

SQL: (use sqlite, Download and Install: <http://sqlitebrowser.org/>)

Structured Query Language is the language we use to issue commands to the database

-Create data(insert):

The screenshot shows the DB Browser for SQLite interface. At the top, there's a toolbar with options like 'New Database', 'Open Database', 'Write Changes', 'Revert Changes', 'Open Project', etc. A red arrow points to the 'New Database' button. Below the toolbar, the main window has tabs for 'Database Structure' and 'Browse Data'. A sub-menu under 'Database Structure' is open, showing 'Edit Pragmas' and 'Execute SQL'. A red arrow points to the 'Edit Pragmas' option. In the center, there's a text input field containing 'kf@umich.edu'. Below it, a message says 'Type of data currently in cell: Text / Numeric' and '12 character(s)'. To the right, there's a 'Mode' dropdown set to 'Text' and a 'Remote' section with a 'Select an identity to connect' dropdown and a table view.

create a New Database

User Advanced **table name**

Add **Remove** **Move...top** **Move up** **Move down** **Move..tton**

Name **Type** **NN** **PK** **AI** **U** **Default** **Check**

add Names, type by click this

1 **CREATE TABLE "User"** (

2 **L**

3 **);**

Cancel OK

Then insert the values:

The screenshot shows two windows of DB Browser for SQLite. The top window is in 'Edit Database Cell' mode, displaying a table named 'Users' with columns 'name' and 'email'. A red arrow points to the 'Edit Database Cell' button in the toolbar. The bottom window is an 'Add New Record' dialog for the 'Users' table, showing fields for 'Name' and 'Type' with values 'Chuck' and 'csev@umich.edu' respectively. Red arrows point to these values. Below the dialog is a SQL log window showing the generated SQL code:

```

1 INSERT INTO "main"."Users"
2 ("name", "email")
3 VALUES ('Chuck', 'csev@umich.edu');

```

At the bottom of the dialog are buttons for Help, Restore Defaults, Cancel, and Save.

Some Steps to Create SQL Tables

CREATE TABLE "Users" ("name" TEXT, "email" TEXT)

INSERT INTO Users (name, email) VALUES ('Chuck', 'csev@umich.edu')

```

INSERT INTO Users (name, email) VALUES ('Colleen', 'cvl@umich.edu')
INSERT INTO Users (name, email) VALUES ('Ted', 'ted@umich.edu')
INSERT INTO Users (name, email) VALUES ('Sally', 'a1@umich.edu')
INSERT INTO Users (name, email) VALUES ('Ted', 'ted@umich.edu')
INSERT INTO Users (name, email) VALUES ('Kristen', 'kf@umich.edu')

```

After inserting all names and emails, get this:

The screenshot shows a database interface with a toolbar at the top. The 'Table' dropdown is set to 'Users'. Below it is a grid showing the following data:

	name	email
1	Chuck	csev@umich.edu
2	Colleen	cvl@umich.edu
3	Ted	ted@umich.edu
4	Sally	a1@umich.edu
5	Ted	ted@umich.edu
6	Kristen	kf@umich.edu

Also, we could insert values through execute SQL by typing

: **INSERT INTO Users (name, email) VALUES ('Alice', 'AL@umich.edu')** through **Execute SQL**

The screenshot shows a database interface with a toolbar at the top. The 'Execute SQL' tab is selected. Below it is a text input field containing the following SQL query:

```
1 INSERT INTO Users (name, email) VALUES ('Alice', 'AL@umich.edu')
```

A red arrow points to the execute button (a blue square with a white play icon) in the toolbar.

Now the table looks like(from Browse Data):

Table: Users

	name	email
1	Chuck	csev@umich.edu
2	Colleen	cvl@umich.edu
3	Ted	ted@umich.edu
4	Sally	a1@umich.edu
5	Ted	ted@umich.edu
6	Kristen	kf@umich.edu
7	Alice	AL@umich.edu

Delete: we could delete a row in a table based on selection criteria by typing:

DELETE FROM Users WHERE email='ted@umich.edu' through **Execute SQL**

Execute SQL

```
1 DELETE FROM Users WHERE email='ted@umich.edu'
```

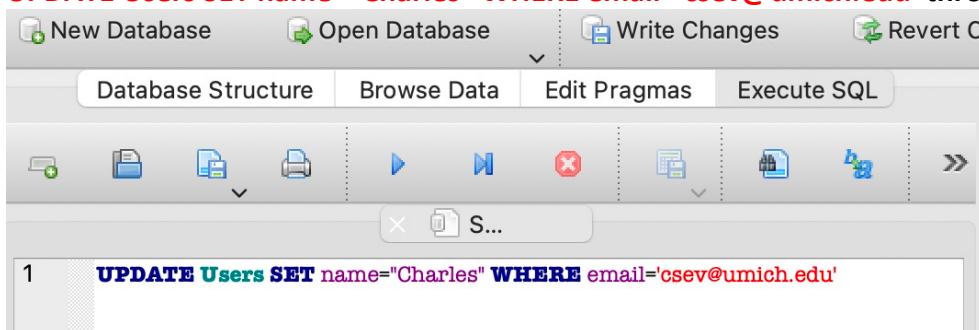
Now the table looks like:

Table: Users

	name	email
1	Chuck	csev@umich.edu
2	Colleen	cvl@umich.edu
3	Sally	a1@umich.edu
4	Kristen	kf@umich.edu
5	Alice	AL@umich.edu

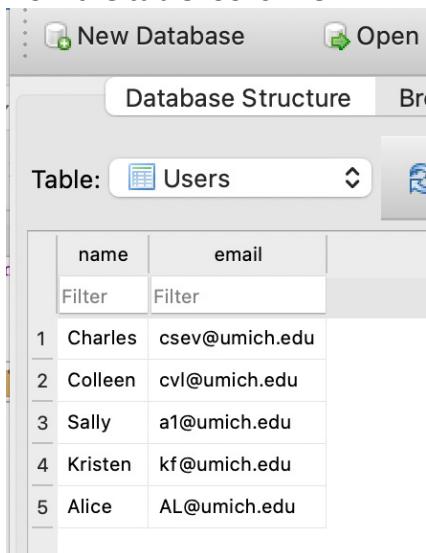
Update: Allow the updating of a field with a where clause by typing:

UPDATE Users SET name="Charles" WHERE email='csev@umich.edu' through Execute SQL



The screenshot shows the SQLite Database Browser interface. The top menu bar includes 'New Database', 'Open Database', 'Write Changes', and 'Revert C...'. Below the menu is a toolbar with icons for creating, opening, writing changes, reverting, and other database operations. A tab bar at the top right has tabs for 'Database Structure', 'Browse Data', 'Edit Pragmas', and 'Execute SQL', with 'Execute SQL' being the active tab. The main window contains a single line of SQL code: '1 UPDATE Users SET name="Charles" WHERE email='csev@umich.edu''. The background of the main window is light gray.

Now the table looks like:



The screenshot shows the SQLite Database Browser interface with the 'Browse Data' tab selected. The 'Table' dropdown is set to 'Users'. The main area displays a table with two columns: 'name' and 'email'. The table contains five rows of data:

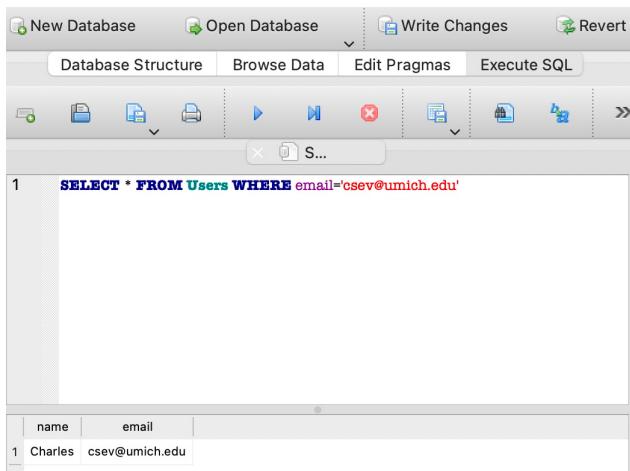
	name	email
1	Charles	csev@umich.edu
2	Colleen	cvl@umich.edu
3	Sally	a1@umich.edu
4	Kristen	kf@umich.edu
5	Alice	AL@umich.edu

You can see that Chuck changed into Charles

Retrieving Records: Select

The select statement retrieves a group of records -you can either retrieve all the records or a subset of the records with a WHERE clause by typing:

SELECT * FROM Users WHERE email='csev@umich.edu' through Execute SQL:



The screenshot shows the SQLite Database Browser interface with the 'Execute SQL' tab selected. The main window contains a single line of SQL code: '1 SELECT * FROM Users WHERE email='csev@umich.edu''. The background of the main window is light gray.

could see under there is a table that selected

SELECT * FROM Users through Execute SQL:

The screenshot shows a software interface for managing databases. At the top, there are buttons for 'New Database' and 'Open Database'. Below that is a toolbar with various icons. A main window contains a SQL query and its results. The SQL query is:

```
1  SELECT * FROM Users
```

The results are displayed in a table with two columns: 'name' and 'email'. The data is as follows:

	name	email
1	Charles	csev@umich.edu
2	Colleen	cvl@umich.edu
3	Sally	a1@umich.edu
4	Kristen	kf@umich.edu
5	Alice	AL@umich.edu

SELECT * FROM Users ORDER BY email through Execute SQL:

The screenshot shows a software interface for managing databases. At the top, there are buttons for 'New Database' and 'Open Database'. Below that is a toolbar with various icons. A main window contains a SQL query and its results. The SQL query is:

```
1  SELECT * FROM Users ORDER BY email
```

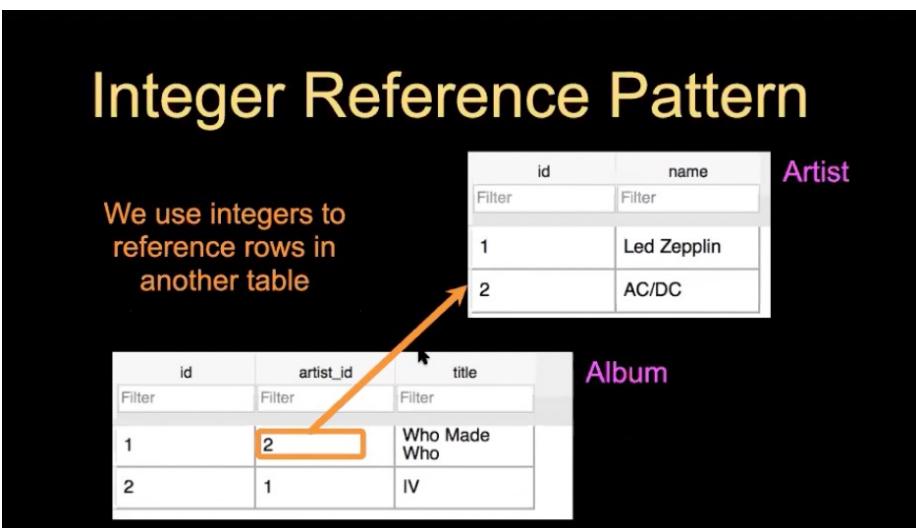
The results are displayed in a table with two columns: 'name' and 'email'. The data is as follows:

	name	email
1	Alice	AL@umich.edu
2	Sally	a1@umich.edu
3	Charles	csev@umich.edu
4	Colleen	cvl@umich.edu
5	Kristen	kf@umich.edu

SELECT * FROM Users ORDER BY name DESC through Execute SQL:

```
1   SELECT * FROM Users ORDER BY name DESC
```

	name	email
1	Sally	a1@umich.edu
2	Kristen	kf@umich.edu
3	Colleen	cvl@umich.edu
4	Charles	csev@umich.edu
5	Alice	AL@umich.edu



Primary key: generally an integer auto-increment field

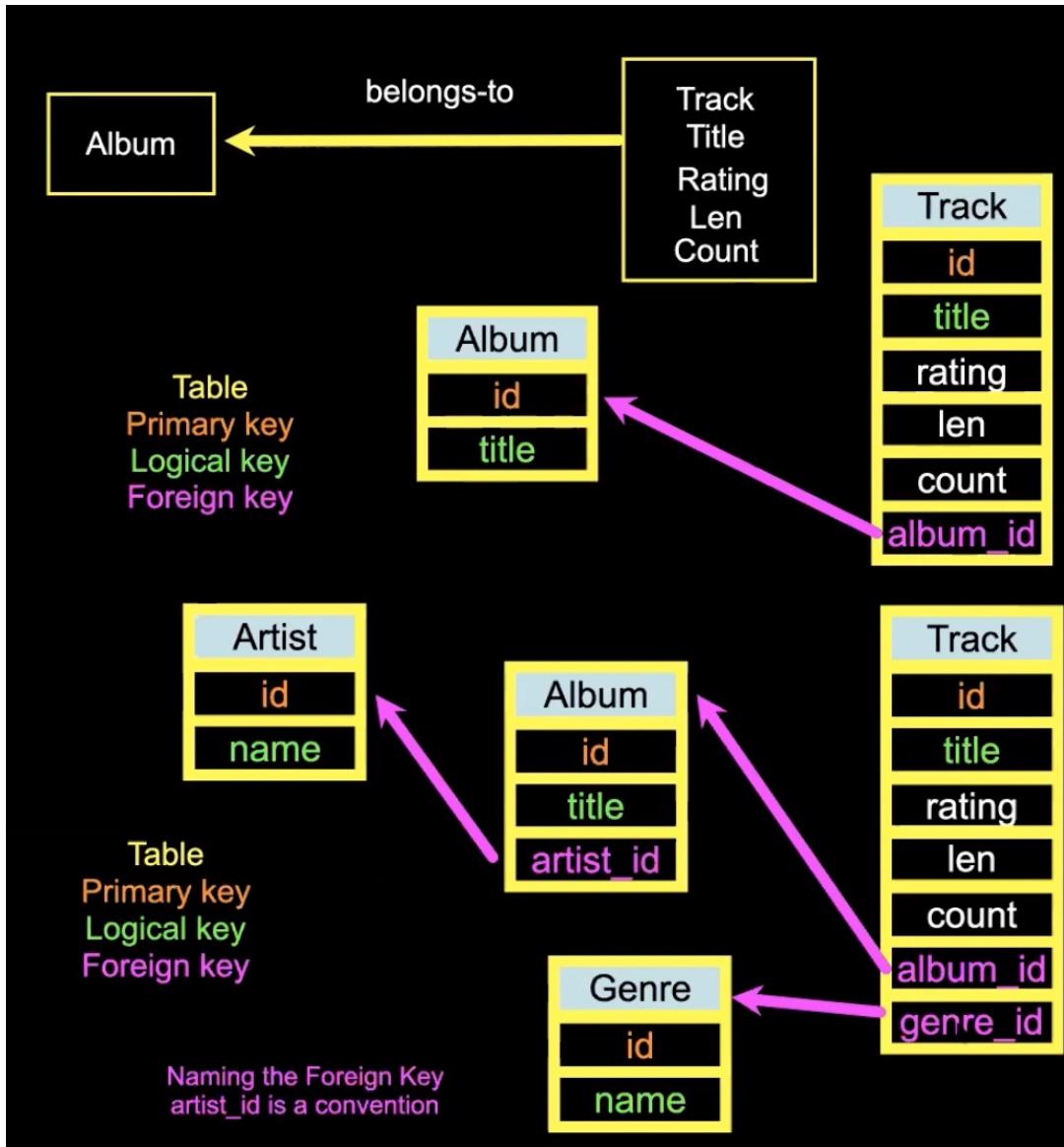
Logical key: what the outside world uses for lookup

Foreign key: generally an integer key pointing to a row in another table

Three Kinds of Keys

- Primary key - generally an integer auto-increment field
- Logical key - What the outside world uses for lookup
- Foreign key - generally an integer key pointing to a row in another table

Album
id
title
artist_id
...



Create multiple tables:

Some Steps to Create multiple SQL Tables:

Multi-Table SQL:

```
CREATE TABLE "Artist" (
  "id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL UNIQUE,
  "name" TEXT)
```

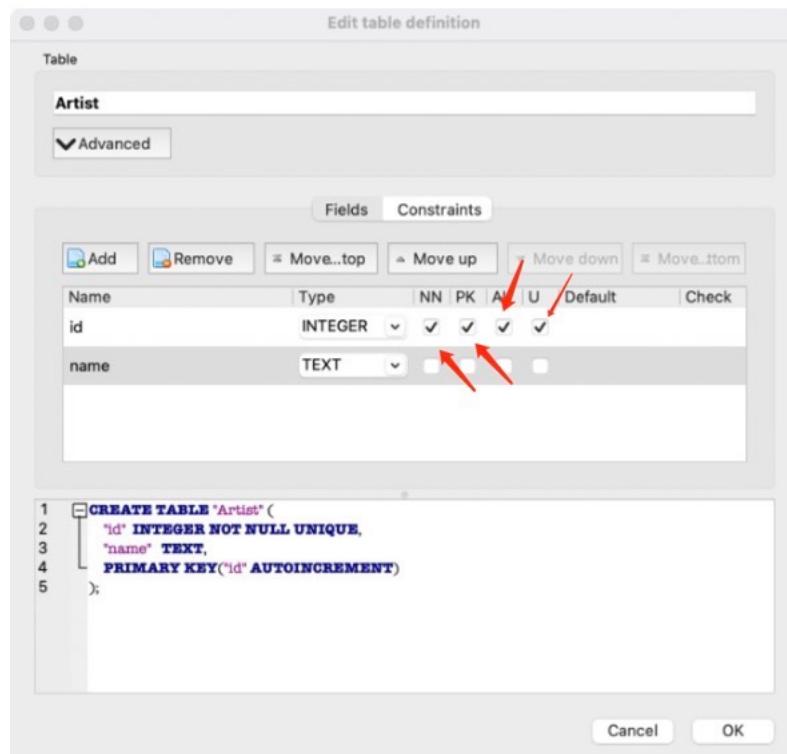
```
CREATE TABLE "Album" (
  "id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL UNIQUE,
  artist_id INTEGER,
```

```
"title" TEXT)
```

```
CREATE TABLE "Genre" (
    "id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL UNIQUE,
    "name" TEXT)
```

```
CREATE TABLE "Track" (
    "id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL UNIQUE,
    album_id INTEGER, genre_id INTEGER, len INTEGER, rating INTEGER,
    "title" TEXT, "count" INTEGER)
```

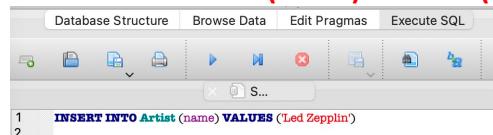
```
CREATE TABLE "Artist" (
    "id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL UNIQUE,
    "name" TEXT)
```



Create others follow above text...

Then insert

INSERT INTO Artist (name) VALUES ('Led Zeppelin') through Execute SQL:



Browse Data:

Database Structure Browse Data Edit Prag

Table: Artist

id	name
1	Led Zeppelin

INSERT INTO Artist (name) VALUES ('AC/DC')

Browse data:

Database Structure Browse Data

Table: Artist

id	name
1	Led Zeppelin
2	AC/DC

DB Browser for SQLite - /Users/csev/Desktop/Music

New Database Open Database Write Changes Revert Changes

Database Structure Browse Data Edit Pragmas Execute SQL

Table: Album New Record Delete Record

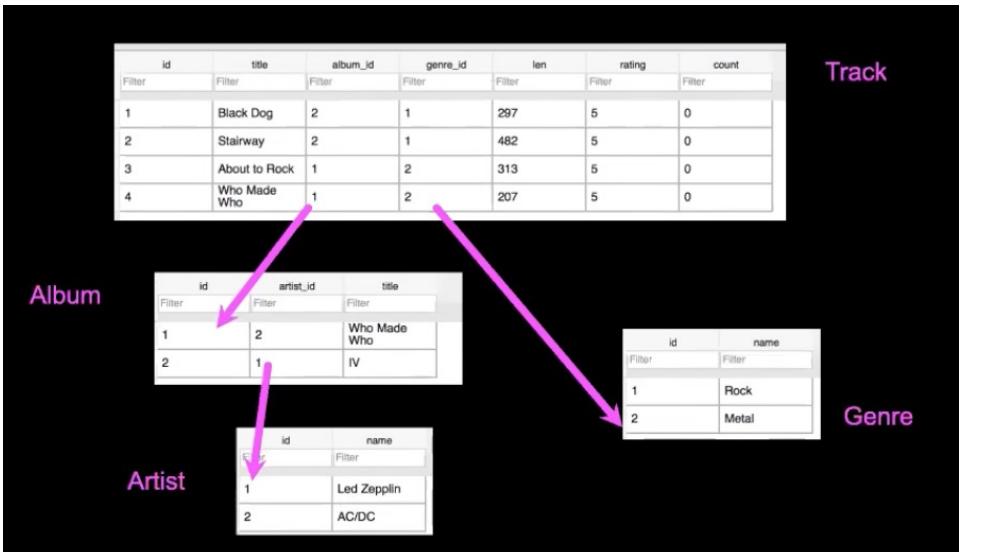
id	artist_id	title
1	1	Who Made Who
2	2	IV

< < 1 - 2 of 2 > > Go to: 1 UTF-8

insert into Album (title, artist_id) values ('Who Made Who', 2)
 insert into Album (title, artist_id) values ('IV', 1)

```
insert into Track (title, rating, len, count, album_id, genre_id)
  values ('Black Dog', 5, 297, 0, 2, 1)
insert into Track (title, rating, len, count, album_id, genre_id)
  values ('Stairway', 5, 482, 0, 2, 1)
insert into Track (title, rating, len, count, album_id, genre_id)
  values ('About to Rock', 5, 313, 0, 1, 2)
insert into Track (title, rating, len, count, album_id, genre_id)
  values ('Who Made Who', 5, 207, 0, 1, 2)
```

id	title	album_id	genre_id	len	rating	count
1	Black Dog	2	1	297	5	0
2	Stairway	2	1	482	5	0
3	About to Rock	1	2	313	5	0
4	Who Made Who	1	2	207	5	0



Summary

- Relational databases allow us to **scale** to very large amounts of data
- The key is to have **one copy of any data element** and use relations and joins to link the data to multiple places
- This greatly **reduces the amount of data which must be scanned** when doing complex operations across large amounts of data
- Database and SQL design is a bit of an **art form**