SQL AND SCRIPTACULOUS

Database and front end tricks
Announcements

- Demo Tuesday, April 1
  - Sprint 1
    - Each person chooses a story
    - The team presents the results to the TA
    - The TA grades the results

- MySQL is now available on Betaweb
  - Log on and change your password.
  - Try the examples
Databases on Betaweb
13.1: Database Basics
13.2: SQL
13.3: Multi-table Queries
13.4: Databases and PHP
SQL (Structured Query Language)

SELECT name FROM cities WHERE id = 17;
INSERT INTO countries VALUES ('SLD', 'ENG', 'T', 100.0);

- A language for searching and updating a database
- Standard syntax used by every DBMS (with minor incompatibilities)
  - generally case-insensitive
- a declarative language: describes the data you are seeking, not how to find it
MySQL on Betaweb

```bash
betaweb:databases> pwd
/home/martin/databases
betaweb:databases> mysql -n martin -p
Enter password:
```

```bash
mysql>
```
Changing Password

mysql> SET PASSWORD = PASSWORD('password');

- Always put a semicolon after a SQL command
- SQL commands are case insensitive

mysql> quit;

- You can exit using the “quit” command.
Loading Databases

databases> mysql -u martin -p martin < simpsons.sql
Enter password:
databases> mysql -u martin -p
Enter password:

mysql>
### Viewing Databases

```sql
mysql> show databases;
+-------------------+
| Database           |
+-------------------+
| information_schema |
| martin             |
+-------------------+
2 rows in set (0.00 sec)
```
Using Databases

mysql> use martin

Reading table information for completion of table and column names

You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;

+------------------+
| Tables_in_martin |
+------------------+
| courses          |
| grades           |
| students         |
| teachers         |
+------------------+

4 rows in set (0.00 sec)
SHOW DATABASES;
USE database;
SHOW TABLES;

mysql> USE world;
Database changed

mysql> SHOW TABLES;
+-----------+
| cities    |
| countries |
| languages |
+-----------+
3 rows in set (0.00 sec)
the **SELECT** statement searches a database and returns a set of results the column name(s) written after SELECT filter which parts of the rows are returned

- table and column names are case-sensitive
- **SELECT * FROM table;** keeps all columns
### SELECT Example

```sql
mysql> SELECT name, code FROM countries;
+----------------------------------------------+------+
| name                                         | code |
+----------------------------------------------+------+
| Afghanistan                                  | AFG  |
| Netherlands                                  | NLD  |
| ....                                         |      |
```
The DISTINCT modifier

- SELECT language FROM languages;
  - language
    - Dutch
    - English
    - English
    - Spanish
    - Spanish
    - ...

- SELECT DISTINCT language FROM languages;
  - language
    - Dutch
    - English
    - Spanish
    - ...

CSC 210
The WHERE clause

- WHERE clause filters out rows based on their columns' data values
- In large databases, it's critical to use a WHERE clause to reduce the result set size
- Suggestion: when trying to write a query, think of the FROM part first, then the WHERE part, and lastly the SELECT part

SELECT columns FROM table WHERE conditions(s)
Conditions

- The WHERE portion of a SELECT statement can use the following operators:
  - =, >, >=, <, <=
  - <> : not equal
  - BETWEEN min AND max
  - LIKE pattern
  - IN (value, value, ..., value)

- Multiple WHERE clauses can be combined with AND, OR
SELECT code, name, population FROM countries WHERE name LIKE 'United%'

- LIKE 'text%' searches for text that starts with a given prefix
- LIKE '%text' searches for text that ends with a given suffix
- LIKE '%text%' searches for text that contains a given substring
Sorting by a column: ORDER BY

- SELECT code, name, population FROM countries WHERE name LIKE 'United%' ORDER BY population

- can write ASC or DESC to sort in ascending (default) or descending order: SELECT * FROM countries ORDER BY population DESC;

- can specify multiple orderings in decreasing order of significance: SELECT * FROM countries ORDER BY population DESC, gnp;

- see also: GROUP BY
Limiting rows: LIMIT

- `SELECT name FROM cities WHERE name LIKE 'K%' LIMIT 5;`
- can be used to get the top-N of a given category (ORDER BY and LIMIT)
- also useful as a sanity check to make sure your query doesn't return $10^7$ rows
The SQL **INSERT** statement

- **INSERT INTO** *table* VALUES (*value*, *value*, ..., *value*);
- adds a new row to the given table
- columns' values should be listed in the same order as in the table
More about INSERT

- `INSERT INTO table (columnName, columnName, ..., columnName) VALUES (value, value, ..., value);`
- some columns have default or auto-assigned values (such as IDs)
- omitting them from the INSERT statement uses the defaults
The SQL **REPLACE** statement

- REPLACE INTO *table* (*columnName*, *columnName*, ..., *columnName*) VALUES (*value*, *value*, ..., *value*);

- just like INSERT, but if an existing row exists for that key (ID), it will be replaced

- can pass optional list of column names, like with INSERT
The SQL **UPDATE** statement

- UPDATE *table* SET *column* = *value*, ..., *column* = *value*
  WHERE *column* = *value*;

- modifies an existing row(s) in a table

- BE CAREFUL! If you omit the WHERE, it modifies ALL rows
The SQL **DELETE** statement

- `DELETE FROM table WHERE condition;`
- removes existing row(s) in a table
- can be used with other syntax like LIMIT, LIKE, ORDER BY, etc.
- BE CAREFUL! If you omit the WHERE, it deletes ALL rows
Creating and deleting an entire database

- `CREATE DATABASE name;`
- `DROP DATABASE name;`

- adds/deletes an entire database from the server
Creating and deleting a table

CREATE TABLE name ( 
    columnName type constraints, 
    ... 
    columnName type constraints 
); 
DROP TABLE name;

- adds/deletes a table from this database
- all columns' names and types must be listed
SQL data types

- BOOLEAN: either TRUE or FALSE
- INTEGER
- DOUBLE
- VARCHAR(length): a string
- ENUM(value, ..., value): a fixed set of values
- DATE, TIME, DATETIME
- BLOB: binary data
CREATE TABLE students (  
id INTEGER UNSIGNED NOT NULL PRIMARY KEY,  
name VARCHAR(20) NOT NULL,  
email VARCHAR(32),  
password VARCHAR(16) NOT NULL DEFAULT "12345"  
);  

- NOT NULL: cannot insert null/empty value in any row for that column  
- PRIMARY KEY / UNIQUE: no two rows can have the same value  
- DEFAULT value: if no value is provided, use the given default  
- AUTO_INCREMENT: default value is last row's value plus 1 (e.g., for IDs)  
- UNSIGNED: don't allow negative numbers (INTEGER only)
Rename a table

- ALTER TABLE name RENAME TO newName;

- changes the name of an existing table
ADD/REMOVE/MODIFY TABLE COLUMN

- **ADD COLUMN** `columnName` `type constraints`;
- **DROP COLUMN** `columnName`;
- **CHANGE COLUMN** `oldColumnName` `newColumnName` `type constraints`;

- **adds/deletes/respecifies** a column in an existing table
- **if a column is added**, all existing rows are given a default value for that column

CSC 210
SQL

- 13.1: Database Basics
- 13.2: SQL
- 13.3: Multi-table Queries
- 13.4: Databases and PHP
Querying multi-table databases

- Datasets spread across multiple tables need queries that can answer high-level questions such as:
  - What courses has Bart taken and gotten a B- or better?
  - What courses have been taken by both Bart and Lisa?
  - Who are all the teachers Bart has had?
  - How many total students has Ms. Krabappel taught, and what are their names?

- To do this, we'll have to join data from several tables in our SQL queries.
Cross product with JOIN

- Cross product or Cartesian product: combines each row of first table with each row of second table.
  - Produces $M \times N$ rows, where table 1 has $M$ rows and table 2 has $N$.
  - Problem: produces too much irrelevant/meaningless data.
### Example Database: Simpsons

<table>
<thead>
<tr>
<th>Tables_in_martin</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>courses</td>
<td></td>
</tr>
<tr>
<td>grades</td>
<td></td>
</tr>
<tr>
<td>students</td>
<td></td>
</tr>
<tr>
<td>teachers</td>
<td></td>
</tr>
</tbody>
</table>

```sql
mysql> show tables;
+------------------+
| Tables_in_martin |
+------------------+
| courses          |
| grades           |
| students         |
| teachers         |
+------------------+
```

```sql
mysql> describe courses;
+------------+------------------+
| Field      | Type             |
+------------+------------------+
| id         | int(10) unsigned |
| name       | varchar(32)      |
| teacher_id | int(10) unsigned |
+------------+------------------+
```

```sql
describe grades;
+------------------+
| Field            |
| student_id       |
| course_id        |
| grade            |
+------------------+
```

```sql
| Field            |
| student_id       |
| course_id        |
| grade            |
+------------------+
```

| Type             |
| int(10) unsigned |
| int(10) unsigned |
| varchar(2)       |
### Example simpsons database

<table>
<thead>
<tr>
<th>students id</th>
<th>name</th>
<th>email</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Bart</td>
<td><a href="mailto:bart@fox.com">bart@fox.com</a></td>
</tr>
<tr>
<td>456</td>
<td>Milhouse</td>
<td><a href="mailto:milhouse@fox.com">milhouse@fox.com</a></td>
</tr>
<tr>
<td>888</td>
<td>Lisa</td>
<td><a href="mailto:lisa@fox.com">lisa@fox.com</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>teachers id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>Krabappel</td>
</tr>
<tr>
<td>5678</td>
<td>Hoover</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>courses id</th>
<th>name</th>
<th>teacher_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>10001</td>
<td>Computer Science 142</td>
<td>1234</td>
</tr>
<tr>
<td>10002</td>
<td>Computer Science 143</td>
<td>5678</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>grades student_id</th>
<th>course_id</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>10001</td>
<td>B-</td>
</tr>
<tr>
<td>123</td>
<td>10002</td>
<td>C</td>
</tr>
<tr>
<td>456</td>
<td>10001</td>
<td>B+</td>
</tr>
<tr>
<td>888</td>
<td>10002</td>
<td>A+</td>
</tr>
</tbody>
</table>
Example Cross Produce

SELECT column(s) FROM table1 JOIN table2;

SELECT * FROM students JOIN grades;

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>email</th>
<th>st_id</th>
<th>c_id</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Bart</td>
<td><a href="mailto:bart@fox.com">bart@fox.com</a></td>
<td>123</td>
<td>10001</td>
<td>B-</td>
</tr>
<tr>
<td>456</td>
<td>Milhouse</td>
<td><a href="mailto:milhouse@f.c">milhouse@f.c</a></td>
<td>123</td>
<td>10001</td>
<td>B-</td>
</tr>
<tr>
<td>888</td>
<td>Lisa</td>
<td><a href="mailto:lisa@fox.com">lisa@fox.com</a></td>
<td>123</td>
<td>10001</td>
<td>B-</td>
</tr>
<tr>
<td>123</td>
<td>Bart</td>
<td><a href="mailto:bart@fox.com">bart@fox.com</a></td>
<td>123</td>
<td>10002</td>
<td>C</td>
</tr>
</tbody>
</table>

... (18 rows returned)
Joining with ON clauses

SELECT column(s) 
FROM table1 
JOIN table2 ON condition(s) 
...
JOIN tableN ON condition(s);

SELECT * 
FROM students 
JOIN grades ON id = student_id;

- join: combines tables ON certain conditions
- the ON clause specifies which records to matches
- the rows are often linked by their key columns (id)
Join example

```
SELECT *
FROM students
JOIN grades ON id = student_id;
```

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>email</th>
<th>student_id</th>
<th>course_id</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Bart</td>
<td><a href="mailto:bart@fox.com">bart@fox.com</a></td>
<td>123</td>
<td>10001</td>
<td>B-</td>
</tr>
<tr>
<td>123</td>
<td>Bart</td>
<td><a href="mailto:bart@fox.com">bart@fox.com</a></td>
<td>123</td>
<td>10002</td>
<td>C</td>
</tr>
<tr>
<td>456</td>
<td>Milhouse</td>
<td><a href="mailto:milhouse@fox.com">milhouse@fox.com</a></td>
<td>456</td>
<td>10001</td>
<td>B+</td>
</tr>
<tr>
<td>888</td>
<td>Lisa</td>
<td><a href="mailto:lisa@fox.com">lisa@fox.com</a></td>
<td>888</td>
<td>10002</td>
<td>A+</td>
</tr>
</tbody>
</table>

- `table.column` can be used to disambiguate column names:
- `SELECT * FROM students JOIN grades ON students.id = grades.student_id;`
Filtering columns in a join

```
SELECT name, course_id, grade
FROM students
JOIN grades ON id = student_id;
```

<table>
<thead>
<tr>
<th>name</th>
<th>course_id</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bart</td>
<td>10001</td>
<td>B-</td>
</tr>
<tr>
<td>Bart</td>
<td>10002</td>
<td>C</td>
</tr>
<tr>
<td>Ralph</td>
<td>10004</td>
<td>D+</td>
</tr>
<tr>
<td>Milhouse</td>
<td>10001</td>
<td>B+</td>
</tr>
<tr>
<td>Lisa</td>
<td>10002</td>
<td>A+</td>
</tr>
<tr>
<td>Lisa</td>
<td>10003</td>
<td>A+</td>
</tr>
</tbody>
</table>
Filtered join (JOIN with WHERE)

```
SELECT name, course_id, grade
FROM students
JOIN grades ON id = student_id
WHERE name = 'Bart';
```

<table>
<thead>
<tr>
<th>name</th>
<th>course_id</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bart</td>
<td>10001</td>
<td>B-</td>
</tr>
<tr>
<td>Bart</td>
<td>10002</td>
<td>C</td>
</tr>
</tbody>
</table>

- JOIN glues tables together; WHERE filters results
- ON vs Where
  - ON directly links columns of the joined tables
  - WHERE sets additional constraints such as particular values (123, 'Bart')
What's wrong with this?

SELECT name, id, course_id, grade
FROM students
JOIN grades ON id = 123
WHERE id = student_id;

<table>
<thead>
<tr>
<th>name</th>
<th>id</th>
<th>course_id</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bart</td>
<td>123</td>
<td>10001</td>
<td>B-</td>
</tr>
<tr>
<td>Bart</td>
<td>123</td>
<td>10002</td>
<td>C</td>
</tr>
</tbody>
</table>

- The above query is poor style. Why?
- The JOIN ON clause doesn't say what connects a grades record to a students record.
  - Grades/students related on a student id.
  - Filtered by a specific ID.
Giving names to tables

SELECT s.name, g.*
FROM students s
JOIN grades g ON s.id = g.student_id
WHERE g.grade <= 'C';

- can give names to tables, like a variable name in Java
- to specify all columns from a table, write table.*
Multi-way join

```
SELECT c.name
FROM courses c
JOIN grades g ON g.course_id = c.id
JOIN students bart ON g.student_id = bart.id
WHERE bart.name = 'Bart' AND g.grade <= 'B-';
```

- More than 2 tables can be joined, as shown above
- This query gets all courses in which Bart has gotten a B- or better
Another example

- What courses have been taken by both Bart and Lisa

SELECT DISTINCT c.name
FROM courses c
JOIN grades g1 ON g1.course_id = c.id
JOIN students bart ON g1.student_id = bart.id
JOIN grades g2 ON g2.course_id = c.id
JOIN students lisa ON g2.student_id = lisa.id
WHERE bart.name = 'Bart'
AND lisa.name = 'Lisa';
Designing a query

- Figure out the proper SQL queries in the following way:
  - Which table(s) contain the critical data? (FROM)
  - Which columns do I need in the result set? (SELECT)
  - How are tables connected (JOIN), filtered (WHERE)?
- Test on a small data set; confirm on real data set.
- Try out the queries first in the MySQL console.
- Write the PHP code to run those same queries.
  - Make sure to check for SQL errors at every step!!
Example simpsons database

<table>
<thead>
<tr>
<th>students id</th>
<th>name</th>
<th>email</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Bart</td>
<td><a href="mailto:bart@fox.com">bart@fox.com</a></td>
</tr>
<tr>
<td>456</td>
<td>Milhouse</td>
<td><a href="mailto:milhouse@fox.com">milhouse@fox.com</a></td>
</tr>
<tr>
<td>888</td>
<td>Lisa</td>
<td><a href="mailto:lisa@fox.com">lisa@fox.com</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>teachers id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>Krabappel</td>
</tr>
<tr>
<td>5678</td>
<td>Hoover</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>courses id</th>
<th>name</th>
<th>teacher_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>10001</td>
<td>Computer Science 142</td>
<td>1234</td>
</tr>
<tr>
<td>10002</td>
<td>Computer Science 143</td>
<td>5678</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>grades student_id</th>
<th>course_id</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>10001</td>
<td>B-</td>
</tr>
<tr>
<td>123</td>
<td>10002</td>
<td>C</td>
</tr>
<tr>
<td>456</td>
<td>10001</td>
<td>B+</td>
</tr>
<tr>
<td>888</td>
<td>10002</td>
<td>A+</td>
</tr>
</tbody>
</table>

CSC 210
1. What are the names of all teachers Bart has had?
2. What are the names of the students Ms. Krabappel has taught?
And the answer is ...
Quiz Answers

SELECT DISTINCT t.name
FROM teachers t
JOIN courses c ON c.teacher_id = t.id
JOIN grades g ON g.course_id = c.id
JOIN students s ON s.id = g.student_id
WHERE s.name = 'Bart';

SELECT DISTINCT s.name
FROM students s
JOIN grades g ON s.id = g.student_id
JOIN courses c ON g.course_id = c.id
JOIN teachers t ON t.id = c.teacher_id
WHERE t.name = 'Krabappel';
Discuss questions with your Scrum Team
Usability

- Summarize
- Organize
- Write compactly
- Don’t be too creative!
Navigation and links

- **Menus:**
  - Horizontal
  - Vertical
  - Flyout

- ** Efficient forms**
  - Proper input elements
  - Min number of fields
  - Javascript/PHP for validation
Visual Effects
Scriptaculous overview

Scriptaculous: a JavaScript library, built on top of Prototype, that adds:

- visual effects (animation, fade in/out, highlighting)
- drag and drop
- some DOM enhancements
- other stuff (unit testing, etc.)
Downloading and using Scriptaculous

- documentation available on their [wiki](#)
- [Scriptaculous Effects Cheat Sheet](#)

```html
<script src="http://ajax.googleapis.com/ajax/libs/scriptaculous/1.9.0/scriptaculous.js" type="text/javascript"></script>
```
Visual Effects

appearing:
appear  blindDown  grow  slideDown

disappearing:
blindUp  dropOut  fade  fold  puff  shrink  slideUp  squish  switchOff

Getting attention:
highlight  pulsate  shake  morph  Effect.Move  Effect.Scale  Effect.toggle (blind)

Click effects above
Adding effects to an element

- The effect will begin to animate on screen (asynchronously) the moment you call it.

- Six core effects are used to implement all effects on the previous slides:
  - `Effect.Highlight`
  - `Effect.Morph`
  - `Effect.Move`
  - ` Effect.Opacity`
  - `Effect.Parallel`
  - `Effect.Scale`

```javascript
// for most effects
element.effectName();

// some effects must be run the following way:
new Effect.name(element or id);

```
Adding effects to an element

- many effects can be customized by passing additional options (note the `{}`)

- options (wiki): delay, direction, duration, fps, from, queue, sync, to, transition
Adding effects to an element

- all effects have the following events that you can handle:
  - `beforeStart`, `beforeUpdate`, `afterUpdate`, `afterFinish`

- the `afterFinish` event fires once the effect is done animating
  - useful to do something to the element (style, remove, etc.) when effect is done
Adding effects to an element

- each of these events receives the Effect object as its parameter
  - its properties: element, options, currentFrame, startOn, finishOn
  - some effects (e.g. Shrink) are technically "parallel effects", so to access the modified element, you write `effect.effects[0].element` rather than just `effect.element`
Drag and drop

Scriptaculous provides several objects for supporting drag-and-drop functionality:

- **Draggable**: an element that can be dragged
- **Droppables**: elements on which a Draggable can be dropped
- **Sortable**: a list of items that can be reordered
- **Puzzle Game demo**
Draggable

new Draggable(element or id, 
   { options }
);

- specifies an element as being able to be dragged
- **options**: handle, revert, snap, zindex, constraint, ghosting, starteffect, reverteffect, endeffect
- **event options**: onStart, onDrag, onEnd
  - each handler function accepts two parameters: the Draggable object, and the mouse event
Draggable Example

```javascript
document.observe("dom:loaded", function() {
    new Draggable("draggabledemo1");
    new Draggable("draggabledemo2", {revert: true, snap: [40, 40]});
});
```

```html
<div id="draggabledemo1">Draggable demo. Default options.</div>
<div id="draggabledemo2">Draggable demo. {snap: [40,40], revert: true}</div>
```
Draggables

- a global helper for accessing/managing all Draggable objects on a page

- **properties:** drags, observers

- **methods:** register, unregister, activate, deactivate, updateDrag, endDrag, keyPress, addObserver, removeObserver, notify
Droppables

To make an element react when a Draggable is dropped onto it, you’ll add it to the Droppables of the page.

- options: accept, containment, hoverclass, overlap, greedy
- event options: onHover, onDrop
  - each callback accepts three parameters: the Draggable, the Droppable, and the event

```javascript
Droppables.add(element or id,
    { options }
);
```
Draggable Example

```html
<img id="product1" src="images/shirt.png" alt="shirt" />
<img id="product2" src="images/cup.png" alt="cup" />
<div id="droptarget"></div>

```
Sortable

```javascript
Sortable.create(element or id of list,
    { options }
);
```

- specifies a list (ul, ol) as being able to be dragged into any order
- implemented internally using Draggables and Droppables
- **options**: tag, only, overlap, constraint, containment, format, handle, hoverclass, ghosting, dropOnEmpty, scroll, scrollSensitivity, scrollSpeed, tree, treeTag

- to make a list un-sortable again, call `Sortable.destroy` on it
Sortable demo

```html
<ol id="simpsons">
    <li id="simpsons_0">Homer</li>
    <li id="simpsons_1">Marge</li>
    <li id="simpsons_2">Bart</li>
    <li id="simpsons_3">Lisa</li>
    <li id="simpsons_4">Maggie</li>
</ol>
```

```javascript
document.observe("dom:loaded", function() {
    Sortable.create("simpsons");
});
```
Sortable demo

<table>
<thead>
<tr>
<th>event</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>onChange</td>
<td>when any list item hovers over a new position while dragging</td>
</tr>
<tr>
<td>onUpdate</td>
<td>when a list item is dropped into a new position (more useful)</td>
</tr>
</tbody>
</table>

document.observe("dom:loaded", function() {
  Sortable.create("simpsons", {
    onUpdate: listUpdate
  });
});

JS
Sortable list events example

document.observe("dom:loaded", function() {
    Sortable.create("simpsons", {
        onUpdate: listUpdate
    });
});

function listUpdate(list) {
    // can do anything I want here; effects, an Ajax request, etc.
    list.shake();
}
Auto-completing text fields

- Scriptaculous offers ways to make a text box that auto-completes based on prefix strings:
  - `Autocompleter.Local`: auto-completes from an array of choices
  - `Ajax.Autocompleter`: fetches and displays list of choices using Ajax
Using Autocompleter.Local

```javascript
new Autocompleter.Local(
    element or id of text box,
    element or id of div to show completions,
    array of choices,
    { options }
);
```

- you must create an (initially empty) div to store the auto-completion matches
  - it will be inserted as a ul that you can style with CSS
  - the user can select items by pressing Up/Down arrows; selected item is given a class of selected
- pass the choices as an array of strings
- pass any extra options as a fourth parameter between {}
  - options: choices, partialSearch, fullSearch, partialChars, ignoreCase
Using Autocompleter.Local

```html
<input id="bands70s" size="40" type="text" />
<div id="bandlistarea"></div>
```

```js
document.observe("dom:loaded", function() {
    new Autocompleter.Local(
        "bands70s",
        "bandlistarea",
        ["ABBA", "AC/DC", "Aerosmith", "America", "Bay City Rollers", ...],
        {}
    );
});
```
Using Autocompleter.Local

HTML

```html
<input id="bands70s" size="40" type="text" />
<div id="bandlistarea"></div>
```

CSS

```css
#bandlistarea {
    border: 2px solid gray;
}
/* 'selected' class is given to the autocomplete item currently chosen */
#bandlistarea .selected {
    background-color: pink;
}
```
Using Ajax.Autocompleter

new Ajax.Autocompleter(
    element or id of text box,
    element or id of div to show completions,
    url,
    { options }
);

- when you have too many choices to hold them all in an array, you can instead fetch subsets of choices from the server using Ajax

- instead of passing choices as an array, pass a URL from which to fetch them
  - the choices are sent back from the server as an HTML ul with li elements in it

- options: paramName, tokens, frequency, minChars, indicator, updateElement, afterUpdateElement, callback, parameters
Playing sounds (API)

<table>
<thead>
<tr>
<th>method</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sound.play(&quot;url&quot;);</td>
<td>plays a sound/music file</td>
</tr>
<tr>
<td>Sound.disable();</td>
<td>stops future sounds from playing (doesn't mute any sound in progress)</td>
</tr>
<tr>
<td>Sound.enable();</td>
<td>re-enables sounds to be playable after a call to Sound.disable()</td>
</tr>
</tbody>
</table>

- to silence a sound playing in progress, use
  ```
  Sound.play('', {replace: true});
  ```

- cannot play sounds from a local computer (must be uploaded to a web site)
new Ajax.InPlaceEditor(element or id, 
    url, 
    { options } 
);
Ajax.InPlaceEditor

new Ajax.InPlaceCollectionEditor(element or id, url, {
    collection: array of choices,
    options
}
);

- a variation of Ajax.InPlaceEditor that gives a collection of choices
- requires collection option whose value is an array of strings to choose from
- all other options are the same as Ajax.InPlaceEditor
**Ajax.InPlaceEditor**

- **Slider control:**
  
  ```javascript
  new Control.Slider("id of knob", "id of track", {options});
  ```

- **Builder** - convenience class to replace document.createElement:
  
  ```javascript
  var img = Builder.node("img", {
      src: "images/lolcat.jpg",
      width: 100, height: 100,
      alt: "I can haz Scriptaculous?"
  });
  
  $("main").appendChild(img);
  ```

- **Tabbed UIs**