triggers*
- languages and systems for querying data
  - complex queries & query semantics*: SQL*
  - over massive data sets*: indexes, query processing, optimization*
- concurrency control for data manipulation
  - ensuring transactional semantics: *ACID properties*
- reliable data storage
  - maintain data semantics even if you pull the plug: *durability*
  - fault tolerance: *RAID*

Relational Data Model

- Most widely used model today
- Main concepts:
  - *relation*: basically a table with rows and columns
  - *schema* (of the relation): description of the columns
- Example:

  courses(dept char(4), courseID integer, name varchar(80), instructor varchar(80))

  students(sid char(9), name varchar(80), …)

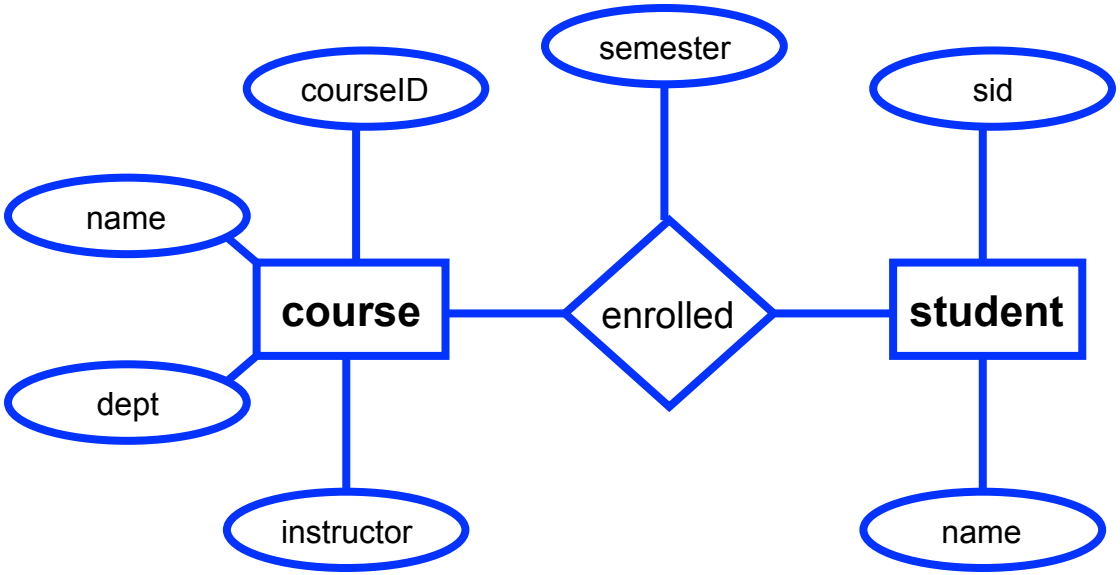
  enrolled(sid char(9), courseID integer, …)
- This is pretty much the only construct

An *instance* of the courses relation

Dept	CourseID	Name	Instructor
CMSC	424	…	…
CMSC	427	…	…

Entity-Relationship (E/R) Data Model

- More powerful model, commonly used during conceptual design
  - Easier and more intuitive for users to work with in the beginning
- Has two main constructs:
  - Entities: e.g. courses, students
  - Relationships: e.g. enrolled
- Diagrammatic representation



Relational Query Languages

- Example schema: *R(A, B)*
- Practical languages
  - <u>SQL</u>
    - select A from R where B = 5;
  - <u>Datalog</u> (sort of practical) – Has seen a resurgence in recent years
    - q(A) :- R(A, 5)
- Formal languages
  - <u>Relational algebra</u>

    $\pi_A (\sigma_{B=5} (R))$ -- You will encounter this in many papers
  - <u>Tuple relational calculus</u>

    $\{ t : \{A\} \mid \exists\ s : \{A, B\} (R(A, B) \wedge s.B = 5) \}$
  - <u>Domain relational calculus</u>
    - Similar to tuple relational calculus

Relational Query Languages

- Important thing to keep in mind:
  - SQL is not SET semantics, it is BAG semantics
  - i.e., duplicates are not eliminated by default
    - With the exception of UNION, INTERSECTION, MINUS
  - Relational model is SET semantics
    - Duplicates cannot exist by definition
- Relational algebra: Six basic operators
  - Select (σ), Project ($\prod$), Carterisan Product ($\times$)
  - Set union (U), Set difference (-)
  - Rename (ρ)

Join Variations (SQL and Relational Alg.)

- Tables: r(A, B), s(B, C)

name	Symbol	SQL Equivalent	RA expression
cross product	$\times$	select * from r, s;	$r \times s$
natural join	$\bowtie$	natural join	$\pi_{r.A,\ r.B,\ s.C}\sigma_{r.B\ =\ s.B}(r\ x\ s)$
theta join	$\bowtie_\theta$	from .. where θ;	$\sigma_\theta(r\ x\ s)$
equi-join	$\bowtie_\theta$ *(theta must be equality)*		
left outer join	$r \mathbin{⟕} s$	left outer join (with "on")	(see previous slide)
full outer join	$r \mathbin{⟗} s$	full outer join (with "on")	–
(left) semijoin	$r \ltimes s$	none	$\pi_{r.A,\ r.B}(r \bowtie s)$
(left) antijoin	$r \triangleright s$	none	$r - \pi_{r.A,\ r.B}(r \bowtie s)$

Relational Model: Normalization

- Goal: What is a "good" schema for a database? How to define and achieve that

- Problems to avoid:
  - Repetition of information
    - For example, a table:
      - *accounts(owner_SSN, account_no, owner_name, owner_address, balance)*
    - Inherently repeats information if a customer is allowed to have more than one account
  - Avoid set-valued attributes

Relational Model: Normalization

1. Encode and list all our knowledge about the schema
   - Functional dependencies (FDs)

     SSN → name (means: SSN "implies" name)
   - If two tuples have the same "SSN", they must have the same "name"

     movietitle → length ???? Not true.
   - But, (movietitle, movieYear) → length --- True.
2. Define a set of rules that the schema must follow to be considered good
   - "Normal forms": 1NF, 2NF, 3NF, BCNF, 4NF, …
   - A normal form specifies constraints on the schemas and FDs
3. If not in a "normal form", we modify the schema

See 424 class notes for more

Semantic Constraints

- SQL supports defining integrity constraints over the data
  - Basically a property that must always be valid
  - E.g., a customer must have an SSN, a customer with a loan must have a sufficiently high balance in checking account, etc.

- Triggers
  - If something happens, then execute something
    - E.g., if a tuple inserted in table *R,* then update table *S* as well
  - Quite frequently used in practice, and surprising not as well optimized for large numbers

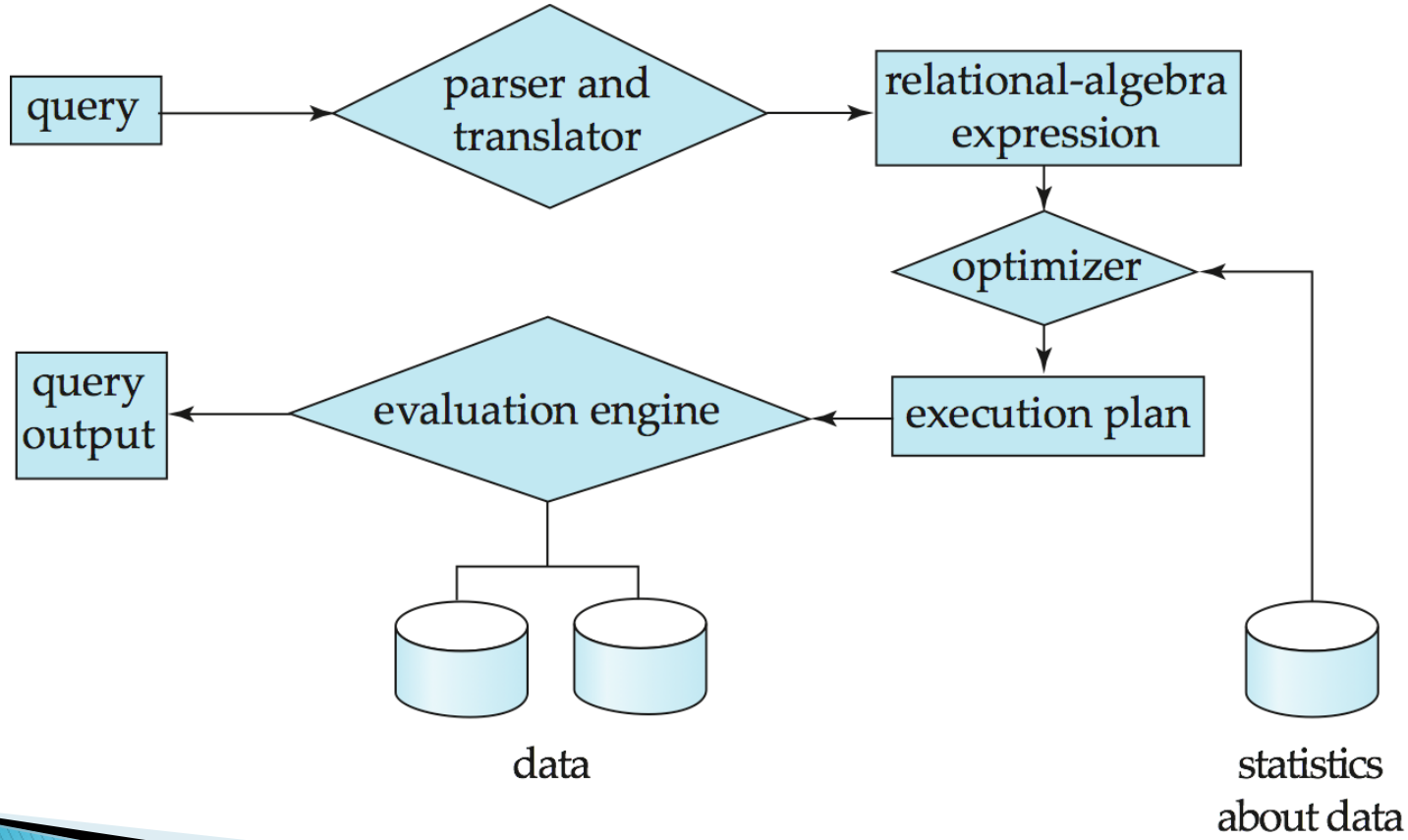
Storage

- Storage:
  - Need to be cognizant of the memory hierarchy
    - Many of traditional DBMS decisions are based on:
      - Disks are cheap, memory is expensive
      - Disks much faster to access sequentially than randomly
    - Much work in recent years on revisiting the design decisions…
  - RAID: Surviving failures through redundancy

- Indexes
  - One of the biggest keys to efficiency, and heavily used
  - **B+-trees** most popular and pretty much the only ones used in most systems
  - Others: R-trees, kD-trees, …

Query Processing

1. Parsing and translation
2. Optimization
3. Evaluation



Transactions

- *Transaction*: A sequence of database actions enclosed within special tags
- Properties:
  - ***Atomicity***: Entire transaction or nothing
  - ***Consistency***: Transaction, executed completely, takes database from one consistent state to another
  - ***Isolation***: Concurrent transactions *appear* to run in isolation
  - ***Durability***: Effects of committed transactions are not lost
- Consistency: programmer needs to guarantee that
  - DBMS can do a few things, e.g., enforce constraints on the data
- Rest: DBMS guarantees

Transactions: How?

- ***Atomicity***: Through "logging" of all operations to "stable storage", and reversing if the transaction did not finish
- ***Isolation***:
  - Locking-based mechanisms
  - Multi-version concurrency control
- ***Durability***: Through "logging" of all operations to "stable storage", and repeating if needed

- Some key concepts:
  - Two-phase locking, Write-ahead logging

Next week..

- More background
- Recommend skimming through the background material
  - Would need to do that for the homework anyway
- Deep dive into research papers after that…