Intro to SQL

Focusing on Sqlite

Richard Morrill

Fordham University CS Society

Wednesday, February 26th, 2020

Game Plan

- Installing Sqlite
- Why you need SQL
- What is SQL
- What is Sqlite
- Basics of SQL syntax and database design
- How adding a SQL[1] database to an existing project can simplify your code

[1] If pronounced "sequel", "a" is the correct article.

CS SOCIETY
FORDHAM UNIVERSITY

Let's get it Installed!

Mac and Linux

There's a 99% chance it's already installed. Open a terminal window and try `sqlite3`, and if that doesn't work, `sqlite`. If for some reason your OS didn't come with it, install it at `https://www.sqlite.org`

Windows

You could have unknowingly installed Sqlite when installing something else. Give it a try in Powershell or CMD, but it's likley you'll need to install Sqlite at: `https://www.sqlite.org`

If it works you will see:
(The specific version does not matter.)

```
SQLite version 3.30.0 2019-10-04 15:03:17
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>
```

What's the point of this, anyways?

Non-CS majors that don't know SQL say, "Can't I just use a GUI tool like MS Access?"

CS majors that don't know SQL say, "There's plenty of tools in the language I'm already using, why do I need another one?"

The power of SQL

- SQL is a *declarative language*.

The power of SQL

- SQL is a *declarative language*.You don't need to worry about *how* the database engine is going to do something, just tell it what you *want it to do*.

The power of SQL

- SQL is a *declarative language*.You don't need to worry about *how* the database engine is going to do something, just tell it what you *want it to do*.
  - SQL lets you think about your data on a higher level than if you were dealing with it in objects and arrays.
  - Learning to think *declaratively* can help you write more readable code, even when working in languages other than SQL.

The power of SQL

- SQL is a *declarative language*.You don't need to worry about *how* the database engine is going to do something, just tell it what you *want it to do*.
  - SQL lets you think about your data on a higher level than if you were dealing with it in objects and arrays.
  - Learning to think *declaratively* can help you write more readable code, even when working in languages other than SQL.
- Even if your language provides its own set of data-manipulation tools, they're probably just different names for things SQL already does.

The power of SQL

- SQL is a *declarative language*.You don't need to worry about *how* the database engine is going to do something, just tell it what you *want it to do*.
  - SQL lets you think about your data on a higher level than if you were dealing with it in objects and arrays.
  - Learning to think *declaratively* can help you write more readable code, even when working in languages other than SQL.
- Even if your language provides its own set of data-manipulation tools, they're probably just different names for things SQL already does.
  - If you learn to use the real thing, you'll be able to figure out pretty much any pseudo-sql stuff you come across.

The power of SQL

- SQL is a *declarative language*.You don't need to worry about *how* the database engine is going to do something, just tell it what you *want it to do*.
  - SQL lets you think about your data on a higher level than if you were dealing with it in objects and arrays.
  - Learning to think *declaratively* can help you write more readable code, even when working in languages other than SQL.
- Even if your language provides its own set of data-manipulation tools, they're probably just different names for things SQL already does.
  - If you learn to use the real thing, you'll be able to figure out pretty much any pseudo-sql stuff you come across.
- SQL gets to the point.

A Bit About SQL

- *Structured Query language*

A Bit About SQL

- *Structured Query language*
- Developed by IBM in the 1970s

CS SOCIETY
FORDHAM UNIVERSITY

A Bit About SQL

- *Structured Query language*
- Developed by IBM in the 1970s
  - Like other languages developed by committee, has its weird bits.
  - Syntax isn't always consistent with itself.

A Bit About SQL

- *Structured Query language*
- Developed by IBM in the 1970s
  - Like other languages developed by committee, has its weird bits.
  - Syntax isn't always consistent with itself.
- The gold standard for managing *relational databases*.

CS SOCIETY
FORDHAM UNIVERSITY

A Bit About SQL

- *Structured Query language*
- Developed by IBM in the 1970s
  - Like other languages developed by committee, has its weird bits.
  - Syntax isn't always consistent with itself.
- The gold standard for managing *relational databases*.
  - A bit of a victim of its popularity, SQL isn't always implemented identically across database software from different vendors.
  - 90% of syntax is the same, but some things you "get away with" on one database you won't in others.

A Bit About SQL

- *Structured Query language*
- Developed by IBM in the 1970s
  - Like other languages developed by committee, has its weird bits.
  - Syntax isn't always consistent with itself.
- The gold standard for managing *relational databases*.
  - A bit of a victim of its popularity, SQL isn't always implemented identically across database software from different vendors.
  - 90% of syntax is the same, but some things you "get away with" on one database you won't in others.
- Technically a fully-fledged programming language, although rarely used this way[2].

[2] With one important exception we'll see later.

Relational Databases

Very few pieces of data exist in a vacuum

Relational Databases

Very few pieces of data exist in a vacuum

Department

Name	Manager	Building

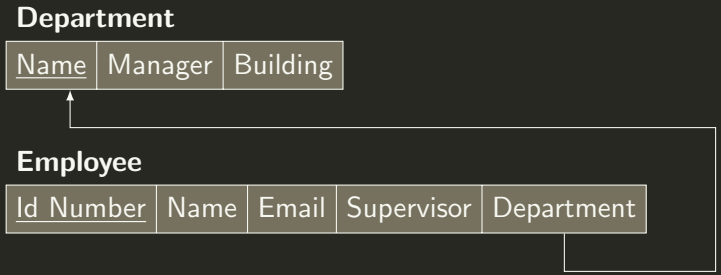
Employee

Id Number	Name	Email	Supervisor	Department

Insert about 100 more tables here . . .

Relational Databases

Very few pieces of data exist in a vacuum

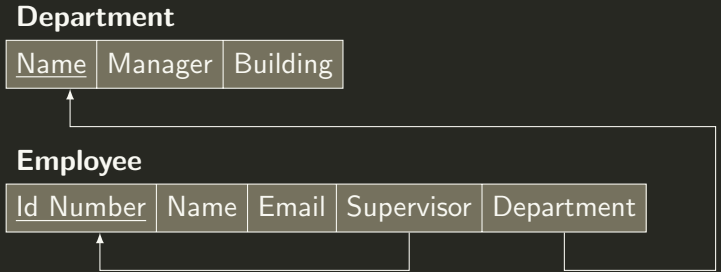


Insert about 100 more tables here . . .

CS SOCIETY
FORDHAM UNIVERSITY

Relational Databases

Very few pieces of data exist in a vacuum



Insert about 100 more tables here . . .

CS SOCIETY
FORDHAM UNIVERSITY

Consistency

- Consistency is key to making relational systems work smoothly.
  - Imagine what would happen if half of the employees called their department “facilities”, while the other half called it “maintenance”.

Consistency

- Consistency is key to making relational systems work smoothly.
  - Imagine what would happen if half of the employees called their department “facilities”, while the other half called it “maintenance”.
- With many different sources of data and users, maintaining consistency can be very difficult.

Consistency

- Consistency is key to making relational systems work smoothly.
  - Imagine what would happen if half of the employees called their department "facilities", while the other half called it "maintenance".
- With many different sources of data and users, maintaining consistency can be very difficult.
  - You can educate users $\rightarrow$ Like that's gonna happen...

Consistency

- Consistency is key to making relational systems work smoothly.
  - Imagine what would happen if half of the employees called their department "facilities", while the other half called it "maintenance".
- With many different sources of data and users, maintaining consistency can be very difficult.
  - You can educate users $\rightarrow$ Like that's gonna happen...
  - You can write code that "gaurds" your data store $\rightarrow$ Closer, but what if you make a mistake?

Consistency

- Consistency is key to making relational systems work smoothly.
  - Imagine what would happen if half of the employees called their department "facilities", while the other half called it "maintenance".
- With many different sources of data and users, maintaining consistency can be very difficult.
  - You can educate users $\rightarrow$ Like that's gonna happen...
  - You can write code that "gaurds" your data store $\rightarrow$ Closer, but what if you make a mistake?
- SQL builds consistency protection directly into the data storage[3].

[3] This is assuming the SQL database is set up properly.

CS SOCIETY
FORDHAM UNIVERSITY

Consistency

- Consistency is key to making relational systems work smoothly.
  - Imagine what would happen if half of the employees called their department "facilities", while the other half called it "maintenance".
- With many different sources of data and users, maintaining consistency can be very difficult.
  - You can educate users $\rightarrow$ Like that's gonna happen...
  - You can write code that "gaurds" your data store $\rightarrow$ Closer, but what if you make a mistake?
- SQL builds consistency protection directly into the data storage[3].
  - Bad data is rejected... No Exceptions!
  - Strict column types
  - Foreign keys: Can't insert an employee if his department is non-existent.
  - Triggers: Arbitrary code that can stop mistakes in their tracks.

[3] This is assuming the SQL database is set up properly.

CS SOCIETY
FORDHAM UNIVERSITY

Basics of SQL: Making a Table

1. Open a terminal.
2. Navigate to a directory you can use for temporary files.
3. Type in `$ sqlite demo.db`
4. On the `sqlite>` prompt that will come up, enter the commands below.

IN SQL WE ALWAYS SHOUT!

```
CREATE TABLE "employee" (
    "id" INTEGER PRIMARY KEY,
    "name" VARCHAR,
    "dob" VARCHAR,
    "joinDate" VARCHAR
);
```

To see if it worked:

```
.tables
.schema "employee"
```

CS SOCIETY
FORDHAM UNIVERSITY

Basics of SQL: Inserting and Reading Data

To make things more readable:

```
.mode column
.nullvalue NULL
.headers on
```

Add Some Data:

```
1 INSERT INTO "employee"
2     VALUES (12, "Sam Smith", "1980-05-06", "2008-06-22");
3 INSERT INTO "employee" (id, "dob")
4     VALUES (44, "1968-04-18");
```

Read It Back:

```
1 SELECT * FROM "employee";
2 SELECT "id", "name" FROM "employee";
```

Basics of SQL: Inserting and Reading Data

WHERE Conditions and ORDER BY

By default, SELECT returns all rows.

Basics of SQL: Inserting and Reading Data

WHERE Conditions and ORDER BY

By default, SELECT returns all rows. While you can simply iterate through the results in your own code to get the ones you want, there is a better way:

```
1 SELECT * FROM "employee" WHERE id = 12;
2 SELECT * FROM "employee" WHERE id = 44 LIMIT 1;
```

There are many different conditions available in SQL.

Basics of SQL: Inserting and Reading Data

WHERE Conditions and ORDER BY

By default, SELECT returns all rows. While you can simply iterate through the results in your own code to get the ones you want, there is a better way:

```
1 SELECT * FROM "employee" WHERE id = 12;
2 SELECT * FROM "employee" WHERE id = 44 LIMIT 1;
```

There are many different conditions available in SQL. The database engine is also very efficient at sorting data:

```
1 SELECT * FROM "Employee" ORDER BY "dob" DESC;
```

How could we get the oldest employee?

Basics of SQL: Inserting and Reading Data

WHERE Conditions and ORDER BY

By default, SELECT returns all rows. While you can simply iterate through the results in your own code to get the ones you want, there is a better way:

```
1 SELECT * FROM "employee" WHERE id = 12;
2 SELECT * FROM "employee" WHERE id = 44 LIMIT 1;
```

There are many different conditions available in SQL. The database engine is also very efficient at sorting data:

```
1 SELECT * FROM "Employee" ORDER BY "dob" DESC;
```

How could we get the oldest employee?

```
1 SELECT * FROM "employee" ORDER BY "dob" ASC LIMIT 1;
```

Note that there is a MAX() function in SQL, but it probably won't work the way you expect.

Basics of SQL: Deleting Data

Deleting in SQL is pretty intuitive:

```
1 DELETE FROM "employee";
```

What just got deleted?

Basics of SQL: Deleting Data

Deleting in SQL is pretty intuitive:

```
1 DELETE FROM "employee";
```

What just got deleted?
If you want to delete specific records, you need to use WHERE:

```
1 DELETE FROM "employee" WHERE id = 12;
```

CS SOCIETY
FORDHAM UNIVERSITY

Basics of SQL: Deleting Data

Deleting in SQL is pretty intuitive:

```
1 DELETE FROM "employee";
```

What just got deleted?
If you want to delete specific records, you need to use WHERE:

```
1 DELETE FROM "employee" WHERE id = 12;
```

To delete a table, you use DROP not DELETE:

```
1 DROP TABLE "employee";
```

The convention of using DELETE for *data* and DROP for *structures* is widely used in SQL.

Basics of SQL: Constraints

A constraint is anything you set up in that controls what data may be put where.

Basics of SQL: Constraints

A constraint is anything you set up in that controls what data may be put where.

- Primary Keys
  - Must be unique
  - *Tables exist to store primary keys, the other fields describe the primary key.*
  - Database engine optimizes queries involving primary key

Basics of SQL: Constraints

A constraint is anything you set up in that controls what data may be put where.

- Primary Keys
  - Must be unique
  - *Tables exist to store primary keys, the other fields describe the primary key.*
  - Database engine optimizes queries involving primary key
- Foreign Keys
  - The primary key[4] of another table, stored as a regular column in your table
  - Database engine will enforce the validity of whatever value you attempt to insert (e.g. can't insert an employee with a nonexistent department)
  - Advanced Topic: You can control what happens to dependant rows on delete

[4] Sometimes it can be another field, but this is not recommended

CS SOCIETY
FORDHAM UNIVERSITY

Basics of SQL: Constraints

A constraint is anything you set up in that controls what data may be put where.

- Primary Keys
  - Must be unique
  - *Tables exist to store primary keys, the other fields describe the primary key.*
  - Database engine optimizes queries involving primary key
- Foreign Keys
  - The primary key[4] of another table, stored as a regular column in your table
  - Database engine will enforce the validity of whatever value you attempt to insert (e.g. can't insert an employee with a nonexistent department)
  - Advanced Topic: You can control what happens to dependant rows on delete
- Other Keys / Indices
  - The terms "key" and "index" have overlapping meanings
  - You can arbitrarily enforce uniqueness on any column
  - You can tell the database engine to optimize queries involving any column

[4] Sometimes it can be another field, but this is not recommended

Foreign Keys

First, enable foreign keys:

```
PRAGMA foreign_keys = ON;
```

Let's make a table with a foreign key:

```
CREATE TABLE customer (
    id INTEGER PRIMARY KEY,
    name VARCHAR,
    contact INTEGER,
    FOREIGN KEY (contact) REFERENCES employee(id)
);
```

Foreign Keys

First, enable foreign keys:

```
PRAGMA foreign_keys = ON;
```

Let's make a table with a foreign key:

```
CREATE TABLE customer (
    id INTEGER PRIMARY KEY,
    name VARCHAR,
    contact INTEGER,
    FOREIGN KEY (contact) REFERENCES employee(id)
);
```

Now try to insert a customer whose contact has Id 99

Joins