Class announcements

- **Database tutorial**
  - Monday, Nov 7th, 7-8:30pm in Classroom Unit 1
  - Tuesday, Nov 8th, 8-9:30pm in Baskin Rm 152

- **Assignment 4 due today!!!!**

- **Reading for Thursday (nov 10th):**
  - Messerschmitt Ch 18 (493-512)

- **Student Presentations Thursday (nov 10th)**
  - Michela To
  - Arnaud Lawson
Student Presentations

- Katie Colburn
Locating things

by

David G. Messerschmitt
Three ways of locating things

Name
- “Joe Bloe”

Address
- “1299 Hearst St, Berkeley, CA”

Reference
- “Postmaster of Berkeley CA”
Name

- Symbolic (character string) representation

- Easy for people to remember or guess

- Identifies, but

  - *Does not locate* directly
    - Distinction important for mobile entities

  - *Not unique:* entities can have more than one name (called aliases)
Hierarchical names

Hierarchy makes names easier to remember or guess

Host domain names:
- “info.sims.berkeley.edu”
- designates administrative hierarchy

File names:
- “c:\My Documents\Docs\Resume.doc”
- designates folder hierarchy
Address

- Route or path to entity
  - is directly specified, or
  - can be inferred

- Independent of who or what is locating entity

- Topological specification
Path from blue to green is (R,D,D,D,R,R,R,R)

Is (R,D,D,D,R,R,R,R) an address?
No! -- not an address, because it depends on starting point
Example

Address of is (6,5)
Route from can be inferred
Reference

Abstract representation of an entity

Interaction is with representation

- infrastructure arranges redirection to actual entity
- especially appropriate for things that move

Example

- A Cell phone number is a *reference*.
- A Wired phone number is an *address*.
Name services

1. name
2. address or reference
3. interaction
Databases

by

David G. Messerschmitt
Databases

Treat data as a separate asset
- May be shared by multiple applications

Provide protection and integrity features appropriate to mission-critical data
- Access control
- Integrity constraints
-Persistence
- etc.
Two capabilities

**Aggregation**: accessing multiple databases

**Sharing**: two or more applications accessing the same databases
# Relational table

<table>
<thead>
<tr>
<th>Table</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Employee</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Address</td>
<td>Dept</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Record</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Field*
SQL interface

- **SQL (Structured Query Language)**
- Presents single abstract interface to the application logic
  - For manipulating, and extracting data from database
- Standardized, not vendor specific

- Encapsulates various internal details
  - Data partitioning and replication
  - Host mapping
  - File representation
  - etc.
Database operations

Each operation results in a new table
Multiple tables
### Database Operations

#### Passengers

<table>
<thead>
<tr>
<th>Name</th>
<th>Dept ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>1</td>
</tr>
<tr>
<td>Bob</td>
<td>1</td>
</tr>
<tr>
<td>Chris</td>
<td>2</td>
</tr>
</tbody>
</table>

#### Departments

<table>
<thead>
<tr>
<th>Dept Name</th>
<th>Dept ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Engineering</td>
<td>1</td>
</tr>
<tr>
<td>Sales</td>
<td>2</td>
</tr>
</tbody>
</table>

#### JOIN

<table>
<thead>
<tr>
<th>Name</th>
<th>Dept ID</th>
<th>Dept Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>1</td>
<td>Engineering</td>
</tr>
<tr>
<td>Bob</td>
<td>1</td>
<td>Engineering</td>
</tr>
<tr>
<td>Chris</td>
<td>2</td>
<td>Sales</td>
</tr>
</tbody>
</table>
Record at a time processing

Application logic (system programming language) 

SQL

Database
<table>
<thead>
<tr>
<th>Year</th>
<th>City</th>
<th>Accommodation</th>
<th>Tourists</th>
</tr>
</thead>
<tbody>
<tr>
<td>2002</td>
<td>Oakley</td>
<td>Bed&amp;Breakfast</td>
<td>14</td>
</tr>
<tr>
<td>2002</td>
<td>Oakley</td>
<td>Resort</td>
<td>190</td>
</tr>
<tr>
<td>2002</td>
<td>Oakland</td>
<td>Bed&amp;Breakfast</td>
<td>340</td>
</tr>
<tr>
<td>2002</td>
<td>Oakland</td>
<td>Resort</td>
<td>230</td>
</tr>
<tr>
<td>2002</td>
<td>Berkeley</td>
<td>Camping</td>
<td>120000</td>
</tr>
<tr>
<td>2002</td>
<td>Berkeley</td>
<td>Bed&amp;Breakfast</td>
<td>3450</td>
</tr>
<tr>
<td>2002</td>
<td>Berkeley</td>
<td>Resort</td>
<td>390800</td>
</tr>
<tr>
<td>2002</td>
<td>Albany</td>
<td>Camping</td>
<td>8790</td>
</tr>
<tr>
<td>2002</td>
<td>Albany</td>
<td>Bed&amp;Breakfast</td>
<td>3240</td>
</tr>
<tr>
<td>2003</td>
<td>Oakley</td>
<td>Bed&amp;Breakfast</td>
<td>55</td>
</tr>
<tr>
<td>2003</td>
<td>Oakley</td>
<td>Resort</td>
<td>320</td>
</tr>
<tr>
<td>2003</td>
<td>Oakland</td>
<td>Bed&amp;Breakfast</td>
<td>280</td>
</tr>
<tr>
<td>2003</td>
<td>Oakland</td>
<td>Resort</td>
<td>210</td>
</tr>
<tr>
<td>2003</td>
<td>Berkeley</td>
<td>Camping</td>
<td>115800</td>
</tr>
<tr>
<td>2003</td>
<td>Berkeley</td>
<td>Bed&amp;Breakfast</td>
<td>4560</td>
</tr>
<tr>
<td>2003</td>
<td>Berkeley</td>
<td>Resort</td>
<td>419000</td>
</tr>
<tr>
<td>2003</td>
<td>Albany</td>
<td>Camping</td>
<td>7650</td>
</tr>
<tr>
<td>2003</td>
<td>Albany</td>
<td>Bed&amp;Breakfast</td>
<td>6750</td>
</tr>
</tbody>
</table>

- Entries are simple data types or compositions of those types
  - Integer, string, etc.
Object/table correspondence

Class implementation

Record-at-a-time program

Class

Employee

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Dept</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Object instance data

Attribute
<table>
<thead>
<tr>
<th>Year</th>
<th>City</th>
<th>Accommodation</th>
<th>Tourists</th>
</tr>
</thead>
<tbody>
<tr>
<td>2002</td>
<td>Oakley</td>
<td>Bed&amp;Breakfast</td>
<td>14</td>
</tr>
<tr>
<td>2002</td>
<td>Oakley</td>
<td>Resort</td>
<td>190</td>
</tr>
<tr>
<td>2002</td>
<td>Oakland</td>
<td>Bed&amp;Breakfast</td>
<td>340</td>
</tr>
<tr>
<td>2002</td>
<td>Oakland</td>
<td>Resort</td>
<td>230</td>
</tr>
<tr>
<td>2002</td>
<td>Berkeley</td>
<td>Camping</td>
<td>120000</td>
</tr>
<tr>
<td>2002</td>
<td>Berkeley</td>
<td>Bed&amp;Breakfast</td>
<td>3450</td>
</tr>
<tr>
<td>2002</td>
<td>Berkeley</td>
<td>Resort</td>
<td>390800</td>
</tr>
<tr>
<td>2002</td>
<td>Albany</td>
<td>Camping</td>
<td>8790</td>
</tr>
<tr>
<td>2002</td>
<td>Albany</td>
<td>Bed&amp;Breakfast</td>
<td>3240</td>
</tr>
<tr>
<td>2003</td>
<td>Oakley</td>
<td>Bed&amp;Breakfast</td>
<td>55</td>
</tr>
<tr>
<td>2003</td>
<td>Oakley</td>
<td>Resort</td>
<td>320</td>
</tr>
<tr>
<td>2003</td>
<td>Oakland</td>
<td>Bed&amp;Breakfast</td>
<td>280</td>
</tr>
<tr>
<td>2003</td>
<td>Oakland</td>
<td>Resort</td>
<td>210</td>
</tr>
<tr>
<td>2003</td>
<td>Berkeley</td>
<td>Camping</td>
<td>115800</td>
</tr>
<tr>
<td>2003</td>
<td>Berkeley</td>
<td>Bed&amp;Breakfast</td>
<td>4560</td>
</tr>
<tr>
<td>2003</td>
<td>Berkeley</td>
<td>Resort</td>
<td>419000</td>
</tr>
<tr>
<td>2003</td>
<td>Albany</td>
<td>Camping</td>
<td>7650</td>
</tr>
<tr>
<td>2003</td>
<td>Albany</td>
<td>Bed&amp;Breakfast</td>
<td>6750</td>
</tr>
</tbody>
</table>

- Rows can be considered object instances with the same attributes
- Restriction to simple data types
- No encapsulated state
- No methods
Object-relational database

• A column can store object instances of a given class rather than data of a given simple or compound data type

• Because of the table structure, SQL can be extended to this case

• Standard SQL queries can be extended to methods returning simple data types

• Many other good ideas
Benefits of ORDBMS

Extension: manage arbitrarily complex data types
Migration: preserve and extend existing databases
Preserve SQL interface
  - OR extensions in latest standard

All the benefits/experience of earlier databases
  - Access control, data integrity, persistence, etc.

Killer app: Behind Web/CGI
  - Images, video, audio, animation, applets, etc.
Markup languages
Definition

A **markup language** describes the structure of a document

- Based on tags
- Tags denote structural elements like sections, subsections, figures, etc

Internationally standardized, so application independent
Example: HTML

```
<html>
<h1>Super Widget</h1>
<h2>Widgets Incorporated</h2>
<em>123456789</em>
<br>
<p>$300</p>
</html>
```
Example: XML

Tags Emphasize what the things *mean* rather than how to *format* their Presentation.

```xml
<xml>
  <product>
    <model>Super Widget</model>
    <make>Widgets Incorporated</make>
    <sku>123456789</sku>
    <price>$300</price>
  </product>
</xml>
```
XML in Ecommerce example

<xml>
  <product>
    <model> Super Widget </model>
    <make> Widgets Incorporated </make>
    <sku> 123456789 </sku>
    <price> $300 </price>
  </product>
  <product>
    <model> Amazing Gadget </model>
    <make> Gadgets Co. </make>
    <sku> 987654321 </sku>
    <price> $500 </price>
  </product>
</xml>

Supplier

Product info
From each Supplier sent in XML

Retailer

Stuff4U

Super Widget $300
Amazing Gadget $500

Consumer
XML in e-commerce example 2

From each Supplier sent in XML, product info is recognized and managed by SCM software.

```
<xml>
  <product>
    <model>Super Widget</model>
    <make>Widgets Incorporated</make>
    <sku>123456789</sku>
    <price>$300</price>
  </product>
</xml>
```
Family lineage

- **SGML**
  - Emphasizes formatting and presentation of documents
  - Standardized in mid 80s by ISO

- **HTML**
  - Introduced in Early 90s
  - Purpose- and presentation of documents

- **XML**
  - Proposed in mid 90s
  - Emphasizes structure of documents
  - Purpose- and industry-specific extensions
Break
MySQL Quiz!!!

1) Who are the top three companies in the database business (in terms of sales revenue)?

2) In 2003, MySQL formed an alliance with what major enterprise software company?

3) According to the case, what operating system was leading the open source software movement?
   a) Linux  b) Windows  c) MAC OS  d) DOS
MySQL Case

- Raymund Rosario: MySQL Database case