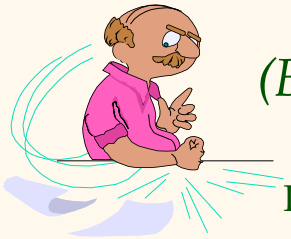
Introduction to Data Management



Lecture #2
(Big Picture, Cont.)

Instructor: Mike Carey
mjcarey@ics.uci.edu

Announcements

- Still hanging in so far? (If you might drop, please *do it soon* so others can enroll – *thanks!*)
- Today's plan:
  - More about DBs and DBMS architectures
  - Then on to logical DB design!
- Reminder:
  - Sign up on Piazza! (You're only 2/3 there...)
  - HW #1 is coming out later this week!
  - A HW turn-in plan is coming soon as well. (☺)
- Any lingering *Q*'s from last time?

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!*

Recall our University DB Example

- 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)*

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., 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 (*when run all by itself*) preserves consistency is the program's job!

Concurrent DBMS Transactions

- DBMS ensures that execution of {T_1, ... , 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. (All locks are released together, at the end of a 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!

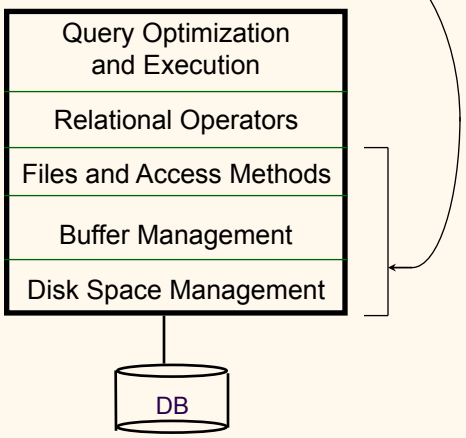
Ensuring Atomicity

- DBMS ensures *atomicity* (all-or-nothing 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.

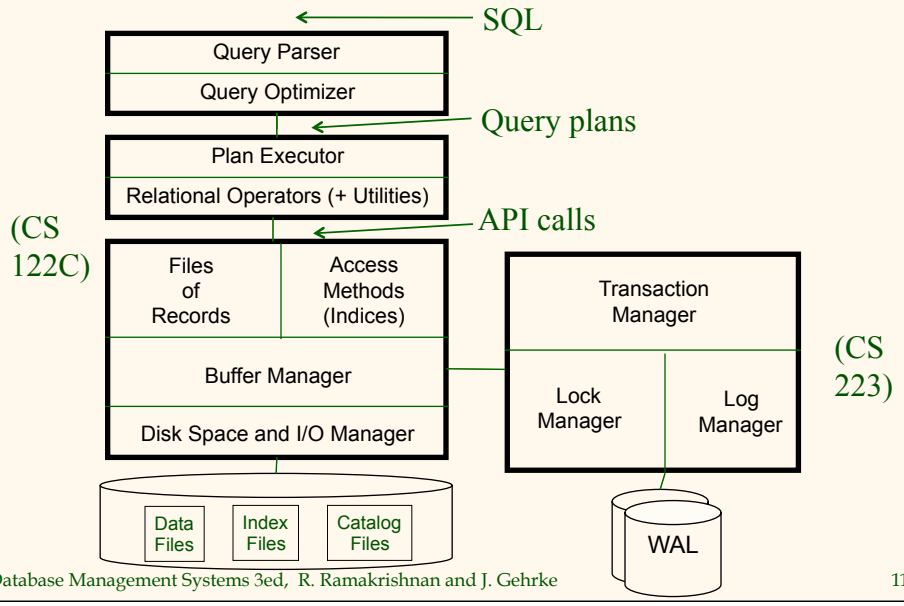
Structure of a DBMS

These layers must consider concurrency control and recovery

- 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

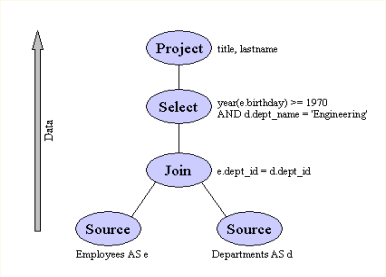
```
SELECT e.title, e.lastname
FROM Employees e, Departments d
WHERE e.dept_id = d.dept_id AND
      year (e.birthday >= 1970) AND
      d.dept_name = 'Engineering'
```

- 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…