Introduction to Database Systems

Jennie Duggan

Winter 2016

Support

- Instructor office hours: Monday 2-4pm in Ford 3.221
- TA hours
  - Peer mentors: Diane Liu, Nevil George, Thurs – 5-7pm in T-Lab
  - Grad TA: Dipendra Jha, MW - 5-7pm in Wilkinson Lab
- Discussion group
  - Nevil George, Friday – 1-2pm, location TBA
- Additional peer mentors: Katherine Lin, Shannon Nachreiner
- Use Piazza

Administrivia

- Grades:
  - 20% problem sets
  - 30% labs
  - 25% midterm
  - 25% final
- See Canvas for more details
- Textbook: Database Management Systems (R. Ramakrishnan, J. Gehrke)

Policies

- Full syllabus on Canvas
- Problem sets & programming assignments due at midnight on specified date
- Individual responsibility to keep up-to-date
  - Pro tip: Canvas is the authoritative info source
- Late days
- Don't cheat!

Why databases?

- Used to be the province of banks and retailers
- Seeing a renaissance from:
  - Web
  - Diversity of data already collected/available
  - Cheap storage!

Class Focus: Relational database management systems (RDBMSs)

Key Database Issues

- Database design and querying
- Data analysis
- Concurrency control
- Scalability and Efficiency

Background

- Ted Codd invented the relational model in the seminal paper “A Relational Model of Data for Large Shared Data Banks”
  - Main concept: *relation* = a table with rows and columns.
  - Every relation has a *schema*, which describes the columns.
- Prior 1970, no standard data model.
  - Network model used by Codasyl
  - Hierarchical model used by IMS
- After 1970, IBM built System R as proof-of-concept for relational model and used **SQL** as the query language. SQL eventually became a standard.

Example of a Traditional Database Application

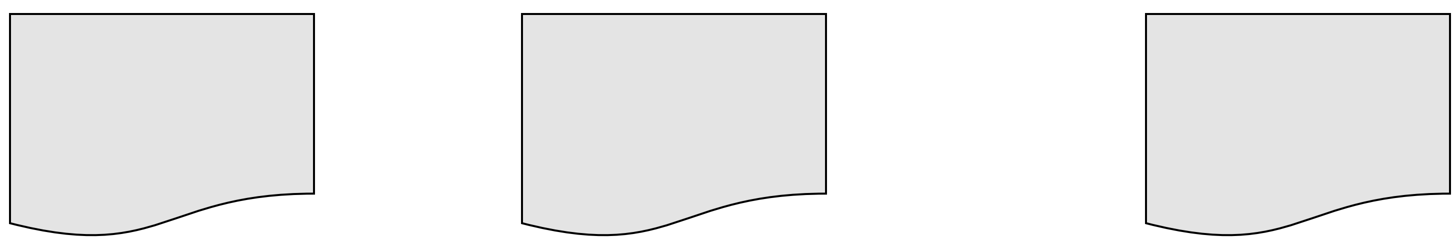
Suppose we are building a system
to store the information about:

- students
- courses
- professors
- who takes what, who teaches what

Can we do it without a DBMS ?

Sure we can! Start by storing the data in files:

students.txt courses.txt professors.txt



Now write C or Java programs to implement specific tasks

Doing it without a DBMS...

- Enroll “Mary Johnson” in “CSE444”:

Write a C program to do the following:

Read ‘students.txt’
Read ‘courses.txt’
Find&update the record “Mary Johnson”
Find&update the record “CSE444”
Write “students.txt”
Write “courses.txt”

Problems without a DBMS...

- System crashes:

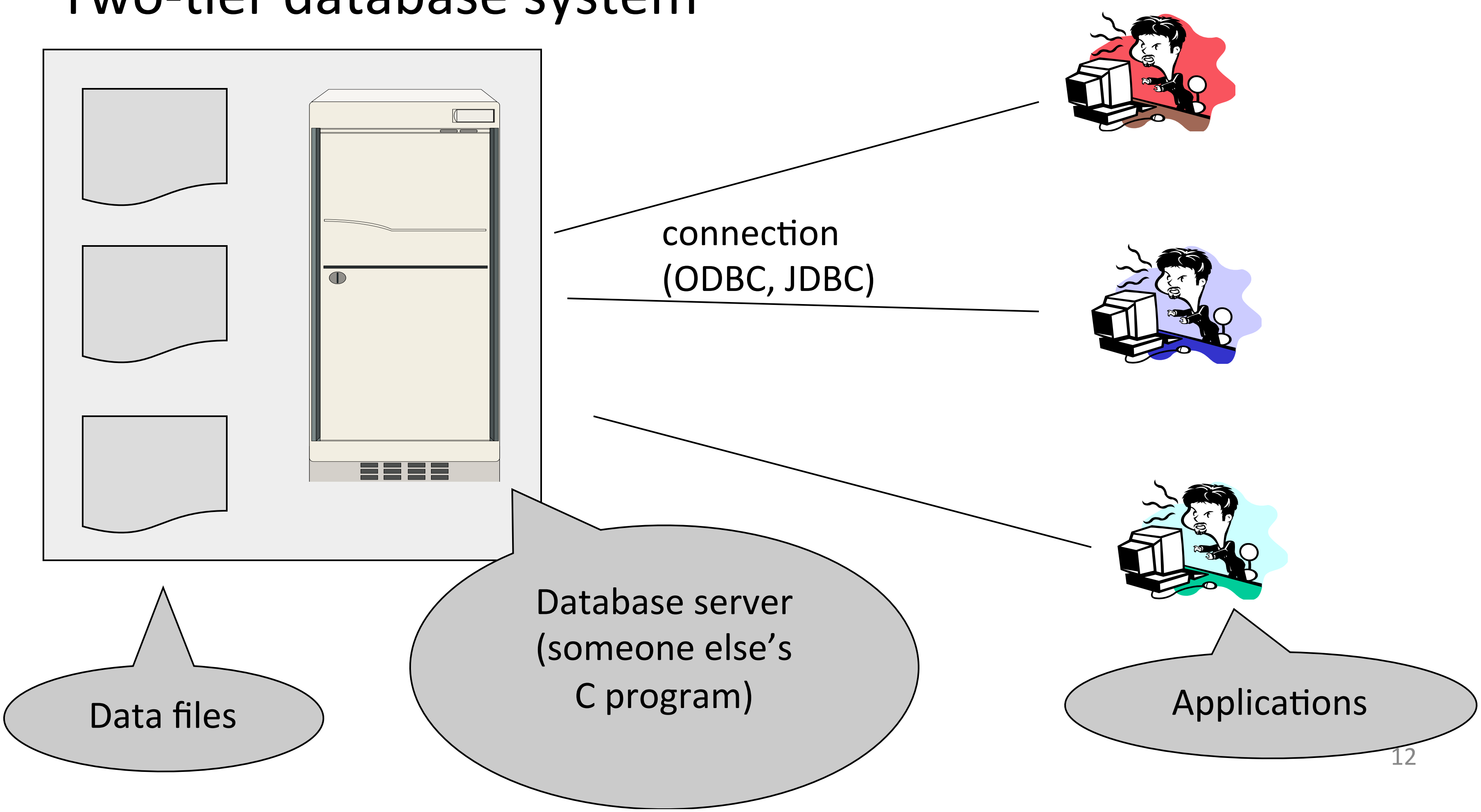
  Read 'students.txt'
  Read 'courses.txt'
  Find&update the record "Mary Johnson"
  Find&update the record "CSE444"
  Write "students.txt"
  Write "courses.txt"

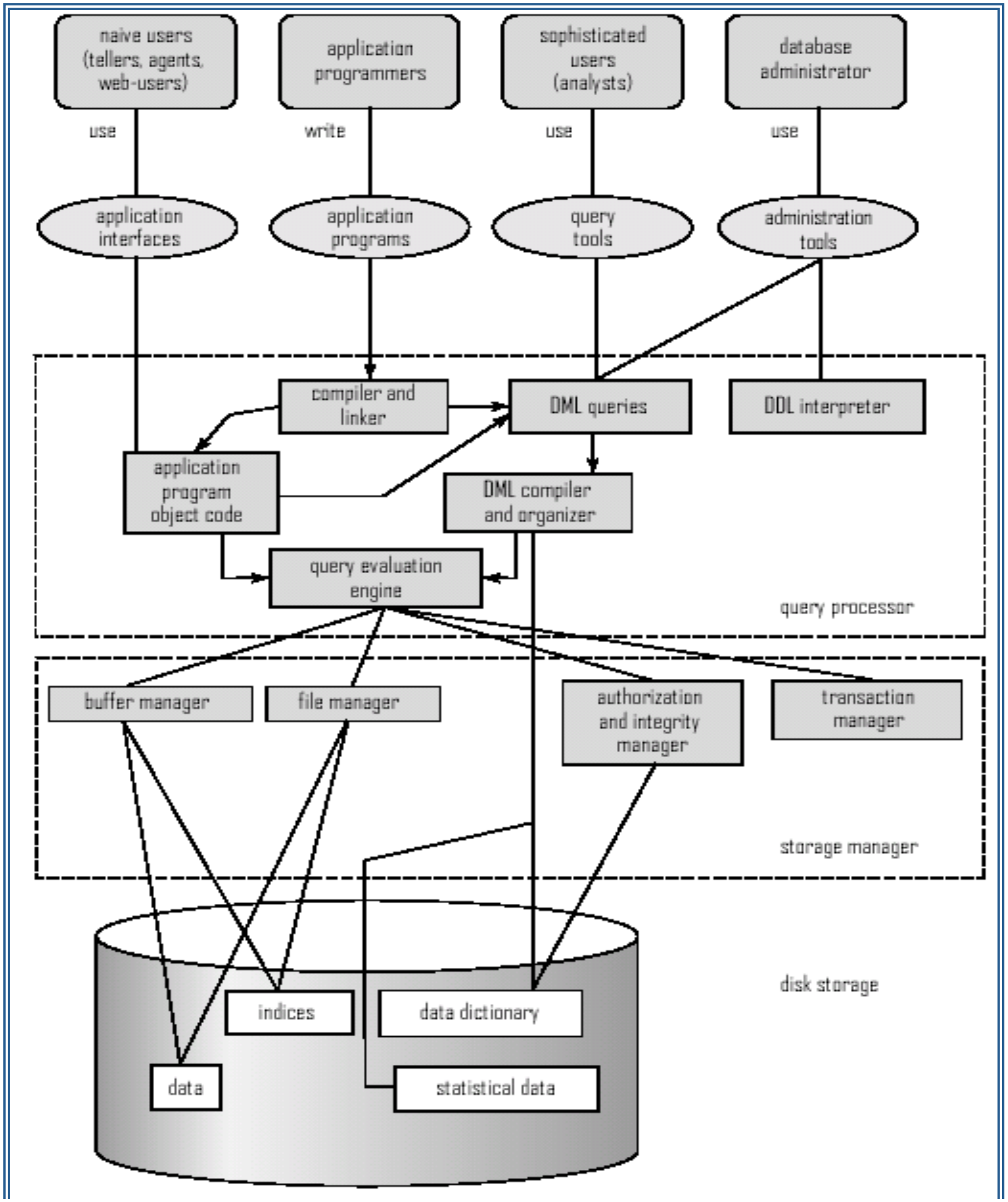
  CRASH !

  – What is the problem ?
- Large data sets (say 50TB)
  – What is the problem ?
- Simultaneous access by many users
  – Need locks: we know them from OS, but now data on disk; and is there any fun to re-implement them ?
- Data- and user-specific access control

Enter a DBMS

"Two-tier database system"




naive users (tellers, agents, web-users)
application programmers
sophisticated users (analysts)
database administrator
use
write
use
use
application interfaces
application programs
query tools
administration tools
compiler and linker
DML queries
DDL interpreter
application program object code
DML compiler and organizer
query evaluation engine
query processor
buffer manager
file manager
authorization and integrity manager
transaction manager
storage manager
indices
data dictionary
data
statistical data
disk storage

Main Advantages of DBMS

- Data independence
- Efficient querying
- Data integrity and security
- Data administration
- Concurrency and crash recovery
- Reduced developer bandwidth

Functionality of a DBMS

The programmer sees SQL, which has two components:

- Data Definition Language - DDL
- Data Manipulation Language - DML
  - query language

Behind the scenes the DBMS has:

- Query optimizer
- Query engine
- Storage management
- Transaction Management (concurrency, recovery)

Functionality of a DBMS

Two things to remember:

- Client-server architecture
  - Slow, cumbersome connection
  - But good for the data
  - But the DBMS is *general* and *convenient*
    - We can do any very specific task faster outside the DBMS

How the User Sees the DBMS

- Start with DDL to *create tables*:

```
CREATE TABLE Students (
      Name CHAR(30)
      SSN CHAR(9) PRIMARY KEY NOT NULL,
      Category CHAR(20)
)   . . .
```

- Continue with DML to *populate tables:*

```
INSERT INTO Students
VALUES('Charles', '123456789', 'undergraduate')
.  .  .  .
```

How the User Sees the DBMS

- Tables:

Students:

SSN	Name	Category
123-45-6789	Charles	undergrad
234-56-7890	Dan	grad
	…	…

Takes:

SSN	CID
123-45-6789	CSE444
123-45-6789	CSE444
234-56-7890	CSE142
	…

Courses:

CID	Name	Quarter
CSE444	Databases	fall
CSE541	Operating systems	winter

- Still implemented as files, but behind the scenes can be quite complex

  "*data independence*" = separate *logical* view from *physical implementation*

Another context…

Accounts

accountNo	*balance*	*type*
12345	1000.00	savings
67890	2846.92	checking

```
SELECT balance
FROM Accounts
WHERE accountNo = 67890;
```

```
SELECT accountNo
FROM Accounts
WHERE type = 'savings'
        AND balance < 0;
```

Key observation:

- Regardless of the context:
  - university settings
  - banking settings

The “structure” of interacting is the same…

Transactions

- Enroll "Mary Johnson" in "CSE444":

```
BEGIN TRANSACTION;

INSERT INTO Takes
   SELECT Students.SSN, Courses.CID
   FROM Students, Courses
   WHERE Students.name = 'Mary Johnson' and
              Courses.name = 'CSE444'

-- More updates here....

IF everything-went-OK
     THEN COMMIT;
ELSE ROLLBACK
```

If system crashes, the transaction is still either committed or aborted

Transactions

- A *transaction* = sequence of statements that either all succeed, or all fail
- Transactions have the ACID properties:

  A = atomicity

  C = consistency

  I = Isolation

  D = durability

Queries

- Find all courses that “Mary” takes

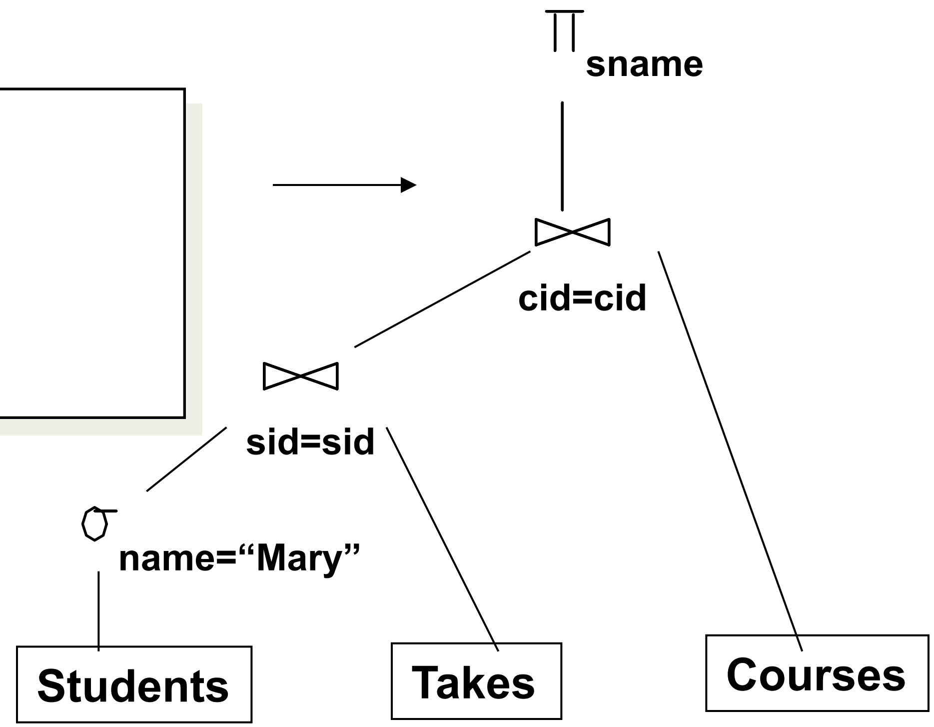
```
SELECT  C.name
FROM     Students S, Takes T, Courses C
WHERE  S.name=“Mary” and
             S.ssn = T.ssn and T.cid = C.cid
```

- What happens behind the scene ?
  - Query processor figures out how to answer the query efficiently.

Queries, behind the scene

Declarative SQL query → *Imperative query execution plan:*

```
SELECT  C.name
FROM Students S, Takes T, Courses C
WHERE S.name="Mary" and
        S.ssn = T.ssn and T.cid = C.cid
```



The **optimizer** chooses the best execution plan for a query

People Who Interact With Databases

- Application developers
- Database administrators

- Database producers

Alternative Data Models

- Hierarchical
- Object-oriented
- Object-relational
- Semistructured
- Unstructured
- Multidimensional
- Graphs
- …many more

Conclusions

- DBMSs are everywhere!
- SQL is the lingua franca of data management
  - But this is slowly changing
- Databases enable people to organize and retrieve their data in a general and consistent fashion.