Introduction to Data Management

Lecture #2
Intro II & Data Models I

Instructor: Mike Carey
mjcarey@ics.uci.edu

Today’s Topics

- The biggest class ever continues…! 😊
- Read (and live by!) the course wiki page:
  - http://www.ics.uci.edu/~cs122a/
  - Note the new dates for the Endterm and last two HW’s (!)
- Also follow (and live by) the Piazza page:
  - https://piazza.com/uci/spring2019/cs122a/home
  - Everyone needs to get signed up! (290/420 at last glance…)
- The first HW assignment will become available at class time on Friday
  - We’ll be supporting a hypothetical personal health logging application (kind of an IoT-ish application) this term
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!
**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)**

- SQL
- Query plans
- API calls

- Query Parser
- Query Optimizer
- Plan Executor
- Relational Operators (+ Utilities)
- Files of Records
- Access Methods (Indices)
- Buffer Manager
- Disk Space and I/O Manager
- Data Files
- Index Files
- Catalog Files
- Transaction Manager
- Lock Manager
- Log Manager
- WAL

These layers must consider concurrency control and recovery.
Components’ Roles

- Query Parser
  - Parse and analyze SQL query
  - Makes sure the query is valid and talking about tables, etc., that indeed exist

- Query optimizer (usually has 2 steps)
  - Rewrite the query logically
  - Perform cost-based optimization
  - Goal is finding a “good” query plan considering
    - Available access paths (files & indexes)
    - Data statistics (if known)
    - Costs of the various relational operations

\[\text{SELECT e.title, e.lastname} \]
\[\text{FROM Employees e, Departments d} \]
\[\text{WHERE e.dept_id = d.dept_id AND} \]
\[\text{year (e.birthday >= 1970) AND} \]
\[\text{d.dept_name = 'Engineering'} \]

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**
  - DBMSs have *record* based APIs under the hood
    - 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 (the most commonly used indexes for both commercial and open source DBMSs)

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

- **Disk Space and I/O Managers**
  - Manage space on *disk* (pages)
  - Also manage I/O (sync, async, prefetch, …)
  - Remember: database data is *persistent* (!)
Components’ Roles (continued)

- System Catalog (or “Metadata”)
  - Info about tables (name, columns, column types, …);
  - Data statistics (e.g., counts, value distributions, …)
  - Info about indexes (tables, index kinds, …)
  - 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 physical DB design, parameter tuning, performance monitoring, backup/restore, user and group authorization management

- **Application Developer**
  - Builds data-centric applications (take 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
Introductory Recap

- 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 data independence.
- 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:
  - Again, be sure that you’re signed up on Piazza
  - And, stockpile sleep – no homework yet 😊
- Let’s switch gears to database design...
Overview of Database Design

- **Conceptual design**: (ER Model used at this stage.)
  - What are the *entities* and *relationships* in the enterprise?
  - What information about these entities and relationships should we store in the database?
  - What are the *integrity constraints* or *business rules* that hold?
  - A database schema in the ER Model can be represented pictorially (using an *ER diagram*).
  - Can map an ER diagram into a relational schema (manually or using a design tool’s automation).

ER Model Basics

- **Entity**: Real-world object, distinguishable from all other objects. An entity is described (in DB-land) using a set of *attributes*.
- **Entity Set**: A collection of similar entities. E.g., all employees.
  - All entities in an entity set have the *same* set of attributes. (Until we get to ISA hierarchies…)
  - Each entity set has a *key* (a unique identifier); this can be either one attribute (an “atomic” key) or several attributes (called a “composite” key)
  - Each attribute has a *domain* (similar to a data type).
**ER Model Basics (Contd.)**

- **Relationship**: Association among two or more entities. E.g., Santa Claus works in the Toy department.
- **Relationship Set**: Collection of similar relationships.
  - An n-ary relationship set R relates n entity sets E1 ... En; each relationship in R involves entities e1:E1, ..., en:En
  - One entity set can participate in different relationship sets – or in different “roles” in the same set.

**Cardinality Constraints**

- Consider Works In: An employee can work in many departments; a dept can have many employees.
- In contrast, each dept has at most one manager, according to the **cardinality constraint** on Manages above.

(Note: A given employee can manage several departments)