Week 1 – Part 1: An Introduction to Database Systems

Databases and DBMSs

Data Models and Data Independence

Concurrency Control and Database Transactions

Structure of a DBMS

DBMS Languages

Databases and DBMSs

- **Database**: A very large, integrated collection of data.
- **Examples**: databases of customers, products, ...
- There are huge databases out there, for satellite and other scientific data, digitized movies, ...; up to hexabytes of data (i.e., $10^{18}$ bytes)
- A database usually models (some part of) a real-world enterprise.
  - Entities (e.g., students, courses)
  - Relationships (e.g., Paolo 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** — You don’t need to know the implementation of the database to access data; queries are optimized.
- **Reduced application development time** — Queries can be expressed declaratively, programmer doesn’t have to specify how they are evaluated.
- **Data integrity and security** — (Certain) constraints on the data are enforced automatically.
- **Uniform data administration.**
- **Concurrent access, recovery from crashes** — Many users can access/update the database at the same time without any interference.

Why Study Databases??

- **Shift from computation to information:**
  Computers were initially conceived as neat devices for doing scientific calculations; more and more they are used as data managers.
- **Datasets increasing in diversity and volume:**
  Digital libraries, interactive video, Human Genome project, EOS project
  ...
  **need for DBMS technology is exploding!**
- **DBMS technology encompasses much of Computer Science:**
  OS, languages, theory, AI, multimedia, logic,...
Data Models

• A data model is a collection of concepts for describing data.
• A database schema is a description of the data that are contained in a particular database.
• The relational model of data is the most widely used data model today.
  • Main concept: relation, basically a table with rows and columns.
  • A relation schema, describes the columns, or attributes, or fields of a relation.

Levels of Abstraction

Many views, single logical schema and physical schema.
• Views (also called external schemas) describe how users see the data.
• Logical schema* defines logical structure.
• Physical schema describes the files and indexes used.

* Called conceptual schema back in the old days.
Example: University Database

• Logical 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.

• (One) External Schema (View):
  CourseInfo(Cid:String, Enrollment:Integer)

Tables Represent Relations

<table>
<thead>
<tr>
<th>Sid</th>
<th>Name</th>
<th>Login</th>
<th>Age</th>
<th>Gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>00243</td>
<td>Paolo</td>
<td>pg</td>
<td>21</td>
<td>4.0</td>
</tr>
<tr>
<td>01786</td>
<td>Maria</td>
<td>mf</td>
<td>20</td>
<td>3.6</td>
</tr>
<tr>
<td>02699</td>
<td>Klaus</td>
<td>klaus</td>
<td>19</td>
<td>3.4</td>
</tr>
<tr>
<td>02439</td>
<td>Eric</td>
<td>eric</td>
<td>19</td>
<td>3.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cid</th>
<th>Cname</th>
<th>Credits</th>
</tr>
</thead>
<tbody>
<tr>
<td>csc340</td>
<td>Rqmts Engineering DBs</td>
<td>4</td>
</tr>
<tr>
<td>csc343</td>
<td>Databases</td>
<td>6</td>
</tr>
<tr>
<td>ece268</td>
<td>Operating Systems</td>
<td>3</td>
</tr>
<tr>
<td>csc324</td>
<td>Programming Langs</td>
<td>4</td>
</tr>
</tbody>
</table>
Data Independence

Applications insulated from how data is structured and stored: (See also 3-layer schema structure.)

- **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 database technology!*

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., cheque is cleared while account balance is being computed.
- DBMS ensures that such problems don’t arise: users can pretend they are using a single-user system.
Database Transactions

• 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!**

Scheduling Concurrent Transactions

DBMS ensures that execution of \{T_1, \ldots , T_n\} is equivalent to some **serial** execution of T_1, \ldots , T_n.
• 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 2-phase locking protocol.**)
• **Idea:** If an action of T_i (say, writing X) affects T_k (which perhaps reads X), one of them, say T_i, will obtain the lock on X first and T_k is forced to wait until T_i completes; this effectively orders the transactions.
• What if T_k already has a lock on Y and T_i later requests a lock on Y? (**Deadlock!**) T_i or T_k is **aborted** and restarted!
Ensuring Atomicity

• DBMSs ensure atomicity (all-or-nothing property), even if system crashes in the middle of a transaction.
• Idea: Keep a log (history) of all actions carried out by the DBMS while executing a set of transactions:
  † 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!)

The Log

• The following actions are recorded in the log:
  † T_i writes an object: the old value and the new value; log record must go to disk before the changed page!
  † T_i commits/aborts: a log record indicating this action.
• Log records chained together by transaction id, so it’s easy to undo a specific transaction (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 Folks Happy...

- End users and DBMS vendors
- Database application programmers, e.g. smart webmasters
- *Database administrators (DBAs)*
  - Design logical /physical schemas
  - Handle 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 variation.
Database Languages

A DBMS supports several languages and several modes of use:

- Interactive textual languages, such as SQL;
- Interactive commands embedded in a host programming language (Pascal, C, Cobol, Java, etc.)
- Interactive commands embedded in ad-hoc development languages (known as 4GL), usually with additional features (e.g., for the production of forms, menus, reports, ...)
- Form-oriented, non-textual user-friendly languages such as QBE.

SQL, an Interactive Language

```
SELECT Course, Room, Building
FROM Rooms, Courses
WHERE Code = Room
AND Floor="Ground"
```
SQL Embedded in Pascal

```pascal
write('city name'?'): readln(city);
EXEC SQL DECLARE E CURSOR FOR
    SELECT NAME, SALARY
    FROM EMPLOYEES
    WHERE CITY = :city ;
EXEC SQL OPEN E ;
EXEC SQL FETCH E INTO :name, :salary ;
while SQLCODE = 0 do begin
    write('employee:', name, 'raise?');
    readln(raise):
    EXEC SQL UPDATE PERSON SET
        SALARY=SALARY+:raise
    WHERE CURRENT OF E
    EXEC SQL FETCH E INTO :name, :salary
end;
EXEC SQL CLOSE CURSOR E
```

SQL Embedded in ad-hoc Language

(Oracle PL/SQL)

```sql
declare  Sal number;
begin
    select Sal into Salary from Emp where
    Code='5788'
    for update of Sal;
    if Salary>30M then
        update Emp set Sal=Salary*1.1 where
    Code='5788';
    else
        update Emp set Sal=Salary*1.2 where
    Code='5788';
    end if;
    commit;
exception
    when no_data_found then
        insert into Errors
        values('No employee has given
        code',sysdate);
end;
```
Form-Based Interface
(in Access)

DBMS Languages

DML — data manipulation language
DDL — data definition language
(allowing definition of database schema)
4GL — fourth generation language,
useful for declarative query processing, report generation
DBMS Technology: Pros and Cons

Pros
• Data are handled as a common resource.
• Centralized management and economy of scale.
• Availability of integrated services, reduction of redundancies and inconsistencies.
• Data independence (useful for the development and maintenance of applications).

Cons
• Costs of DBMS products (and associated tools), also of data migration.
• Difficulty in separating features and services (with potential lack of efficiency.)

Conventional Files vs Databases

<table>
<thead>
<tr>
<th>Files</th>
<th>Databases</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Advantages</strong></td>
<td><strong>Advantages</strong> — Good for data integration; allow for more flexible formats (not just records)</td>
</tr>
<tr>
<td>already exist; good for simple applications; very efficient</td>
<td><strong>Disadvantages</strong> — high cost; drawbacks in a centralized facility</td>
</tr>
<tr>
<td><strong>Disadvantages</strong></td>
<td>data duplication; hard to evolve; hard to build for complex applications</td>
</tr>
</tbody>
</table>

*The future is with databases!*
Types of DBMSs

- **Conventional** – relational, network, hierarchical, consist of records of many different record types (database looks like a collection of files)
- **Object-Oriented** – database consists of objects (and possibly associated programs); database schema consists of classes (which can be objects too).
- **Multimedia** – database can store formatted data (i.e., records) but also text, pictures,...
- **Active databases** – database includes event-condition-action rules
- **Deductive databases** – like large Prolog programs, not available commercially

The Hierarchical Data Model

Database consists of **hierarchical record structures**; a field may have as value a list of records; every record has at most one parent
The Network Data Model

A database now consists of records with pointers (links) to other records. Offers a navigational view of a database.

cycles of links are allowed

Comparing Data Models

• The oldest DBMSs were hierarchical, dating back to the mid-60s. IMS (IBM product) is the most popular among them. Many old databases are hierarchical.
• The network data model came next (early ‘70s). Views database programmer as “navigator”, chasing links (pointers, actually) around a database.
• The network model was found to be too implementation-oriented, not insulating sufficiently the programmer from implementation features of network DBMSs.
• The relational model is the most recent arrival. Relational databases are cleaner because they don’t allow links/pointers (necessarily implementation-dependent).
• Even though the relational model was proposed in 1970, it didn’t take over the database market till the 80s.
Summary

• DBMSs used to maintain and 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.