Web Programming Step by Step

Lecture 24
SQL Joins
Reading: 11.4 - 11.5; Appendix A

References: SQL syntax reference, w3schools tutorial

Appendix A: Database Design

- 11.1: Database Basics
- 11.2: SQL
- 11.3: Databases and PHP
- **Appendix A: Database Design**
- 11.4: Multi-table Queries

Database design principles (Appendix A)

- **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 by which any row can be uniquely fetched
  - eliminate redundancy, especially of lengthy data (strings)
    - integers are smaller than strings and better to repeat
  - favor integer data for comparisons and repeated values
    - integers are smaller than strings and better to repeat
    - integers can be compared/searched more quickly than strings, real numbers

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?
  - good: simple (one table), can see all data in one place
  - bad: redundancy (name, email, course repeated frequently)
  - bad: most searches (e.g. find a student's courses) will have to rely on string comparisons
  - bad: there is no single column whose value will be unique in 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
- **normalizing**: splitting tables to improve structure and remove redundancy / anomalies
- 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+

- **primary key**: a table column guaranteed to be unique for each record
  - record in `Student` table with `id` of 888 is Lisa Simpson's student info
- records of one table may be associated with record(s) in another table
- **foreign key**: a column in table A that stores a value of a primary key from another table B
  - records in `Grade` table with `student_id` of 888 are Lisa Simpson's course grades

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

11.4: Multi-table Queries

- 11.1: Database Basics
- 11.2: SQL
- 11.3: Databases and PHP
- **11.4: Multi-table Queries**

Example `simpsons` database

students

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

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

grades

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

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` (11.4.1)

```
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** or **Cartesian 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 (11.4.2)

```
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) (11.4.3)

```
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

```
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';
```

Example `imdb` database (11.1.2)

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
...		

- 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)

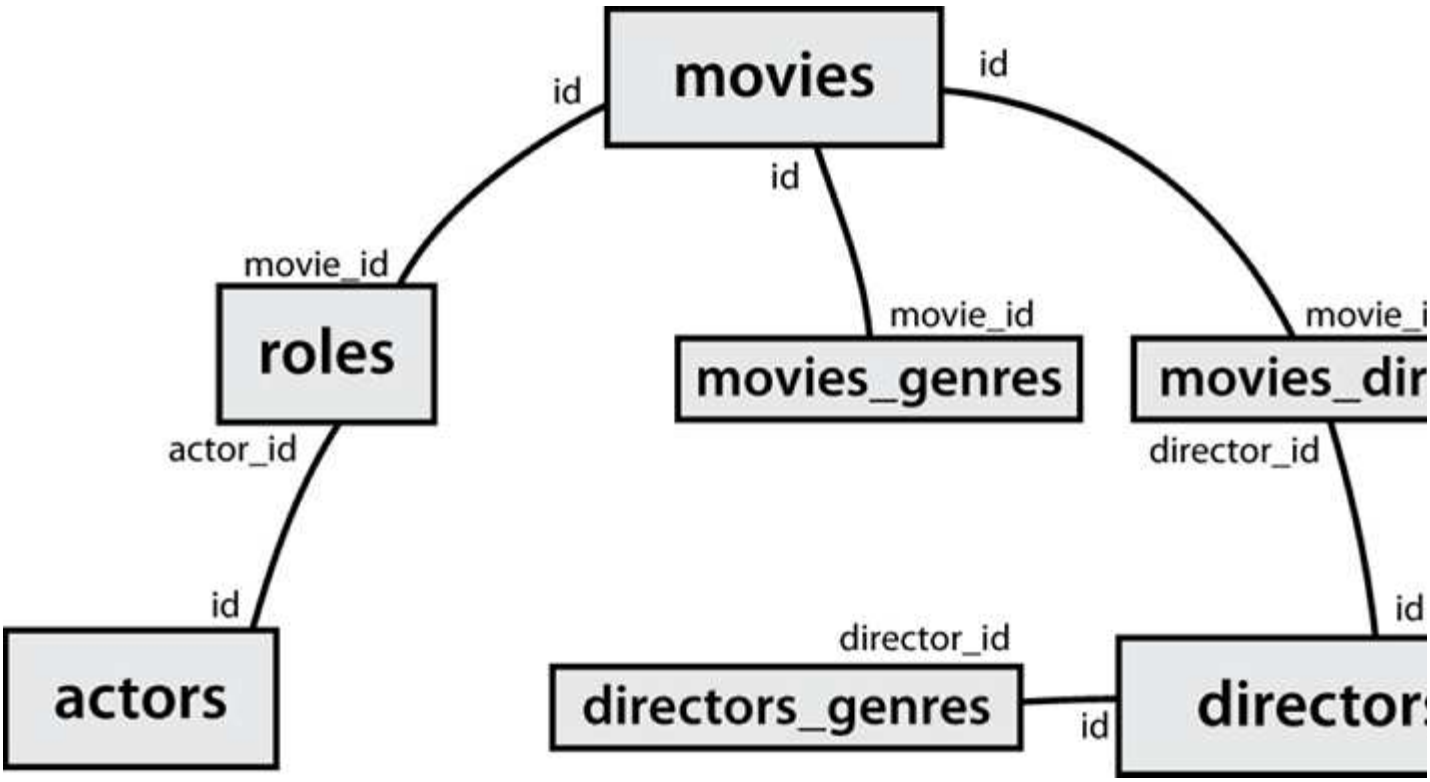
IMDb query example

```
[stepp@webster ~]$ mysql -u myusername -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.

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)
```

IMDb table relationships / ids (11.4.3)



Designing a query (11.4.4)

- 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!!