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>
```
HTML

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>
```
HTML

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

Column 1	*Column 2*
1,1	*1,2 okay*
2,1 real wide	*2,2*

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

Without `border-collapse`

Column 1	Column 2
1,1	1,2
2,1	2,2

With `border-collapse`

Column 1	Column 2
1,1	1,2
2,1	2,2

- 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>
```

Column 1	Column 2	Column 3
1,1-1,2		
2,1	2,2	1,3-3,3
3,1	3,2	

- `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 /><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>
```

Column 1	Column 2	Column 3
1,1	1,2	1,3
2,1	2,2	2,3

- `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

student_grades

name	email	course	grade
Bart	bart@fox.com	Computer Science 142	B-
Bart	bart@fox.com	Computer Science 143	C
Milhouse	milhouse@fox.com	Computer Science 142	B+
Lisa	lisa@fox.com	Computer Science 143	A+
Lisa	lisa@fox.com	Computer Science 190M	A+
Ralph	ralph@fox.com	Informatics 100	D+

- 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

students

id	name	email
123	Bart	bart@fox.com
456	Milhouse	milhouse@fox.com
888	Lisa	lisa@fox.com
404	Ralph	ralph@fox.com

courses

id	name
10001	Computer Science 142
10002	Computer Science 143
10003	Computer Science 190M
10004	Informatics 100

grades

student_id	course_id	grade
123	10001	B-
123	10002	C
456	10001	B+
888	10002	A+
888	10003	A+
404	10004	D+

- 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

students

id	name	email
123	Bart	bart@fox.com
456	Milhouse	milhouse@fox.com
888	Lisa	lisa@fox.com
404	Ralph	ralph@fox.com

courses

id	name
10001	Computer Science 142
10002	Computer Science 143
10003	Computer Science 190M
10004	Informatics 100

grades

student_id	course_id	grade
123	10001	B-
123	10002	C
456	10001	B+
888	10002	A+
888	10003	A+
404	10004	D+

- 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

students

id	name	email
123	Bart	bart@fox.com
456	Milhouse	milhouse@fox.com
888	Lisa	lisa@fox.com
404	Ralph	ralph@fox.com

courses

id	name
10001	Computer Science 142
10002	Computer Science 143
10003	Computer Science 190M
10004	Informatics 100

grades

student_id	course_id	grade
123	10001	B-
123	10002	C
456	10001	B+
888	10002	A+
888	10003	A+
404	10004	D+

- 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

teachers

id	name
1234	Krabappel
5678	Hoover
9012	Stepp

courses

id	name	teacher_id
10001	Computer Science 142	1234
10002	Computer Science 143	5678
10003	Computer Science 190M	9012
10004	Informatics 100	1234

- 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;
```
SQL

```
SELECT * FROM students JOIN grades;
```
SQL

id	name	email	student_id	course_id	grade
123	Bart	bart@fox.com	123	10001	B-
404	Ralph	ralph@fox.com	123	10001	B-
456	Milhouse	milhouse@fox.com	123	10001	B-
888	Lisa	lisa@fox.com	123	10001	B-
123	Bart	bart@fox.com	123	10002	C
404	Ralph	ralph@fox.com	123	10002	C
... (24 rows returned)					

- **cross product**: combines each row of first table with each row of second
  - produces *M* * *N* rows, where table 1 has *M* rows and table 2 has *N*
  - problem: produces too much irrelevant/meaningless data

Joining with `ON` clauses

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

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

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

Join example

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

id	name	email	student_id	course_id	grade
123	Bart	bart@fox.com	123	10001	B-
123	Bart	bart@fox.com	123	10002	C
404	Ralph	ralph@fox.com	404	10004	D+
456	Milhouse	milhouse@fox.com	456	10001	B+
888	Lisa	lisa@fox.com	888	10002	A+
888	Lisa	lisa@fox.com	888	10003	A+

- *table*`.`*column* can be used to disambiguate column names:

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

Filtering columns in a join

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

name	course_id	grade
Bart	10001	B-
Bart	10002	C
Ralph	10004	D+
Milhouse	10001	B+
Lisa	10002	A+
Lisa	10003	A+

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

Giving names to tables

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

name	student_id	course_id	grade
Bart	123	10001	B-
Bart	123	10002	C
Ralph	404	10004	D+
Milhouse	456	10001	B+
Lisa	888	10002	A+
Lisa	888	10003	A+

- 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`)

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

name	course_id	grade
Bart	10001	B-
Bart	10002	C

- `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

```SQL
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-';
```

name
Computer Science 142

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

A suboptimal query

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

```SQL
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

actors

id	first_name	last_name	gender
433259	William	Shatner	M
797926	Britney	Spears	F
831289	Sigourney	Weaver	F
...			

movies

id	name	year	rank
112290	Fight Club	1999	8.5
209658	Meet the Parents	2000	7
210511	Memento	2000	8.7
...			

roles

actor_id	movie_id	role
433259	313398	Capt. James T. Kirk
433259	407323	Sgt. T.J. Hooker
797926	342189	Herself
...		

- database name is `imdb` on server `webster.cs.washington.edu`
  - 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 (!$results) {
  die("SQL query failed:\n$query\n" . mysql_error());
}

# loop through each actor
while ($row = mysql_fetch_array($results)) {
?>
  <li>
    <?= $row["last_name"] ?>, <?= $row["first_name"] ?>
  </li>
<?php
}
?>
```

PHP

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!!