

Sam Madden
Robert Morris
Nizameddin Ordulu (office hours Monday
Adam Seering 11-12, Tues 4-5)

<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

9/10/09

- Readings

- please do them
- weekly questions -- come to class prepared to answer (not going to collect written answers unless needed)
- 4 problem sets, 2 exams
- 3 or 5 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)

Sign up sheet

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 approximate 1/3 of the students have been seniors

This year : 6.814 to satisfy the Advanced Undergraduate Subject requirement. 2 additional labs instead of the final project. You may opt to do a final project instead of the labs.

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

Declarative Query Language,
Query Processing

Transactions

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

- Fundamental concepts:

- *Data models*

- Systematic approach to structuring / representing data
- Important for consistency, sharing, efficiency of access

- *Declarative Querying and Query Processing*

- High level language for accessing data
- "Data Independence"
- Optimization Techniques for efficiently accessing data

- *Transactions*

- related actions that we want to succeed or fail together -- "atomicity"
- recoverability (after a crash, what is supposed to be on disk is on disk)
- consistency (no partial related actions on disk)

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

- This course will look in detail at first three areas, as well as a number of papers current in DBMS research, e.g., streaming, large scale data processing.

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

why not just use a file system? what does a database give the developer?

1,000 animals, 5,000 pages, 10 admins, 200 zookeepers, 10,000 hits per day
why not just create a separate set of pages for each animal, store it in FS
(one page for zookeepers, one page for public)

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

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

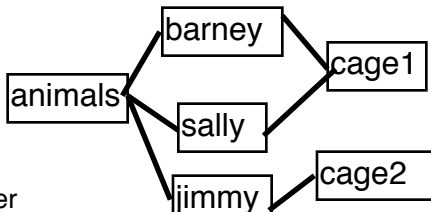
name	age	species	cageno
barney	13	bear	1
jimmy	3	antelope	2
sally	1	salamander	1

cages

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

schema: tables, fields, names, types

cage 1
barney
bear
13
sally
salamander
1



cage 2...
barney isa bear
barney livesin cage1
sally isa salamander...

Logical vs physi-

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

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?

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

“data model”

“schema”

tables:

what else?

hierarchy

network

triplets

6.830: Relational Model

Many possible representations of a given data set

name	age	species	cageno	feedtime	bldg
barney	13	bear	1	1:30	1
jimmy	3	antelope	2	2:30	2
sally	1	salama.	1	1:30	1

“Normalization”

User’s perspective: Querying
“names of snakes”

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

“selection query”

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

1

caged in bldg 2

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
```

2

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?

find the names of animals that are snakes. (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

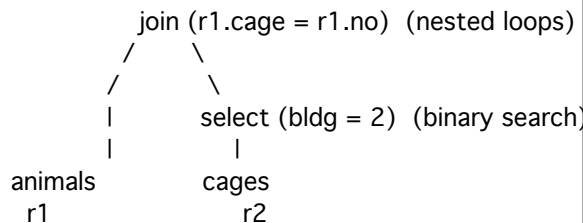
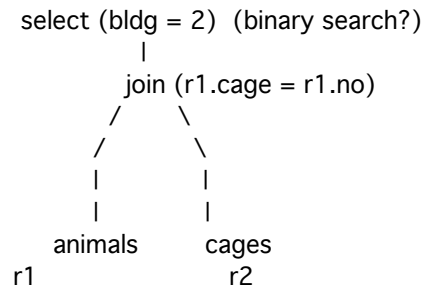
Under the covers -- Declarative queries:
multiple procedural plans

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



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!

When could I do something besides the procedural programs shown above:

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

Is there a cost to doing this?

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

PS1 -- learn SQL; due 1 week from next Tuesday, will post over the weekend