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
- **Fundamental concepts:**
  - **Data models**
    - 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.

- 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)
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:

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
<th>species</th>
<th>cageno</th>
</tr>
</thead>
<tbody>
<tr>
<td>mike</td>
<td>13</td>
<td>giraffe</td>
<td>1</td>
</tr>
<tr>
<td>sam</td>
<td>3</td>
<td>salam</td>
<td>2</td>
</tr>
<tr>
<td>sally</td>
<td>1</td>
<td>student</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>no</th>
<th>feedtime</th>
<th>bdlg</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1:30</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2:30</td>
<td>2</td>
</tr>
</tbody>
</table>

keeps

<table>
<thead>
<tr>
<th>keeper</th>
<th>cage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

keepers

<table>
<thead>
<tr>
<th>keeper</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>jenny</td>
</tr>
<tr>
<td>2</td>
<td>joe</td>
</tr>
</tbody>
</table>

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.

Let's 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

what else? hierarchy (json) network triplets
(break)
6.830: Relational Model

Many possible representations of a given data set

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
<th>species</th>
<th>cageno</th>
<th>feedtime</th>
<th>bldg</th>
</tr>
</thead>
<tbody>
<tr>
<td>mike</td>
<td>13</td>
<td>giraffe</td>
<td>1</td>
<td>1:30</td>
<td>1</td>
</tr>
<tr>
<td>sam</td>
<td>3</td>
<td>salam</td>
<td>2</td>
<td>2:30</td>
<td>2</td>
</tr>
<tr>
<td>sally</td>
<td>1</td>
<td>student</td>
<td>1</td>
<td>1:30</td>
<td>1</td>
</tr>
</tbody>
</table>

“Normalization”

User’s perspective: Querying
“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

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 queries:
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

Query optimization -- 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 early next week