Introduction to Data Management

Week 1

What Is a DBMS?
❖ A very large, integrated collection of data.
❖ Models real-world enterprise.
   – Entities (e.g., students, courses)
   – Relationships (e.g., Madonna is taking CS564)
❖ A Database Management System (DBMS) is a software package designed to store and manage databases.

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??
❖ Shift from computation to information
   – at the “low end”: scramble to webspace (a mess!)
   – at the “high end”: scientific applications
❖ Datasets increasing in diversity and volume.
   – Digital libraries, interactive video, Human Genome project, EOS project
   – ... need for DBMS exploding
❖ DBMS encompasses most of CE/CS
   – OS, languages, theory, “A”I, 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 the a given data model.
❖ The relational model of data 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.

Need for Data Models
❖ Data Any representation to which meaning may be assigned. Itself does not have any meanings.
   Ex. 25, RED, 7.25
❖ Information Meaning assigned to data by known conventions.
   – John is 25 years old
   – The cat is red
❖ Knowledge (semantics) Perception of fact or truth.
   – Children have parents
   – Automobile have engines
**Need for Data Models**
- **Information precision**: Ex: I saw a man on the hill with a telescope.
- **Relation**: Grouping a chunk of knowledge of the same type so that the group has a fixed meaning. – Name, Age, Sex, Married, No. of Children

**Logical Modeling**
- **Entity-Relation Model**
- **Relational Model**
- **Hierarchical Models (Generalization to Object-Oriented Model)**
- **Network Model**

**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.

- Schemas are defined using DDL, data is modified/queried using DML.

**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**
- Applications insulated from how data is structured and stored.
- **Logical data independence**: Protection from changes in logical structure of data.
- **Physical data independence**: Protection from changes in physical structure of data.

- One of the most important benefits of using a DBMS!

**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 don’t arise: users can pretend they are using a single-user system.
**Transaction: An Execution of a DB Program**

- Key concept is *transaction*, which is an 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 some simple 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!

**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: OS support for this is often inadequate.)
  - After a crash, the effects of partially executed transactions are undone using the log. (Thanks to WAL, if log entry wasn’t saved before the crash, corresponding change was not applied to database?)

**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!

**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 CC related activities such as lock/unlock, dealing with deadlocks etc.) are handled transparently by the DBMS.

**Databases make these folks happy ...**

- End users and DBMS vendors
  - E.g. smart webmasters
- DB application programmers
  - Database administrator (DBA)
    - Designs logical/physical schemas
    - Handles security and authorization
    - Data availability, crash recovery
    - Database tuning as needs evolve
  - Must understand how a DBMS works!

**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.
Summary
❖ DBMS 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 are well-paid!
❖ DBMS R&D is one of the broadest, most exciting areas in CE/CS.

Why Study the Relational Model?
❖ Most widely used model.
   – Vendors: IBM, Informix, Microsoft, Oracle, Sybase, etc.
❖ “Legacy systems” in older models
   – e.g., IBM’s IMS
❖ Recent competitor: Object-Oriented model
   – ObjectStore, Versant, Ontos
   – a synthesis emerging: object-relational model
     • Informix Universal Server, UniSQL, O2

Relational Database: Definitions
❖ Relational database: a set of relations.
❖ Relation: made up of 2 parts:
   – Instance: a table, with rows and columns. #rows = cardinality, #fields = degree
   – Schema: specifies name of relation, plus name and type of each column.
     • E.g. S(#, SName: char, Status: integer, City: char)
     • Can think of a relation as a set of rows or tuples (i.e., all rows are distinct)

Example Instance of Supplier Relation

<table>
<thead>
<tr>
<th>S#</th>
<th>SName</th>
<th>Status</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>Smith</td>
<td>20</td>
<td>London</td>
</tr>
<tr>
<td>S2</td>
<td>Jones</td>
<td>10</td>
<td>Paris</td>
</tr>
<tr>
<td>S3</td>
<td>Blake</td>
<td>30</td>
<td>Paris</td>
</tr>
<tr>
<td>S4</td>
<td>Clark</td>
<td>20</td>
<td>London</td>
</tr>
<tr>
<td>S5</td>
<td>Adams</td>
<td>30</td>
<td>Athens</td>
</tr>
</tbody>
</table>

❖ Cardinality = 5, degree = 4, all rows distinct
❖ Do all columns in a relation instance have to be distinct?

Relational Query Languages
❖ A major strength of the relational model:
   supports simple, powerful querying of data.
❖ Queries can be written intuitively, and the DBMS is responsible for efficient evaluation.
   – The key allows precise semantics for relational queries.
   – Allows the optimizer to extensively re-order operations, and still ensure that the answer does not change.

Chapter 4: The SQL Query Language
The SQL Query Language

❖ To find all suppliers in London

Select *
FROM S
WHERE city = "London"

• To find just names and status, replace the first line:
  SELECT name, status

Semantics of Selection and Projection

Selection (WHERE)     Projection (FROM)

Querying Multiple Relations

❖ What does the following Query Computes

SELECT S.name, J.name
FROM S, J
WHERE S.city = J.city AND S.status > 20

Semantics of a Query

❖ A conceptual evaluation method for the previous query:
  1. do FROM clause: compute cross-product of S and J relations
  2. do WHERE clause: Check conditions, discard tuples that fail
  3. do SELECT clause: Delete unwanted fields
❖ Remember, this is conceptual. Actual evaluation will be much more efficient, but must produce the same answers.

Creating Relations in SQL

❖ Creates the S relation.
  Observe that the type (domain) of each field is specified, and enforced by the DBMS whenever tuples are added or modified.
❖ As another example, the P table holds information about Parts.

Destroying and Altering Relations

DROP TABLE S
❖ Destroys the relation S. The schema information and the tuples are deleted.
ALTER TABLE S
ADD COLUMN No_of_Outlets: integer
❖ The schema of S is altered by adding a new field, every tuple in the current instance is extended with a null value in the new field
**Adding and Deleting Tuples**

- Can insert a single tuple using:

  ```sql
  INSERT INTO S(s#, sname, status, city)
  VALUES ('S6', 'Mike', 15, 'Lafayette')
  ```

- Can delete all tuples satisfying some condition (e.g., name = Smith):

  ```sql
  DELETE
  FROM S
  WHERE sname = 'Smith'
  ```

- Powerful variants of these commands are available; more later!