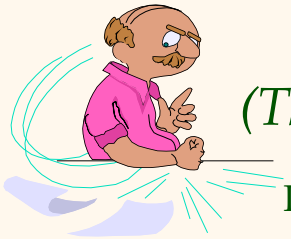
# *Introduction to Data Management*



## *Lecture #1*
## *(The Course "Trailer")*

Instructor: Mike Carey
mjcarey@ics.uci.edu



## *Today's Topics*


- Welcome to my biggest class *ever*!
- Read (and live by!) the course wiki page:
  - http://www.ics.uci.edu/~cs122a/
- Also follow (and live by) the Piazza page:
  - https://piazza.com/uci/spring2017/cs122a/home
- Let's peek at the wiki page, and then lets also preview what lies ahead...
- ***Note*:** There *will* be a quiz in this week's initial discussion sessions...!
  - *Note*: You must attend the one you registered for.

## *What is a Database System?*

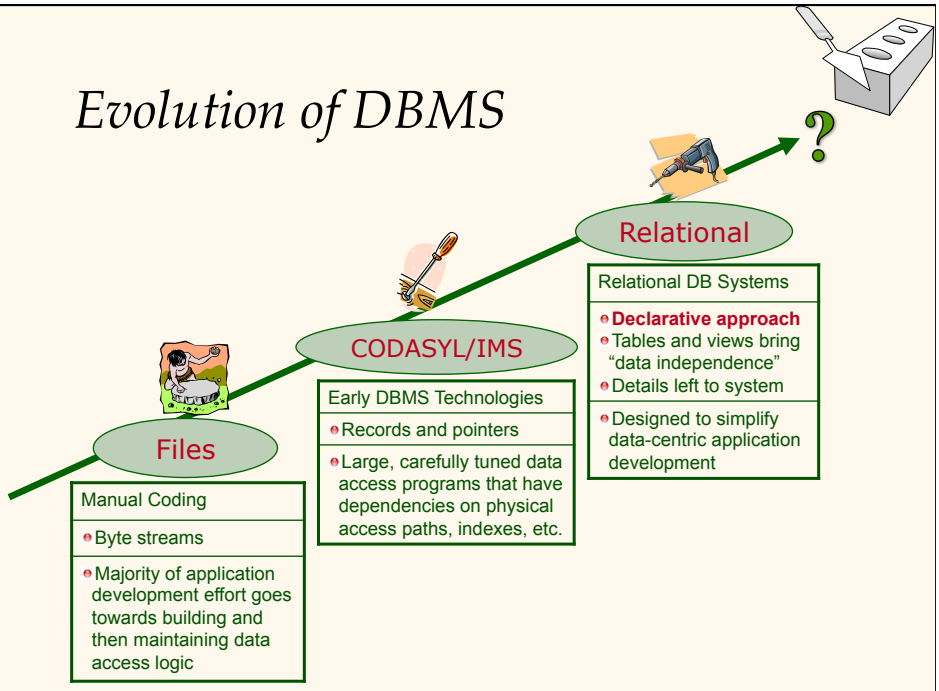
- What's a *database*?
  - A very large, integrated collection of data
- Usually a model of a *real-world enterprise*
  - ***Entities*** (*e.g.*, students, courses, Facebook users, …) with attributes (*e.g.*, name, birthdate, GPA, …)
  - ***Relationships*** (*e.g.*, Susan is *taking* CS 234, Susan is a *friend of* Lynn, …)
- What's a *database management system* (DBMS)?
  - A software system designed to store, manage, and provide access to one or more databases



## *File Systems vs. DBMS*

- Application programs must sometimes *stage large datasets* between main memory and secondary storage (for buffering huge data sets, getting page-oriented access, etc.)
- *Special code needed* for different queries, and that code must be (stay) correct and efficient
- Must *protect data from inconsistency* due to multiple concurrent users
- *Crash recovery* is important since data is now the currency of the day (corporate jewels)
- *Security and access control* are also important(!)

# *Evolution of DBMS*






# *Why Use a DBMS?*

- Data independence.
- Efficient data access.
- Reduced application development time.
- Data integrity and security.
- Uniform data administration.
- Concurrent access, recovery from crashes.

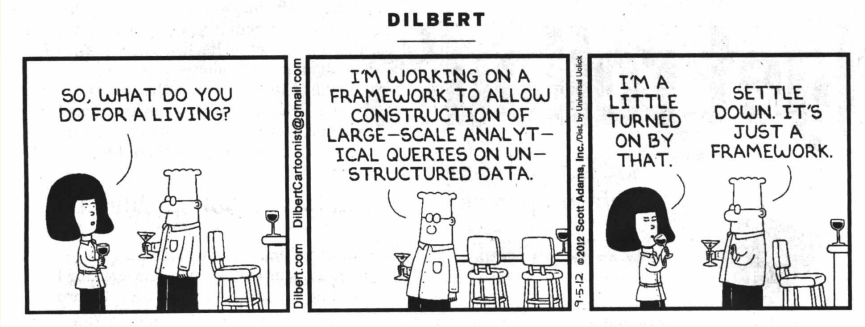
# *Why Study Databases?*

- Shift from *computation* to *information*
  - At the "low end": explosion of the web (a mess!)
  - At the "high end": scientific applications, social data analytics, …
- Datasets increasing in diversity and volume
  - Digital libraries, interactive video, Human Genome project, EOS project , the Web itself, …
  - Mobile devices, Internet of Things, …
  - ***… need for DBMS exploding!***


- DBMS field encompasses most of CS!!
  - OS, languages, theory, AI, multimedia, logic, …



# *Why Study Databases (Really)?*




***Big Data!*** ☺

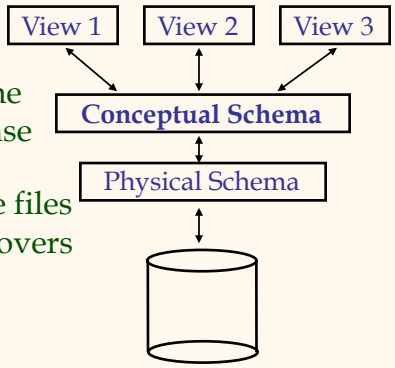
# *Data Models*

- A *data model* is a collection of concepts for describing data
- A *schema* is a description of a particular collection of data, using a given data model
- The *relational model* is (still) the most widely used data model today
  - *Relation* – basically a table with rows and (named) columns
  - *Schema* – describes the tables and their columns



# *Levels of Abstraction*

- Many *views* of one *conceptual (logical) schema* and an underlying *physical schema*
  - Views describe how different users see the data.
  - Conceptual schema defines the logical structure of the database
  - Physical schema describes the files and indexes used under the covers



# *Example: University DB*

- Conceptual schema:
  - *Students(sid: string, name: string, login: string, age: integer, gpa: real)*
  - *Courses(cid: string, cname: string, credits: integer)*
  - *Enrolled(sid: string, cid: string, grade: string)*
- Physical schema:
  - Relations stored as unordered files
  - Index on first and third columns of *Students*
- External schema (*a.k.a.* view):
  - *CourseInfo(cid: string, cname: string, enrollment: integer)*



# *Data Independence*

- Applications are *insulated* (at multiple levels) from how data is actually structured and stored, thanks to schema layering and high-level queries
  - *Logical data independence*: Protection from changes in the *logical* structure of data
  - *Physical data independence*: Protection from changes in the *physical* structure of data
- *One of the most important benefits of DBMS use!*
  - Allows *changes* to occur – *w/o application rewrites!*

# University DB Example (cont.)

- User query (in SQL, against the external schema):
  - *SELECT c.cid, c.enrollment*
    *FROM CourseInfo c*
    *WHERE c.cname = 'Computer Game Design'*
- Equivalent query (against the conceptual schema):
  - *SELECT e.cid, count(e.*)*
    *FROM Enrolled e, Courses c*
    *WHERE e.cid = c.cid AND c.cname = 'Computer Game Design'*
    *GROUP BY c.cid*
- Under the hood (against the physical schema)
  - Access *Courses* – use index on *cname* to find associated *cid*
  - Access *Enrolled* – use index on *cid* to count the enrollments



# Concurrency and Recovery

- *Concurrent execution* of user programs is essential to achieve good DBMS performance.
  - Disk accesses are frequent and slow, so it's important to keep the CPUs busy by serving multiple users' programs concurrently.
  - Interleaving multiple programs' actions can lead to inconsistency: e.g., a bank transfer while a customer's assets are being totalled.
- *Errors or crashes* may occur during, or soon after, the execution of users' programs.
  - This could lead to undesirable partial results or to lost results.
- DBMS answer: Users/programmers can pretend that they're using a reliable, single-user system!

## *Transaction: An Execution of a DB Program*

- Key concept is *transaction*: An *atomic* sequence of database actions (e.g., SQL operations on records).
- Each transaction, when run to completion, is expected to leave the DB in a *consistent state* if the DB was consistent before it started to run.
  - Users can specify simple *integrity constraints* on the data, and the DBMS will enforce these constraints.
  - Beyond this, the DBMS is happily clueless about the data's semantics (e.g., how bank interest is computed).
  - Note: Ensuring that any one transaction (*when run all by itself*) preserves consistency is the programmers job!



## *Concurrent DBMS Transactions*

- The DBMS ensures that its execution of {T1, ... , Tn} is equivalent to some (in fact, any!) *serial* execution.
  - Before reading/writing a record, a transaction must request a **lock** on the record and wait until the DBMS grants it. (All locks are released together, at the end of the transaction.)
  - Key Idea: If any action of a transaction Ti (e.g., writing record X) impacts Tj (e.g., reading record X), one of them will lock X first and the other will have to wait until the first one is done – which orders the transactions!

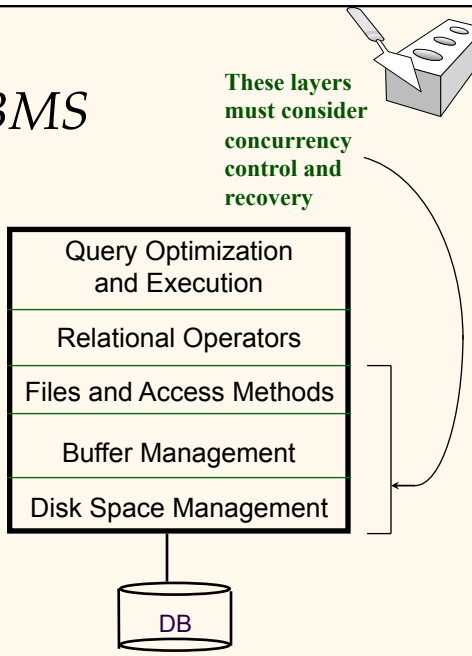
# *Ensuring Atomicity*


- DBMS ensures *atomicity* (an all-or-nothing outcome) even if the system crashes in the middle of a Xact.
- Idea: Keep a *log* (history) of all actions carried out by the DBMS while executing a set of Xacts:
  - Before a change is made to the database, a log entry (old value, new value) is forced to a safe (different) location.
  - In the event of a crash, the effects of partially executed transactions can first be *undone* using the log.
  - In the event of a data loss following a successful finish, lost transaction effects can also be *redone* using the log.
  - Note: The DBMS does all of this transparently!

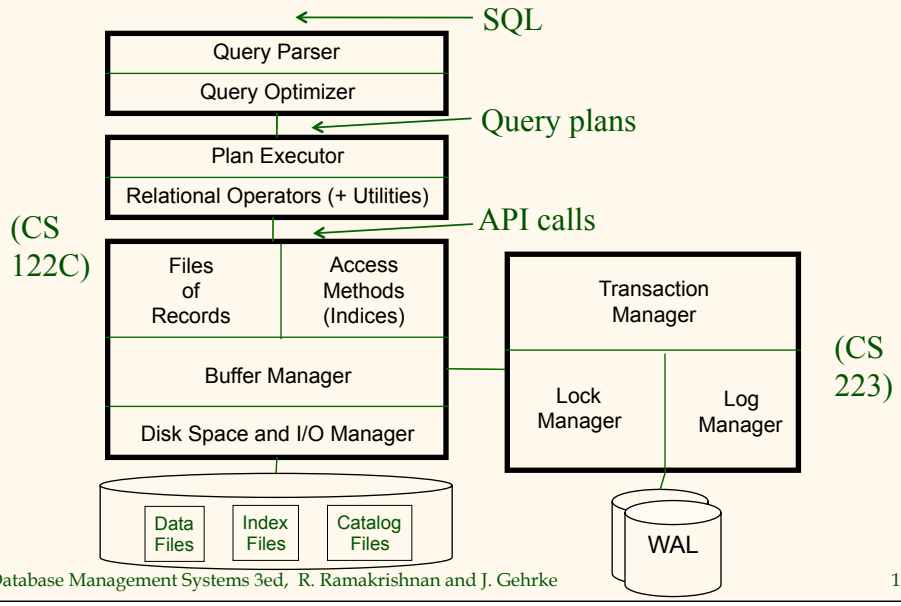


# *Structure of a DBMS*

- A typical DBMS has a layered architecture.
- The figure does not show the concurrency control and recovery components (CS 223).
- This is one of several possible architectures; each system has its own variations.



# DBMS Structure (More Detail)






# Components' Roles

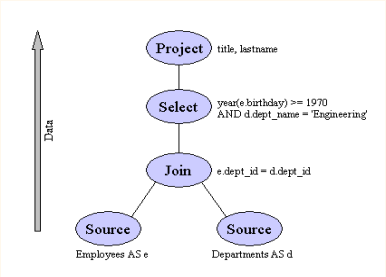
```
SELECT e.title, e.lastname
FROM Employees e, Departments d
WHERE e.dept_id = d.dept_id AND
      year (e.birthday >= 1970) AND
      d.dept_name = 'Engineering'
```

- Query Parser
  - Parse and analyze *SQL query*
  - Makes sure the query is valid and talking about tables, etc., that indeed exist
- Query optimizer (often w/2 steps)
  - *Rewrite* the query logically
  - Perform cost-based *optimization*
  - Goal is a "good" query plan considering
    - Physical table structures
    - Available access paths (indexes)
    - Data statistics (if known)
    - Cost model (for relational operations)

*(Cost differences can be orders of magnitude!!!)*

# Components' Roles (continued)

- Plan Executor + Relational Operators
  - Runtime side of query processing
  - Query plan is a tree of relational operators (drawn from the ***relational algebra***, which you will learn all about in this class)






# Components' Roles (continued)

- Files of Records
  - OSs usually have *byte-stream* based APIs
  - DBMSs instead provide ***record*** based APIs
    - Record = set of fields
    - Fields are typed
    - Records reside on pages of files
- Access Methods
  - Index structures for lookups based on field values
  - We'll look in more depth at ***B+ tree*** indexes in this class (as they are the most commonly used indexes across all commercial and open source systems)

## *Components' Roles (continued)*

- Buffer Manager
  - The DBMS answer to *main memory* management!
  - All disk page accesses go through the buffer pool
  - Buffer manager caches pages from files and indices
  - "DB-oriented" page replacement scheme(s)
  - Also interacts with logging (so undo/redo possible)
- Disk Space and I/O Managers
  - Manage space on *disk* (pages), including extents
  - Also manage I/O (sync, async, prefetch, …)
  - Remember: database data is *persistent (!)*



## *Components' Roles (continued)*

- System Catalog (or "Metadata")
  - Info about physical data (file system stuff)
  - Info about tables (name, columns, types, … ); also, info about any constraints, keys, etc.
  - Data statistics (e.g., value distributions, counts, …)
  - Info about indexes (kinds, target tables, …)
  - And so on! (Views, security, …)
- Transaction Management
  - ACID (Atomicity, Consistency, Isolation, Durability)
  - Lock Manager for Consistency + Isolation
  - Log Manager for Atomicity + Durability

## *Miscellany: Some Terminology*

- Data Definition Language (DDL)
  - Used to express views + logical schemas (using a syntactic form of a a data model, e.g., relational)
- Data Manipulation Language (DML)
  - Used to access and update the data in the database (again in terms of a data model, e.g., relational)
- Query Language (QL)
  - Synonym for DML or its retrieval (i.e., data access or query) sublanguage



## *Miscellany (Cont'd.): Key Players*

- Database Administrator (DBA)
  - The "super user" for a database or a DBMS
  - Deals with things like physical DB design, tuning, performance monitoring, backup/restore, user and group authorization management
- Application Developer
  - Builds data-centric applications (CS122b!)
  - Involved with logical DB design, queries, and DB application tools (e.g., JDBC, ORM, …)
- Data Analyst or End User
  - Non-expert who uses tools to interact w/the data

## *A Brief History of Databases*

- Pre-relational era: 1960's, early 1970's
- Codd's seminal paper: 1970
- Basic RDBMS R&D: 1970-80 (System R, Ingres)
- RDBMS improvements: 1980-85
- Relational goes mainstream: 1985-90
- Distributed DBMS research: 1980-90
- Parallel DBMS research: 1985-95
- Extensible DBMS research: 1985-95
- OLAP and warehouse research: 1990-2000
- Stream DB and XML DB research: 2000-2010
- "Big Data" R&D (also including "NoSQL"): 2005-present



## *Summary*

- DBMS is used to maintain & query large datasets.
- Benefits include recovery from system crashes, concurrent access, quick application development, data integrity and security.
- Levels of abstraction give <u>data independence</u>.
- A DBMS typically has a layered architecture.
- DBAs (and friends) hold responsible jobs and they are also well-paid! (☺)
- Data-related *R&D* is one of the broadest, most exciting areas in CS.

# *So Now What?*


- Time to dive into the first tech topic:
  - Logical DB design (ER model)
- Read the first two chapters of the book
  - Intro and ER – see the syllabus on the wiki
- Immediate to-do's for you are:
  - Be sure you're signed up on Piazza
  - Stockpile sleep – no homework ***yet*** (☺)
- Let's move on to database design…