Carnegie Mellon Univ.  
Dept. of Computer Science  
15-415/615 - DB Applications  

C. Faloutsos – A. Pavlo  
Lecture#1: Introduction  

Outline  
• Introduction to DBMSs  
• The Entity Relationship model  
• The Relational Model  
• SQL: the commercial query language  
• DB design: FD, 3NF, BCNF  
• indexing, q-opt  
• concurrency control & recovery  
• advanced topics (data mining, multimedia)  

We’ll learn:  
• What are RDBMS  
  – when to use them  
  – how to model data with them  
  – how to store and retrieve information  
  – how to search quickly for information  
• Internals of an RDBMS: indexing, transactions  

We’ll learn (cnt’d)  
• Advanced topics  
  – multimedia indexing (how to find similar, eg., images)  
  – data mining (how to find patterns in data)
Administrivia

- Weights: as announced

Course grade

30%  
ASGN

30%  
MT

40%  
Final exam

Sum=  
100%

5%  
ASGN1

5%  
...  
ASGN8

Administrivia - II

- FYI: ASGN3 and ASGN7 are heavy
- Late policy: 4 ‘slip days’
- Exams: no aids allowed, except
  - 1 page with your notes (both sides) for MT
  - 2 such pages for Final

Detailed outline

- Introduction
  - Motivating example
  - How do DBMSs work? DDL, DML, views.
  - Fundamental concepts
  - DBMS users
  - Overall system architecture
  - Conclusions

What is the goal of rel. DBMSs

(eg., you have 50 friends + phone#;
Or a dentist has 100 customers, addresses,
visit-info, treatment-info)
How can RDBMSs help?
What is the goal of rel. DBMSs

Electronic record-keeping:
Fast and convenient access to information.

Definitions

• ‘DBMS’ = ‘Data Base Management System’:
  the (commercial) system, like:
  DB2, Oracle, MS SQL-server, ...
• ‘Database system’: DBMS + data + application programs

Motivating example

Eg.: students, taking classes, obtaining grades;
• find my gpa
• <and other ad-hoc queries>

Obvious solution: paper-based

• advantages?
• disadvantages?
  eg., student folders, alpha sorted
Obvious solution: paper-based

- advantages?
  - cheap; easy to use
- disadvantages?
  eg., student folders, alpha sorted

Next obvious solution

- computer-based (flat) files +
- C (Java, ...) programs to access them
  e.g., one (or more) UNIX/DOS files, with student records and their courses
Next obvious solution

your layout for the student records?
(eg., comma-separated values ‘csv’
  Smith,John,123,db,A,os,B
  Tompson,Peter,234
  Atkinson,Mary,345,os,B,graphics,A

Problems?

• inconvenient access to data (need ‘C++’
  expertize, plus knowledge of file-layout)
  – data isolation
• data redundancy (and inconsistencies)
• integrity problems
• atomicity problems
Problems? (cont’d)

- ...  
- concurrent-access anomalies  
- security problems

[ why?  
because of two main reasons:  
- file-layout description is buried within the C programs and  
- Transactions: there is no support for them (concurrency and recovery)  
]

DBMSs handle exactly these two problems

Main vendors/products

Commercial
- Oracle  
- IBM/DB2  
- MS SQL-server  
- Sybase  
- (MS Access, ...)  

Open source
- Postgres (UCB)  
- mySQL/mariaDB  
- sqlite (sqlite.org)  
- miniBase (Wisc)  
- Predator (Cornell)  

(www.acm.org/sigmod)
<Demo with sqlite3>

- Insert ‘student’ and ‘takes’ records
- Find the ‘os’ class roster
- Find the GPA of ‘Smith’

www.cs.cmu.edu/~christos/courses/dbms.S15/files/sqldemo.zip

Detailed outline

- Introduction
  - Motivating example
  - How do DBMSs work? DDL, DML, views.
  - Fundamental concepts
  - DBMS users
  - Overall system architecture
  - Conclusions

How do DBs work?

Pictorially:

```
% sqlite3 miniu.sql
sqlite>create table student ( ssn fixed;
Tompson,Peter,234
Atkinson,Mary,345,os,B,graphics,A
```

How do DBs work?

```
select * from student
```

and meta-data = catalog = data dictionary

<table>
<thead>
<tr>
<th>student</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith,John, 123,db,A,os,B</td>
<td></td>
</tr>
<tr>
<td>Tompson,Peter,234</td>
<td></td>
</tr>
<tr>
<td>Atkinson,Mary,345,os,B,graphics,A</td>
<td></td>
</tr>
</tbody>
</table>
How do DBs work?

% sqlite3 miniu.sql
sqlite>create table student (  
    ssn fixed;  
    name char(20) );

<table>
<thead>
<tr>
<th>ssn</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Smith</td>
</tr>
<tr>
<td>234</td>
<td>Tompson</td>
</tr>
<tr>
<td>345</td>
<td>Atkinson</td>
</tr>
</tbody>
</table>

create table student (ssn fixed, name char(20));
insert into student values(123, "Smith");
insert into student values(234, "Tompson");
insert into student values(345, "Atkinson");

-- see what we have inserted
select * from student;

<table>
<thead>
<tr>
<th>ssn</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Smith</td>
</tr>
<tr>
<td>234</td>
<td>Tompson</td>
</tr>
<tr>
<td>345</td>
<td>Atkinson</td>
</tr>
</tbody>
</table>

How do DBs work?

sqlite>create table takes (  
    ssn fixed,  
    cid char(10),  
    grade fixed) ;

sqlite>insert into student
values (123, "Smith");
sqlite>select * from student;

<table>
<thead>
<tr>
<th>student</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Smith</td>
</tr>
</tbody>
</table>

sqlite>insert into student
values (234, "Tompson");
sqlite>select * from student;

<table>
<thead>
<tr>
<th>student</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Smith</td>
</tr>
</tbody>
</table>

sqlite>insert into student
values (345, "Atkinson");
sqlite>select * from student;

<table>
<thead>
<tr>
<th>student</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Smith</td>
</tr>
</tbody>
</table>

-- see what we have inserted
select * from student;

<table>
<thead>
<tr>
<th>ssn</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Smith</td>
</tr>
<tr>
<td>234</td>
<td>Tompson</td>
</tr>
<tr>
<td>345</td>
<td>Atkinson</td>
</tr>
</tbody>
</table>

sqlite>create table takes (  
    ssn fixed,  
    cid char(10),  
    grade fixed) ;

<table>
<thead>
<tr>
<th>takes</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
</table>

sqlite>create table takes (  
    ssn fixed,  
    cid char(10),  
    grade fixed) ;

sqlite>insert into student
values (123, "Smith");
sqlite>select * from student;

<table>
<thead>
<tr>
<th>student</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Smith</td>
</tr>
</tbody>
</table>

sqlite>insert into student
values (234, "Tompson");
sqlite>select * from student;

<table>
<thead>
<tr>
<th>student</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Smith</td>
</tr>
</tbody>
</table>

sqlite>insert into student
values (345, "Atkinson");
sqlite>select * from student;

<table>
<thead>
<tr>
<th>student</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Smith</td>
</tr>
</tbody>
</table>

-- see what we have inserted
select * from student;

<table>
<thead>
<tr>
<th>ssn</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Smith</td>
</tr>
<tr>
<td>234</td>
<td>Tompson</td>
</tr>
<tr>
<td>345</td>
<td>Atkinson</td>
</tr>
</tbody>
</table>

sqlite>create table takes (  
    ssn fixed,  
    cid char(10),  
    grade fixed) ;

<table>
<thead>
<tr>
<th>takes</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
</table>
-- register students in classes and give them grades

drop table if exists takes;
create table takes (ssn fixed, cid char(10), grade fixed);

insert into takes values( 123, "db", 4);
insert into takes values( 123, "os", 3);
insert into takes values( 345, "os", 3);
insert into takes values( 345, "graphics", 4);

Smith,John,123,db,A,os,B
Tompson,Peter,234
Atkinson,Mary,345,os,B,graphics,A

-- see what we inserted
select * from takes;

ssn     cid     grade
-------- -------- --------
123      db      4
123      os      3
345      os      3
345      graphics 4

How do DBs work - cont’d

More than one tables - joins
Eg., roster (names only) for ‘os’

<table>
<thead>
<tr>
<th>student</th>
<th>takes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssn</td>
<td>name</td>
</tr>
<tr>
<td>ssn</td>
<td>cid</td>
</tr>
</tbody>
</table>

sqlite> select name
from student, takes
where student.ssn = takes.ssn
and takes.c-id = ‘os’
-- find the os class roster

select name from student, takes
where student.ssn = takes.ssn
and cid="os";

name
-------
Smith
Atkinson

Views - a powerful tool!

what and why?

• suppose secy is allowed to see only ssn’s
  and GPAs, but not individual grades
• -> VIEWS!

Views

sqlite> create view fellowship as ( 
select ssn,  avg(grade)
from takes   group by ssn);

Views = ‘virtual tables’

takes
<table>
<thead>
<tr>
<th>ssn</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>db</td>
<td>4</td>
</tr>
<tr>
<td>123</td>
<td>os</td>
<td>3</td>
</tr>
<tr>
<td>345</td>
<td>os</td>
<td>3</td>
</tr>
<tr>
<td>345</td>
<td>graphics</td>
<td>4</td>
</tr>
</tbody>
</table>
Views

sqlite> select * from fellowship;

<table>
<thead>
<tr>
<th>takes</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssn</td>
<td>cid</td>
<td>grade</td>
</tr>
<tr>
<td>123</td>
<td>db</td>
<td>4</td>
</tr>
<tr>
<td>123</td>
<td>os</td>
<td>3</td>
</tr>
<tr>
<td>345</td>
<td>os</td>
<td>3</td>
</tr>
<tr>
<td>345</td>
<td>graphics</td>
<td>4</td>
</tr>
</tbody>
</table>

Views

sql> grant select on fellowship to secy;

('grant' not supported in sqlite)

Iterating: advantages over (flat) files

- **logical** and **physical** data independence, because data layout, security etc info: stored **explicitly** on the disk
- concurrent access and transaction support

Disadvantages over (flat) files?
Disadvantages over (flat) files

- Price
- additional expertise (SQL/DBA)

hence: over-kill for small, single-user data sets

But: mobile phones (e.g., android) use sqlite; some versions of firefox do, too: /mozilla/.../cookies.sqlite etc

Detailed outline

- Introduction
  - Motivating example
  - How do DBMSs work? DDL, DML, views.
- Fundamental concepts
  - DBMS users
  - Overall system architecture
  - Conclusions

Fundamental concepts

- 3-level architecture
- logical data independence
- physical data independence

3-level architecture

- view level
- logical level
- physical level
3-level architecture

- view level
- logical level: eg., tables
  - STUDENT(ssn, name)
  - TAKES (ssn, cid, grade)
- physical level:
  - how are these tables stored, how many bytes / attribute etc

3-level architecture

- view level, eg:
  - v1: select ssn from student
  - v2: select ssn, c-id from takes
- logical level
- physical level

3-level architecture

- view level -> ‘fellowship’
- logical level -> ‘student’ ‘takes’
- physical level -> indices, hash, …

3-level architecture

- -> hence, physical and logical data independence:
  - logical D.I.:
    - ???
  - physical D.I.:
    - ???
3-level architecture

• -> hence, **physical** and **logical** data independence:
  • logical D.I.:
    – can add (drop) column; add/drop table
  • physical D.I.:
    – can add index; change record order

Database users

• ‘naive’ users
• casual users
• application programmers
• [ DBA (Data base administrator)]

Detailed outline

• Introduction
  – Motivating example
  – How do DBMSs work? DDL, DML, views.
  – Fundamental concepts
  – DBMS users
    – Overall system architecture
    – Conclusions

Casual users

```
select * from student
```

and meta-data = catalog
``Naive'' users

Pictorially:

DBMS

data

app. (eg., report generator)

and meta-data = catalog

App. programmers

• Authors of applications (like the ‘report generator’)

DBMS

data

app. (eg., report generator)

and meta-data = catalog

DB Administrator (DBA)

• Duties?

DBMS

data

app. (eg., report generator)

and meta-data = catalog
**DB Administrator (DBA)**

- schema definition ('logical' level)
- physical schema (storage structure, access methods)
- schema modifications
- granting authorizations
- integrity constraint specification

**Detailed outline**

- Introduction
  - Motivating example
  - How do DBMSs work? DDL, DML, views.
  - Fundamental concepts
  - DBMS users
  - Overall system architecture
  - Conclusions

**Overall system architecture**

- [Users]
- DBMS
  - query processor
  - storage manager
- [Files]

![Diagram of database system architecture](image-url)
Overall system architecture

- query processor
  - DML compiler
  - embedded DML pre-compiler
  - DDL interpreter
  - Query evaluation engine

Overall system architecture (cont’d)

- storage manager
  - authorization and integrity manager
  - transaction manager
  - buffer manager
  - file manager

Overall system architecture (cont’d)

- Files
  - data files
  - data dictionary = catalog (= meta-data)
  - indices
  - statistical data

Some examples:

- DBA doing a DDL (data definition language) operation, eg.,
  create table student ...
Some examples:

- casual user, asking for an update, eg.: update student set name to 'smith' where ssn = '345'
Some examples:

• app. programmer, creating a report, eg
  main()
  {
    ....
    exec sql "select * from student"
    ...
  }

Some examples:

• ‘naive’ user, running the previous app.
Conclusions

- (relational) DBMSs: electronic record keepers
- customize them with `create table` commands
- ask SQL queries to retrieve info

Conclusions cont’d

main advantages over (flat) files & scripts:

- **logical + physical data independence** (ie., flexibility of adding new attributes, new tables and indices)
- **concurrency control** and recovery