Relational Databases
Introduction to SQL

Sophie Engle

November 23, 2015

Introduction

This is the script used to demonstrate SQL. The tables we will be building and using in this example are similar to:

userid	name
1	Cathy
2	Alice
3	Emily
4	Billy
5	David

id	area	number	description	userid
1	555	111-1111	Work	1
2	555	222-2222	Cell	1
3	555	333-3333	Home	2
4	555	444-4444	Home	4
5	555	555-5555	Cell	5

The exact tables will depend on which point you are at in the demo script. Refer to the lecture on SQL and JDBC for more information on what each command means.

Getting Started

First, `ssh` in to one of the CS lab computers and run `mysql`. Remember to use the command `rusers -a` to determine which lab computers are free after you log on to `stargate`.

```
ssh [username]@stargate.cs.usfca.edu
ssh [username]@hrn#####.cs.usfca.edu
mysql -h sql.cs.usfca.edu -u user## -p
```

Next, indicate the database `mysql` should start using. In this class, the database will always be the same as your username, so use the following command (replace `##` with your assigned number):

```
USE user##;
```

You will always use these first few commands to get started using `mysql` on the lab computers.

Creating Tables

Now, we can start creating our tables. Start by creating the `demo_users` table. Each user should have a unique id, which will be the primary key of the table. Copy/paste the following at the prompt:

```
CREATE TABLE demo_users (
id INTEGER NOT NULL AUTO_INCREMENT PRIMARY KEY,
name VARCHAR(15) NOT NULL );
```

You will see output similar to the snippet below. Notice that the `mysql>` text is the actual prompt, and the `->` symbol indicates a multi-line command and appears automatically after you press Enter. You will also get a status message after each command. In this case, we see that the query was okay. If there was an issue, you'll see an error instead.

```
mysql> CREATE TABLE demo_users (
    -> id INTEGER NOT NULL AUTO_INCREMENT PRIMARY KEY,
    -> name VARCHAR(15) NOT NULL );
Query OK, 0 rows affected (0.12 sec)
```

Verify that the `demo_users` table was created correctly with the following commands. Remember, do not copy/paste the `mysql>` text:

```
mysql> SHOW TABLES;
+------------------+
| Tables_in_user01 |
+------------------+
| demo_users       |
+------------------+

mysql> DESCRIBE demo_users;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| id    | int(11)     | NO   | PRI | NULL    | auto_increment |
| name  | varchar(15) | NO   |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
```

We can insert multiple values into our table at once:

```
INSERT INTO demo_users (name) VALUES
('Cathy'), ('Alice'), ('Emily'), ('Billy'), ('David');
```

To see all of the rows you entered into your table, use the statement:

```
mysql> SELECT * FROM demo_users;
+----+-------+
| id | name  |
+----+-------+
|  1 | Cathy |
|  2 | Alice |
|  3 | Emily |
|  4 | Billy |
|  5 | David |
+----+-------+
```

To sort the rows, use the command:

```
mysql> SELECT * FROM demo_users ORDER BY name ASC;
+----+-------+
| id | name  |
+----+-------+
|  2 | Alice |
|  4 | Billy |
|  1 | Cathy |
|  5 | David |
|  3 | Emily |
+----+-------+
```

Use the `DESC` keyword if you want to sort in descending order instead.

Now, create the `demo_phones` table:

```
CREATE TABLE demo_phones (
id INTEGER NOT NULL AUTO_INCREMENT PRIMARY KEY,
area CHAR(3) NOT NULL DEFAULT '555',
number CHAR(8) NOT NULL,
description VARCHAR(15),
userid INTEGER NOT NULL);
```

Make sure everything looks right:

```
mysql> DESCRIBE demo_phones;
+-------------+-------------+------+-----+---------+----------------+
| Field       | Type        | Null | Key | Default | Extra          |
+-------------+-------------+------+-----+---------+----------------+
| id          | int(11)     | NO   | PRI | NULL    | auto_increment |
| area        | char(3)     | NO   |     | 555     |                |
| number      | char(8)     | NO   |     | NULL    |                |
| description | varchar(15) | YES  |     | NULL    |                |
| userid      | int(11)     | NO   |     | NULL    |                |
+-------------+-------------+------+-----+---------+----------------+
```

Now, time to insert a row into this table:

```
INSERT INTO demo_phones
(area, number, description, userid)
VALUES ('555', '111-1111', 'Work', 1);
```

When inserting, you can mix up the order of the columns:

```
INSERT INTO demo_phones
(userid, area, number, description)
VALUES (1, '555', '222-2222', 'Cell');
```

When inserting, you can take advantage of the default value for `area`:

```
INSERT INTO demo_phones
(number, description, userid)
VALUES ('333-3333', 'Home', 2);
```

You can verify the default value did get set correctly by looking at only the last row:

```
mysql> SELECT * FROM demo_phones WHERE id=3;
+----+------+----------+-------------+--------+
| id | area | number   | description | userid |
+----+------+----------+-------------+--------+
|  3 | 555  | 333-3333 | Home        |      2 |
+----+------+----------+-------------+--------+
```

You can also insert a `NULL` into the `description` column:

```
INSERT INTO demo_phones
(area, number, description, userid)
VALUES ('555', '444-4444', null, 4);
```

You can skip specifying the columns, but then you have to provide all columns (including ones that are `AUTO_INCREMENT`, may be `NULL`, or have default values):

```
INSERT INTO demo_phones
VALUES (5, '555', '555-5555', 'Cell', 5);
```

Lets make sure everything looks right so far:

```
mysql> SELECT * FROM demo_phones;
+----+------+----------+-------------+--------+
| id | area | number   | description | userid |
+----+------+----------+-------------+--------+
|  1 | 555  | 111-1111 | Work        |      1 |
|  2 | 555  | 222-2222 | Cell        |      1 |
|  3 | 555  | 333-3333 | Home        |      2 |
|  4 | 555  | 444-4444 | NULL        |      4 |
|  5 | 555  | 555-5555 | Cell        |      5 |
+----+------+----------+-------------+--------+
```

Actually, lets get rid of that `NULL` value. We can update a row as follows:

```
UPDATE demo_phones
SET description='Home'
WHERE ISNULL(description);
```

You should see the following messages, indicating 1 row was changed:

```
Query OK, 1 row affected (0.04 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Now to double-check:

```
mysql> SELECT * FROM demo_phones WHERE id=4;
+----+------+----------+-------------+--------+
| id | area | number   | description | userid |
+----+------+----------+-------------+--------+
|  4 | 555  | 444-4444 | Home        |      4 |
+----+------+----------+-------------+--------+
```

Joining Data

It would be nice if we could see the names associated with each phone number. A simple way to combine tables is:

```
mysql> SELECT * FROM demo_users, demo_phones;
+----+-------+----+------+----------+-------------+--------+
| id | name  | id | area | number   | description | userid |
+----+-------+----+------+----------+-------------+--------+
|  1 | Cathy |  1 | 555  | 111-1111 | Work        |      1 |
|  2 | Alice |  1 | 555  | 111-1111 | Work        |      1 |
|  3 | Emily |  1 | 555  | 111-1111 | Work        |      1 |
|  4 | Billy |  1 | 555  | 111-1111 | Work        |      1 |
|  5 | David |  1 | 555  | 111-1111 | Work        |      1 |
|  1 | Cathy |  2 | 555  | 222-2222 | Cell        |      1 |
...
```

However, this is not what we want. It performs something similar to a cross product. We could filter out the undesired rows follows:

```
SELECT * FROM demo_users, demo_phones
WHERE demo_users.id = demo_phones.userid;
```

We use the `WHERE` constraint to only show those rows where the `id` columns match. The resulting output should be:

```
+----+-------+----+------+----------+-------------+--------+
| id | name  | id | area | number   | description | userid |
+----+-------+----+------+----------+-------------+--------+
|  1 | Cathy |  1 | 555  | 111-1111 | Work        |      1 |
|  1 | Cathy |  2 | 555  | 222-2222 | Cell        |      1 |
|  2 | Alice |  3 | 555  | 333-3333 | Home        |      2 |
|  4 | Billy |  4 | 555  | 444-4444 | Home        |      4 |
|  5 | David |  5 | 555  | 555-5555 | Cell        |      5 |
+----+-------+----+------+----------+-------------+--------+
```

We can even have multiple criteria:

```
SELECT * FROM demo_users, demo_phones
WHERE demo_users.id = demo_phones.userid
AND description = 'Cell';
```

You should see the following output:

```
+----+-------+----+------+----------+-------------+--------+
| id | name  | id | area | number   | description | userid |
+----+-------+----+------+----------+-------------+--------+
|  1 | Cathy |  2 | 555  | 222-2222 | Cell        |      1 |
|  5 | David |  5 | 555  | 555-5555 | Cell        |      5 |
+----+-------+----+------+----------+-------------+--------+
```

We can also achieve this using a `INNER JOIN` on the `userid` and `id` columns:

```
SELECT * FROM demo_users
INNER JOIN demo_phones
ON demo_phones.userid = demo_users.id;
+----+-------+----+------+----------+-------------+--------+
| id | name  | id | area | number   | description | userid |
+----+-------+----+------+----------+-------------+--------+
|  1 | Cathy |  1 | 555  | 111-1111 | Work        |      1 |
|  1 | Cathy |  2 | 555  | 222-2222 | Cell        |      1 |
|  2 | Alice |  3 | 555  | 333-3333 | Home        |      2 |
|  4 | Billy |  4 | 555  | 444-4444 | Home        |      4 |
|  5 | David |  5 | 555  | 555-5555 | Cell        |      5 |
+----+-------+----+------+----------+-------------+--------+
```

The result of this is the same, just the columns are in different orders:

```
SELECT * FROM demo_phones
INNER JOIN demo_users
ON demo_users.id = demo_phones.userid;
+----+------+----------+-------------+--------+----+-------+
| id | area | number   | description | userid | id | name  |
+----+------+----------+-------------+--------+----+-------+
|  1 | 555  | 111-1111 | Work        |      1 |  1 | Cathy |
|  2 | 555  | 222-2222 | Cell        |      1 |  1 | Cathy |
|  3 | 555  | 333-3333 | Home        |      2 |  2 | Alice |
|  4 | 555  | 444-4444 | Home        |      4 |  4 | Billy |
|  5 | 555  | 555-5555 | Cell        |      5 |  5 | David |
+----+------+----------+-------------+--------+----+-------+
```

We can specify which columns we want displayed:

```
SELECT name, area, number, description FROM demo_users
INNER JOIN demo_phones
ON demo_users.id = demo_phones.userid;
```

```
+-------+------+----------+-------------+
| name  | area | number   | description |
+-------+------+----------+-------------+
| Cathy | 555  | 111-1111 | Work        |
| Cathy | 555  | 222-2222 | Cell        |
| Alice | 555  | 333-3333 | Home        |
| Billy | 555  | 444-4444 | Home        |
| David | 555  | 555-5555 | Cell        |
+-------+------+----------+-------------+
```

We can skip the `ON` clause with a `NATURAL JOIN`:

```
SELECT * FROM demo_phones
NATURAL JOIN demo_users;
+----+------+----------+-------------+--------+-------+
| id | area | number   | description | userid | name  |
+----+------+----------+-------------+--------+-------+
|  1 | 555  | 111-1111 | Work        |      1 | Cathy |
|  2 | 555  | 222-2222 | Cell        |      1 | Alice |
|  3 | 555  | 333-3333 | Home        |      2 | Emily |
|  4 | 555  | 444-4444 | Home        |      4 | Billy |
|  5 | 555  | 555-5555 | Cell        |      5 | David |
+----+------+----------+-------------+--------+-------+
```

However, it joined on `id` in both tables instead of `userid`. We can fix our table as follows:

```
ALTER TABLE demo_users CHANGE id userid INTEGER;
```

Now we can see the new column name:

```
DESCRIBE demo_users;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| userid | int(11)     | NO   | PRI | 0       |       |
| name   | varchar(15) | NO   |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
```

And our natural join works as expected:

```
SELECT * FROM demo_phones NATURAL JOIN demo_users;
+--------+----+------+----------+-------------+-------+
| userid | id | area | number   | description | name  |
+--------+----+------+----------+-------------+-------+
|      1 |  1 | 555  | 111-1111 | Work        | Cathy |
|      1 |  2 | 555  | 222-2222 | Cell        | Cathy |
|      2 |  3 | 555  | 333-3333 | Home        | Alice |
|      4 |  4 | 555  | 444-4444 | Home        | Billy |
|      5 |  5 | 555  | 555-5555 | Cell        | David |
+--------+----+------+----------+-------------+-------+
```

There are also `OUTER JOIN`s. Inner joins only include rows where both tables match. An outer join will include all rows from the outer table, and show the matches from the other table. If there is not a match, you will see a `NULL` value:

```
SELECT * FROM demo_users
LEFT OUTER JOIN demo_phones
ON demo_users.userid = demo_phones.userid;
```

```
+--------+-------+------+------+----------+-------------+--------+
| userid | name  | id   | area | number   | description | userid |
+--------+-------+------+------+----------+-------------+--------+
|      1 | Cathy |    1 | 555  | 111-1111 | Work        |      1 |
|      1 | Cathy |    2 | 555  | 222-2222 | Cell        |      1 |
|      2 | Alice |    3 | 555  | 333-3333 | Home        |      2 |
|      4 | Billy |    4 | 555  | 444-4444 | Home        |      4 |
|      5 | David |    5 | 555  | 555-5555 | Cell        |      5 |
|      3 | Emily | NULL | NULL | NULL     | NULL        |   NULL |
+--------+-------+------+------+----------+-------------+--------+
```

And, of course, we can make those joins `NATURAL`:

```
SELECT * FROM demo_users
NATURAL LEFT OUTER JOIN demo_phones;
+--------+-------+------+------+----------+-------------+
| userid | name  | id   | area | number   | description |
+--------+-------+------+------+----------+-------------+
|      1 | Cathy |    1 | 555  | 111-1111 | Work        |
|      1 | Cathy |    2 | 555  | 222-2222 | Cell        |
|      2 | Alice |    3 | 555  | 333-3333 | Home        |
|      4 | Billy |    4 | 555  | 444-4444 | Home        |
|      5 | David |    5 | 555  | 555-5555 | Cell        |
|      3 | Emily | NULL | NULL | NULL     | NULL        |
+--------+-------+------+------+----------+-------------+
```

There is a `RIGHT OUTER JOIN` too, but we don't have anything extra to display. Lets add an extra row so that we see how `RIGHT OUTER JOIN` works:

```
INSERT INTO demo_phones
(number, userid)
VALUES ('777-7777', 6);
```

Now for the `RIGHT OUTER JOIN`:

```
SELECT * FROM demo_users
NATURAL RIGHT OUTER JOIN demo_phones;
+--------+----+------+----------+-------------+-------+
| userid | id | area | number   | description | name  |
+--------+----+------+----------+-------------+-------+
|      1 |  1 | 555  | 111-1111 | Work        | Cathy |
|      1 |  2 | 555  | 222-2222 | Cell        | Cathy |
|      2 |  3 | 555  | 333-3333 | Home        | Alice |
|      4 |  4 | 555  | 444-4444 | Home        | Billy |
|      5 |  5 | 555  | 555-5555 | Cell        | David |
|      6 |  6 | 555  | 777-7777 | NULL        | NULL  |
+--------+----+------+----------+-------------+-------+
```

We can get a full outer join using the `UNION` operation:

```
SELECT * FROM demo_users
LEFT OUTER JOIN demo_phones
ON demo_users.userid = demo_phones.userid
UNION
SELECT * FROM demo_users
RIGHT OUTER JOIN demo_phones
ON demo_users.userid = demo_phones.userid;
```

```
+--------+-------+------+------+----------+-------------+--------+
| userid | name  | id   | area | number   | description | userid |
+--------+-------+------+------+----------+-------------+--------+
|      1 | Cathy |    1 | 555  | 111-1111 | Work        |      1 |
|      1 | Cathy |    2 | 555  | 222-2222 | Cell        |      1 |
|      2 | Alice |    3 | 555  | 333-3333 | Home        |      2 |
|      4 | Billy |    4 | 555  | 444-4444 | Home        |      4 |
|      5 | David |    5 | 555  | 555-5555 | Cell        |      5 |
|      3 | Emily | NULL | NULL | NULL     | NULL        |   NULL |
|   NULL | NULL  |    6 | 555  | 777-7777 | NULL        |      6 |
+--------+-------+------+------+----------+-------------+--------+
```

There are other operations, like `CONCAT`:

```
SELECT userid, name,
CONCAT('(', area, ') ', number)
FROM demo_users
NATURAL LEFT OUTER JOIN demo_phones
ORDER BY name;
```

The results are as follows. Note the column names, which will be how we access results later when we connect this to Java.

```
+--------+-------+---------------------------------+
| userid | name  | CONCAT('(', area, ') ', number) |
+--------+-------+---------------------------------+
|      2 | Alice | (555) 333-3333                  |
|      4 | Billy | (555) 444-4444                  |
|      1 | Cathy | (555) 111-1111                  |
|      1 | Cathy | (555) 222-2222                  |
|      5 | David | (555) 555-5555                  |
|      3 | Emily | NULL                            |
+--------+-------+---------------------------------+
```

For a better column name, use the `AS` clause:

```
SELECT userid, name,
CONCAT('(', area, ') ', number) AS phone
FROM demo_users
NATURAL LEFT OUTER JOIN demo_phones
ORDER BY name;
+--------+-------+----------------+
| userid | name  | phone          |
+--------+-------+----------------+
|      2 | Alice | (555) 333-3333 |
|      4 | Billy | (555) 444-4444 |
|      1 | Cathy | (555) 111-1111 |
|      1 | Cathy | (555) 222-2222 |
|      5 | David | (555) 555-5555 |
|      3 | Emily | NULL           |
+--------+-------+----------------+
```

We can also use aggregate functions in combination with `GROUP BY` to do some interesting things:

```
SELECT name, description,
count(number) AS 'phone numbers'
FROM demo_users
NATURAL LEFT OUTER JOIN demo_phones
GROUP BY name;
```

```
+-------+-------------+---------------+
| name  | description | phone numbers |
+-------+-------------+---------------+
| Alice | Home        |             1 |
| Billy | Home        |             1 |
| Cathy | Work        |             2 |
| David | Cell        |             1 |
| Emily | NULL        |             0 |
+-------+-------------+---------------+
```

This is especially useful for calculating averages, maximums, etc.

Enforcing Relationships

But wait, there is something weird here now:

```
SELECT * FROM demo_phones;
+----+------+----------+-------------+--------+
| id | area | number   | description | userid |
+----+------+----------+-------------+--------+
|  1 | 555  | 111-1111 | Work        |      1 |
|  2 | 555  | 222-2222 | Cell        |      1 |
|  3 | 555  | 333-3333 | Home        |      2 |
|  4 | 555  | 444-4444 | Home        |      4 |
|  5 | 555  | 555-5555 | Cell        |      5 |
|  6 | 555  | 777-7777 | NULL        |      6 |
+----+------+----------+-------------+--------+
```

Do we have a user 6 in our table?

```
mysql> SELECT * FROM demo_users WHERE userid=6;
Empty set (0.00 sec)
```

How did this happen? Lets get rid of it:

```
mysql> DELETE FROM demo_phones WHERE id=6;
Query OK, 1 row affected (0.05 sec)
```

So, if we want MySQL to enforce relationships between our table we need to make sure we are using the `InnoDB` engine and use foreign key constraints. Let's fix our tables so this works, starting with `demo_users`. To see the details about this table, use the following command:

```
mysql> SHOW CREATE TABLE demo_users;
+-------+--------------------------------+
| Table | Create Table
+-------+--------------------------------+
| users | CREATE TABLE `users` (
  `userid` int(11) NOT NULL DEFAULT '0',
  `name` varchar(15) NOT NULL,
  PRIMARY KEY (`userid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 |
+-------+--------------------------------+
1 row in set (0.02 sec)
```

If it isn't using `InnoDB` as the `ENGINE`, then run the following commands:

```
mysql> ALTER TABLE demo_users ENGINE=InnoDB;
Query OK, 5 rows affected (0.22 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

```
mysql> ALTER TABLE demo_phones ENGINE=InnoDB;
Query OK, 5 rows affected (0.22 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

Next, we have to add in the `FOREIGN KEY` constraint:

```
ALTER TABLE demo_phones ADD FOREIGN KEY (userid)
REFERENCES demo_users(userid);
```

Now, when we try to add a row with a non existent user, we get an error. Try it out:

```
mysql> INSERT INTO demo_phones (number, userid) VALUES ('777-7777', 6);
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint
     fails (`user01`.`demo_phones`, CONSTRAINT `demo_phones_ibfk_1` FOREIGN
    KEY (`userid`) REFERENCES `demo_users` (`userid`))
```

Now that we are done with this example, go ahead and drop the tables:

```
mysql> DROP TABLE demo_phones, demo_users;
Query OK, 0 rows affected (0.09 sec)

mysql> SHOW TABLES;
Empty set (0.00 sec)
```