Web Programming Step by Step

Chapter 11
Relational Databases and SQL

References: SQL syntax reference, w3schools tutorial

11.1: Database Basics

Relational databases

- **relational database**: A method of structuring data as tables associated to each other by shared attributes.
- a table row corresponds to a unit of data called a **record**; a column corresponds to an attribute of that record
- relational databases typically use **Structured Query Language** (SQL) to define, manage, and search data

Why use a database? (11.1.1)

- **powerful**: can search it, filter data, combine data from multiple sources
- **fast**: can search/filter a database very quickly compared to a file
- **big**: scale well up to very large data sizes
- **safe**: built-in mechanisms for failure recovery (e.g. **transactions**)
- **multi-user**: concurrency features let many users view/edit data at same time
- **abstract**: provides layer of abstraction between stored data and app(s)
    - many database programs understand the same SQL commands

Database software

- Oracle
- Microsoft SQL Server (powerful) and Microsoft Access (simple)
- PostgreSQL (powerful/complex free open-source database system)
- SQLite (transportable, lightweight free open-source database system)
- MySQL (simple free open-source database system)
  - many servers run "LAMP" (Linux, Apache, MySQL, and PHP)
  - Wikipedia is run on PHP and MySQL
  - we will use MySQL in this course

Example `world` database (11.1.2)

Countries

Other columns: **region**, **surface_area**, **life_expectancy**, **gnp_old**, **local_name**, **government_form**, **capital**, **code2**

code	name	continent	independance_year	population	gnp	head_of_state	...
AFG	Afghanistan	Asia	1919	22720000	5976.0	Mohammad Omar	...
NLD	Netherlands	Europe	1581	15864000	371362.0	Beatrix	...
...	...	...	...	...	...	...	...

Cities

id	name	country_code	district	population
3793	New York	USA	New York	8008278
1	Los Angeles	USA	California	3694820
...	...	...	...	...

CountriesLanguages

country_code	language	official	percentag
AFG	Pashto	T	52.4
NLD	Dutch	T	95.6
...	...	...	...

11.2: SQL

SQL basics

```sql
SELECT name FROM Cities WHERE id = 17;

INSERT INTO Countries VALUES ('SLD', 'ENG', 'T', 100.0);
```

- **Structured Query Language (SQL)**: a language for searching and updating a database
- a standard syntax that is used by all database software (with minor incompatiblities)
- a **declarative** language: describes what data you are seeking, not exactly how to find it

Issuing SQL commands directly in MySQL (11.2.1 - 11.2.2)

```
SHOW DATABASES;
USE database;
SHOW TABLES;
```
SQL

- SSH to a web server, then type:

```
$ mysql -u yourusername -p
Password:
Welcome to the MySQL monitor.  Commands end with ; or \g.

mysql> USE world;
Database changed

mysql> SHOW TABLES;
+--------------------+
| Cities             |
| Countries          |
| CountriesLanguages |
+--------------------+
3 rows in set (0.00 sec)
```

The SQL `SELECT` statement (11.2.3)

```
SELECT column(s) FROM table;
```
SQL

```
SELECT name, code FROM Countries;
```
SQL

name	code
China	CHN
United States	IND
Indonesia	USA
Brazil	BRA
Pakistan	PAK
...	...

- 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

The `DISTINCT` modifier

```
SELECT DISTINCT column(s) FROM table;
```

```
SELECT language
FROM CountriesLanguages;
```

language
Dutch
English
English
Papiamento
Spanish
Spanish
Spanish
...

```
SELECT DISTINCT language
FROM CountriesLanguages;
```

language
Dutch
English
Papiamento
Spanish
...

- eliminates duplicates from the result set

The `WHERE` clause (11.2.4)

```
SELECT column(s) FROM table WHERE condition(s);
```

```
SELECT name, population FROM Cities WHERE country_code = "FSM";
```

name	population
Weno	22000
Palikir	8600

- `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

More about the `WHERE` clause

```
WHERE column operator value(s)
```

```
SELECT name, gnp FROM Countries WHERE gnp > 2000000;
```

code	name	gnp
JPN	Japan	3787042.00
DEU	Germany	2133367.00
USA	United States	8510700.00
...	...	...

- 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: AND, OR

```
SELECT * FROM Cities WHERE code = 'USA' AND population >= 2000000;
```

id	name	country_code	district	population
3793	New York	USA	New York	8008278
3794	Los Angeles	USA	California	3694820
3795	Chicago	USA	Illinois	2896016
...	...	...	...	...

- multiple `WHERE` conditions can be combined using `AND` and `OR`

Approximate matches: `LIKE`

```
WHERE column LIKE pattern
```

```
SELECT code, name, population FROM Countries WHERE name LIKE 'United%';
```

code	name	population
ARE	United Arab Emirates	2441000
GBR	United Kingdom	59623400
USA	United States	278357000
UMI	United States Minor Outlying Islands	0

- `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` (11.2.5)

```
ORDER BY column(s)
```

```
SELECT code, name, population FROM Countries
WHERE name LIKE 'United%' ORDER BY population;
```

code	name	population
UMI	United States Minor Outlying Islands	0
ARE	United Arab Emirates	2441000
GBR	United Kingdom	59623400
USA	United States	278357000

- 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;
```

The SQL `INSERT` statement (11.2.6)

```
INSERT INTO table
VALUES (value, value, ..., value);
```
SQL

```
INSERT INTO student
VALUES (789, "Nelson", "muntz@fox.com");
```
SQL

- adds a new row to the given table

The SQL `UPDATE` and `DELETE` statements

```
UPDATE table
SET column = value,
    ...,
    column = value
WHERE condition;

DELETE FROM table
WHERE condition;
```
SQL

```
UPDATE student
SET email = "lisasimpson@gmail.com"
WHERE SID = 888;

DELETE FROM student WHERE SID < 800;
```
SQL

- modifies or deletes an existing row(s) in a table

11.3: Databases and PHP

PHP MySQL functions

name	description
`mysql_connect`	connects to a database server
`mysql_select_db`	chooses which database on server to use (similar to SQL `USE` *`database;`* command)
`mysql_query`	performs a SQL query on the database
`mysql_real_escape_string`	encodes a value to make it safe for use in a query
`mysql_fetch_array`, ...	returns the query's next result row as an associative array
`mysql_close`	closes a connection to a database

Complete PHP MySQL example

```php
# connect to world database on local computer
$db = mysql_connect("localhost", "traveler", "packmybags");
mysql_select_db("world");

# execute a SQL query on the database
$results = mysql_query("SELECT * FROM Countries WHERE population > 10000(

# loop through each country
while ($row = mysql_fetch_array($results)) {
  ?>

  <li> <?= $row["name"] ?>, ruled by <?= $row["head_of_state"] ?> </li>

  <?php
}
?>
```

Connecting to MySQL: `mysql_connect` (11.3.1)

```php
mysql_connect("host", "username", "password");
mysql_select_db("database name");
```

```php
# connect to world database on local computer
mysql_connect("localhost", "traveler", "packmybags");
mysql_select_db("world");
```

- `mysql_connect` opens connection to database on its server
  - any/all of the 3 parameters can be omitted (default: `localhost`, anonymous)
- `mysql_select_db` sets which database to examine

Performing queries: `mysql_query` (11.3.2)

```
mysql_connect("host", "username", "password");
mysql_select_db("database name");

$results = mysql_query("SQL query");
...
```
PHP

```
$results = mysql_query("SELECT * FROM Cities WHERE code = 'USA'
                        AND population >= 2000000;");
```
PHP

- `mysql_query` sends a SQL query to the database
- returns a special result-set object that you don't interact with directly, but instead pass to later functions
- SQL queries are in `" "`, end with `;`, and nested quotes can be `'` or `\"`

Result rows: `mysql_fetch_array`

```
mysql_connect("host", "username", "password");
mysql_select_db("database name");
$results = mysql_query("SQL query");

while ($row = mysql_fetch_array($results)) {
  do something with $row;
}
```
PHP

- `mysql_fetch_array` returns one result row as an associative array
  - the column names are its keys, and each column's values are its values
  - example: `$row["population"]` gives the population from that row of the results

Error-checking: `mysql_error` (11.3.3)

```php
if (!mysql_connect("localhost", "traveler", "packmybags")) {
  die("SQL error occurred on connect: " . mysql_error());
}
if (!mysql_select_db("world")) {
  die("SQL error occurred selecting DB: " . mysql_error());
}
$query = "SELECT * FROM Countries WHERE population > 100000000;";
$results = mysql_query($query);
if (!$results) {
  die("SQL query failed:\n$query\n" . mysql_error());
}
```

- SQL commands can fail: database down, bad password, bad query, ...
- for debugging, always test the results of PHP's `mysql` functions
  - if they fail, stop script with `die` function, and print `mysql_error` result to see what failed
  - give a descriptive error message and also print the query, if any

Complete example w/ error checking

```php
# connect to world database on local computer
check(mysql_connect("localhost", "traveler", "packmybags"), "connect");
check(mysql_select_db("world"), "selecting db");

# execute a SQL query on the database
$query = "SELECT * FROM Countries WHERE population > 100000000;";
$results = mysql_query($query);
check($results, "query of $query");

# loop through each country
while ($row = mysql_fetch_array($results)) {
  ?>
  <li> <?= $row["name"] ?>, ruled by <?= $row["head_of_state"] ?> </li>
  <?php
}

# makes sure result is not false/null; else prints error
function check($result, $message) {
  if (!$result) {
    die("SQL error during $message: " . mysql_error());
  }
}
?>
```

Other MySQL PHP functions

name	description
`mysql_num_rows`	returns number of rows matched by the query
`mysql_num_fields`	returns number of columns per result in the query
`mysql_list_dbs`	returns a list of databases on this server
`mysql_list_tables`	returns a list of tables in current database
`mysql_list_fields`	returns a list of fields in the current data
complete list	

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;
```
SQL

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?

```
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)

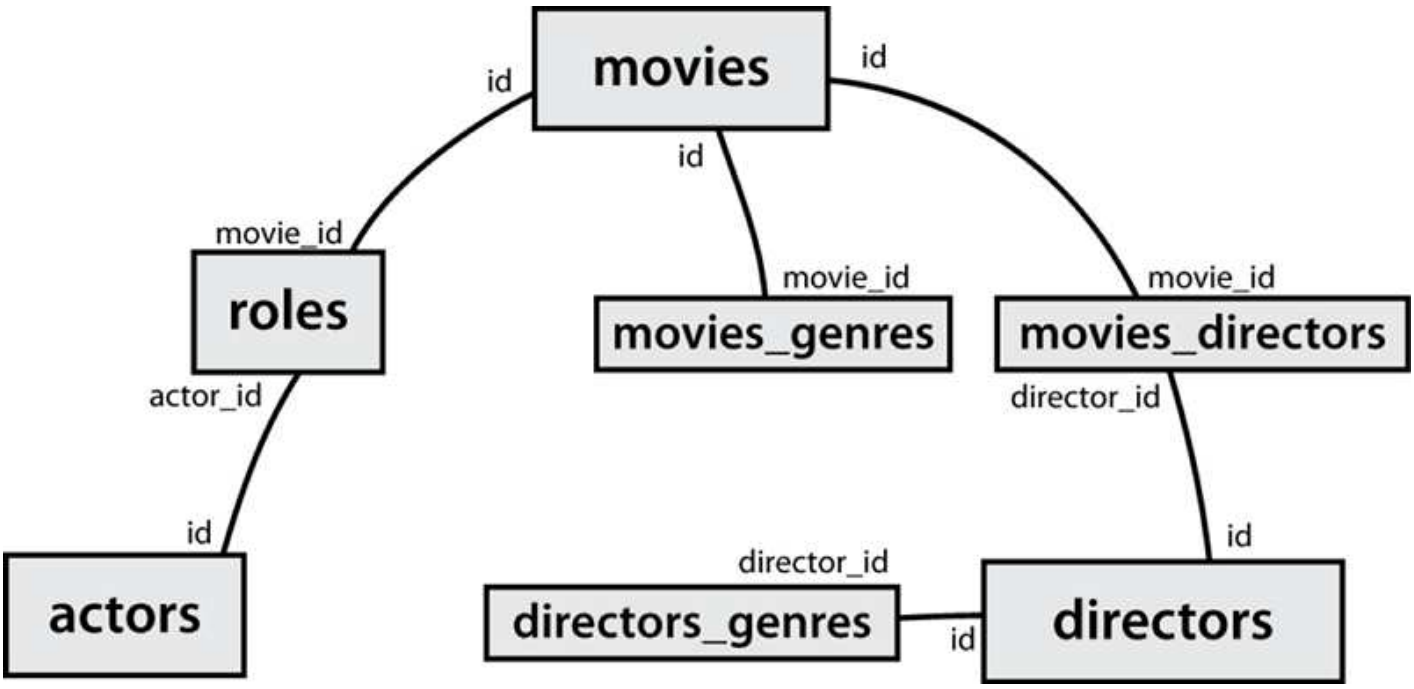
IMDb query example

```
[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!!