

CS 2316 Data Manipulation for Engineers

SQL

Christopher Simpkins

`chris.simpkins@gatech.edu`

Small. Fast. Reliable. Choose Any Three.

Using

SQLite



O'REILLY*

Jay A. Kreibich

1

¹The material in this lecture is taken from, [Using SQLite3, by Jay Kreibich](#), which is an excellent introduction to databases in general and SQLite in particular.

The SQL Language

Relational databases and the SQL language are inexorably tied.

- Issuing SQL commands is the way you create and manipulate data in relational databases.
- Proficiency in using relational databases means proficiency in SQL.

We'll learn the basics of SQL today, using SQLite:

- creating tables (`create table`),
- inserting new data into tables (`insert`),
- updating existing data in tables (`update`),
- deleting data from tables (`delete`), and
- querying tables (`select`).

Most of these commands are quite simple. `select` is like a language unto itself, so we'll spend most of our time with `select`.

create table

The basic syntax of the create table command is

```
CREATE TABLE table_name (  
    column_name column_type [column_constraints...],  
    [...] );
```

Here's an example

```
create table if not exists author (  
    author_id integer primary key autoincrement,  
    first_name text not null check(first_name != ''),  
    last_name text not null check(last_name != '')  
);
```

The names of the fields, or *columns* of this table are `author_id`, `first_name`, and `last_name`. Now we'll learn what the rest of the column definitions mean.

Column Types

- Attributes have data types, a.k.a. domains
- SQLite is manifest-typed, meaning it can store any type of value in any column, but most RDBMSes are statically typed
- SQLite has typed storage classes and type affinities for columns (a suggestion to convert data to specified type)
- SQLite supports the following type affinities:
 - text - NULL, text, or BLOB
 - numeric - integers, floats, NULLs, and BLOBs
 - integer - like numeric, but floats with no fractional part are converted to integers
 - float - like numeric, but integers are converted to floats
 - none - no preference over storage class

Column Constraints

```
create table if not exists author (  
    author_id integer primary key autoincrement,  
    first_name text not null check(first_name != ''),  
    last_name text not null check(last_name != '')  
);
```

- `primary key` means that the column uniquely identifies each record, or *row* in the table. Note that *integer* primary key fields are assigned automatically so that you don't have to assign them manually when inserting new rows into the table (we'll learn about composite keys later)
- `autoincrement` means that each new automatically assigned primary key will be greater than any existing primary key
- `not null` means the column cannot contain any `null` values
- `check(first_name != '')` means that you cannot insert an empty string into the field

Foreign Keys

One-to-one or one-to-many relationship can be established between tables using *foreign keys*. Consider the following tables.

```
create table if not exists venue (  
    venue_id integer primary key,  
    booktitle text not null not null check(booktitle != ''),  
    month text not null check(month != ''),  
    year integer not null  
);  
  
create table if not exists pub (  
    pub_id integer primary key,  
    title text not null check(title != ''),  
    venue_id integer not null references venue(venue_id)  
);
```

A single venue can have many publications.

- The references venue(venue_id) constraint on the venue_id field in pub table makes venue_id a foreign key which references the primary key, venue_id, of the venue table

Many-to-Many Relationships

A single author can write many publications, and a single publication can have many authors. This is a many-to-many relationship, which is modeled in relational databases with a *link table* or *bridge table*.

Consider:

```
create table if not exists author_pub (  
    author_id integer not null references author(author_id),  
    pub_id integer not null references publication(pub_id),  
    author_position integer not null, -- first author, second, etc?  
  
    primary key (author_id, pub_id)  
);
```

author_pub links the author and publication tables

- author_id and pub_id are both foreign keys
- Together, author_id and pub_id comprise a composite key for the table, which is created with the table level constraint primary key (author_id, pub_id)

Inserting Data

Inserting new rows into a table is done with the `insert` command:

```
insert into table_name [(column_name [, ...])]  
values (new_value [, ...]);
```

Can insert data without specifying column names, which means you must include all values, which are resolved by position in the table schema:

```
insert into author values (1, "Jenny", "McCarthy");
```

Or you can specify column names, which means the values are resolved by position relative to the list of column names:

```
insert into author (first_name, second_name) values ("Jenny",  
"McCarthy");
```

The second version is easier to maintain and allows auto-generated fields (like the primary key) to do their job

Updating Data

You update existing rows of a table using the `update` command:

```
update table_name set column_name=new_value [, ...]  
where expression;
```

Surely we meant Lisp inventor, AI co-founder, and Turing Award winner John McCarthy instead of anti-vaxxer Jenny McCarthy:

```
update author set first_name='John' where last_name='McCarthy';
```

Deleting Table Rows

We can delete table rows with the `delete` command:

```
delete from table_name where expression;
```

We can also delete an entire table with the `drop table` command:

```
drop table table_name;
```

Be careful with these commands, because they execute without any "are you sure?" message or indication of what will be deleted.

Querying a Database

Querying a database is done with the `select` command, whose general form is:

```
SELECT [DISTINCT] select_header FROM source_tables
WHERE filter_expression
GROUP BY grouping_expressions HAVING filter_expression
ORDER BY ordering_expressions
LIMIT count OFFSET count
```

- The table is the fundamental data abstraction in a relational database.
- The `select` command returns its result as a table
- Think of a `select` statement as creating a pipeline, each stage of which produces an intermediate *working table*

The `select` Pipeline

The evaluation order of select clauses is approximately:

- `FROM` *source_tables* - Designates one or more source tables and combines them together into one large working table.
- `WHERE` *filter_expression* - Filters specific rows of working table
- `GROUP BY` *grouping_expressions* - Groups sets of rows in the working table based on similar values
- `SELECT` *select_heading* - Defines the result set columns and (if applicable) grouping aggregates.
- `HAVING` *filter_expression* - Filters specific rows of the grouped table. Requires a `GROUP BY`
- `DISTINCT` - Eliminates duplicate rows.
- `ORDER BY` *ordering_expressions* - Sorts the rows of the result set
- `OFFSET` *count* - Skips over rows at the beginning of the result set. Requires a `LIMIT`.
- `LIMIT` *count* - Limits the result set output to a specific number of rows.

The `from` Clause

The `FROM` clause takes one or more source tables from the database and combines them into one (large) table using the `JOIN` operator.

Three kinds of joins:

- `CROSS JOIN`
- `INNER JOIN`
- `OUTER JOIN`

The join is the most important part of a query. We'll discuss cross joins and inner joins, the most important and common joins.

To make our discussion of queries concrete, we'll see examples using a the publication database. Download [create-pubs.sql](#) and [seed-pubs.sql](#) to play along. The query examples are in [query-pubs.sql](#)

Simple selects

The simplest `select` query returns all rows from a single table:

```
sqlite> select * from author;
```

author_id	first_name	last_name
1	John	McCarthy
2	Dennis	Ritchie
3	Ken	Thompson
4	Claude	Shannon
5	Alan	Turing
6	Alonzo	Church

Notice the `*` character in the select header. It means return all columns.

Cross Joins

A `CROSS JOIN` matches every row of the first table with every row of the second table. Think of a cross join as a cartesian product.

The general syntax for a cross join is:

```
SELECT select_header FROM table1 CROSS JOIN table2
```

```
sqlite> select * from pub cross join venue;
```

pub_id	title	venue_id	venue_id	booktitle	month	year
1	Recursive Funct	1	1	Communications	April	1960
1	Recursive Funct	1	2	Communications	July	1974
1	Recursive Funct	1	3	Bell System Tec	July	1948
1	Recursive Funct	1	4	Proceedings of	November	1936
1	Recursive Funct	1	5	Mind	October	1950
1	Recursive Funct	1	6	Annals of Mathe	Month	1941
2	The Unix Time-s	2	1	Communications	April	1960
2	The Unix Time-s	2	2	Communications	July	1974
2	The Unix Time-s	2	3	Bell System Tec	July	1948
2	The Unix Time-s	2	4	Proceedings of	November	1936
2	The Unix Time-s	2	5	Mind	October	1950
2	The Unix Time-s	2	6	Annals of Mathe	Month	1941
...						

Inner Joins

A simple inner join uses an `on` condition.

```
sqlite> select * from pub join venue on pub.venue_id = venue.venue_id;
```

pub_id	title	venue_id	venue_id	booktitle	month	year
1	Recursive Functions of	1	1	Communications	April	1960
2	The Unix Time-sharing	2	2	Communications	July	1974
3	A Mathematical Theory	3	3	Bell System Te	July	1948
4	On computable numbers,	4	4	Proceedings of	November	1936
5	Computing machinery and	5	5	Mind	October	1950
6	The calculi of lambda-	6	6	Annals of Math	Month	1941

Notice that `venue_id` appears twice, because we get one from each source table. We can fix that ...

Join Using

The `using` clause, also called a natural join, joins on a like-named column from each table and includes it only once.

```
sqlite> .width 6 15 10 15 6 6
sqlite> select * from pub join venue using (venue_id);
```

pub_id	title	venue_id	booktitle	month	year
1	Recursive Funct	1	Communications	April	1960
2	The Unix Time-s	2	Communications	July	1974
3	A Mathematical	3	Bell System Tec	July	1948
4	On computable n	4	Proceedings of	Novemb	1936
5	Computing machi	5	Mind	Octobe	1950
6	The calculi of	6	Annals of Mathe	Month	1941

Notice how we limited the column width in the printout with the `.width` command.

Select Headers

We can limit the columns returned in the final result

```
sqlite> select title, year from pub join venue using (venue_id);
title                                year
-----
Recursive Functions of Symbolic ... 1960
The Unix Time-sharing System          1974
A Mathematical Theory of Communicat 1948
On computable numbers, with an appl 1936
Computing machinery and intelligenc 1950
The calculi of lambda-conversion     1941
```

The `select title, year` means that the result table will only have those two columns.

Joining Multiple Tables

Remember that we linked the `author` table to the `pub` table using the `author_pub` table to model the many-to-many relationship between authors and publications. We can join all three tables by chaining `join` clauses:

```
sqlite> select * from author join author_pub using (author_id) join pub using (pub_id);
```

author_id	first_name	last_name	pub_id	author_position	title	venue
1	John	McCarthy	1	1	Recursive	1
2	Dennis	Ritchie	2	1	The Unix	2
3	Ken	Thompson	2	2	The Unix	2
4	Claude	Shannon	3	1	A Mathem	3
5	Alan	Turing	4	1	On compu	4
5	Alan	Turing	5	1	Computin	5
6	Alonzo	Church	6	1	The calc	6

where Clauses

We can filter the results using a `where` clause.
"Which papers did Turing write?"

```
sqlite> select first_name, last_name, title
...>   from author join author_pub using (author_id) join pub using (pub_id)
...>   where last_name = 'Turing';
first_name  last_name  title
-----
Alan        Turing     On computable numbers, with an application ro the Entscheidu
Alan        Turing     Computing machinery and intelligence
```

Here we get only the publications by Turing.

like and distinct

Our `where` condition can match a pattern with `like`. Use a `%` for wildcard, i.e., matching any character sequence.

"Which papers did Turing write?"

```
sqlite> select booktitle from venue where booktitle like 'Communications%';
booktitle
-----
Communications of the ACM
Communications of the ACM
```

Notice that the query returned both entries (two different months). We can get distinct rows by adding `distinct` before the select header:

```
sqlite> select distinct booktitle from venue where booktitle like 'Communications%';
booktitle
-----
Communications of the ACM
```

The `glob` operator is like the `like` operator, but uses `*` as its wildcard character.

group by and Aggregate Functions

The `group by` clause groups the results in the working table by the specified column value

- we can apply an aggregate function to the resulting groups
- if we don't apply an aggregate function, only the last row of a group is returned
 - Since the order of rows unspecified, failing to apply an aggregate function would essentially give us a random result

"How many papers did each author write?"

```
sqlite> select author_id, last_name, count(author_id)
...> from author join author_pub using (author_id) join pub using (pub_id)
...> group by author_id;
```

author_id	last_name	count(author_id)
1	McCarthy	1
2	Ritchie	1
3	Thompson	1
4	Shannon	1
5	Turing	2
6	Church	1

Sorting, Aliasing, and Limiting

We can use the `order by` clause to sort the result table.
"Who wrote the most papers?"

```
sqlite> select author_id, last_name, count(author_id) as pub_count
...>      from author join author_pub using (author_id) join pub using (pub_id)
...>      group by author_id
...>      order by pub_count desc
...>      limit 1;
author_id  last_name  pub_count
-----
5          Turing      2
```

Notice that we also used an alias so we could reference the count in the `order by` clause, we sorted in descending order, and we limited the output to 1 so we got only the top result.

Query Exercises

Download the [dorms.sql](#) script, which creates and seeds a database of students and dorms. Create a SQLite database with it and write queries to answer the following questions:

- What are the names of all the dorms?
- What is the total capacity (number of spaces) for all dorms?
- Which students are in Armstrong?
- Which student has the highest GPA?
- Which dorm has the most students?
- Which dorm's students have the highest average GPA?

Closing Thoughts

- Relational databases are pervasive
- A firm grasp of SQL is essential to mastering databases
- You now have a good start
- Believe it or not, we've only scratched the surface.