Advanced Online Media
Dr. Cindy Royal
Texas State University - San Marcos
School of Journalism and Mass Communication

Using SQLite Manager

SQL or Structured Query Language is a powerful way to communicate with relational databases. A database is relational if it contains multiple tables that relate to one another. Having relational databases makes for easier maintenance and simpler ways to work with data. MySQL and SQLite are popular relational database systems.

General characteristics of relational databases:

- Data is organized into tables (relations) that represent a collection of similar objects (e.g. bands).
- The columns of the table represent the attributes that members of the collection share (name, city, genre, contact).
- Each row in the table represents an individual member of the collection (one band).
- And the values in the row represent the attributes of that individual (Old 97s, Dallas, Rock, 1-512-555-1212).

Once you have a database, you can work with the data in the tables using SQL commands. There are a few basic queries:
INSERT
UPDATE
DELETE

These are things you can do to data. They are fairly self explanatory, but we will do examples of each.

You can also manipulate tables with the following:
DROP (this deletes the entire table)
ALTER (this modified a table)

Capitalizing the keywords is not required, but it is good coding. It helps you differentiate the keywords from the context names.

Always end the statement with a ;

Installing SQLite Manager
We will be using SQLite Manager, a free plug-in for Firefox. To install, visit https://addons.mozilla.org/en-US/firefox/addon/sqlite-manager/ and download and install the plug-in.You will probably need to restart Firefox.

To access SQLite Manager, open Firefox and look for it in the Tools menu.

When you are in SQLite Manager, you can view the basic interface. Mouse over the icons to see their functions.

Use the Create Database tool to create a database. We'll work with a "test" database for now.
Create a table. Use the Create Table icon to make your first table. For this table, name it "bands." We will add a series of fields to this table. id, name, city, genre, web.
id – a unique identifier that will be setup by SQLite Manager. Make it an integer, primary key, autoincrement and say it cannot be NULL.
name – VARCHAR
city – VARCHAR
genre – VARCHAR
web - VARCHAR

Click OK. What you are actually doing is executing this SQL statement – except the program is doing it for you.
CREATE TABLE "main"."band" ("id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL , "name" VARCHAR, "city" VARCHAR, "genre" VARCHAR, "web" VARCHAR);

You should now see the bands table.

Now we can use SQL to insert some data into the table.
INSERT INTO bands (name, city, genre, web) VALUES ('Old 97s', 'Dallas', 'alt-country', 'http://www.old97s.com');

Values must be surrounded properly by single quotes. Separate values with commas, outside the quotes.

We excluded a value for id because the system will assign it, based on the Auto Increment attribute.

Click Run SQL and you should see the data writing to the table. Go to Browse and Search to see your data.
Let's add a few more bands to our table:

INSERT INTO bands (name, city, genre, web) VALUES ('Quiet Company', 'Austin', 'pop', 'http://www.quietcompanymusic.com');

INSERT INTO bands (name, city, genre, web) VALUES ('Bob Schneider', 'Austin', 'rock', 'http://www.bobschneidermusic.com');

INSERT INTO bands (name, city, genre, web) VALUES ('Buttercup', 'San Antonio', 'pop', 'http://www.buttercult.com');

You can copy all these statements and run them at once. Of course, later we will learn how you can import an entire file into a table.

Now, let's run a basic query to select all the records in the table:
SELECT * FROM bands;

Now, let's just select the cities:
SELECT city FROM bands;

Let's get a list of distinct cities, using the DISTINCT keyword:

SELECT DISTINCT city FROM bands;

We can use the WHERE clause to filter the results
SELECT * from bands WHERE city='Austin';

SELECT * from bands WHERE genre='pop';

Play around with a few of these select queries using the WHERE modifier.

If we had a column that included amounts or numbers, we could also use other operators:
=Equal
<>Not equal*
>Greater than
<Less than
>=Greater than or equal
<=Less than or equal

(*Many database systems also use != for "Not equal.")

You can combine conditions with booleans AND and OR.
SELECT * FROM bands WHERE city = 'Austin' AND genre='pop';

You can specify the order field with ORDER:
SELECT * FROM bands ORDER BY city;

The order is assumed to be ascending, but you can changed to descending:
SELECT * FROM bands ORDER BY city DESC;
or
SELECT * FROM bands ORDER BY city, name DESC;

You can also update a record with a SQL command, but be careful to use the WHERE clause to filter it properly.
UPDATE bands SET genre = 'rock' WHERE name = 'Old 97s';

And you can delete a record:

DELETE FROM bands WHERE id = 1;

That would remove the entire record for Old 97s.

Importing a file
You can import a table or an entire database of multiple tables. Individual files can be plain text, using a number of separators (comma, pipe, space, etc.). You can also import a range of dbs. Pay attention to the defaults, change the first row option, if your first row has data or headers. Select the proper separator. You will get a couple of pop ups, say OK to them. Then you should have your data.

Importing a database
If a complete database is available, you can Connect to it and get all associated tables. Under the Database Menu, choose Connect Database. Find the file music2.sqlite on your Desktop. It will upload several tables, one of which is bands.

Look at the table structure, you can see a separate table for bands and genres, and then tables that bands to other fields, like websites and genres. Run a few queries on the bands table to get information by city.

SELECT * FROM bands WHERE city='Austin';
SELECT * FROM bands WHERE genre_id=1;

etc.

NULL
The keyword NULL is a special value in SQL. It's a placeholder for an empty field. If a field is NULL, it's really empty. That means it's not 0. It's not an empty string ("").

We can look to see if we have any NULL values in any of the bands fields.

SELECT * FROM bands WHERE name IS NULL;
Etc. We use IS NULL instead of = NULL. The opposite is IS NOT NULL. It is important to understand how NULL works to make sure you craft the proper queries.

LIKE and Wildcards

You can select fields with partial text by using a combination of LIKE and the wildcard indicator:
SELECT name FROM bands WHERE name LIKE 'A%';
This will find all the bands that start with A.

You can also use the SUBSTR() function for similar results:
SELECT * FROM bands WHERE SUBSTR(name, 1, 1) = 'A';

This gives us all the records of bands where the first letter is A. The first argument is the starting point, and the second is the number of characters to match.

Sometimes the data isn't very clean, so you can use some functions to find all the data. For example, maybe some cities are capitalized, some are not. You will need to use UPPER() or LOWER()

SELECT * from bands WHERE UPPER(city) = 'AUSTIN';

Concatenate

SELECT name ll ' - ' ll city FROM bands ORDER BY city, name;

Count
SELECT COUNT(id) FROM bands WHERE city = 'Austin';
SELECT COUNT(id) from bands WHERE genre_id = '5';

If we had a column with values, we could also use MIN(), MAX(), SUM(), AVG().

Joins
JOINS are very powerful ways that you can combine data from multiple tables. In a relational database environment, you have multiple tables that have a relationship between a primary and foreign key.

For example, the bands table has a column for genre, but those are numbers. We made a genre table, so we didn't have to restate the genre each time and we could maintain one table of genres. To get a list of the band and genre, we can do this:

SELECT bands.name, genres.name FROM bands, genres WHERE bands.genre_id = genres.id;

You can also write this as an explicit JOIN:
SELECT bands.name, genres.name FROM bands JOIN genres ON bands.genre_id = genres.id;

And, to get a list of all the counts of artists in genres, we can use this:

SELECT count(bands.id), genres.name FROM bands JOIN genres ON bands.genre_id = genres.id GROUP BY genres.id, genres.name;

In actuality, the genres in the bands table are just the first genre the artist indicated. Artists could supply 2 genres. So, that's what the bands_genres table does (for that matter, that is also what the bands_links table does – bands can have multiple links, website, myspace, facebook, etc.)

To do this, we need to join the bands_genres table with genres:

```
SELECT count(band_genres.band_id), genres.name FROM band_genres JOIN genres
ON band_genres.genre_id = genres.id GROUP BY genres.id, genres.name;
```

Now, let's say we want to export this to another program to do further analysis or to do a visualization (like Google Fusion Tables). The best way to do this is to use the query above in creating a View. Then we can Export the View to a csv file that can be imported into another program.