Database Management Systems

Chapter 1

Mirek Riedewald

Many slides based on textbook slides by Ramakrishnan and Gehrke

Logistics

- Go to http://www.ccs.neu.edu/~mirek/classes/2010-F-CS3200 for all course-related information
  - Slides will be posted there as well
- Grading
  - Homework: 50%
    - Project, incl. report, and exercises
  - Midterm: 20%
  - Final exam: 30%
- TA: Yue Huang
- Office hours will be announced soon
- Can always email us with questions or to set up appointments

Project

- Work with a real DBMS: MSFT SQL Server 2008
- Work with database using SQL and Java (JDBC)
- Deliverables: code and reports
- Supported environment: Windows Lab machines with SQL Server 2008 client tools and MSFT JDBC driver
- What about working on my own machine, using Linux, MySQL, Python, C++ etc.?
  - Ok, but do it at your own risk
  - Contact me ASAP, no later than 09/15
  - We simply cannot provide support for all possible configurations

Goals for This Course

- Learn about the foundations of relational DBMS; also relevant to other fields
  - Declarative programming: specify WHAT you want, not HOW to get it
    - Set-oriented processing and query optimization
  - Data independence
  - Recovery from crashes to a consistent state
  - Programming for concurrent execution: transactions
- Be able to create, access, and manipulate a database through SQL and from an application
- Have enough background to more quickly become an expert on any DBMS
- Be better able to understand and critically evaluate features of competing data management offerings

What This Course Cannot Do

- Make you a DB admin
  - Beyond the scope of this course: requires a lot of practice and deep understanding of a specific product
    - Short-term specialized knowledge versus long-term principles
- Make you an expert on the DBMS from vendor XYZ
  - Employers can train you for their specific environment
  - This course cannot (and should not) be product specific
- Make you an SQL guru
  - Requires extensive practice (like programming in general)
  - This course will give you a good start
- Provide details about DBMS internals
  - That's a whole different course

Any Questions So Far?

What Is a DBMS?

- Database = very large, integrated collection of data.
  - Entities (e.g., students, courses)
  - Relationships (e.g., Joe is taking CS 3200)
- Database Management System (DBMS) = software package designed to store and manage databases.

Files vs. DBMS

- Special file access code for different queries
  - Find income of all young customers in a large customer file
  - Now find income of all Boston customers, where addresses are stored in a different large file
    - Two nested loops (does one data set fit in memory?) versus sort-merge implementation, or maybe create an index?
  - Once your Java program finally works, what if data layout or file size changes? Need to make significant code changes…
- Writing code for managing very large files is difficult
  - Application must stage large datasets between main memory and secondary storage (e.g., buffering, page-oriented access)
- Protect data from inconsistency due to multiple concurrent users
- Crash recovery
- Security and access control

Why Use a DBMS?

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

Why Study Databases??

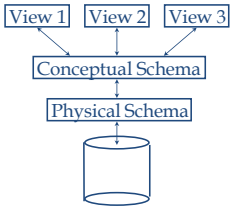
- Ubiquitous in enterprises and daily life
  - ATMs, banking, retail transactions, flight booking, customer databases
- Shift from computation to information
  - Simplify data management tasks
  - Enable efficient data processing at large scale
- Datasets increasing in diversity and volume.
  - Digital libraries, Human Genome project, Sloan Digital Sky Survey
- DBMS encompasses most of CS
  - OS, languages, theory, AI, multimedia, logic

Data Models

- Data model = collection of concepts for describing data.
- Schema = description of a particular collection of data, using a given data model.
- The relational data model is the most widely used model today.
  - Main concept: relation, basically a table with rows and columns.
  - Every relation has a schema, which describes the columns, or fields.

Levels of Abstraction

- Many views, single conceptual (logical) schema and physical schema.
  - Views describe how users see the data.
  - Conceptual schema defines logical structure
  - Physical schema describes the files and indexes used.



Example: University Database

- 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 column of Students.
- External Schema (View):
  - *Course_info(cid: string, enrollment: integer)*

Data Independence

- One of the most important benefits of using a DBMS
- Applications insulated from how data is structured and stored.
- Logical data independence: Protection from changes in *logical* structure of data.
  - If logical structure changes, create view with old structure
  - Works fine for queries, but might be tricky for updates
- Physical data independence: Protection from changes in *physical* structure of data.
  - Query and update logical structure, not physical structure

Concurrency Control

- Concurrent execution of user programs is essential for good DBMS performance.
  - Because disk accesses are frequent and relatively slow, it is important to keep the CPU humming by working on several user programs concurrently.
- Interleaving actions of different user programs can lead to inconsistency
  - E.g., check is cleared while account balance is being computed.
- DBMS ensures such problems do not arise: users and programmers can pretend they are using a single-user system.

Transaction: An Execution of a DB Program

- Transaction = *atomic* sequence of database actions (reads/writes).
- Each transaction, executed completely, must leave the DB in a consistent state if DB is consistent when the transaction begins.
  - Users can specify integrity constraints on the data, and the DBMS will enforce these constraints.
  - Beyond this, the DBMS does not really understand the semantics of the data.
    - E.g., it does not understand how the interest on a bank account is computed.
  - Thus, ensuring that a transaction (run alone) preserves consistency is ultimately the user's responsibility!

Scheduling Concurrent Transactions

- DBMS ensures that execution of {T1,..., Tn} is equivalent to some serial execution T1',..., Tn'.
  - 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 at the end of the transaction. (Strict 2PL locking protocol.)
  - Idea: If an action of Ti (say, writing X) affects Tj (which perhaps reads X), one of them, say Ti, will obtain the lock on X first and Tj is forced to wait until Ti completes; this effectively orders the transactions.
  - What if Tj already has a lock on Y and Ti later requests a lock on Y? (Deadlock!) Ti or Tj is aborted and restarted!

Ensuring Atomicity

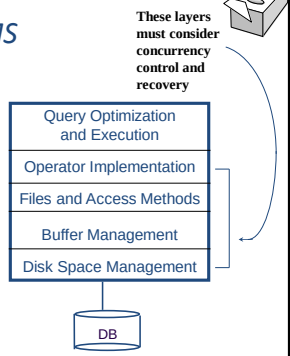
- DBMS ensures atomicity (all-or-nothing property) even if system crashes in the middle of a Xact.
- Idea: Keep a log (history) of all actions carried out by the DBMS while executing a set of Xacts:
  - Before a change is made to the database, the corresponding log entry is forced to a safe location. (WAL protocol)
  - After a crash, the effects of partially executed transactions are undone using the log. (Thanks to WAL, if log entry was not saved before the crash, corresponding change was not applied to database!)

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 page!
  - Ti commits/aborts: A log record indicating this action.
- Log records chained together by Xact id, so it's easy to undo a specific Xact (e.g., to resolve a deadlock).
- Log is often *duplexed* and *archived* on "stable" storage.
- All log related activities (and in fact, all concurrency-control related activities such as lock/unlock, dealing with deadlocks etc.) are handled transparently by the DBMS.

Structure of a DBMS

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



Databases make these folks happy

- End users and DBMS vendors
- Many enterprises
- DB application programmers
- Database administrator (DBA)
  - Designs logical/physical schemas
  - Handles security and authorization
  - Data availability, crash recovery
  - Database tuning as needs evolve

Databases And Startups

- DBMS perfect as data management system for startups
- LAMP stack: Linux OS, Apache Web server, MySQL DBMS, PHP (or Perl, Python)
- Why LAMP?
  - The price is right
  - Easy to code
    - MySQL and scripting language
  - Easy to deploy
    - Set up LAMP on laptop, build app locally, then deploy on the Web
  - Ubiquitous hosting
    - Even cheapest Web hosting options allow running PHP, MySQL

Example: eBay

- 1995—1997: GDBM (GNU library of DB functions)
- 1997—1999: Oracle (biggest DBMS vendor)
- 1999—2001: still Oracle, but now multiple servers
- 2001—present: split DBs by functionality, pull most functionality from DBMS up into application layer
- DBMS still important component
  - Initially the data management entity, scaling well…
  - …until eBay grew so much that customized solutions were needed
  - DBMS is general-purpose, and extreme challenges require more customized solutions

NoSQL Movement

- Growing popularity of non-relational data stores
  - Document stores, key-value stores, eventually consistent stores, graph DB, object-oriented DB, XML DB
- Examples: MongoDB, CouchDB, Google's BigTable, Amazon's Dynamo
- Many of them driven by performance challenges
  - Inherent tradeoff between consistency, availability, and tolerance to network partitions (Eric Brewer, UC Berkeley)
    - Maintaining consistent state across 100s of machines requires expensive agreement (communication)
    - Failures reduce availability, unless consistency is weakened (1000 machines => failures happen all the time)
- Solutions: weaker consistency guarantees or tailored solution for specific workload

MapReduce vs. DBMS

- Google's answer to scalable data processing challenges
- Programming paradigm for distributed computation on large clusters
- Two phases
  - Map: map each input record independently to a set of (key, value) pairs
  - Reduce: process set of all values with the same key together
- Less expressive than distributed DBMS, but highly popular
  - Read what two DBMS luminaries think about it and how readers reacted
    - http://databasecolumn.vertica.com/database-innovation/mapreduce-a-major-step-backwards/
    - http://databasecolumn.vertica.com/database-innovation/mapreduce-ii/
- Active research area in databases
  - High-level programming languages for MapReduce, processing DB queries in MapReduce-style system

Exciting Times

- Worldwide relational DBMS software revenue $15.2B in 2006 (source: Gartner)
  - Dominant players: Oracle, IBM, Microsoft, Teradata
- Smaller companies with specialized data management solutions
  - Vertica, Greenplum, Netezza, and many more
- Virtually every enterprise relies on DBMS
- Close relative of data warehousing
  - Crucial for business success, e.g., Wal-Mart
- Mushrooming of noSQL alternatives and parallel/distributed data management solutions
- Knowing the principles of relational DBMS is essential for understanding these trends.

Summary

- DBMS are 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 data independence.
- A DBMS typically has a layered architecture.
- DBAs hold responsible jobs and are well-paid ☺
- DBMS R&D is one of the broadest, most exciting areas in CS.