Sam Madden

TAs: Cheng Chen, Yi Lu

http://db.csail.mit.edu/6.830/

Readings in Database Systems
Database Management Systems

Database

Structured data collection
Records
Relationships

Database management system (DBMS)

- Overview course format -- see syllabus online

Lec1 9/7/2016

- Readings
    - please do them
    - weekly questions -- come to class prepared to answer (not going to collect written answers unless needed)
    - 3 problem sets, 2 exams -- first problem set out next week
    - 4 or 6 labs (Java programming)
- Final project (6.830 only)
- Participation
- Text book -- “Readings in Database Systems” (The Red Book), also “Database Management Systems” (Ramakrishnan and Gehrke)

Redbook online (books24x7.com) (show)
RSS Feed -- linked from main page; lecture notes online, readings and suggested questions to think about for each lecture linked from schedule page

6.814 -- in the past, seniors have done well in the 6.830, and approximately 1/3 of the students have been seniors. 2 additional labs instead of the final project. You may opt to to do a final project instead of the labs.

Don’t cheat -- will check code using code similarity tools. Fine to talk to each other.

Late days.

Piazza. Stellar. Github.

what is a database?

- collection of structured data
    - typically organized as "records" (traditionally, large #, on disk)
    - and relationships between records

this class is about database management systems
(system for creating, manipulating, accessing a database)

Why should you care?
_There are lots of **applications** that we don't offer classes on at MIT.

Why are databases any different?
- Ubiquity + real world impact + software market (roughly same size as OS market)
(most web sites, most big companies)
manage both day to day ops as well as business intelligence + data mining

Core 6.830 Concepts

Data modeling / layout

Declarative Query Language,
Query Processing, Efficient
Data Access

Consistency / Transactions (“ACID”)

Today:

Why database systems?

User’s perspective:
Modeling data

Querying data

Zoo
admin interface
edit
add animal
public
pictures + maps
zookeeper
feeding

1K animals, 5K pages, 10 admins, 200 keepers

ZooFS: store each page in a text file
ZooFS Ops:
move each snake to a new bldg
custom code, consistency issues
multiple simultaneous admins
“concurrency control”
system crashes
pages in uncertain state
hungriest animal
custom code, slow

- Fundamental concepts:
    - *Data modeling & layout*
        - Systematic approach to structuring / representing data
        - Important for consistency, sharing, efficiency of access to persistent data

    - *Declarative Querying and Query Processing*
        - High level language for accessing data
            - Say what I want, not how to do it
        - "Data Independence"
        - Compiler that finds optimal plan for data access
        - Many low-level techniques for efficiently getting at data

- *Consistency / Transactions + Concurrency Control*
    - *Atomicity* -- Complex operations can be thought of as a single atomic operation that either completes or fails; partial state is never exposed
    - *Consistency and Isolation* -- Semantics of concurrent operations are well defined -- equivalent to a serial execution, respecting invariants over time
    - Durability -- Completed operations persist after a failure

Makes programming applications MUCH easier, since you don’t have to reason about arbitrary interleavings of concurrent code, and you know that the database will always be in a consistent state

- *Distributed data processing*

- A bit of many fields: systems, algorithms and data structures, languages + language design, more recently AI + learning, distributed systems….

- This course will look in detail at these areas, as well as a number of papers current in DBMS research, e.g., streaming, large scale data processing (Spark etc).

Importance of these concepts far exceeds the specific artifact that most of us would call a database (SQL, transactions, etc) — we will learn about those artifacts and also the “big ideas” (mapd demo)

suppose i am creating a web site that stores information about a zoo. has :
- admin interface that allows me to add new animals, edit animals
- public interface that allows me to look at pictures and maps
- zookeeper interface to find the animals that need to be fed

Modeling

Features to capture

How to (logically) represent data

Features:

Animals: name, age species, cage

Cages: feedtime, bldgs

Data Model: logical structures used for data

Tabular:

animals

name	age	species	cageno
peter	13	giraffe	1
sam	3	salam	2
sally	1	student	1

cages

no	feedtime	bdlg
1	1:30	1
2	2:30	2

schema: tables, fields, names, types

keeps

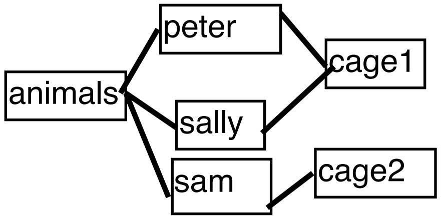
keeper	cage
1	1
1	2
2	1

keepers

keeper	name
1	jenny
2	joe

cage 1
mike
giraffe
13
sally
student
1

cage 2...



sam isa salam
sam livesin cage2
sally isa student...

Logical vs physical

Operations

- suppose move all the snakes to a new building
  - database => queries

- suppose multiple admins try to edit the same page at the same time
  - need some kind of locking

  database => ("concurrency control")

- suppose the system crashes mid-update
  - pages might be in uncertain states

  database provides

  transactions + recovery

  groups of actions that happen atomically -- "all or nothing"

- suppose i want to find the animal that was fed the longest ago
  - have to write a complex program
  - could be very slow if it has to read and search all of the pages

long history of file system research that tries to fix these issues

database: simple high level program compiled into efficient plan

Databases address all of these issues.

Lets look at how data might be structured in database

What features of our zoo do we want to capture? ("Entity Relationship Diagram" — see slide)

each animal has a name, age, species, and is in a cage

each cage gets fed at a particular time, is in a particular building

each cage is kept by many keepers

each keeper keeps many cages

each animal is in one cage, each cage has many animals

"data model" --> "schema"

Relational data model -- tables that represent entities and their properties

(Show slide)

Translates into tables by taking all of the one-to-one relations and putting them in table named for object.

Many to many relationships require an intermediate mapping table

(BREAK)

what else? hierarchy (json)   network   triplets

6.830: Relational Model

Many possible representations of a given data set

name	age	species	cageno	feedtime	bldg
peter	13	giraffe	1	1:30	1
sam	3	salam	2	2:30	2
sally	1	student .	1	1:30	1

“Normalization”

User’s perspective: Querying

1

“names of giraffes”

```
          for each row r in animals
                    if r.type = giraffes
                              output r.name
```

“selection query”

```
          SELECT r.name FROM animals
          WHERE r.type = giraffes
```

2

caged in bldg 32

```
for each row r1 in animals
          for each row r2 in cages
            if r1.bldg = r2.no and r2.bldg = 32
                              output r1
```

join operator (join)

SQL:

```
SELECT r FROM animals AS r1, cages AS r2
WHERE r1.bldg = r2.no AND r2.bldg = 32
```

avg bear age

```
SELECT AVG(age)  FROM animals
WHERE type = 'bear'
```

3

INSERT, DELETE, UPDATE

Why might I prefer one representation over the other? Are they equivalent?
Think about writing a program that manipulates these structures
Think about expressing certain complex relationships in some of these models?

This logical representation is different from the physical representation -- e.g., the layout in memory or on disk -- is different than the logical representation the programs and users see.

E.g., can represent a hierarchy via an XML file. Can represent a graph as a C struct with pointers to related nodes. Can represent a table as row-wise files of bytes, or as a linked list, or as a tree.

What is the advantage a logical/physical separation? Disadvantages?
Suppose we are using tables? Which logical representation is best? Why? Which physical representation is best?

Mostly, in this class, we will talk about the tabular -- **relational** -- approach.

why is it called relational?
  because each record is a relation between fields (“keys” capture relations)

note that there are many possible relations for a given set of data
  (example with joined column)

rules for choosing the best set of relations for a given data set
  "schema normalization"

For now, we’ll use a physical representation similar to the logical representation -- e.g., rows in a file.

what kind of operations might i want to perform on a relation? (see slides)

find the names of animals that are giraffes. (1)

find the animals in a cage in bldg 32. (you guys) -- “join” (2)

find the average age of the bears. (3)

insert an a new snake named bill INSERT
delete barney DELETE
move the snakes to a new cage UPDATE

Under the covers -- Declarative queries:
multiple procedural plans

sorted animals on type => binary search

+ search performance
- update performance

indices: map from (value) -> (record list)

declarative query -> unoptimized procedural plan ->
optimized plan -> compiled program

```
              select (bldg = 2)  (binary search?)
                         |
                   join (r1.cage = r1.no)
                     /     \
                    /       \
                    |        |
                    |        |
                 animals    cages
           r1                  r2
```

```
             join (r1.cage = r1.no)  (nested loops)
               /     \
              /       \
              |        select (bldg = 2)  (binary search)
              |           |
      animals            cages
        r1                  r2
```

Database systems provide
efficient access and updating
recoverability
consistency

Relational Model + Schema Design
Declarative Queries
Query Optimization

Declarative:
Notice, however, that our procedural programs are not the only way to compute the answers to these queries!

<u>When could I do something besides the procedural programs shown above:</u>

For example, if we store animals in animal type order, we can use binary search to find the animals of a particular type quickly.

<u>Is there a cost to doing this?</u>

Have to store in sorted order (more expensive inserts)

Lots of other possibilities -- e.g., can have hash table (index) that maps from type -> records

Declarative query -> unoptimized plan -> optimized plan -> physical plan

<u>Query optimization</u> -- Depending on physical representation of data, and type of query, DBMS selects what it believes to be the *best* plan. Uses a *cost model* to estimate how long different plans will take to run.

Optimization selects which implementation of each operation to use, as well as order of individual operations -- e.g., can move selection below join.

In *declarative* programming, the physical representation -- e.g., the layout in memory or on disk -- is different than the logical representation the user's programs interact with. Optimizer's job is to implement the logical query effectively on physical representation.

in standard *imperative* programming, logical and physical representation are typically more closely aligned.

E.g. can store the table in sorted order, or not. Repr is not exposed in SQL, or app!

Decoupling of logical model from physical representation is known as “data independence”
Can store the data in different ways on disk, don't have to change
program

PS1 -- learn SQL; due 1 week from next Tuesday, will release early next week