# Chapter 11 Relational Databases and SQL

## Introduction

Up until now the data that we have been accessing with our web applications have been stored on the server as flat files such as text or XML files. Storing data as flat files is okay for small, single-user web applications, but to support web applications that access huge amounts of data accessed by millions of users, a more advanced data storage system is needed. Enter the relational database.

Databases have been around since the 1970s and are the backbone of the majority of the web applications out there. To properly design, create, and maintain a busy database takes years of experience and is usually done by high-paid, database administrator geek types. In this chapter, we will leave the complicated stuff to them and will focus on the tasks of accessing data in a database from a web application.

Chapter Outline

## 11.1 Database Basics

> **relational database**
>
> A set of data organized into tables, with rows representing records, columns representing data fields within each record, and relationships connecting tables to each other.

A *database* is a structured collection of related data. The most common type of database is the *relational database*, invented by E. F. Codd in 1970 at IBM. Relations are connections between pieces of data. In relational databases the data are stored in two-dimensional *tables* of rows and columns. Each row (also called a *tuple* or *record*) represents a single data item or object, and each column (or *field*) represents the attributes of that object. A table is sort of like a class in C++ or Java, where each row is one object of the class and the columns of that row store the fields of data inside the object.

> **relational database management system (RDBMS)**
>
> A software package for managing databases and allowing you to query them.

In order to use a database on your web server, you must install software called a Relational Database Management System (*RDBMS*). There are a number of commercial database management systems such as Oracle Corp.'s Oracle, IBM's DB2, and Microsoft's SQL Server. There are also very good open-source RDBMSs including PostgreSQL and MySQL from Sun Microsystems. Which RDBMS you choose depends on your needs and your budget. We prefer MySQL because it is a simple, solid, free, open-source product used by many industry leaders.

Throughout this chapter we will be using data from an Internet Movie Database (IMDb) which stores information about movies, actors, and directors. For example, it has an `actors` table where each row represents information about one individual actor that has appeared in one or more movies. The `actors` table has columns for the first name, the last name, and the gender of each actor.

> **query**
>
> A request for data submitted to a database.

Since databases may contain thousands or millions of rows of data, it isn't practical to write a program that loops over each row to search for a specific piece(s) of data. Instead, you ask the RDBMS to search the database for you by sending it a request called a *query*. A query can be thought of as a declarative statement such as, "Show me all rows of actors where the actor's first name is Jessica," or, "Show me the titles of all movies that were released in the year 2005."

Some students get confused about the difference between a database and a table. A database is a collection of one or more tables. IMDb contains tables `actors`, `movies`, `directors`, and more.

Students also get confused about the difference between a database and a spreadsheet such as from Microsoft Excel. Both store data in 2D tables, but each is optimized for different usage. Spreadsheets are often for smaller amounts of data to be viewed in charts or reports. Databases are for large amounts of data to be searched or used for fast calculations. There are also structural differences: database rows are unordered (but can be retrieved in a variety of orders); database columns hold data of a specific type; and databases don't allow swapping rows for columns, ensuring that rows are objects and columns are attributes.

### 11.1.1 Motivation

Up until now we have been using simple files to store the data for our web applications. You may wonder why databases are necessary. Databases provide the following advantages over regular files:

- **Power**. With a database you can quickly and easily learn the answer to complex questions as, "How many of our customers bought both diapers and beer?" or, "Which users' accounts have shown no activity in the last 2 months other than checking deposits?"
- **Speed**. Databases are specifically designed to allow data to be searched and filtered very quickly, even with large amounts of data. For example, consider a web app for a medical company with hundreds of thousands of patients. If you use a file to store the data, the system would have to search it sequentially for a particular patient a doctor is treating.

- **Reliability**. Imagine that a large bank's web app is in the middle of transferring $1000 from one customer's account to another and the server power goes out. The money could be lost or doubly credited, which is not acceptable in a professional system. Databases provide "transactions" that guarantee that any actions performed will leave data in a reliable state.
- **Concurrency**. Databases can be accessed by up to thousands or millions of users at once, unlike ordinary file systems that often lock an entire file when one user is making changes to it. If a customer is making a purchase from your online store, you wouldn't want to lock the entire "shopping carts" file and leave other customers unable to add items to their carts.
- **Abstraction**. Databases provide a standard layer between data and applications and use a common language (SQL) understood by programmers and by applications and libraries. Even if you were able to create your own home-grown file system that had similar power to a database, it would be unfamiliar to all but the few people who created it.

### 11.1.2 Example Database: `imdb`

Figure 11.1 shows the `imdb` database tables we'll use in this chapter and a few rows from each table. Tables can contain data and/or information about relationships. For example, the `roles` table contains information about the relationship of which actors appeared in which movies.

| id | first_name | last_name | gender |
|---|---|---|---|
| 172424 | Mel | Gibson | M |
| 666662 | Scarlett | Johansson | F |

`actors`

| id | first_name | last_name |
|---|---|---|
| 15901 | Francis Ford | Coppola |
| 78273 | Quentin | Tarantino |

`directors`

| director_id | genre | prob |
|---|---|---|
| 15901 | Mystery | 0.0689655 |
| 78273 | Romance | 0.125 |

`directors_genres`

| id | name | year | rank |
|---|---|---|---|
| 46169 | Braveheart | 1995 | 8.3 |
| 194874 | Lost in Translation | 2003 | 8 |

`movies`

| director_id | movie_id |
|---|---|
| 15906 | 194874 |
| 28395 | 46169 |

`movies_directors`

| actor_id | movie_id | role |
|---|---|---|
| 172424 | 46169 | William Wallace |
| 666662 | 194874 | Charlotte |

`roles`

| movie_id | genre |
|---|---|
| 46169 | Action |
| 194874 | Drama |

`movies_genres`

**Figure 11.1 `imdb` database tables**

If you choose to set up MySQL on your own computer, you may download the `imdb` database (along with other databases we use in this chapter) from the book's accompanying web site. In the next section, we will focus on writing queries to search for information in this database and talk a bit about adding, updating, and deleting data to give you a flavor for data manipulation.

## Self-Check

1. What is a relational database? How are data organized in a relational database?
2. In RDBMSs, what is another name for a table? A column? A row?
3. Name a few of the RDBMSs out there. Which are commercial? Which are open source?
4. Which table(s) from the `imdb` database might we look at if we wanted to find out which actors have appeared in one or more comedy movies?
5. What are the standard capabilities of SQL regardless of which software you use?

## 11.2 SQL

Database queries are written in a standard declarative language called *Structured Query Language* (*SQL*), maintained as an international standard by ISO (International Organization for Standardization). SQL provides a standard way to interact with RDBMS software to define, manage, and search for data. Unfortunately each RDBMS vendor has slightly different implementations of and unique extensions to the language, but the majority of the common syntax is equivalent across all systems. The SQL code we show in this book will work properly on MySQL and other major software.

SQL provides the following capabilities:

- Queries: Retrieving desired information from the database
- Data Manipulation: Adding, updating, and deleting data in tables
- Transactions: Ensuring that data is always in a consistent state
- Data Definition: Creating and altering the structure of tables
- Data Control: Specifying permissions for access to databases

SQL differs from most other programming languages because it is a *declarative programming language* rather than an *imperative programming language*. In an imperative programming language such as Java, C, or JavaScript, you tell the computer exactly how to perform the desired computation step by step. In a declarative language like SQL, we describe to the computer what we want but not how to do it. In our SQL statements we will tell the RDBMS, "I want the data that meet the following criteria," and the RDBMS figures out how best to get that data for us.

Since SQL is a declarative programming language, writing SQL comes down to writing commands called *statements* as opposed to programs, scripts, or functions. A statement is made up of a number of *clauses*. Every clause begins with a SQL keyword. Figure 11.2 shows the components of a basic SQL statement. Don't worry about the specifics for now – we'll break those down step by step in the following sections.

**statement**

An SQL command. Statements perform queries and changes to database tables.

**clause**

A component of a SQL statement. Some are optional.

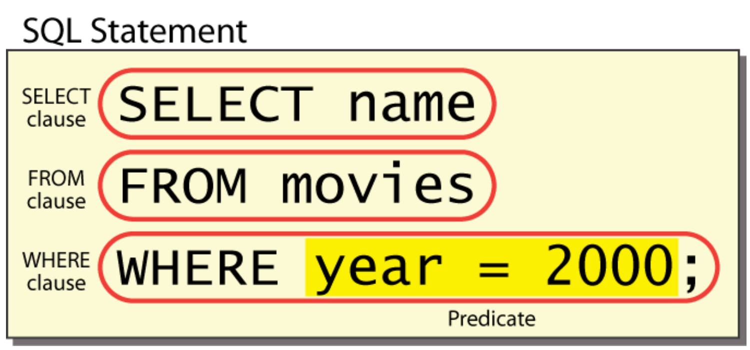


**Figure 11.2 Components of a SQL Statement**

### 11.2.1 Connecting to MySQL

Eventually we will write code in a server-side language such as PHP to connect to a database and query it. But that code will contain SQL statements in it, and before we begin to write such code we should practice bare SQL syntax in the simplest environment possible. The easiest way to practice SQL queries is to log in to a web server using an SSH terminal program, type the commands directly in to the MySQL command-line interpreter, and examine the results.

First you will need to connect to a MySQL database server that has a database to query. There may already be a database server set up for you to which you have credentials (a user name and password). If not, you can install MySQL on your own computer from the MySQL web site listed in the References section of this chapter.

First, log onto the server that is hosting the MySQL database. Once logged into the server, you should be able to connect to MySQL.

- Run your computer's SSH or terminal program and connect to your database server.
- Run the `mysql` program to start the MySQL client. Type your password when prompted.
- At the `mysql>` prompt, type an SQL query and press Enter to view its results.

Example 11.1 shows the syntax for the `mysql` command-line interpreter program. The `-u` option specifies the MySQL user name to use. The `-p` option specifies that your user requires a password. After logging in successfully, you should see some introductory information followed by a `mysql>` prompt where you can now type queries. Figure 11.3 shows the appearance on one Windows system.

```
mysql -u username -p
```

**Example 11.1 Syntax template for MySQL command line**

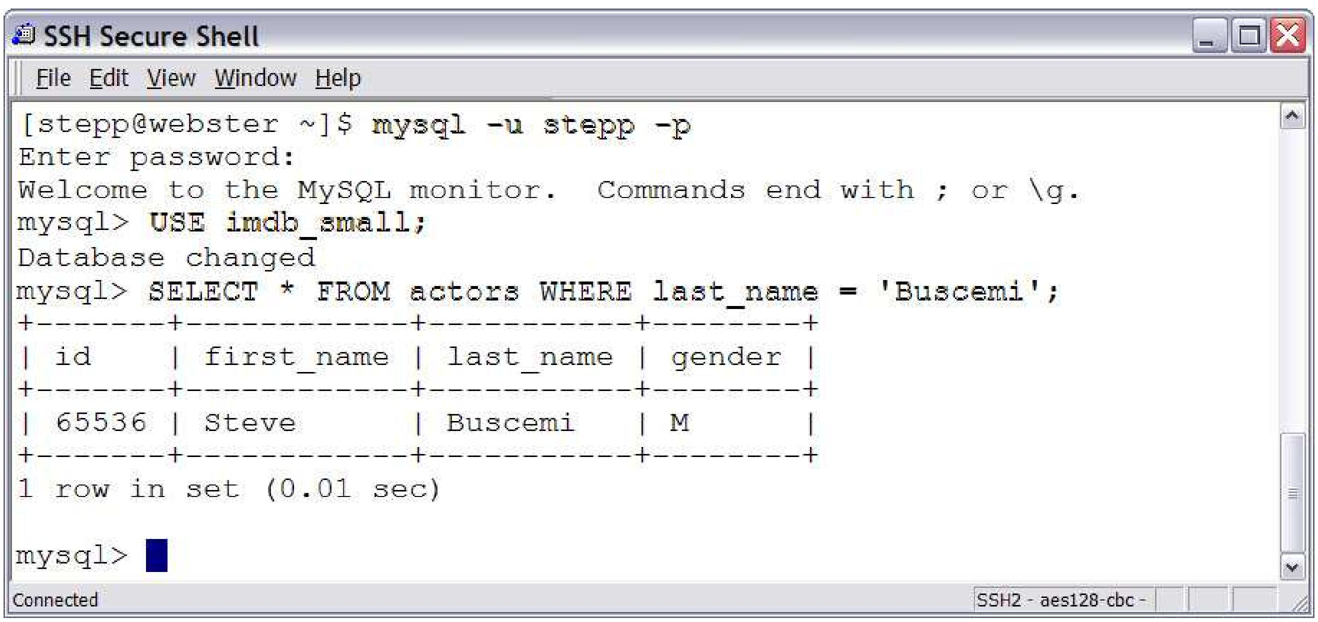


**Figure 11.3 Connecting to database server with SSH**

You can quit the MySQL client at any time by typing `QUIT` or `\q` at the MySQL prompt.

### 11.2.2 Database/Table Information

| SQL Statement | Meaning |
| --- | --- |
| `SHOW DATABASES;` | Lists all available databases on this server |
| `USE` **database**`;` | Chooses a database to use as the target of queries |
| `SHOW TABLES;` | Lists all tables in the current database (must `USE` a database first) |
| `DESCRIBE` **table**`;` | Lists information about the columns of a table (must `USE` first) |

**Table 11.1 SQL statements for database/table information**

Once successfully logged into MySQL, you must choose a database to query. You can find out the names of all available databases on a server by typing the `SHOW DATABASES` statement. An example call to this command on our IMDb server is shown in Example 11.2.

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| animalgame         |
| imdb               |
| imdb_small         |
| mysql              |
| simpsons           |
| world              |
+--------------------+
7 rows in set (0.01 sec)
```

**Example 11.2 `SHOW DATABASES` command**

`SHOW DATABASES` is our first SQL statement. All statements consist of a sequence of SQL keywords and operators, ending with a semicolon and a line break (pressing Enter). The keywords do not need to be capitalized, but many programmers do so for consistency and for good style.

To choose a database to query, you use the `USE` statement. For the rest of this chapter we'll use the `imdb_small` database, so you can type the text in Example 11.3 into the MySQL client. If the database you want to query was successfully selected, you will see a `Database changed` message.

```
mysql> USE imdb_small;
Database changed
```

**Example 11.3 `USE` command**

MySQL also has commands that allow you to get information about tables in a database. The first is the `SHOW TABLES` command, which lists all of the tables in the selected database. The output of the `SHOW TABLES` command on the `imdb_small` database is shown in Example 11.4.

```
mysql> USE imdb_small;
Database changed
mysql> SHOW TABLES;
+----------------------+
| Tables_in_imdb_small |
+----------------------+
| actors               |
| directors            |
| directors_genres     |
| movies               |
| movies_directors     |
| movies_genres        |
| roles                |
+----------------------+
7 rows in set (0.00 sec)
```

**Example 11.4 SHOW TABLES command**

If you want to get information about a particular table's columns, you can use the DESCRIBE command. The output of this command for the directors table is shown in Example 11.5.

```
mysql> DESCRIBE directors;
+------------+--------------+------+-----+---------+-------+
| Field      | Type         | Null | Key | Default | Extra |
+------------+--------------+------+-----+---------+-------+
| id         | int(11)      | NO   | PRI | 0       |       |
| first_name | varchar(100) | YES  |     | NULL    |       |
| last_name  | varchar(100) | YES  |     | NULL    |       |
+------------+--------------+------+-----+---------+-------+
3 rows in set (0.03 sec)
```

**Example 11.5 DESCRIBE command**

The DESCRIBE output has six columns, but for now we'll only focus on the first two. The Field column lists the names of the columns. The second column, Type, tells the type of the data found in each column. In the example, id is an integer of up to 11 bytes, and first_name is a string of up to 100 characters. We'll discuss SQL data types in more detail later in the chapter.

### 11.2.3 The SELECT Statement

The query is the most commonly performed SQL operation. To perform a query, you use the SELECT statement, which retrieves information from a specified table and returns the result of the query in another table. The basic syntax of the SELECT statement is shown in Example 11.6.

```
SELECT column1, column2, ..., columnN
FROM   table
```

**Example 11.6 Syntax template for SELECT statement**

In the SELECT clause you designate which table columns you want to see in the query results. In the FROM clause you designate the table to query. Example 11.7 shows a query that retrieves the first and last names of all of directors in the directors table of the imdb_small database.

```
mysql> SELECT first_name, last_name
    -> FROM directors;
+--------------+------------+
| first_name   | last_name  |
+--------------+------------+
| Andrew       | Adamson    |
| Darren       | Aronofsky  |
| Zach         | Braff      |
| James (I)    | Cameron    |
| Ron          | Clements   |
| Ethan        | Coen       |
| Joel         | Coen       |
| ...          | ...        |
| Paul (I)     | Verhoeven  |
| Andy         | Wachowski  |
| Larry        | Wachowski  |
+--------------+------------+
33 rows in set (0.00 sec)
```

**Example 11.7 Simple `SELECT` statement (some rows omitted)**

Though the SQL keywords and column names are not case-sensitive, the database and table names are case-sensitive in MySQL and some other RDBMS programs. Since the table name is `directors` with a lowercase `d`, we're careful to type it that way in our query.

A SQL statement is not considered finished until a semi-colon followed by a newline is entered. In the above example, we pressed Enter after `last_name`. The interpreter moved to the second line and indicated that the query was still incomplete by leading the next line with a `->` arrow marker. Once we entered the semicolon after `directors` and pressed Enter, the query was executed.

**Common Error**
Incorrect casing of database/tables

Although SQL keywords and column names are not case-sensitive, database and table names sometimes are. This can depend on the underlying operating system of the database server hosting the MySQL RDBMS. In some operating systems (e.g., Windows) database and table names are not case-sensitive, but in others (e.g., most varieties of Unix) database and table names are case-sensitive. Our `imdb_small` database is stored on a server that is running the Fedora Linux operating system, so case sensitivity is an issue. Example 11.8 shows the result when we run the previous query while capitalizing the d of the `directors` table.

```
mysql> SELECT first_name, last_name
    -> FROM Directors;
ERROR 1146 (42S02): Table 'imdb_small.Directors' doesn't exist
```

**Example 11.8 Common error: Incorrect capitalization of database or table name**

It is easy to incorrectly capitalize a table or database name, but the error message you get doesn't tell you that is the problem. It tells you that the database or table doesn't exist. Before you freak out and think that you lost your data, check the capitalization of your database and table names first.

You can use * as a wildcard to specify all columns of a table. The query in Example 11.9 displays everything about directors in the `imdb_small` database, including the `id`, `first_name`, and `last_name` columns. The same number of rows (33) are returned, but with more columns in each.

```
mysql> SELECT *
    -> FROM directors;
+-------+-------------+-----------+
| id    | first_name  | last_name |
+-------+-------------+-----------+
|   429 | Andrew      | Adamson   |
|  2931 | Darren      | Aronofsky |
| ...   | ...         | ...       |
| 83617 | Larry       | Wachowski |
+-------+-------------+-----------+
33 rows in set (0.00 sec)
```

**Example 11.9 `SELECT` statement with * wildcard**

### `DISTINCT` Modifier

To prevent the `SELECT` statement from returning repetitions in the results, you can use the `DISTINCT` modifier. The syntax for using the `DISTINCT` modifier is shown in Example 11.10.

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

**Example 11.10 Syntax template for `DISTINCT` modifier**

For example, from Example 11.7 we can see that there are some directors in our database with the same last name, such as Coen. The query in Example 11.11 gives all unique last names of directors.

```
mysql> SELECT DISTINCT last_name
    -> FROM directors;
+------------+
| last_name  |
+------------+
| Adamson    |
| Aronofsky  |
| Braff      |
| Cameron    |
| Clements   |
| Coen       |
| Coppola    |
| ...        |
| Verhoeven  |
| Wachowski  |
+------------+
30 rows in set (0.00 sec)
```

**Example 11.11 DISTINCT modifier**

Notice that there are now only 30 rows returned as opposed to 33. This is because there is now only one copy of the last names Coen, Coppola, and Wachowski. If you specify more than one column after `SELECT DISTINCT`, the query will strip out any records in which all columns are duplicates of another record already being returned.