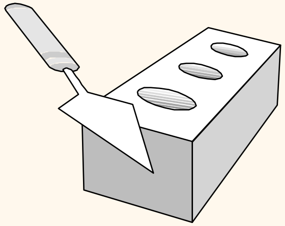
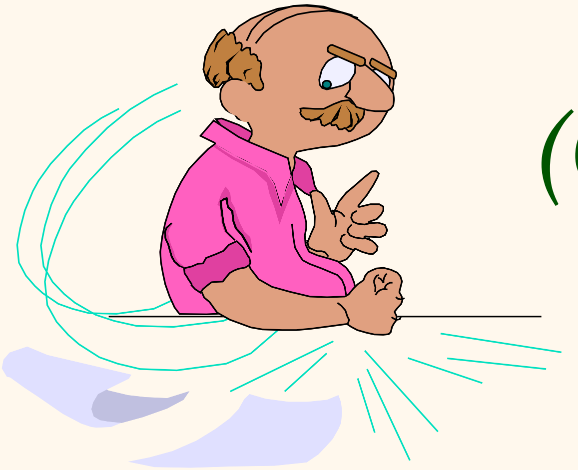


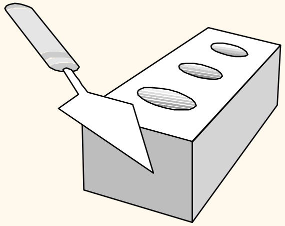
Introduction to Data Management

Lecture #1
(Course "Trailer")

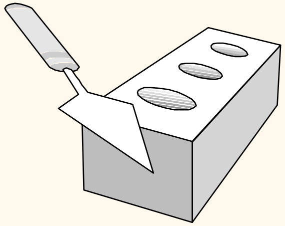


Instructor: Chen Li

Today's Topics



- Welcome to one of my biggest classes *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/spring2016/cs122a/home
  - Let's look at both of these, and then lets also look at a preview of what lies ahead.
- ***Note*:** There *will* be a quiz in this week's discussions – and, you *will* need to prepare (by reading about *Academic Honesty*)…!

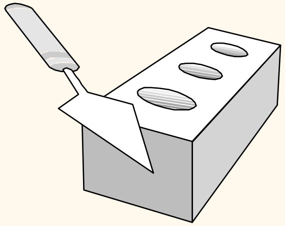


What is a Database System?

- 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 122A, 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

Relational

Relational DB Systems

- **Declarative approach**
- Tables and views bring "data independence"
- Details left to system

- Designed to simplify data-centric application development

CODASYL/IMS

Early DBMS Technologies

- Records and pointers

- Large, carefully tuned data access programs that have dependencies on physical access paths, indexes, etc.

Files

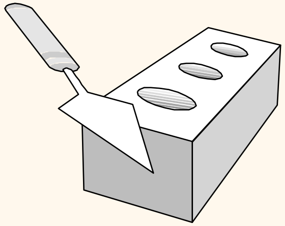
Manual Coding

- Byte streams

- Majority of application development effort goes towards building and then maintaining data access logic

Why Use a DBMS?

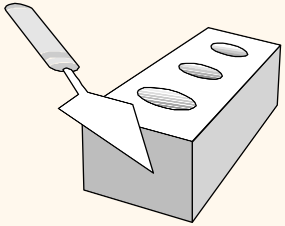
- Data independence.
- Efficient data access.
- Reduced application development time.
- Data integrity and security.
- Uniform data administration.
- Concurrent access, recovery from crashes.



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, …

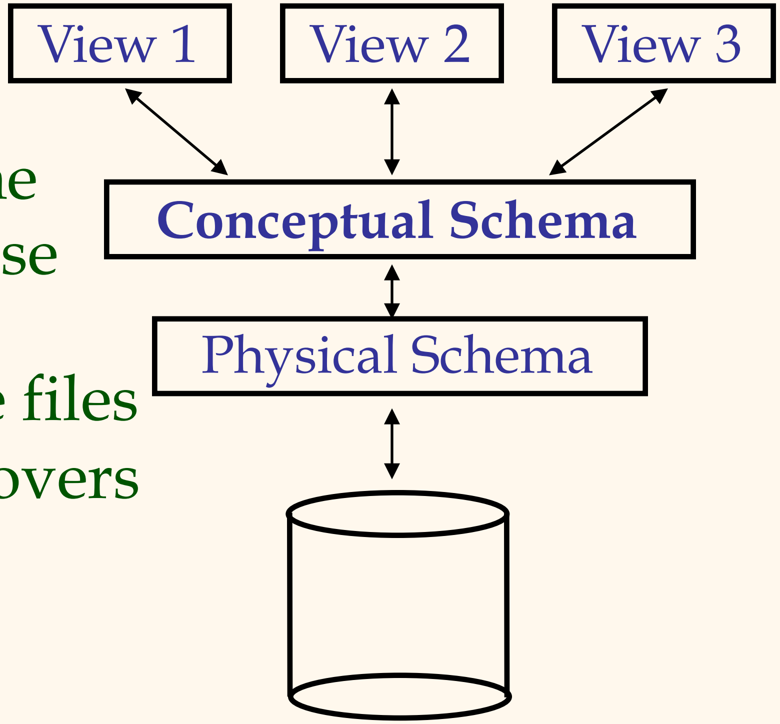


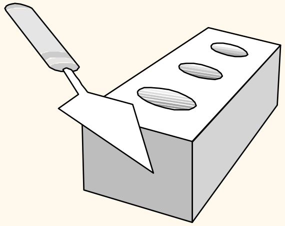
Data Models

- 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

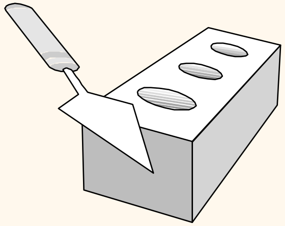
- Many *views* of one *conceptual (logical) schema* and an underlying *physical schema*
  - Views describe how different users see the data.
  - Conceptual schema defines the logical structure of the database
  - Physical schema describes the files and indexes used under the covers





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)*

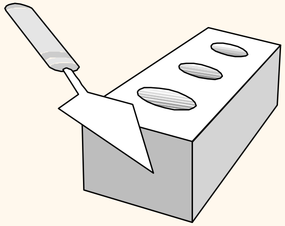


Data Independence

- Applications are *insulated* (at multiple levels) from how data is actually structured and stored
  - *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!*

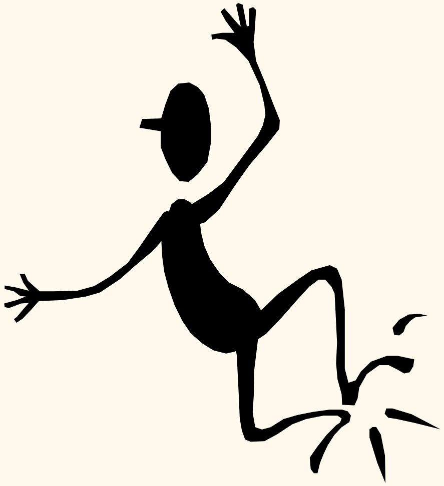
Example: University DB (cont.)

- 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

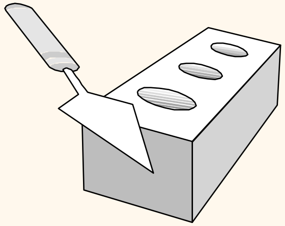


Databases: The Cast

- End users and DBMS software vendors
- DB application programmers
  - *E.g.*, smart webmasters
- Database administrator (DBA)
  - Designs logical and physical schemas
  - Handles security and authorization
  - Ensures data availability, crash recovery
  - Tunes the database (physical schema) as needs evolve

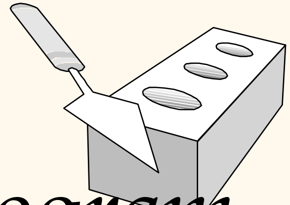


→ (DBA must understand how a DBMS works!) ←



Concurrency Control

- *Concurrent execution* of user programs is essential for good DBMS performance.
  - Because disk accesses are frequent, and relatively slow, it is crucial to keep the CPUs (cores!) humming by working on multiple users' programs concurrently.
- Interleaving actions of different user programs can lead to inconsistency: e.g., a bank transfer is run while a customer's assets are being totalled.
- DBMS ensures that such problems don't arise: users/programmers can pretend they're using a single-user system.

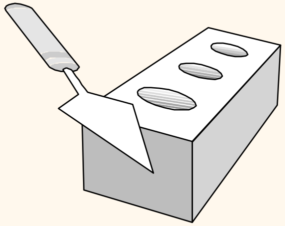


Transaction: An Execution of a DB Program

- Key concept is ***transaction***: An *atomic* sequence of database actions (e.g., reads/writes).
- Each transaction, when executed completely, must leave the DB in a ***consistent state*** if the DB is consistent before it was executed.
  - 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 semantics (e.g., how bank interest is computed).
  - Note: Ensuring that a given transaction (*if run all by itself*) preserves consistency is the user's (app's) job!

Concurrent DBMS Transactions

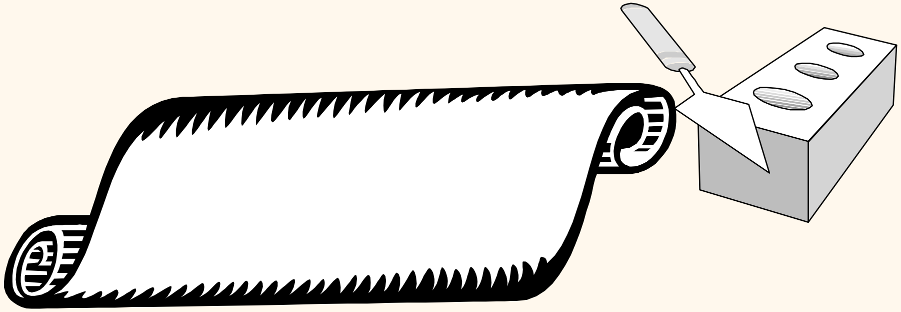
- DBMS ensures that execution of {T1, ... , Tn} is equivalent to some (in fact, any!) *serial* execution.
  - Before reading/writing an object, a transaction requests a **lock** on the object and waits till the DBMS gives it the lock. (Locks are released together at end of transaction.)
  - Key Idea: If any action of Ti (e.g., writing X) impacts Tj (e.g., reading X), one will get a lock on X first and the other will wait until the first one is done; this orders the transactions!



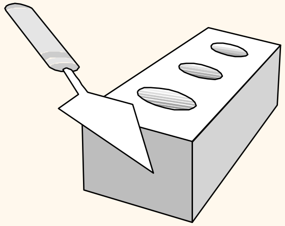
Ensuring Atomicity

- DBMS ensures *atomicity* (all-or-nothing property) even if 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 corresponding log entry 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.

The Log

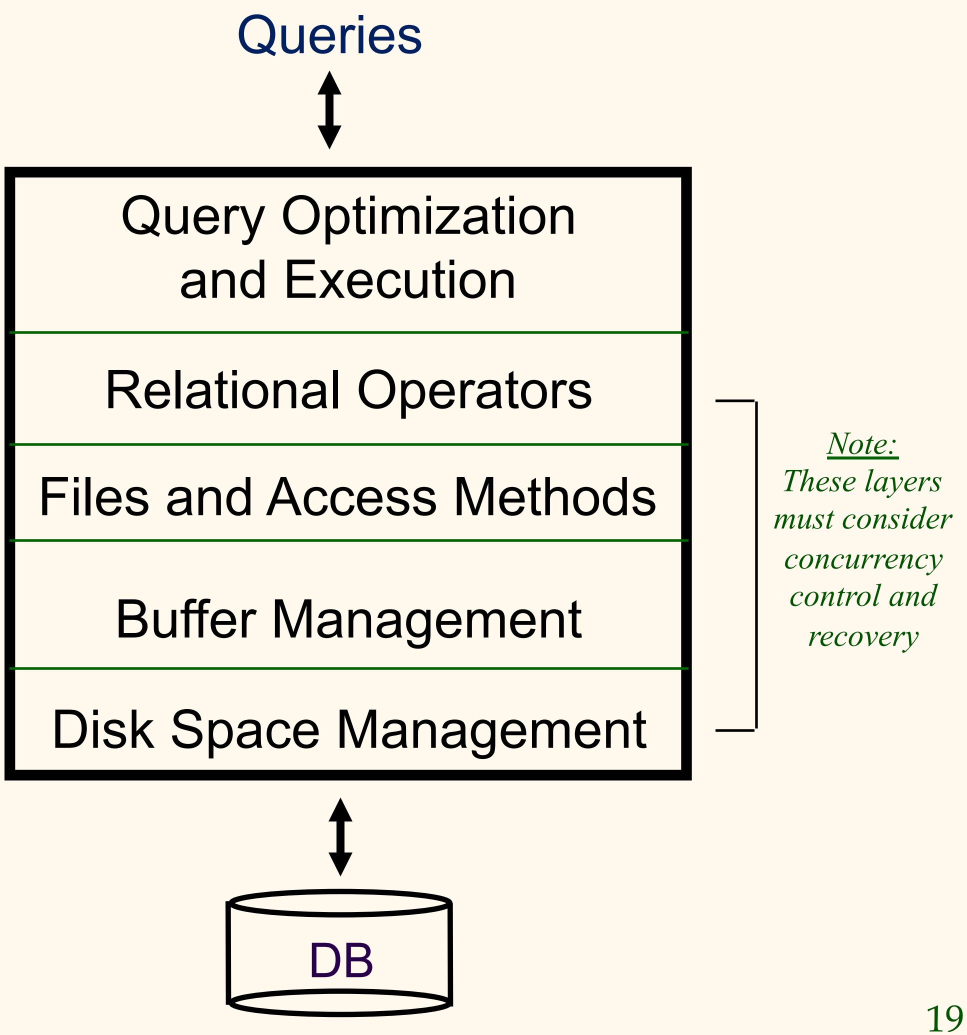


- The following actions are recorded in the log:
  - *Ti writes an object*: The old value and the new value.
    - Log record must go to disk *before* the changed DB page!
  - *Ti commits/aborts*: A log record indicating the action.
- Log records are linked by Xact id, so it's easy to undo a specific Xact (e.g., if it has to abort, or following a crash).
- Log is usually replicated on "stable" storage.
- All logging (and in fact, all the stuff we're talking about) is handled transparently by the DBMS.

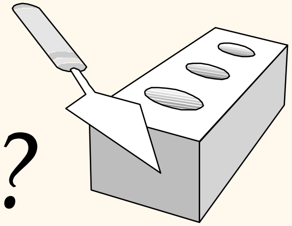


Architecture of a DBMS

- A typical DBMS has a layered architecture.
- Note: This figure doesn't show the locking and recovery components.
- This is one of several possible architectures; each actual system has its own variations.

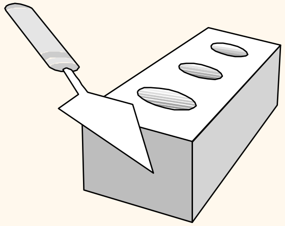


What's <u>Exciting</u> in DB Land Today?



- The Web is full of database challenges
  - Click streams and social networks generate lots of data
    - How can I query and analyze all of that data?
  - A box for keywords only goes so far…
    - How can I query the web, e.g., "Find me 5-string Fender bass guitars for sale in the $1500-2000 price range"
- Ubiquitous computing is data-rich, too
  - Build, deploy, and use location-based data services
  - Query and aggregate streams of sensor or video data
  - "Internet of things", SoLoMo (Social/Local/Mobile), …

- There's data everywhere, and of all shapes and sizes
  - How do we integrate it, *e.g.*, for rapid crisis response?
  - And when we do, how do we ensure privacy/security?



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.