CS 250
Advanced SQL and SQLite

Benjamin Dicken

Advanced SQL

- There are many options to configure how SQLite behaves when interacting through the command prompt
- The commands used to change these configuration options are often referred to as "**dot-commands**"
    - Because you type a dot, then the name of the option, and then what you want the option to be set to

Advanced SQL

- In these slides we’ll use the schema and data from lab 10 when demonstrating how these dot-command work
- Recall:

```sql
CREATE TABLE director (
  first_name TEXT,
  last_name TEXT,
  age INT,
  director_id INT);
```

```sql
CREATE TABLE movie (
  title TEXT,
  year INT,
  rt_rating INT,
  movie_id INT,
  director_id INT);
```

Advanced SQL

- The first dot-command we will discuss is **.mode**
- The **.mode** option allows us to change the format in which SQL queries format the results
- There are 8 different **.mode** options
  - csv column html insert line list quote tabs tcl
- The default is “list”

Advanced SQL

- By default, the results of a **SELECT** are kinda ugly (**list** mode)

```
sqlite> SELECT * FROM movie;
King Kong|2005|84|1|4
Flags of Our Fathers|2006|73|2|3
Man of Steel|2013|55|3|1
Super 8|2011|82|4|5
Open Range|2003|79|5|7
The Kings Speech|2010|95|6|2
Hacksaw Ridge|2016|87|7|6
```

Advanced SQL

- **csv** mode outputs the rows in valid CSV format

```
sqlite> .mode csv
sqlite> SELECT * FROM movie;
"King Kong",2005,84,1,4
"Flags of Our Fathers",2006,73,2,3
"Man of Steel",2013,55,3,1
"Super 8",2011,82,4,5
"Open Range",2003,79,5,7
"The Kings Speech",2010,95,6,2
"Hacksaw Ridge",2016,87,7,6
```

Advanced SQL

- **html** mode outputs the rows in an html table (for you web programmers)

```
sqlite> .mode html
sqlite> SELECT * FROM movie;
<TR><TD>King Kong</TD>
<TD>2005</TD>
<TD>84</TD>
<TD>1</TD>
<TD>4</TD>
</TR>
<TR><TD>Flags of Our Fathers</TD>
<TD>2006</TD>
<TD>73</TD>
...
```

Advanced SQL

- **column** mode formats the columns for easy reading!

```
sqlite> .mode column
sqlite> SELECT * FROM movie;
King Kong   2005        84          1           4
Flags of O  2006        73          2           3
Man of Ste  2013        55          3           1
Super 8     2011        82          4           5
Open Range  2003        79          5           7
The Kings   2010        95          6           2
Hacksaw Ri  2016        87          7           6
```

Advanced SQL

- **insert** mode generates insert commands to replicate the table

```
sqlite> .mode insert
sqlite> SELECT * FROM movie;
INSERT INTO table VALUES('King Kong',2005,84,1,4);
INSERT INTO table VALUES('Flags of Our Fathers',2006,73,2,3);
INSERT INTO table VALUES('Man of Steel',2013,55,3,1);
INSERT INTO table VALUES('Super 8',2011,82,4,5);
INSERT INTO table VALUES('Open Range',2003,79,5,7);
INSERT INTO table VALUES('The Kings Speech',2010,95,6,2);
INSERT INTO table VALUES('Hacksaw Ridge',2016,87,7,6);
```

Advanced SQL

- **line** mode generates lines with variable assignments

```
sqlite> SELECT * FROM movie;
      title = King Kong
       year = 2005
  rt_rating = 84
   movie_id = 1
director_id = 4

      title = Flags of Our Fathers
       year = 2006
  rt_rating = 73
   movie_id = 2
director_id = 3
...
```

Advanced SQL

- Notice that the “wide” columns get cut off!
- By default, each column is between 1 and 10 characters wide, depending on the column header name and the width of the first column of data
- Data that is too wide to fit in a column is truncated
- Use the **.width** dot-command to adjust column widths

Advanced SQL

- **.width** specifies the column width for each column
  - The width of each column is controlled individually

```
sqlite> .mode column
sqlite> .width 20 4 4 4 4
sqlite> SELECT * FROM movie;
King Kong             2005  84    1     4
Flags of Our Fathers  2006  73    2     3
Man of Steel          2013  55    3     1
Super 8               2011  82    4     5
Open Range            2003  79    5     7
The Kings Speech      2010  95    6     2
Hacksaw Ridge         2016  87    7     6
```

Advanced SQL

- It is easy to lose track of the named of each column, and the order that they are printed in
- The **.header** dot-command allows you to optionally show/hide the names of each column in the output
- This is set to **off** by default

Advanced SQL

- The **.header** option on

```
sqlite> .header on
sqlite> SELECT * FROM movie;
title                 yea  rt_rating   movie_id    director_id
--------------------  ---  ----------  ----------  -----------
King Kong             200  84          1           4
Flags of Our Fathers  200  73          2           3
Man of Steel          201  55          3           1
Super 8               201  82          4           5
Open Range            200  79          5           7
The Kings Speech      201  95          6           2
Hacksaw Ridge         201  87          7           6
```

Advanced SQL

- The **.header** option off

```
sqlite> .header off
sqlite> SELECT * FROM movie;
King Kong             200  84         1           4
Flags of Our Fathers  200  73         2           3
Man of Steel          201  55         3           1
Super 8               201  82         4           5
Open Range            200  79         5           7
The Kings Speech      201  95         6           2
Hacksaw Ridge         201  87         7           6
```

Advanced SQL

- The **.databases** option prints info about current the database(s)

```
sqlite> .databases
seq  name             file
---  ---------------  ----------------------------------------------------------
0    main             /Users/bddicken/dev/personal-site/courses/cs250/labs/lab-1
```

Advanced SQL

- The **.tables** dot-command shows all of the tables in the current database file

```
sqlite> .tables
director  movie
```

Advanced SQL

- The **.schema** option shows the full schema (CREATE statements) for the current database

```
sqlite> .schema
CREATE TABLE director (
  first_name TEXT,
  last_name TEXT,
  age INT,
  director_id INT);
CREATE TABLE movie (
  title TEXT,
  year INT,
  rt_rating INT,
  movie_id INT,
  director_id INT);
```

Advanced SQL

- There is also a dot-command for loading data from a file directly into a database table
- We need to use **.mode** and **.import** together
    - First need to set the **.mode** to csv
    - Then, import the file

Advanced SQL

- Say we have a csv file named city.csv with the following format
- We want to quickly load all of this data into a table
- Do not want to run a bunch of individual INSERT statements!

```
name,population
Abilene,115930
Akron,217074
Albany,93994
Albuquerque,448607
Alexandria,128283
Allentown,106632
Amarillo,173627
Anaheim,328014
...
```

Advanced SQL

- Start up sqlite3 with a new (or existing) database file
- Set the **.mode** to csv

```
$ sqlite3 citydb
SQLite version 3.14.0
2016-07-26 15:17:14
Enter ".help" for usage
hints.
sqlite> .mode csv
```

Advanced SQL

- Use **.import** to load the contents of the file into a table
    - First type **.import**
    - Then write the file name
    - Last, put the table name

```
sqlite> .import city.csv city
```

Advanced SQL

- SQLite created a table and put all of the CSV rows into it!

```
sqlite> SELECT * FROM city;
Abilene,115930
Akron,217074
Albany,93994
Albuquerque,448607
Alexandria,128283
Allentown,106632
Amarillo,173627
...
```

Advanced SQL

- There is also a dot-command for dumping data from a table to a csv file
- We need to use **.mode**, **.headers**, and **.out** together
  - First need to set the **.mode** to csv
  - Enable headers with **.headers on**
  - Use **.out** to save the data

Advanced SQL

- Say we have the same city table from before

```
sqlite> SELECT * FROM city;
Abilene,115930
Akron,217074
Albany,93994
Albuquerque,448607
Alexandria,128283
Allentown,106632
Amarillo,173627
...
```

Advanced SQL

- Ensure sqlite3 is in CSV mode
- Ensure headers are turned on
- Use **.out** and then specify a name of an output file
- SELECT all of the rows, which will be sent to the file
- exit!

```
sqlite> .mode csv
sqlite> .headers on
sqlite> .out save-cities.csv
sqlite> SELECT * FROM city;
sqlite> .exit
```

Advanced SQL

- SQLite can write data files in other supported formats
- To do so, just change the **.mode** to the desired format
- Let’s try (to the command line!)

Advanced SQL

- SQLite has *many* dot-commands
- For a listing of the available dot commands, you can enter **.help** at any time

```
sqlite> .help
.auth ON|OFF             Show authorizer callbacks
.backup ?DB? FILE        Backup DB (default "main") to FILE
.bail on|off             Stop after hitting an error.  Default OFF
.binary on|off           Turn binary output on or off.  Default OFF
.changes on|off          Show number of rows changed by SQL
.clone NEWDB             Clone data into NEWDB from the existing database
.databases               List names and files of attached databases
...
```

Advanced SQL

- The **DROP** command is used to remove tables from a database

```
sqlite> CREATE TABLE movie (
   ...>   title TEXT,
   ...>   year INT,
   ...>   rt_rating INT,
   ...>   movie_id INT,
   ...>   director_id INT);
sqlite>
sqlite> .tables
movie
sqlite>
sqlite> DROP TABLE movie;
sqlite>
sqlite> .tables
sqlite>
```

Advanced SQL

- The **UPDATE** command is used to modify value(s) in a row that already exists in a database table
- A well-formed UPDATE command has three main parts
- The table to update, the column(s) to change, and the condition

```
UPDATE table_name
SET column1 = value1, column2 = value2, …
WHERE condition;
```

Advanced SQL

- Say we have these rows in the movie table
- Want to change the **rt_rating** of "Super 8" to 64

```
title                  year  rt_rating  movie_id  director_id
---------------------  ----  ---------  --------  -----------
King Kong              2005  84         1         4
Flags of Our Fathers   2006  73         2         3
Man of Steel           2013  55         3         1
Super 8                2011  82         4         5
Open Range             2003  79         5         7
The Kings Speech       2010  95         6         2
Hacksaw Ridge          2016  87         7         6
```

Advanced SQL

```
UPDATE movie
  SET rt_rating = 64
  WHERE title == 'Super 8';
```

```
title                 year  rt_rating  movie_id  director_id
--------------------  ----  ---------  --------  -----------
King Kong             2005  84         1         4
Flags of Our Fathers  2006  73         2         3
Man of Steel          2013  55         3         1
Super 8               2011  64         4         5
Open Range            2003  79         5         7
The Kings Speech      2010  95         6         2
Hacksaw Ridge         2016  87         7         6
```

Advanced SQL

- We can change multiple columns at once
- Want to change the **year** and **rt_rating** of “Hacksaw Ridge”

```
title                 year  rt_rating  movie_id  director_id
--------------------  ----  ---------  --------  -----------
King Kong             2005  84         1         4
Flags of Our Fathers  2006  73         2         3
Man of Steel          2013  55         3         1
Super 8               2011  82         4         5
Open Range            2003  79         5         7
The Kings Speech      2010  95         6         2
Hacksaw Ridge         2016  87         7         6
```

Advanced SQL

```
UPDATE movie
  SET rt_rating = 82, year = 2007
  WHERE title == 'Hacksaw Ridge';
```

```
title                 year  rt_rating  movie_id  director_id
--------------------  ----  ---------  --------  -----------
King Kong             2005  84         1         4
Flags of Our Fathers  2006  73         2         3
Man of Steel          2013  55         3         1
Super 8               2011  64         4         5
Open Range            2003  79         5         7
The Kings Speech      2010  95         6         2
Hacksaw Ridge         2010  82         7         6
```

Advanced SQL

- Exercise: Update the **rt_rating** to 90 of each movie made after 2007

Advanced SQL

- Exercise: Update the **rt_rating** to 90 of each movie made after 2007

```
UPDATE movie
  SET rt_rating = 90
  WHERE year > 2007;
```

Advanced SQL

- **Aggregate Functions** can be used to “aggregate” the values in one or more columns in SELECT statements
- SQLite supports several aggregate functions, including
  - **avg   count   group_concat   max   min   sum**
- Useful for gathering statistics and discovering the characteristics of a data set
- Let’s try them out (to the command-line!)

Advanced SQL

- **Reading Materials**
    - https://sqlite.org/cli.html (SQLite command line help)
    - http://www.sqlitetutorial.net/sqlite-import-csv/ (Loading files)
    - https://sqlite.org/lang_aggfunc.html (Aggregate functions)