Multi-table Databases and Joins

CSE 190 M (Web Programming) Spring 2008
University of Washington

References: SQL syntax reference, w3schools tutorial

Except where otherwise noted, the contents of this presentation are © Copyright 2008 Marty Stepp, Jessica Miller, and Amit Levy, and are licensed under the Creative Commons Attribution 2.5 License.

Lecture outline

- HTML tables
- database design
- IMDb database
- Multi-table queries and joins
HTML tables

displaying data in tables of rows and columns

Tables: <table>, <tr>, <td>

A 2D table of rows and columns of data (block element)

```
<table>
  <tr><td>1,1</td><td>1,2 okay</td></tr>
  <tr><td>2,1 real wide</td><td>2,2</td></tr>
</table>
```

1,1 1,2 okay
2,1 real wide 2,2

- `<table>` defines the overall table, `<tr>` each row, and `<td>` each cell's data
- Tables are often useful for displaying the results of SQL queries

Headers, captions: <th>, <caption>

```
<table>
  <caption>My important data</caption>
  <tr><th>Column 1</th><th>Column 2</th></tr>
  <tr><td>1,1</td><td>1,2 okay</td></tr>
  <tr><td>2,1 real wide</td><td>2,2</td></tr>
</table>
```

My important data

**Column 1   Column 2**

1,1 1,2 okay
2,1 real wide 2,2

- `<th>` cells in a row are considered headers; by default, they appear bold
- A caption at the start of the table labels its meaning
Styling tables

```css
table { border: 2px solid black; caption-side: bottom; }
tr { font-style: italic; }
td { background-color: yellow; text-align: center; width: 30%; }
```

My important data

<table>
<thead>
<tr>
<th>Column 1</th>
<th>Column 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,1</td>
<td>1,2 okay</td>
</tr>
<tr>
<td>2,1 real wide</td>
<td>2,2</td>
</tr>
</tbody>
</table>

- all standard CSS styles can be applied to a table, row, or cell
- table specific CSS properties:
  - `border-collapse`, `border-spacing`, `caption-side`, `empty-cells`, `table-layout`

### The border-collapse property

```css
table, tr, td, th { border: 2px solid black; }
table { border-collapse: collapse; }
```

<table>
<thead>
<tr>
<th>Without border-collapse</th>
<th>With border-collapse</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Column 1</strong></td>
<td><strong>Column 1</strong></td>
</tr>
<tr>
<td>1,1</td>
<td>1,1</td>
</tr>
<tr>
<td>2,1</td>
<td>2,2</td>
</tr>
</tbody>
</table>

- by default, the overall table has a separate border from each cell inside
- the `border-collapse` property merges these borders into one
The **rowspan** and **colspan** attributes

```html
<table>
  <tr><th>Column 1</th><th>Column 2</th><th>Column 3</th></tr>
  <tr><td colspan="2">1,1-1,2</td> <td rowspan="3">1,3-3,3</td></tr>
  <tr><td>2,1</td><td>2,2</td></tr>
  <tr><td>3,1</td><td>3,2</td></tr>
</table>
```

- **colspan** makes a cell occupy multiple columns; **rowspan** multiple rows
- **text-align** and **vertical-align** control where the text appears within a cell

### Column styles: `<col>`, `<colgroup>`

```html
<table>
  <col class="pinkhighlight" />
  <colgroup class="yellowhighlight">
    <col />
  </colgroup>
  <tr><th>Column 1</th><th>Column 2</th><th>Column 3</th></tr>
  <tr><td>1,1</td><td>1,2</td><td>1,3</td></tr>
  <tr><td>2,1</td><td>2,2</td><td>2,3</td></tr>
</table>
```

- **col** tag can be used to define styles that apply to an entire column
- **colgroup** tag groups several columns to apply a style to all of them

### Don't use tables for layout!

- (borderless) tables appear to be an easy way to achieve grid-like page layouts
- many "newbie" web pages do this (including some of our old web pages...)
- but, a `table` has semantics: it should be used only when you are actually representing a table of data
- instead of tables, use divs, widths/margins, floats, etc. to perform layout as much as possible

- tables should not be used for layout!
- Tables should not be used for layout!!
- **TABLES SHOULD NOT BE USED FOR LAYOUT!!**
Database design

Choosing the proper tables and columns for a database

Database design principles

- **database design**: the act of deciding the schema for a database
- **database schema**: a description of what tables a database should have, what columns each table should contain, which columns' values must be unique, etc.
- Some database design principles:
  - keep it simple, stupid (KISS)
  - provide an identifier, or **key**, by which any row can be uniquely fetched
  - eliminate redundancy, especially redundancy of lengthy data (strings)
    - when redundancy is needed, integers are smaller than strings and better to repeat

First database design

<table>
<thead>
<tr>
<th>name</th>
<th>email</th>
<th>course</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bart</td>
<td><a href="mailto:bart@fox.com">bart@fox.com</a></td>
<td>Computer Science 142</td>
<td>B-</td>
</tr>
<tr>
<td>Bart</td>
<td><a href="mailto:bart@fox.com">bart@fox.com</a></td>
<td>Computer Science 143</td>
<td>C</td>
</tr>
<tr>
<td>Milhouse</td>
<td><a href="mailto:milhouse@fox.com">milhouse@fox.com</a></td>
<td>Computer Science 142</td>
<td>B+</td>
</tr>
<tr>
<td>Lisa</td>
<td><a href="mailto:lisa@fox.com">lisa@fox.com</a></td>
<td>Computer Science 143</td>
<td>A+</td>
</tr>
<tr>
<td>Lisa</td>
<td><a href="mailto:lisa@fox.com">lisa@fox.com</a></td>
<td>Computer Science 190M</td>
<td>A+</td>
</tr>
<tr>
<td>Ralph</td>
<td><a href="mailto:ralph@fox.com">ralph@fox.com</a></td>
<td>Informatics 100</td>
<td>D+</td>
</tr>
</tbody>
</table>

- What's good and bad about this design?
  - Contains redundancy (name, email, course repeated frequently)
  - There is no "key" column unique to each row
Second database design

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>email</th>
<th>id</th>
<th>name</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>10001</td>
<td>Computer Science 142</td>
<td>123</td>
<td>10001</td>
<td>B-</td>
</tr>
<tr>
<td>10002</td>
<td>Computer Science 143</td>
<td>123</td>
<td>10002</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>888</td>
<td>Lisa</td>
<td><a href="mailto:lisa@fox.com">lisa@fox.com</a></td>
<td>10003</td>
<td>Computer Science 190M</td>
<td>456</td>
<td>10001</td>
<td>B+</td>
</tr>
<tr>
<td>404</td>
<td>Ralph</td>
<td><a href="mailto:ralph@fox.com">ralph@fox.com</a></td>
<td>10004</td>
<td>Informatics 100</td>
<td>888</td>
<td>10002</td>
<td>A+</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>888</td>
<td>10003</td>
<td>A+</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>404</td>
<td>10004</td>
<td>D+</td>
</tr>
</tbody>
</table>

- splitting data into multiple tables avoids redundancy
- this is also called normalizing the database
- normalized tables are often linked by unique integer IDs

Related tables and keys

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>email</th>
<th>id</th>
<th>name</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>10001</td>
<td>Computer Science 142</td>
<td>123</td>
<td>10001</td>
<td>B-</td>
</tr>
<tr>
<td>10002</td>
<td>Computer Science 143</td>
<td>123</td>
<td>10002</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>888</td>
<td>Lisa</td>
<td><a href="mailto:lisa@fox.com">lisa@fox.com</a></td>
<td>10003</td>
<td>Computer Science 190M</td>
<td>456</td>
<td>10001</td>
<td>B+</td>
</tr>
<tr>
<td>404</td>
<td>Ralph</td>
<td><a href="mailto:ralph@fox.com">ralph@fox.com</a></td>
<td>10004</td>
<td>Informatics 100</td>
<td>888</td>
<td>10002</td>
<td>A+</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>888</td>
<td>10003</td>
<td>A+</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>404</td>
<td>10004</td>
<td>D+</td>
</tr>
</tbody>
</table>

- records of one table may be associated with record(s) in another table
  - record in Student table with student_id of 888 is Lisa Simpson’s student info
  - records in Grade table with student_id of 888 are Lisa Simpson’s course grades
- primary key: a table column guaranteed to be unique for each record
Design question

- suppose we want to keep track of the teachers who teach each course
  - e.g. Ms. Krabappel always teaches CSE 142 and INFO 100
  - e.g. Ms. Hoover always teaches CSE 143
  - e.g. Mr. Stepp always teaches CSE 190M
- what tables and/or columns should we add to the database?

Design answer

- add a teachers table containing information about instructors
- link this to courses by teacher IDs
- why not just skip the teachers table and put the teacher's name as a column in courses?
  - repeated teacher names are redundant and large in size
Multi-table queries
Extracting and consolidating data from multi-table databases

Querying multi-table databases

When we have larger datasets spread across multiple tables, we 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`

```
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>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>404</td>
<td>Ralph</td>
<td><a href="mailto:ralph@fox.com">ralph@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@fox.com">milhouse@fox.com</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>
<tr>
<td>404</td>
<td>Ralph</td>
<td><a href="mailto:ralph@fox.com">ralph@fox.com</a></td>
<td>123</td>
<td>10002</td>
<td>C</td>
</tr>
</tbody>
</table>

- **cross product**: combines each row of first table with each row of second
  - produces $M \times N$ rows, where table 1 has $M$ rows and table 2 has $N$
  - problem: produces too much irrelevant/meaningless data
Joining with \texttt{ON} clauses

\begin{verbatim}
SELECT \texttt{column(s)} FROM \texttt{table1}
JOIN \texttt{table2} ON \texttt{condition(s)}
...
JOIN \texttt{tableN} ON \texttt{condition(s)};
\end{verbatim}

\begin{verbatim}
SELECT *
FROM \texttt{students}
JOIN \texttt{grades} ON \texttt{id} = \texttt{student_id};
\end{verbatim}

- \texttt{JOIN}: a relational database operation that combines records from two or more tables if they satisfy certain conditions
- the \texttt{ON} clause specifies which records from each table are matched
- often the rows are linked by their \texttt{key} columns

Join example

\begin{verbatim}
SELECT *
FROM \texttt{students}
JOIN \texttt{grades} ON \texttt{id} = \texttt{student_id};
\end{verbatim}

<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>404</td>
<td>Ralph</td>
<td><a href="mailto:ralph@fox.com">ralph@fox.com</a></td>
<td>404</td>
<td>10004</td>
<td>D+</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>
<tr>
<td>888</td>
<td>Lisa</td>
<td><a href="mailto:lisa@fox.com">lisa@fox.com</a></td>
<td>888</td>
<td>10003</td>
<td>A+</td>
</tr>
</tbody>
</table>

- \texttt{table.column} can be used to disambiguate column names:

\begin{verbatim}
SELECT * FROM \texttt{students}
JOIN \texttt{grades} ON \texttt{students.id} = \texttt{grades.student_id};
\end{verbatim}
Filtering columns in a join

```sql
SELECT name, course_id, grade
FROM   students
JOIN   grades ON students.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>

- if a column exists in multiple tables, it may be written as `table.column`

Giving names to tables

```sql
SELECT name, g.*
FROM   students s
JOIN   grades g ON s.id = g.student_id;
```

<table>
<thead>
<tr>
<th>name</th>
<th>student_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>
<tr>
<td>Ralph</td>
<td>404</td>
<td>10004</td>
<td>D+</td>
</tr>
<tr>
<td>Milhouse</td>
<td>456</td>
<td>10001</td>
<td>B+</td>
</tr>
<tr>
<td>Lisa</td>
<td>888</td>
<td>10002</td>
<td>A+</td>
</tr>
<tr>
<td>Lisa</td>
<td>888</td>
<td>10003</td>
<td>A+</td>
</tr>
</tbody>
</table>

- can give names to tables, like a variable name in Java
- to specify all columns from a table, write `table.*`
Filtered join (JOIN with WHERE)

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

<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>

- FROM/JOIN glue the proper tables together, and WHERE filters the results
- what goes in the ON clause, and what goes in WHERE?
  - ON directly links columns of the joined tables
  - WHERE sets additional constraints such as particular values (123, 'Bart')

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-';
```

<table>
<thead>
<tr>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer Science 142</td>
</tr>
</tbody>
</table>

- grade column sorts alphabetically, so grades better than B- are ones <= it

A suboptimal query

```
SELECT bart.course_id
FROM   grades bart
JOIN   grades lisa ON lisa.course_id = bart.course_id
WHERE  bart.student_id = 123
AND     lisa.student_id = 888;
```

- problem: requires us to know Bart/Lisa's Student IDs, and only spits back course IDs, not names.
- Write a version of this query that gets us the course names, and only requires us to know Bart/Lisa's names, not their IDs.
**Improved query**

- What courses have been taken by both Bart and Lisa?

```sql
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';
```

**Practice queries**

- What are the names of all teachers Bart has had?

```sql
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';
```

- How many total students has Ms. Krabappel taught, and what are their names?

```sql
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';
```

**IMDb database**

<table>
<thead>
<tr>
<th>id</th>
<th>first_name</th>
<th>last_name</th>
<th>gender</th>
</tr>
</thead>
<tbody>
<tr>
<td>433259</td>
<td>William</td>
<td>Shatner</td>
<td>M</td>
</tr>
<tr>
<td>797926</td>
<td>Britney</td>
<td>Spears</td>
<td>F</td>
</tr>
<tr>
<td>831289</td>
<td>Sigourney</td>
<td>Weaver</td>
<td>F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>year</th>
<th>rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>112290</td>
<td>Fight Club</td>
<td>1999</td>
<td>8.5</td>
</tr>
<tr>
<td>209658</td>
<td>Meet the Parents</td>
<td>2000</td>
<td>7</td>
</tr>
<tr>
<td>210511</td>
<td>Memento</td>
<td>2000</td>
<td>8.7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>actor_id</th>
<th>movie_id</th>
<th>role</th>
</tr>
</thead>
<tbody>
<tr>
<td>433259</td>
<td>313398</td>
<td>Capt. James T. Kirk</td>
</tr>
<tr>
<td>433259</td>
<td>407323</td>
<td>Sgt. T.J. Hooker</td>
</tr>
<tr>
<td>797926</td>
<td>342189</td>
<td>Herself</td>
</tr>
</tbody>
</table>

- database name is imdb on server
- also available, imdb_small with fewer records (for testing queries)
- other tables:
  - directors (id, first_name, last_name)
  - movies_directors (director_id, movie_id)
  - movies_genres (movie_id, genre)
**IMDb query example**

[stepp@webster ~]$ mysql -u stepp -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Server version: 5.0.45-log Source distribution

mysql> use imdb_small;
Database changed

mysql> select * from actors where first_name like '%mick%';

+--------+------------+-----------+--------+
| id     | first_name | last_name | gender |
|--------+------------+-----------+--------|
|  71699 | Mickey     | Cantwell  | M      |
| 115652 | Mickey     | Dee       | M      |
| 470693 | Mick       | Theo      | M      |
| 716748 | Mickie     | McGowan   | F      |
+--------+------------+-----------+--------+
4 rows in set (0.01 sec)

**PHP IMDB example**

```php
# connect to IMDB (substitute your user name / pw)
$db = mysql_connect("localhost", "USERNAME", "PASSWORD");
if (!$db) {
    die("Connect failed: " . mysql_error());
}
if (!mysql_select_db("imdb")) {
    die("Selecting DB failed: " . mysql_error());
}

# query the database
$query = "SELECT * FROM actors WHERE first_name = 'Ezekiel'";
$results = mysql_query($query);
if (!$results) {
    die("SQL query failed:
$query
" . mysql_error());
}

# loop through each actor
while ($row = mysql_fetch_array($results)) {
    <?php
        if ($row["last_name"] != '') {
            echo "<li>
            $row["last_name"]", $row["first_name"]
        </li>
    <?php
    }
}else
    ?>
```
Practice problem: Movie search

- Write a PHP script that connects to the imdb database on webster and searches for all movies whose names match a given prefix, displaying them as an HTML table. Assume that the prefix is a query string parameter passed into the script.
- Consider modifying the code so that, if only one movie matches, it will print the IDs of all actors who acted in that movie. (This isn't very useful, but we'll improve it next time.)

Practice problem: Cast list for a movie

Write a PHP script that, when given a movie, shows the names of all female actors that appeared in it. (To do this, you will need to perform an SQL query with join operations.)

Development strategy

- 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) and values filtered (WHERE)?
- Test on a small data set (imdb_small).
- Confirm on the real data set (imdb).
- 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!!