CS 250
SQL and SQLite

Benjamin Dicken

SQL

- **SQL (Structured Query Language)** is a programming language designed for managing **schema** and **data** held in a relational database management system (RDBMS)

SQLite

- **SQLite** is a relational database management system written in the C programming language. In contrast to many other database management systems, SQLite is not a client–server database engine. Rather, it is embedded into the end program.
- We use SQL to insert, remove, and access the data in a SQLite database
- Specifically, we are using SQLite version 3
- Installation instructions: tutorialspoint.com/sqlite/sqlite_installation.htm

SQLite

- **SQLite** is a C library that acts as a lightweight DBMS
  - doesn’t require a separate server process and allows accessing the database
  - Uses a standard variant of the SQL query language
  - Some applications use SQLite for internal data storage
  - Since so lightweight, often prototype an application using SQLite and then port the code to a larger database such as PostgreSQL or Oracle

Learning SQL with SQLite

SQLite

- To start the SQLite 3 from bash:

```
$ sqlite3 ex1
```

- This starts up the sqlite3 program
  - If a SQL database file with the name ex1 exists in the current directory, this will user that database
  - If not, will create a new database named ex1

SQLite

- At this points, we are **connected** to the **DB** using the SQLite **DBMS**
- From here, the user must use the SQL programming language to
  - Create tables
  - Add data
  - Remove data
  - Query the tables
  - Create relationships

SQL - create tables

- The first SQL command we will learn about is **CREATE**
- Use the CREATE command to create a new table
- The format is

```
CREATE TABLE table_name(
  column1 datatype,
  column2 datatype,
  ...
  columnN datatype );
```

SQL - create tables

- Each attribute (column) has a **TYPE**.
- SQLite supports many, many types, but we will focus on only a few in this class
  - **INT** - integer numbers (like python int)
  - **FLOAT** - floating-point numbers (like python float)
  - **TEXT** - sequence of characters (like python str)
  - **BOOLEAN** - true (1) or false (0) (like python bool)

SQL - create tables

```
CREATE TABLE character(
  name TEXT,
  cid INT,
  pid INT);
```

```
CREATE TABLE person(
  name TEXT,
  pid INT);
```

```
CREATE TABLE movie(
  title TEXT,
  mid INT);
```

SQL - create tables

```
CREATE TABLE character(
  name TEXT,
  description TEXT,
  is_good BOOLEAN,
  appearances INT,
  cid INT);
```

SQL - create tables

- SQL CREATE resources:
  - sqlite.org/lang_createtable.html
  - tutorialspoint.com/sqlite/sqlite_create_table.htm

SQL - inserting

- The next SQL command we will learn about is **INSERT**
- Use the INSERT command to add rows to an existing table
- The format is

```
INSERT INTO TABLE_NAME
  (column1, column2, ... columnN)
VALUES
  (value1, value2, ... valueN);
```

SQL - inserting

```
INSERT INTO character (name, cid, pid)
VALUES (‘Thor’, 1, 4);
```

```
INSERT INTO character (name, cid, pid)
VALUES (‘Batman’, 2, 7);
```

```
INSERT INTO character (name, cid, pid)
VALUES (Superman, 3, 1);
```

SQL - inserting

```
INSERT INTO character
  (name, description, is_good, appearances, cid)
VALUES
  (‘Batman’, ‘Rich Hero Dude’, 1, 137, 2);
```

```
INSERT INTO character VALUES
  (‘Batman’, ‘Rich Hero Dude’, 1, 137, 2);
```

SQL - inserting

- SQL INSERT resources:
  - sqlite.org/lang_insert.html
  - tutorialspoint.com/sqlite/sqlite_insert_query.htm

SQL - deleting

- The next SQL command we will learn about is **DELETE**
- Use the DELETE command to delete row(s) from an existing table
- The format is

```
DELETE FROM table_name
WHERE condition;
```

- Will delete all rows that match the condition

SQL - deleting

```
DELETE FROM character where cid = 1;
```

```
DELETE FROM movie where title = ‘Batman Begins’;
```

```
DELETE FROM person where name = ‘Chris Nolan’;
```

SQL - deleting

- SQL DELETE resources:
  - sqlite.org/lang_delete.html
  - tutorialspoint.com/sqlite/sqlite_delete_query.htm

SQL - select

- The next SQL command we will learn about is **SELECT**
- Use the SELECT command to extract information from tables in the DB
- The format is

```
SELECT column1, column2, ... columnN
FROM table_name;
```

- Will return each of the specified columns from table_name

SQL - select

```
SELECT name FROM person;
```

```
SELECT cid, name FROM character;
```

```
SELECT * FROM movie;
```

SQL - select

- The SELECT comment has an optional **WHERE** clause
- A condition is specified after the WHERE keyword
- Only rows in which the condition holds true will be returned

```
SELECT column1, column2, ... columnN
FROM table_name
WHERE condition;
```

SQL - select

- Multiple WHERE clauses can be specified with AND / OR

```
SELECT column1, column2, ... columnN FROM table_name
WHERE condition1 AND condition2 AND conditionN;
```

```
SELECT column1, column2, ... columnN FROM table_name
WHERE condition1 OR condition2 OR conditionN;
```

SQL - select

```
SELECT name FROM person WHERE cid = 1;
```

```
SELECT * FROM character
WHERE description <> ‘A Rich Dude’ OR name == ‘Thor’;
```

```
SELECT * FROM character
WHERE description = ‘something’ AND cid > 3;
```

SQL - select

- SQL SELECT resources:
  - https://www.tutorialspoint.com/sqlite/sqlite_select_query.htm
  - https://sqlite.org/lang_select.html

SQL - full example

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

SQL - full example

```
sqlite> CREATE TABLE character(name TEXT, description TEXT, is_good
BOOLEAN, appearances INT, cid INT);
sqlite>
sqlite> INSERT INTO character (name, description, is_good, appearances,
cid) VALUES ('Batman', 'Bat-like super hero', 1, 1284, 1);
sqlite>
sqlite> INSERT INTO character (name, description, is_good, appearances,
cid) VALUES ('Thor', 'Hero from another planet', 1, 572, 2);
sqlite>
sqlite> INSERT INTO character (name, description, is_good, appearances,
cid) VALUES ('Superman', 'Hero from another planet', 1, 1752, 3);
```

SQL - full example

```
sqlite> SELECT * FROM character;
Batman|Bat-like super hero|1|1284|1
Thor|Hero from another planet|1|572|2
Superman|Hero from another planet|1|1752|3
sqlite>
```

SQL - full example

```
sqlite> SELECT description, name, is_good FROM character;
Bat-like super hero|Batman|1
Hero from another planet|Thor|1
Hero from another planet|Superman|1
sqlite>
```

SQL - full example

```
sqlite> SELECT name, description FROM character WHERE
description = 'Hero from another planet';
Thor|Hero from another planet
Superman|Hero from another planet
sqlite>

sqlite> SELECT name, description FROM character WHERE
description <> 'Hero from another planet';
Batman|Bat-like super hero
sqlite>
```

SQL - full example

```
sqlite> SELECT cid, name, description FROM character WHERE
cid = 1 OR cid = 2;
1|Batman|Bat-like super hero
2|Thor|Hero from another planet
sqlite>

sqlite> SELECT cid, name, description FROM character WHERE
cid = 1 AND cid = 2;
sqlite>
```

SQL - full example

```
$ sqlite3 ex1
SQLite version 3.8.5 2014-05-29 12:36:14 Enter ".help" for
usage hints.
sqlite> create table tbl1(one varchar(10), two smallint);
sqlite> insert into tbl1 values('hello!',10);
sqlite> insert into tbl1 values('goodbye', 20);
sqlite> select * from tbl1;
hello!|10
goodbye|20
sqlite>
```

SQL - specifying relationships

- We can create multiple tables in a single database
- cid, mid, and pid are supposed to reference IDs from other tables
- But there is nothing keeping us from putting invalid IDs in there!

```
CREATE TABLE character(
  name TEXT,
  cid INT,
  pid INT);

CREATE TABLE person(
  name TEXT,
  pid INT);

CREATE TABLE movie(
  title TEXT,
  mid INT);
```

SQL - specifying relationships

- We should specify a **PRIMARY KEY** for each table
- A primary key specifies which column uniquely identifies each row
- Often this is an integer ID, as it is in this case

```
CREATE TABLE character(
  name TEXT,
  cid INT PRIMARY KEY,
  pid INT);

CREATE TABLE person(
  name TEXT,
  pid INT PRIMARY KEY);

CREATE TABLE movie(
  title TEXT,
  mid INT PRIMARY KEY);
```

SQL - specifying relationships

- Relationships should be specified with a **FOREIGN KEY**
- Create a **FOREIGN KEY** from the ID in one table to a primary key in another table

```
CREATE TABLE character(
  name TEXT,
  cid INT PRIMARY KEY,
  pid INT,
  FOREIGN KEY(pid) REFERENCES
    person(pid));

CREATE TABLE person(
  name TEXT,
  pid INT PRIMARY KEY);

CREATE TABLE movie(
  title TEXT,
  mid INT PRIMARY KEY);
```

SQL - specifying relationships

- Use this command to make sure that FOREIGN KEY restrictions are enabled

```
PRAGMA foreign_keys = ON;
```

- In-class Exercise (picking up from previous lecture)
  - Model Ford Motor Company's manufacturing database
  - Ford manufactures cars at multiple locations in the US
    - Flat Rock Assembly Plant 1, Michigan
    - Chicago Assembly, Illinois
    - Dearborn Truck, Michigan
    - Kansas City Assembly, Missouri
  - Ford manufactures many types of cars
    - F-150, Mustang, Focus, Explorer, Flex, ...
  - Ford sells to many customers
    - Dealerships, Companies, Gov't, Individuals, ...

- In-class Exercise (picking up from previous lecture)
  - Tables: **Facility Item Purchase Customer**
  - Relationships to model
    - Facility <-> Item
    - Item <-> Purchase
    - Purchase <-> Customer

- In-class Exercise (picking up from previous lecture)
  - Define the schema with **CREATE** statements
    - Make sure to specify the **FOREIGN KEYS**
  - Insert a few rows into each table with **INSERT**

SQL and SQLite

- **Reading Materials**
  - www.w3schools.com/sql
  - www.tutorialspoint.com/sql