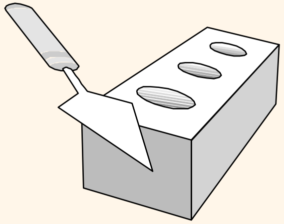
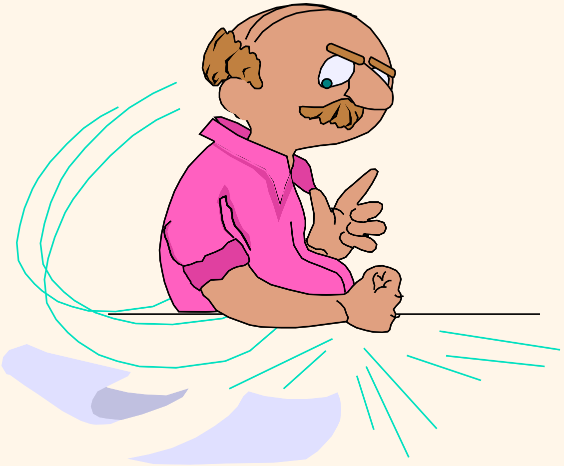


CSCI 4707: *Practice of Database Systems*



Spring 2024

Instructor:

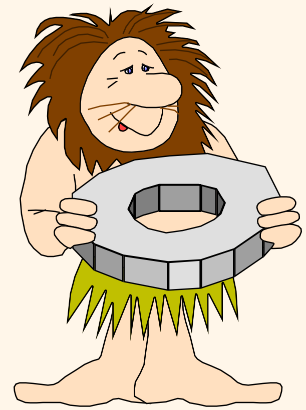
Chang Ge (cge@umn.edu)

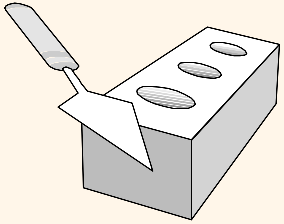
What we will study in the Course

- PART 1: *Outside* the Database Engine as a *User*
  - How to do conceptual database design (Chapter 2)
  - How to do a logical database design (Chapter 3)
  - How to query the database (Chapters 4, 5)
  - How to enhance the logical design (Chapter 19)

- PART 2: *Inside* the Database Engine
  - Storage and indexing modules (Chapters 8-10)
  - Query processing & optimization (Chapters 12-15)
  - Transaction Processing (Chapters 16-18)

What Is a DBMS?



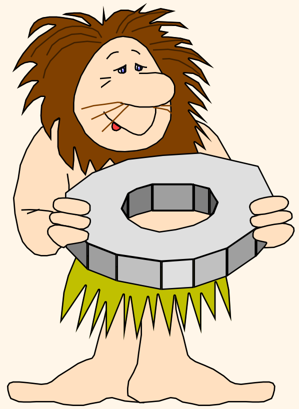


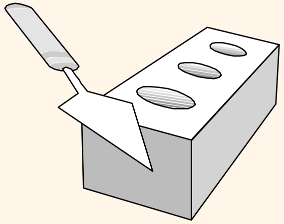
- How do you store your phone contacts:

Name	Home	Cell	Address	Email

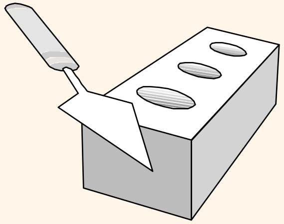
- Designing, storing, and managing such "simple" tables is the core of DBMS

What Is a DBMS?



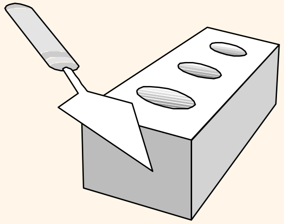


- A very large, integrated collection of data.
- Models real-world *enterprise.*
  - Entities (e.g., students, courses)
  - Relationships (e.g., Madonna is taking CSCI 1234)
- A *Database Management System (DBMS)* is a software package designed to store and manage databases.



History of DBMS

- Early 60's
  - First general-purpose DBMS by Charles Bachman at General Electric (First recipient of ACM Turing Award in databases in 1973)
- Late 60's
  - Hierarchical data model developed at IBM
  - The SABRE system for airline reservation is jointly developed by American Airlines and IBM where several people can access the same data through a network

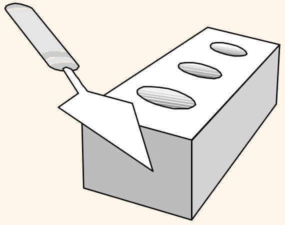


History of DBMS

- 70's
  - The relational data model is proposed by Edgar Codd at IBM (ACM Turing Award at 1981)
  - Two main prototypes for relational database management systems are developed. Ingres at UCB and System R at IBM (Mike Stonebraker received the ACM Turing Award at 2014)
  - Peter Chen (MIT) proposed the entity-relationship model
- 80's
  - SQL query language is developed (part of System R)
  - The concept of read/write transactions is developed to allow concurrent execution of database operations (Jim Gray received the ACM Turing Award at 1998)
  - Commercial databases are in the market (DB2, Oracle, Informix)

History of DBMS

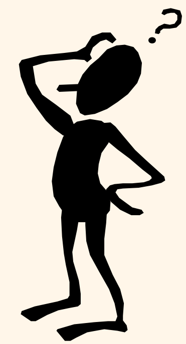
- 90’s – 10’s
  - DBMSs are well-established in industry and academia
  - More applications
  - Big data: volume, velocity, and variety
- 10’s – present
  - New DBMS architectures
  - Cloud native
  - DBMS as a service

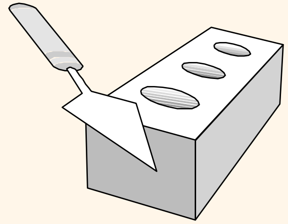


Files vs. DBMS

- Special code for different queries
- Must protect data from inconsistency due to multiple concurrent users
- Crash recovery
- Security and access control

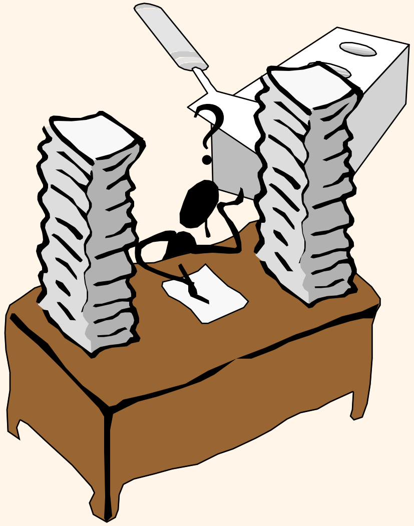
Why Use a DBMS?





- Data independence
  - Applications are independent from data representation
- Efficient data access
  - Through indexing and query optimization techniques
- Data integrity and security
  - DBMS enforce integrity constraints and access control
- Concurrent access
  - Multiples users are allowed to use the same tables
- Crash recovery
  - DBMS protects the user from the system failure
- Reduced application development time
  - DBMS supports functions that are common to many applications

Why Study Databases??



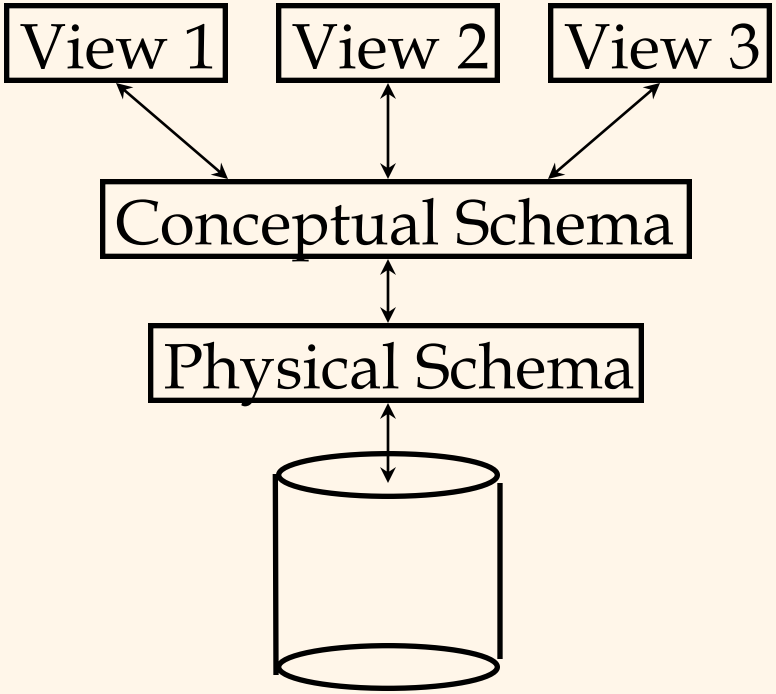
- Shift from *computation* to *information*
  - at the "low end": scramble to webspace (a mess!)
  - at the "high end": scientific applications
- Datasets increasing in diversity and volume.
  - Digital libraries, interactive video, Human Genome project, EOS project
  - ... need for DBMS exploding
- DBMS encompasses most of CS
  - OS, languages, theory, AI, networking, systems

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.

Levels of Abstraction

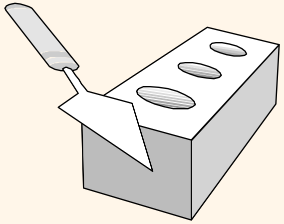
- Many *views*, single *conceptual (logical) schema* and *physical schema*.
  - Views describe how users see the data.
  - Conceptual schema defines logical structure
  - Physical schema describes the files and indexes used.



☛ *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!

Example

- Consider two transactions (*Xacts*):

```
T1:   BEGIN   A=A+100,   B=B-100   END
T2:   BEGIN   A=1.06*A,  B=1.06*B  END
```

- Intuitively, the first transaction is transferring $100 from B's account to A's account. The second is crediting both accounts with a 6% interest payment.
- There is no guarantee that T1 will execute before T2 or vice-versa, if both are submitted together. However, the net effect *must* be equivalent to these two transactions running serially in some order.

Example (Contd.)

- Consider a possible interleaving (*schedule*):

T1:	A=A+100,		B=B-100	
T2:		A=1.06*A,		B=1.06*B

- This is OK. But what about:

T1:	A=A+100,		B=B-100
T2:		A=1.06*A, B=1.06*B	

- The DBMS's view of the second schedule:

T1:	R(A), W(A),		R(B), W(B)
T2:		R(A), W(A), R(B), W(B)	

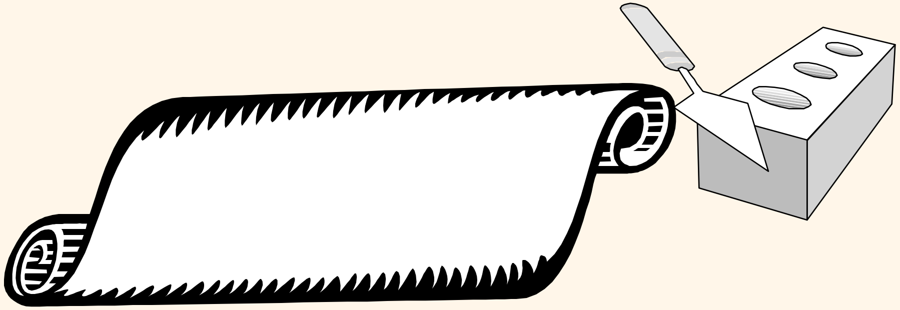
Scheduling Concurrent Transactions

- 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*; 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

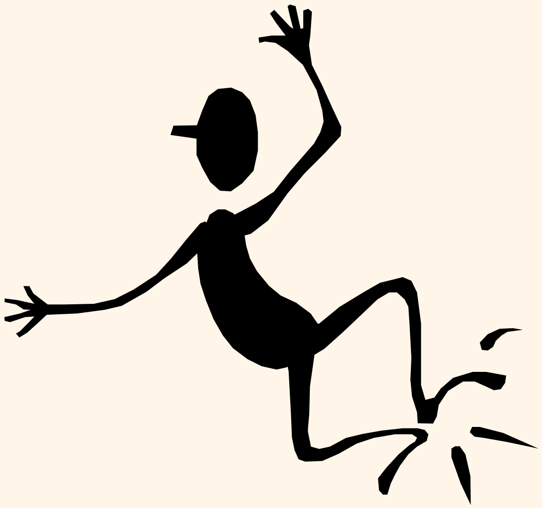


- The following actions are recorded in the log:
  - *Ti writes an object*: The old value and the new value.
    - Log record must go to disk *before* the changed page!
  - *Ti commits/aborts*: A log record indicating this action.
- Log records chained together by Xact id, so it's easy to undo a specific 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.

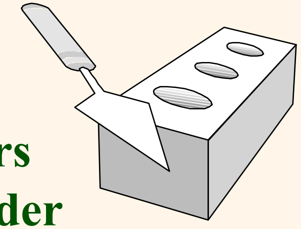
Databases make these folks happy ...

- End users and DBMS vendors
- DB application programmers
  - E.g., smart webmasters
- *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!

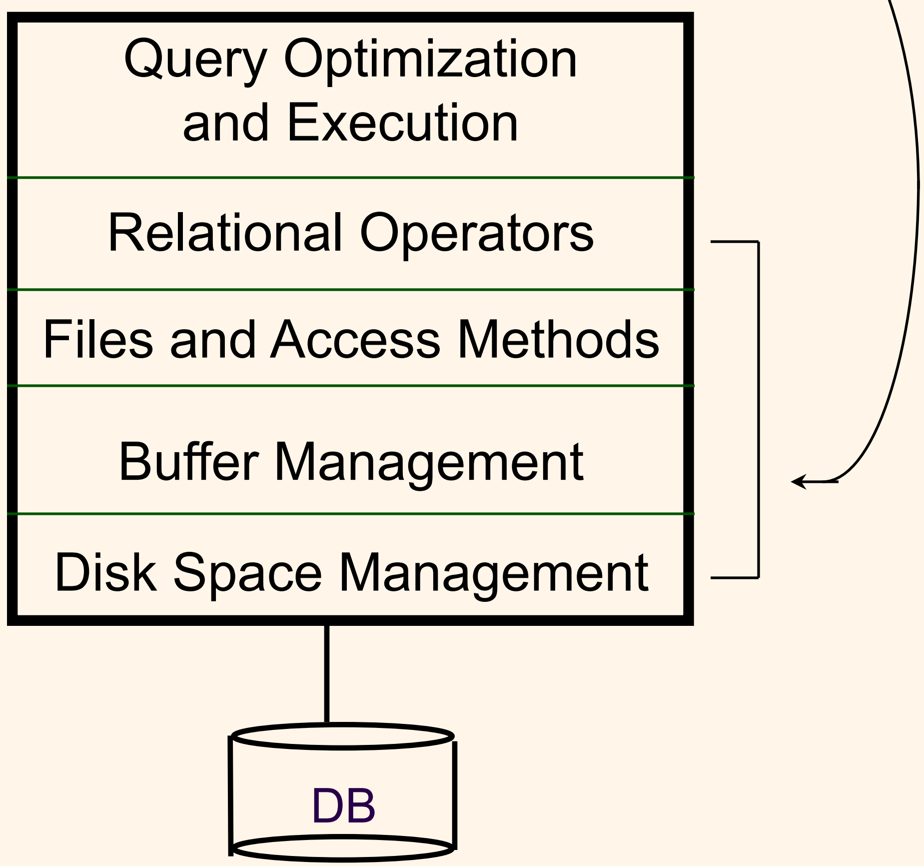


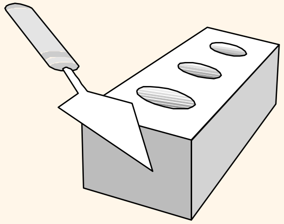
Structure of a DBMS



These layers must consider concurrency control and recovery

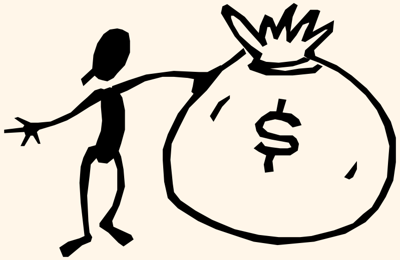
- A typical DBMS has a layered architecture.
- The figure does not show the concurrency control and recovery components.
- This is one of several possible architectures; each system has its own variations.





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 and data scientists jobs are well-paid! ☺
- DBMS R&D is one of the broadest, most exciting areas in CS.



Acknowledgement

- The slides are adapted from textbook’s instructor materials, as well as previous teachings by Prof. Mohamed Mokbel