CMSC 424: Summary of Topics

Instructor: Amol Deshpande
amol@cs.umd.edu

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 ….

- What is a Database ?
  - 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: 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

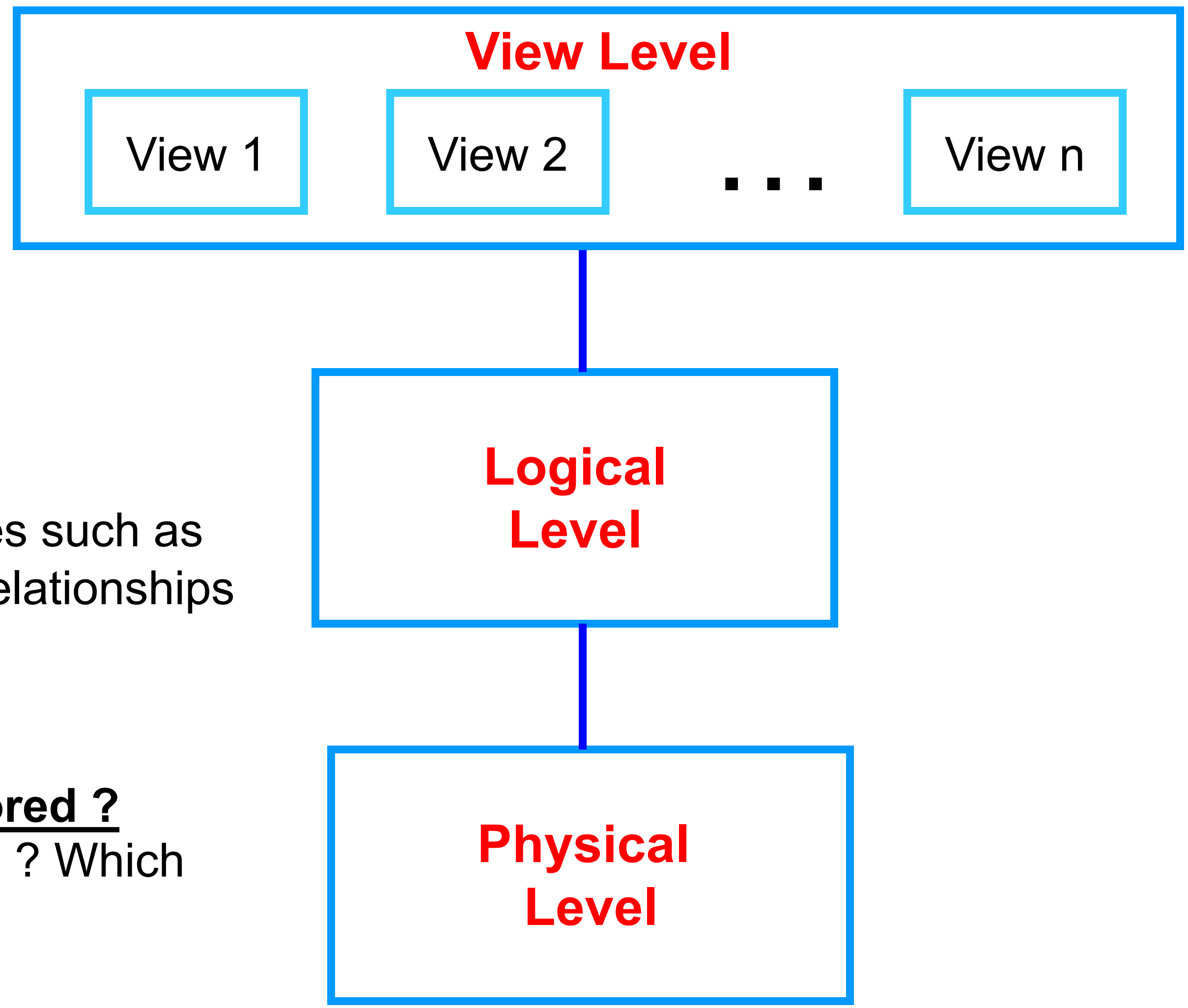
- Also called "Data Independence"

- 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

<u>What data users and application programs see ?</u>



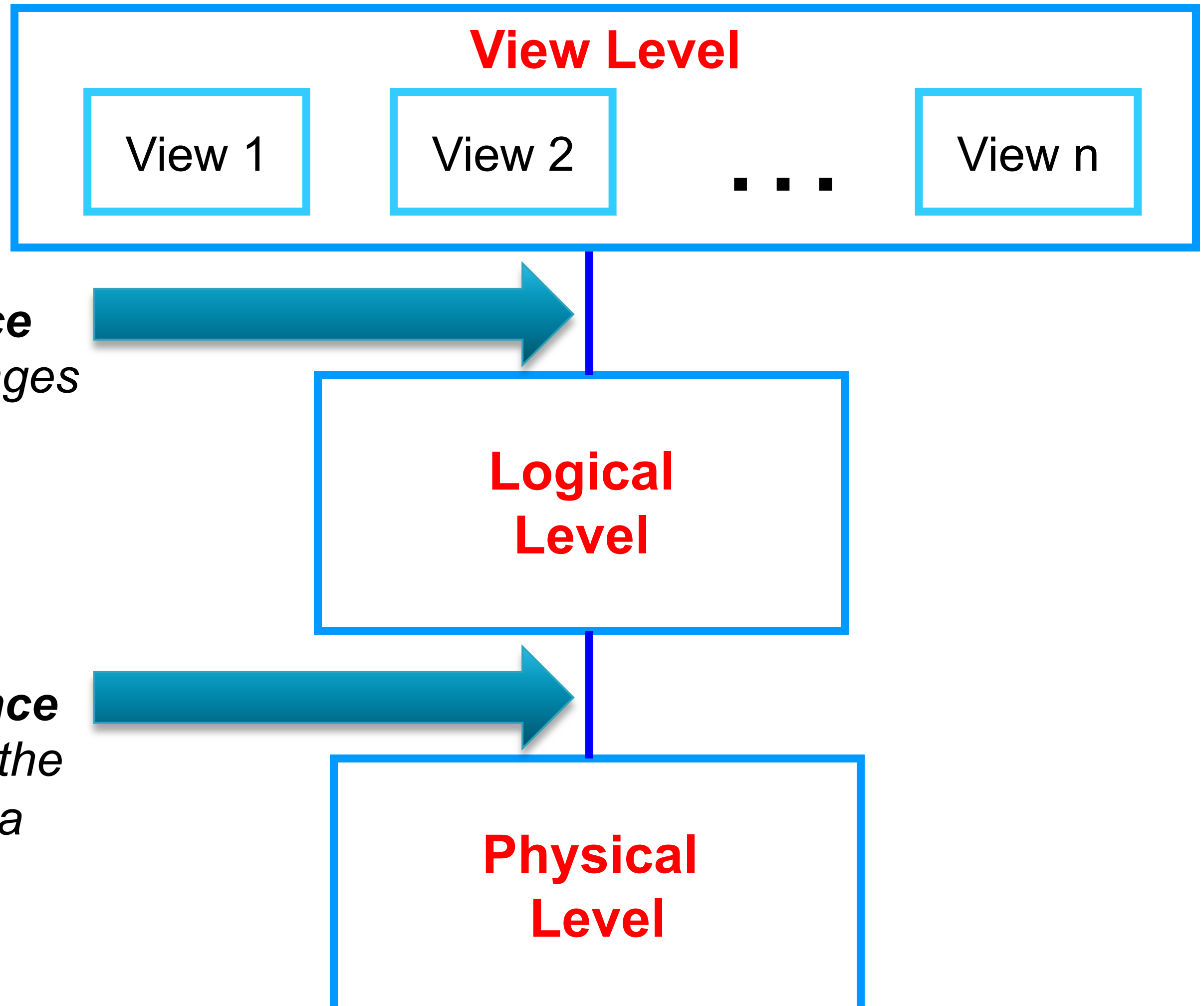
<u>What data is stored ?</u>

describe data properties such as data semantics, data relationships

<u>How data is actually stored ?</u>

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

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**)

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), …)

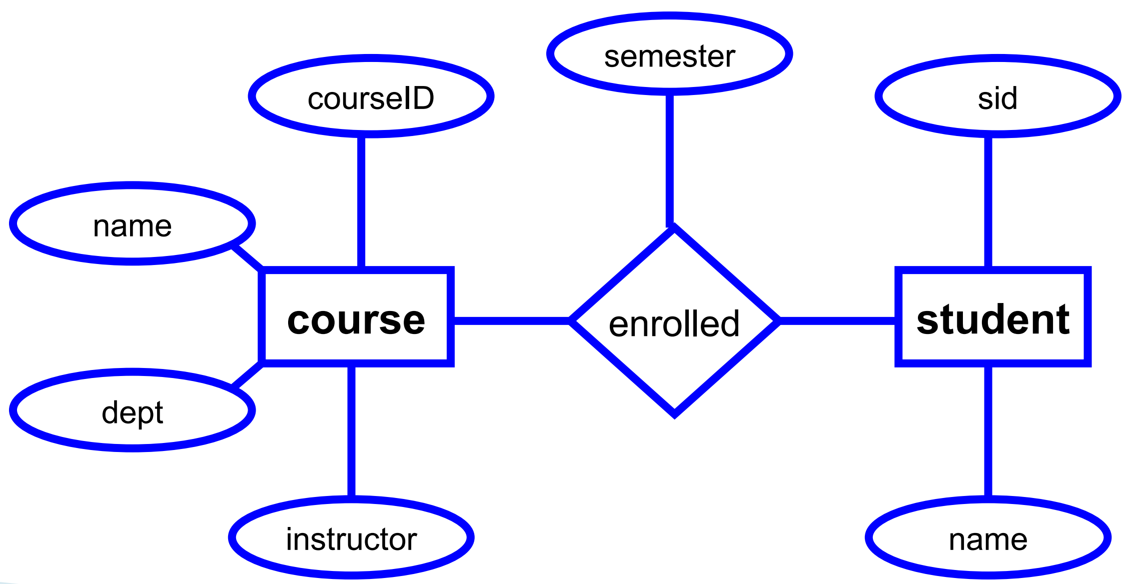
  enrolled(sid char(9), courseID integer, …)
- This is pretty much the only construct

An *instance* of the courses relation

Dept	CourseID	Name	Instructo r
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
  - SQL
    - select A from R where B = 5;
  - Datalog (sort of practical) – Has seen a resurgence in recent years
    - q(A) :- R(A, 5)
- Formal languages
  - Relational algebra

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

    $\{ t : \{A\} \mid \exists\, s : \{A, B\} (R(A, B) \wedge s.B = 5) \}$
  - Domain relational calculus
    - 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 (π), Carterisan Product (×)
  - 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	$r ⟕ s$	left outer join (with “on”)	(see previous slide)
full outer join	$r ⟗ s$	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

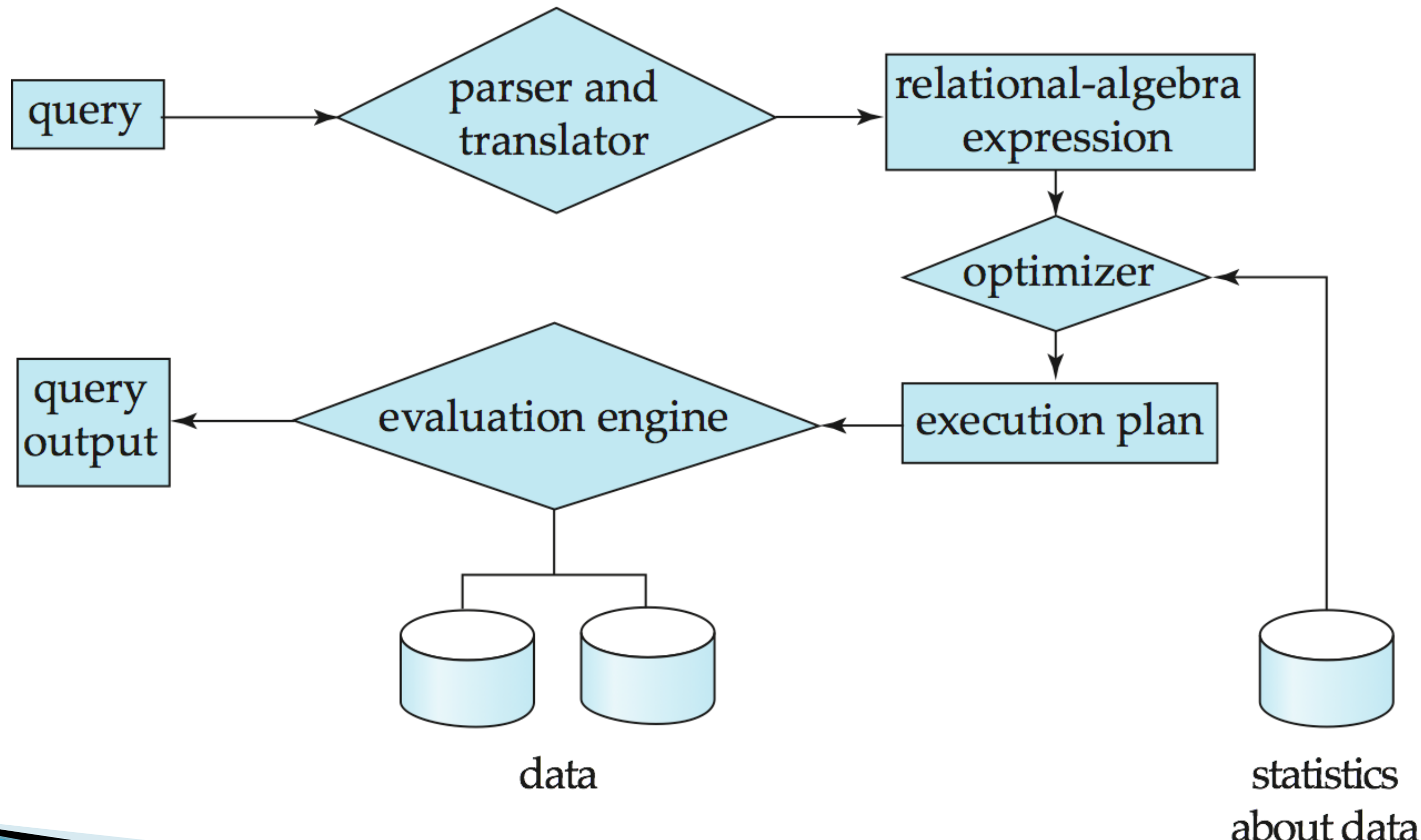
Storage

- 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



Parallel and NoSQL

- Parallel and Distributed Environments
  - Shared-nothing vs Shared-memory vs Shared-disk
  - Speedup vs Scaleup
- How to ”parallelize” different relational operations
- Motivation for emergence of NoSQL Systems
- Map-reduce Framework for Large-scale Data Analysis
- Apache Spark: Resilient Distributed Dataset (RDD) Abstraction
- MongoDB
  - JSON Data Model
  - MongoDB Query Language

Transactions

- *<u>Transaction</u>*: A sequence of database actions enclosed within special tags
- Properties:
  - ***<u>Atomicity</u>***: Entire transaction or nothing
  - ***<u>Consistency</u>***: Transaction, executed completely, takes database from one consistent state to another
  - ***<u>Isolation</u>***: Concurrent transactions *<u>appear</u>* to run in isolation
  - ***<u>Durability</u>***: 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:
  - Serializability, Recoverability, Snapshot Isolation, Two-phase locking, Write-ahead logging, …