Databases

Barbara Ericson

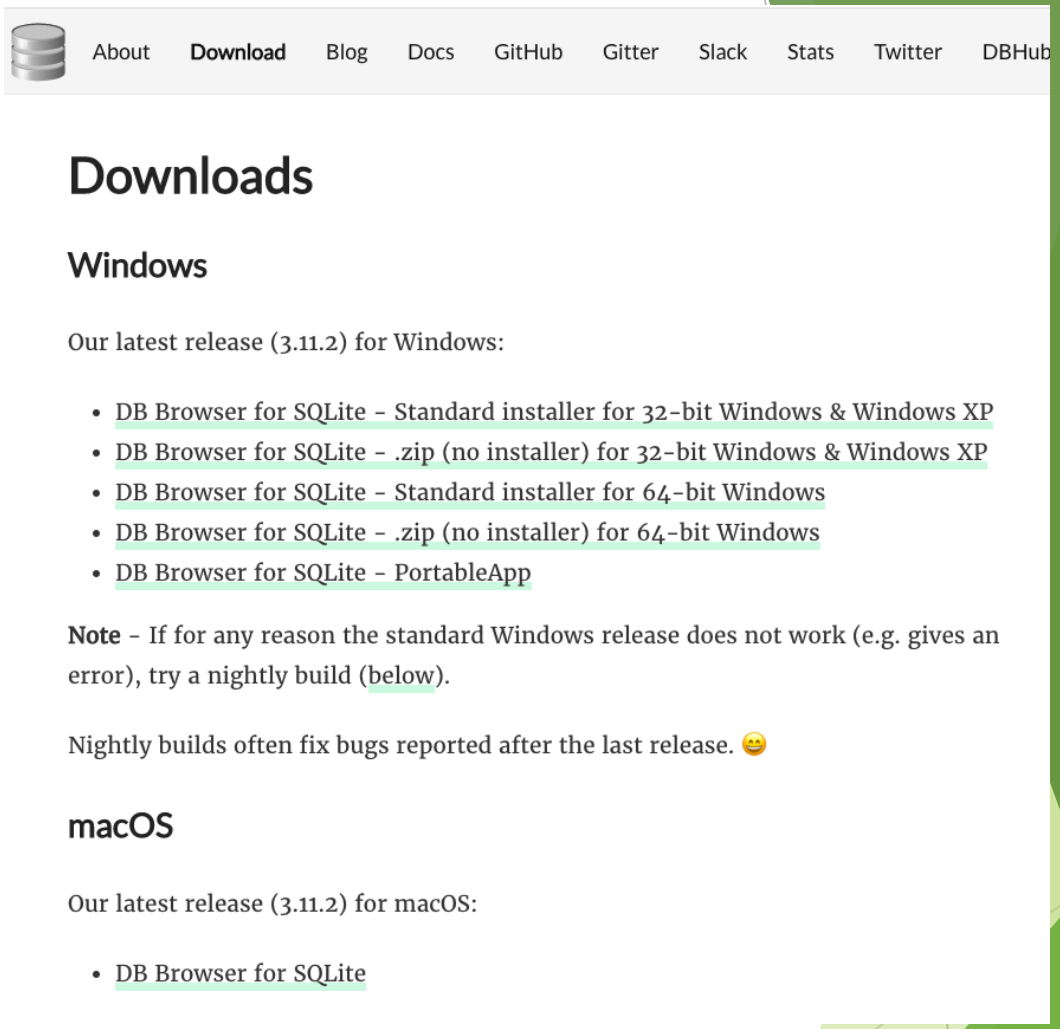
Why Use a Database?

- Databases can store more data than dictionaries and that storage is permanent
  - Databases organize data into tables, rows, and columns
- Can efficiently retrieve data from a database
  - Especially when a query (search) involves multiple tables
  - Using indexes
- Multiple users can read and modify data at the same time

http://en.wikipedia.org/wiki/Relational_database

DB Browser for SQLite

- Install from http://sqlitebrowser.org/dl/



Terminology

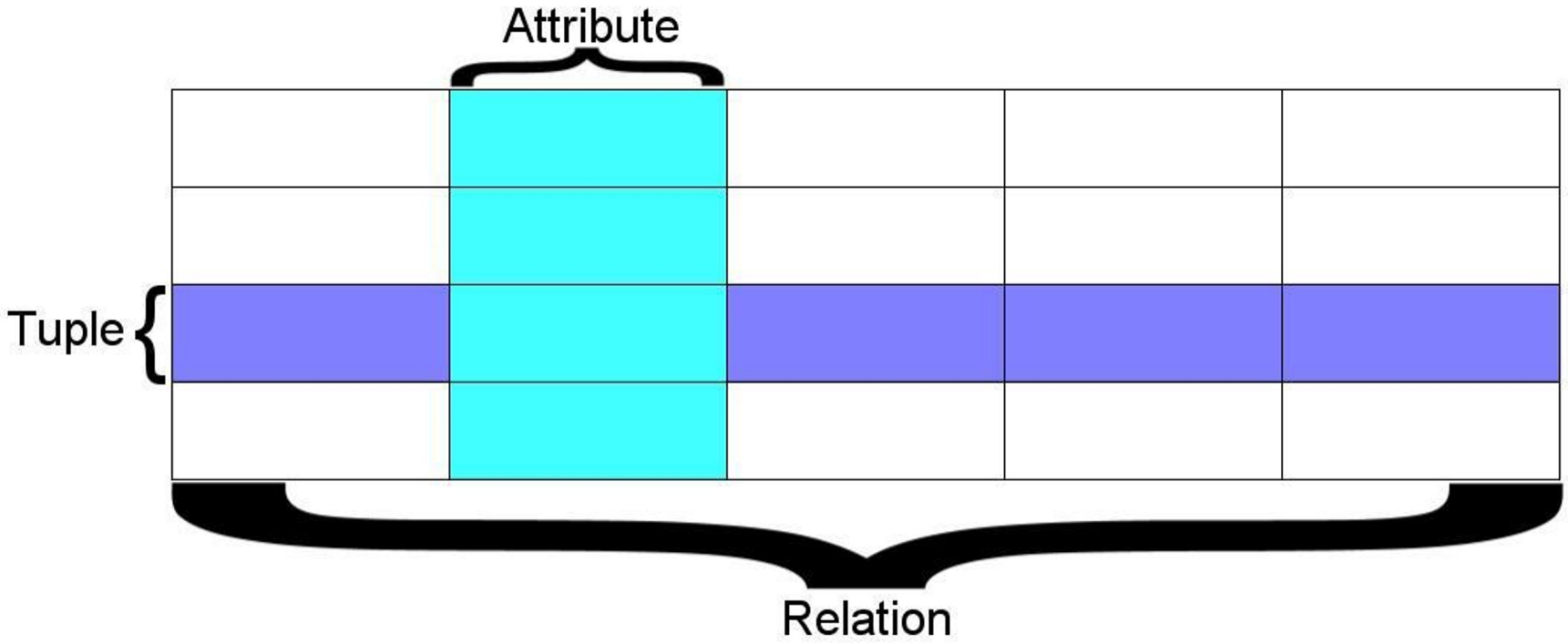
Database - contains many tables

Table (relation) - contains tuples and attributes. You can think of this as a class.

Row (tuple) - a set of fields that generally represents an “object” like a person or a music track

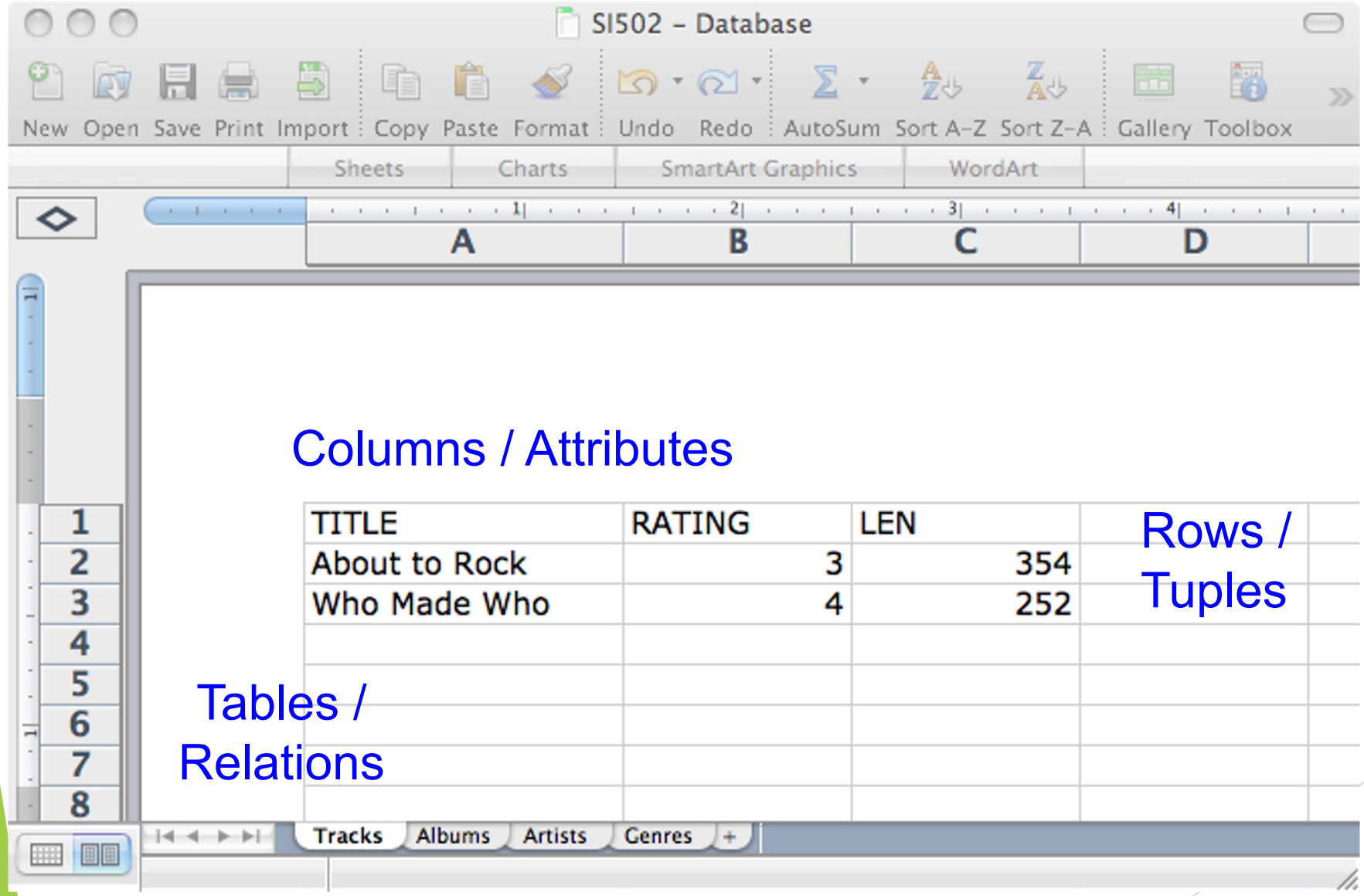
Column (attribute) - one of possibly many elements of data corresponding to the object represented by the row

Defined by Edgar Codd of IBM in 1970



A ***relation (table)*** is defined as a *set of tuples* that have the *same attributes*. A ***tuple*** usually represents an *object* and information about that object. ***Objects*** are typically physical objects or concepts. A ***relation*** is usually described as a *table*, which is organized into *rows and columns*. All the data referenced by an attribute are in the same domain and conform to the same constraints. (Wikipedia)

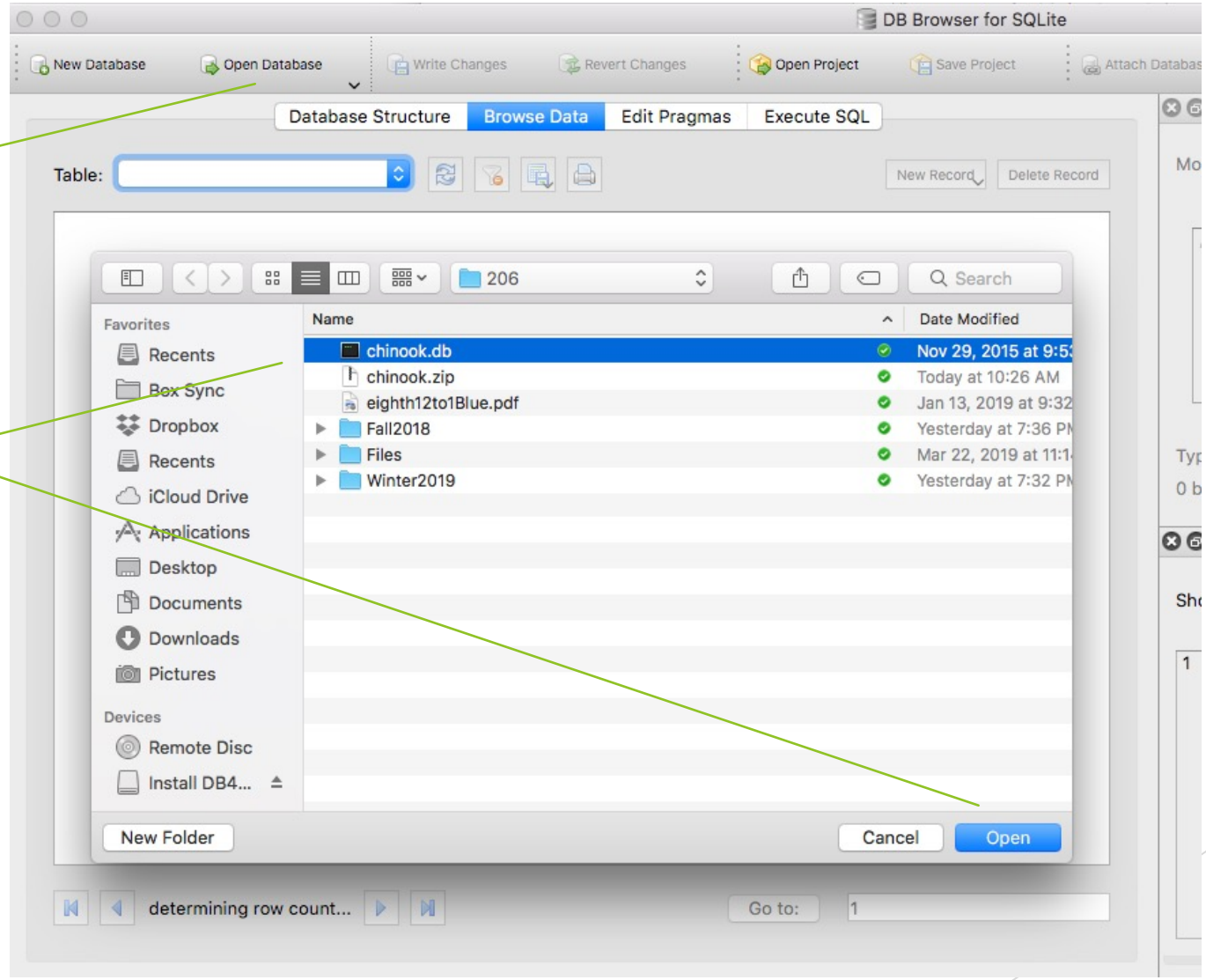
Each table represents one type of thing (like a class)



Example Database – chinook.db

Click on Open Database

Select chinook.db and Open



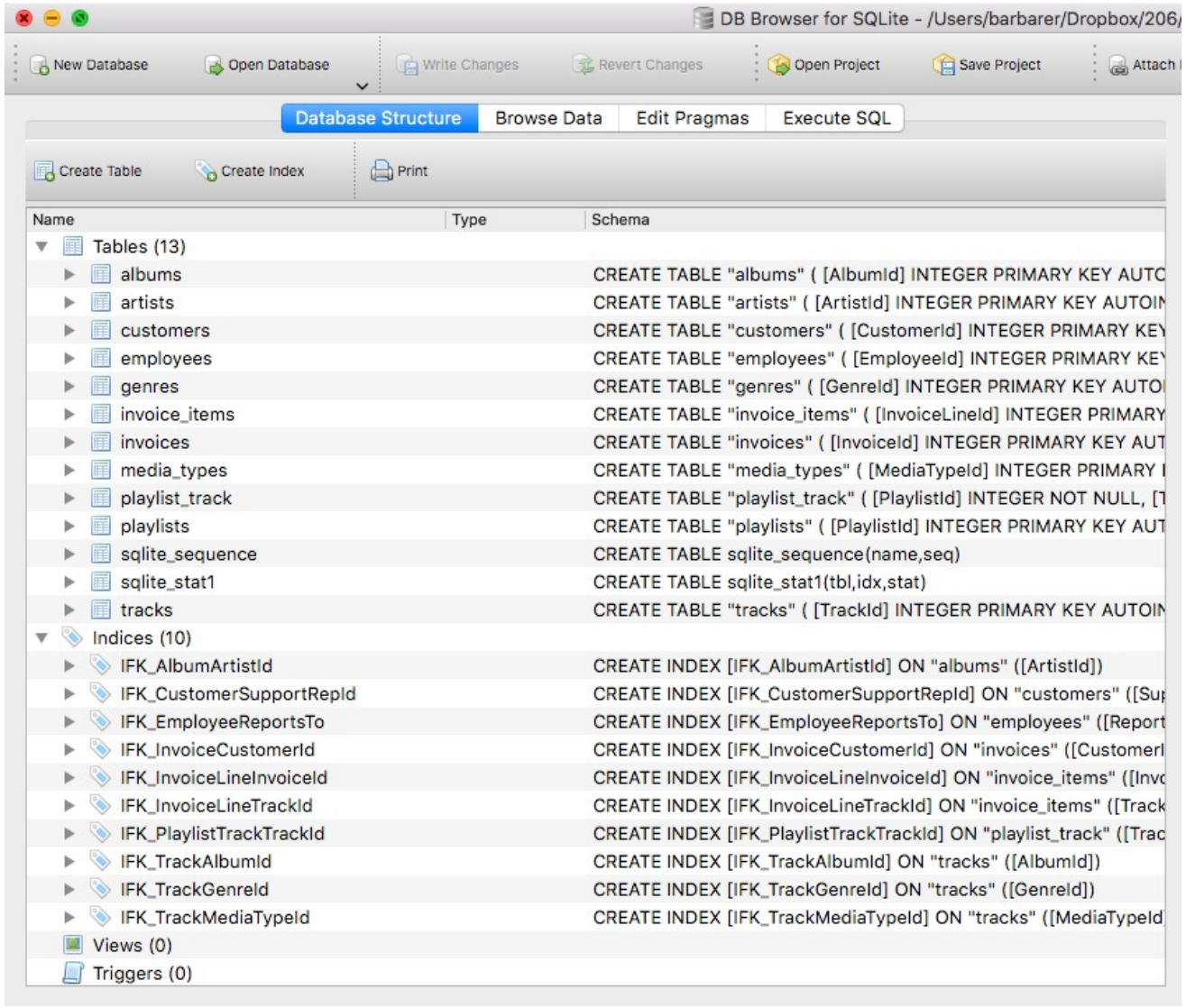
Database Structure

Chinook sample database tables

There are 11 tables in the chinook sample database.

- `employees` table stores employees data such as employee id, last name, first name, etc. It also has a field named `ReportsTo` to specify who reports to whom.
- `customers` table stores customers data.
- `invoices` & `invoice_items` tables: these two tables store invoice data. The `invoices` table stores invoice header data and the `invoice_items` table stores the invoice line items data.
- `artists` table stores artists data. It is a simple table that contains only artist id and name.
- `albums` table stores data about a list of tracks. Each album belongs to one artist. However, one artist may have multiple albums.
- `media_types` table stores media types such as MPEG audio and AAC audio file.
- `genres` table stores music types such as rock, jazz, metal, etc.
- `tracks` table store the data of songs. Each track belongs to one album.
- `playlists` & `playlist_track` tables: `playlists` table store data about playlists. Each playlist contains a list of tracks. Each track may belong to multiple playlists. The relationship between the `playlists` table and `tracks` table is many-to-many. The `playlist_track` table is used to reflect this relationship.

Database Structure



Peer Instruction #1 – PI-Database

Q-1: Which of the following would be a row (tuple) in a database?

- A. The information about a track on an album
- B. All the data in the database for all albums
- C. The album title
- D. All the track information for an album

Check Me

Activity: 1 Multiple Choice (db-pi-which-is-row)

Common Database Systems

Three major Database Management Systems in wide use

- Oracle - Large, commercial, enterprise-scale, very very tweakable
- MySql - Simpler, but very fast and scalable - commercial open source
- SqlServer - Very nice - from Microsoft (also Access)

Many other smaller projects, free and open source

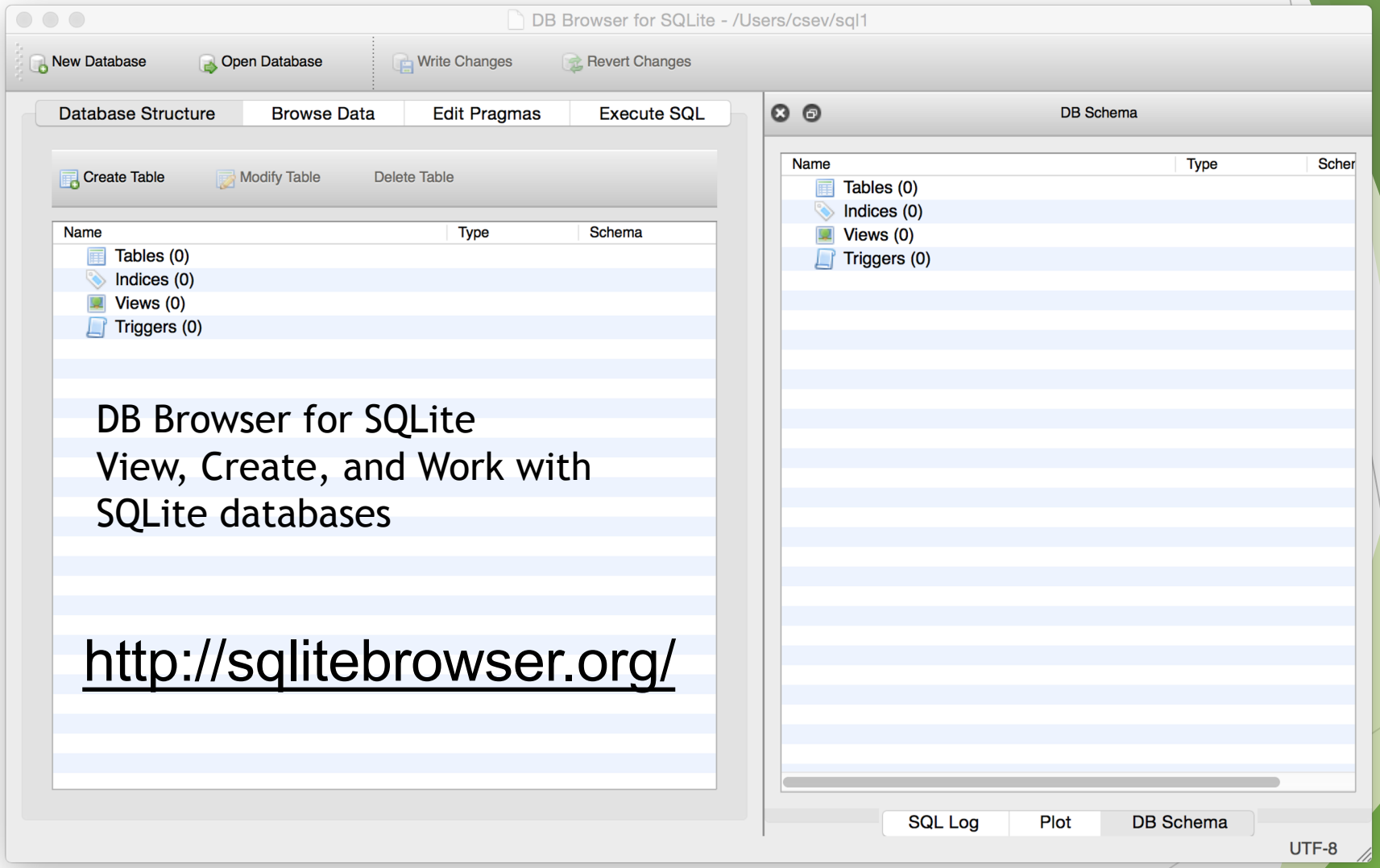
- HSQL, SQLite, Postgres, ...

SQLite is in Lots of Software...

http://www.sqlite.org/famous.html

SQLite

- SQLite is a very popular database - it is free and fast and small
- SQLite is embedded in Python and in a number of other languages


DB Browser for SQLite - /Users/csev/sql1
New Database
Open Database
Write Changes
Revert Changes
Database Structure
Browse Data
Edit Pragmas
Execute SQL
Create Table
Modify Table
Delete Table
Name
Type
Schema
Tables (0)
Indices (0)
Views (0)
Triggers (0)
DB Browser for SQLite
View, Create, and Work with
SQLite databases
http://sqlitebrowser.org/
DB Schema
Name
Type
Scher
Tables (0)
Indices (0)
Views (0)
Triggers (0)
SQL Log
Plot
DB Schema
UTF-8

SQL

http://en.wikipedia.org/wiki/SQL

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

- Create data (a.k.a Insert)
- Retrieve data (a.k.a. Select)
- Update data (a.k.a Update)
- Delete data (a.k.a Delete)

Let's Make a Database

- See

https://www.py4e.com/lectures3/Pythonlearn-15-Database-Handout.txt

```
Single Table SQL

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')

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

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

SELECT * FROM Users

SELECT * FROM Users WHERE email='csev@umich.edu'

SELECT * FROM Users ORDER BY email

SELECT * FROM Users ORDER BY name DESC
```

Peer Instruction #2

Q-1: How many users will be in the Users table after the following executes?

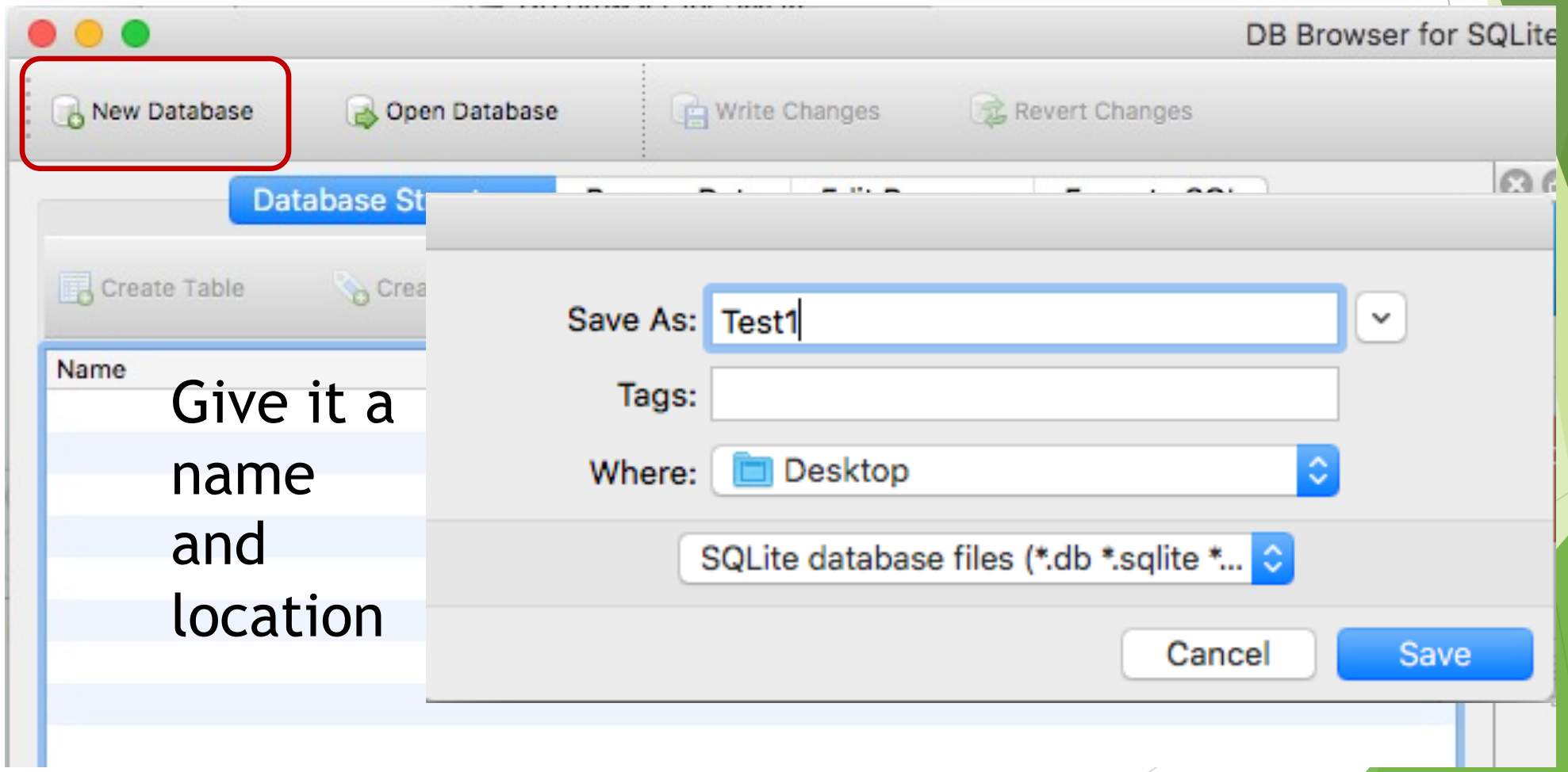
```
CREATE TABLE "Users" ("name" TEXT, "email" TEXT);
INSERT INTO Users (name, email) VALUES ('Chuck', 'csev@umich.edu');
INSERT INTO Users (name, email) VALUES ('Colleen', 'cvl@umich.edu');
INSERT INTO Users (name, email) VALUES ('Ted', 'ted@umich.edu');
INSERT INTO Users (name, email) VALUES ('Sally', 'a1@umich.edu');
INSERT INTO Users (name, email) VALUES ('Ted', 'ted@umich.edu');
INSERT INTO Users (name, email) VALUES ('Kristen', 'kf@umich.edu');
```

- ○ A. 1
- ○ B. 4
- ○ C. 5
- ○ D. 6

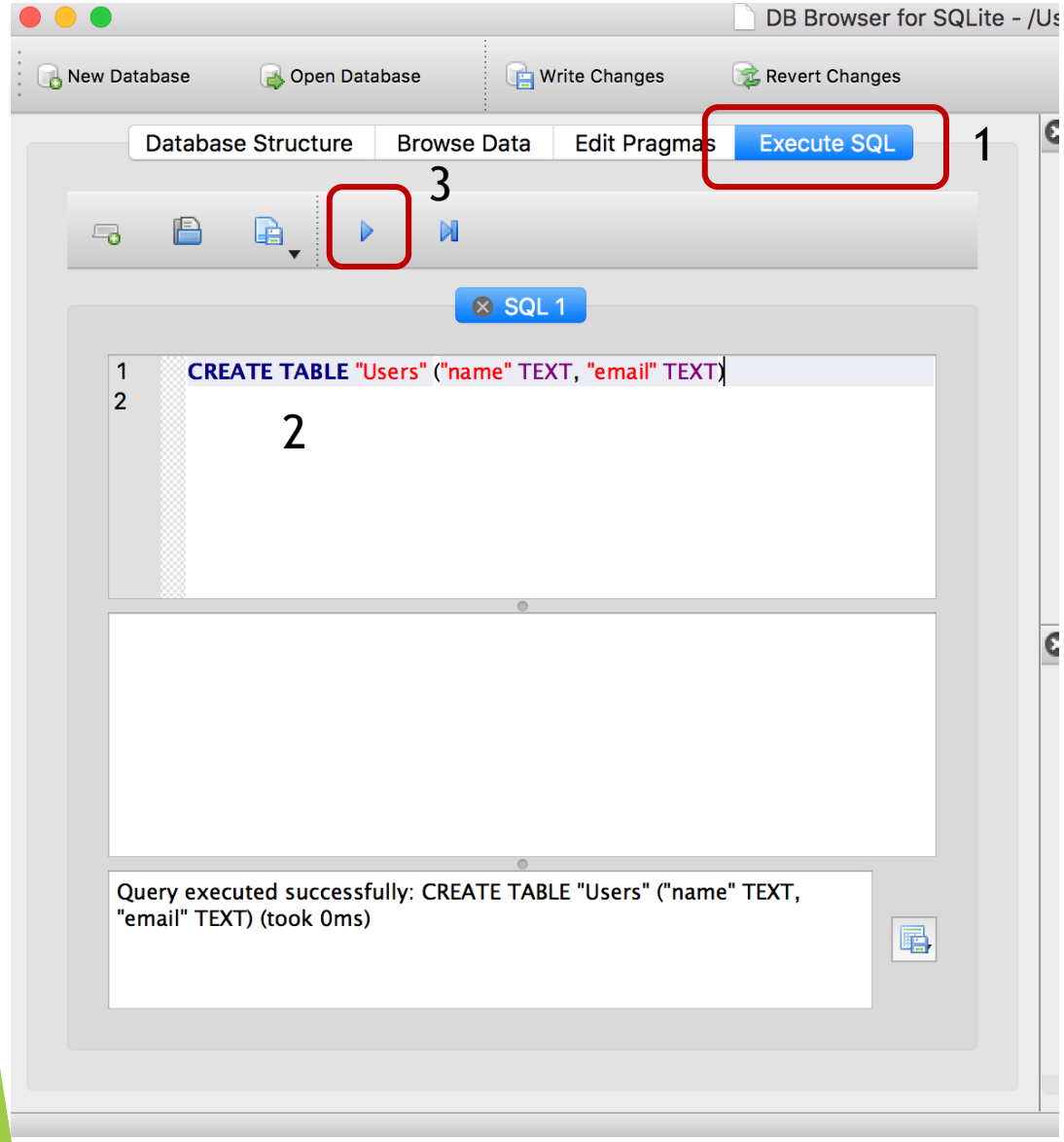
Check Me

Activity: 1 Multiple Choice (db-pi-insert-num-rows)

Create a New Database

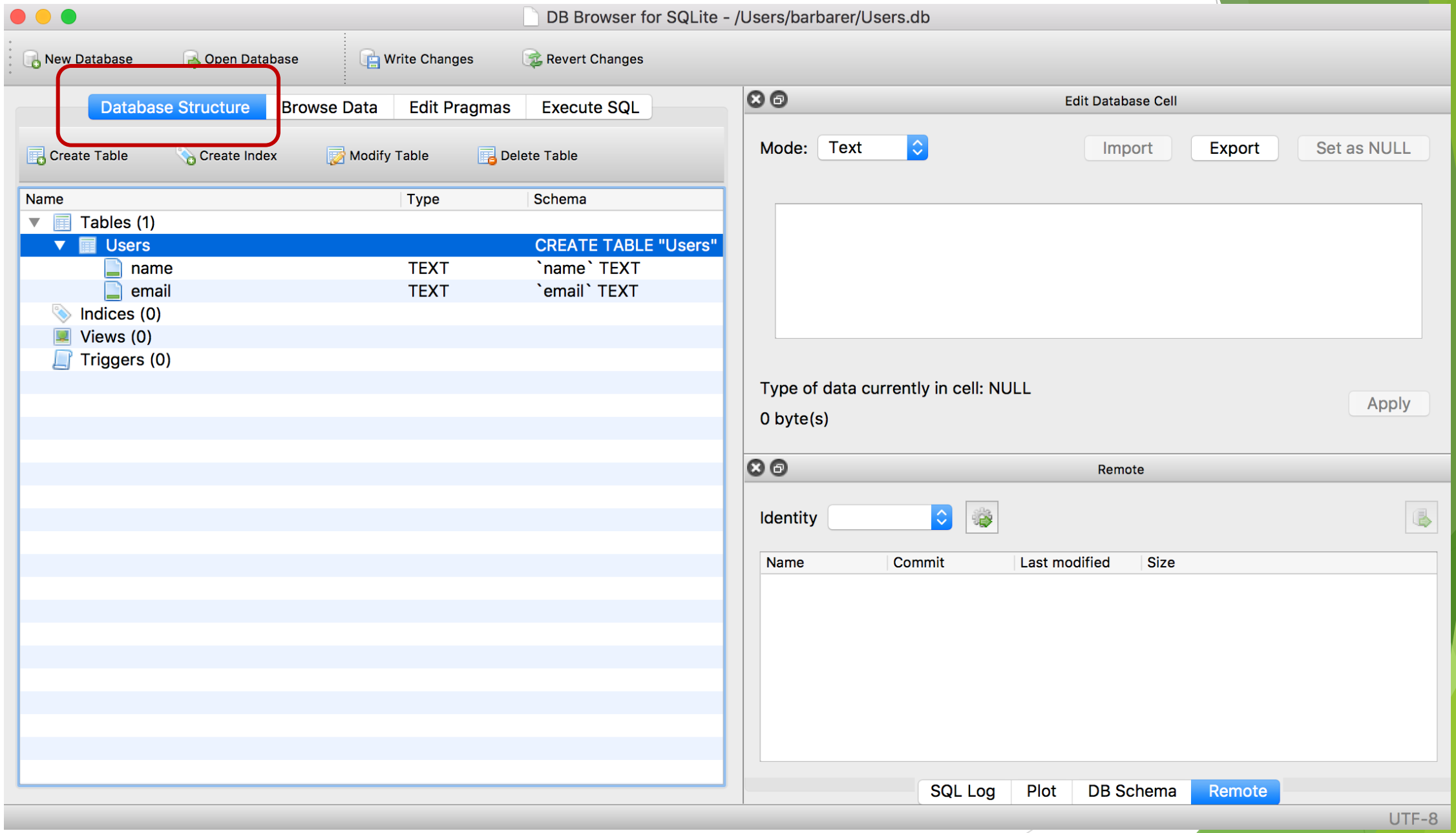


Start Simple - A Single Table



1) Go to the Execute SQL tab
2) Enter some SQL
3) Click the play button

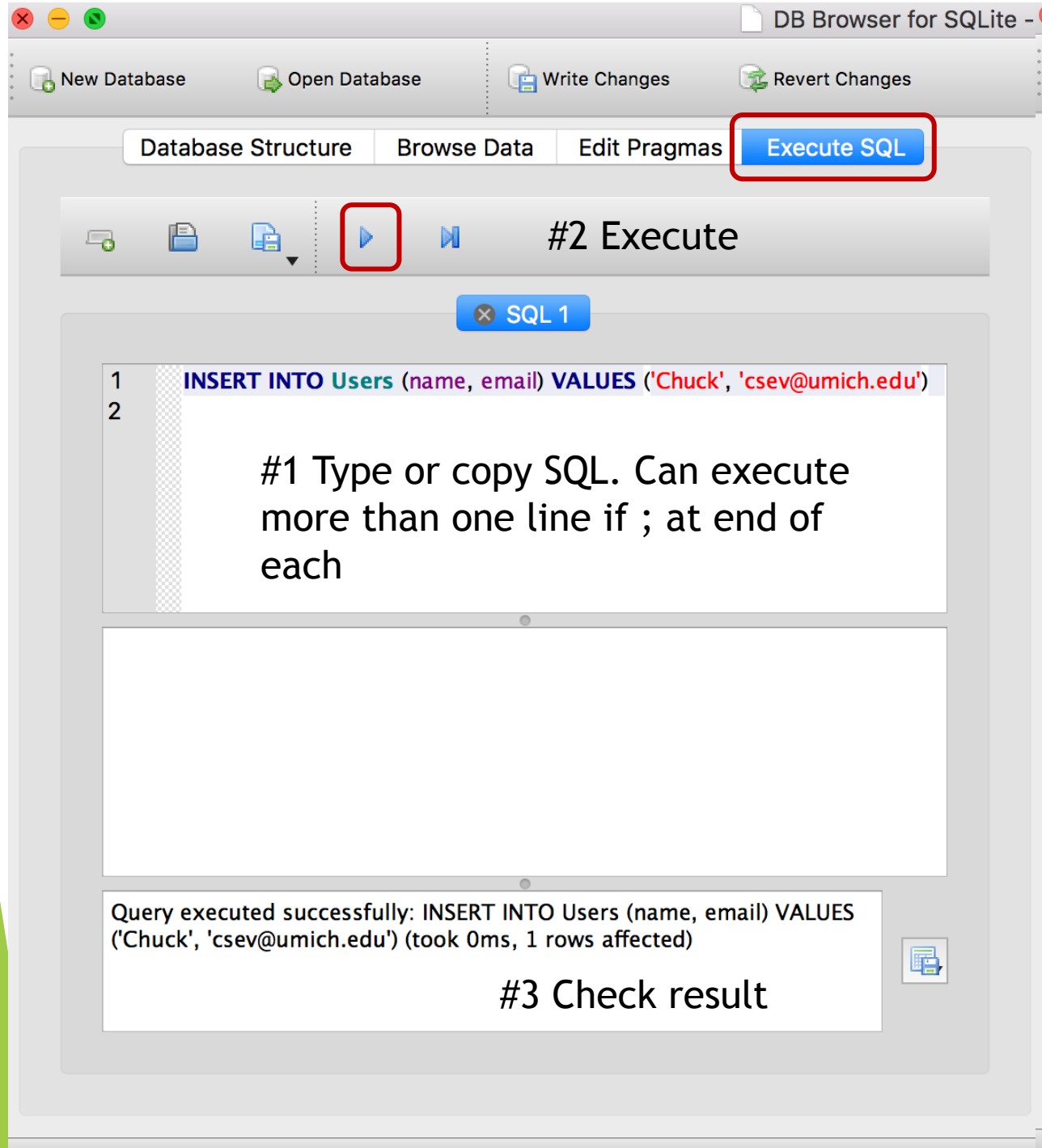
```
CREATE TABLE
"TableName" (
   "attr1Name" TYPE,
   "attr2Name" TYPE)
```

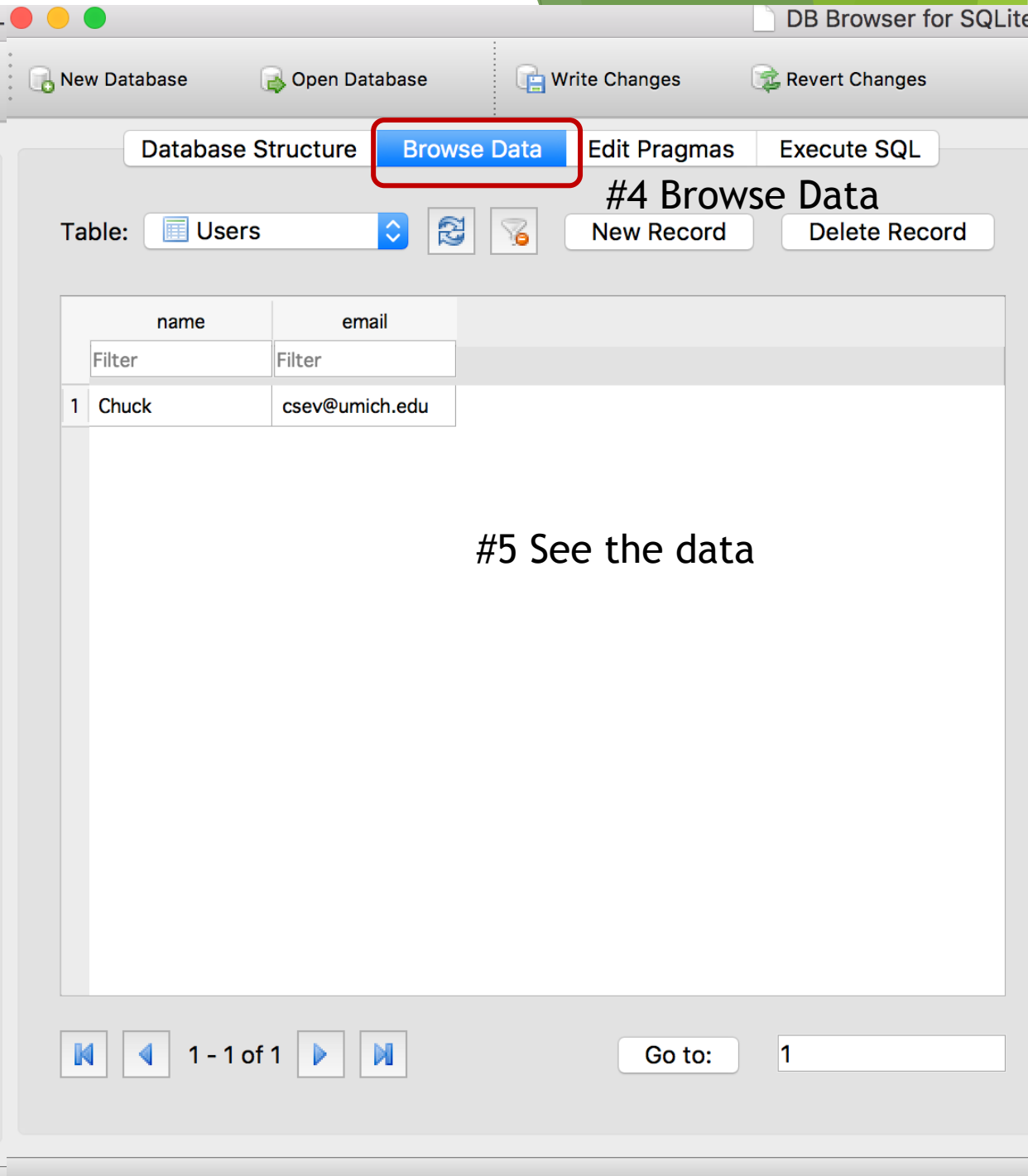

DB Browser for SQLite - /Users/barbarer/Users.db
New Database
Open Database
Write Changes
Revert Changes
Database Structure
Browse Data
Edit Pragmas
Execute SQL
Create Table
Create Index
Modify Table
Delete Table
Name
Type
Schema
Tables (1)
Users
CREATE TABLE "Users"
name
TEXT
`name` TEXT
email
TEXT
`email` TEXT
Indices (0)
Views (0)
Triggers (0)
Edit Database Cell
Mode:
Text
Import
Export
Set as NULL
Type of data currently in cell: NULL
0 byte(s)
Apply
Remote
Identity
Name
Commit
Last modified
Size
SQL Log
Plot
DB Schema
Remote
UTF-8

SQL: Insert

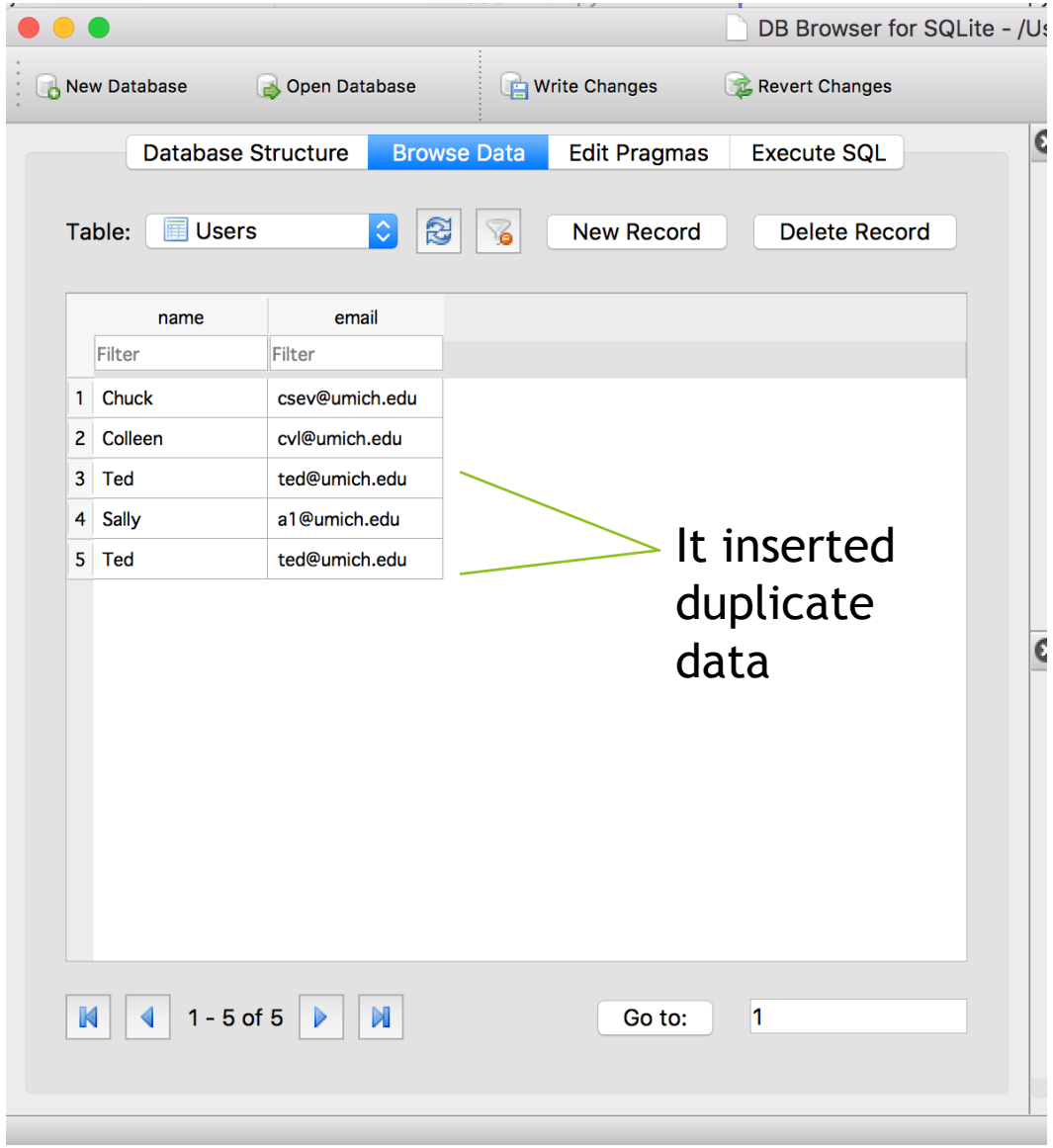
The Insert statement inserts a row into a table

```
INSERT INTO Users (name, email)
VALUES ('Kristin', 'kf@umich.edu')
```


DB Browser for SQLite -
New Database
Open Database
Write Changes
Revert Changes
Database Structure
Browse Data
Edit Pragmas
Execute SQL
#2 Execute
SQL 1
1 INSERT INTO Users (name, email) VALUES ('Chuck', 'csev@umich.edu')
2
#1 Type or copy SQL. Can execute more than one line if ; at end of each
Query executed successfully: INSERT INTO Users (name, email) VALUES ('Chuck', 'csev@umich.edu') (took 0ms, 1 rows affected)
#3 Check result


DB Browser for SQLite
New Database
Open Database
Write Changes
Revert Changes
Database Structure
Browse Data
Edit Pragmas
Execute SQL
#4 Browse Data
Table: Users
New Record
Delete Record
name
email
Filter
Filter
1 Chuck csev@umich.edu
#5 See the data
1 - 1 of 1
Go to: 1

Duplicate Data?

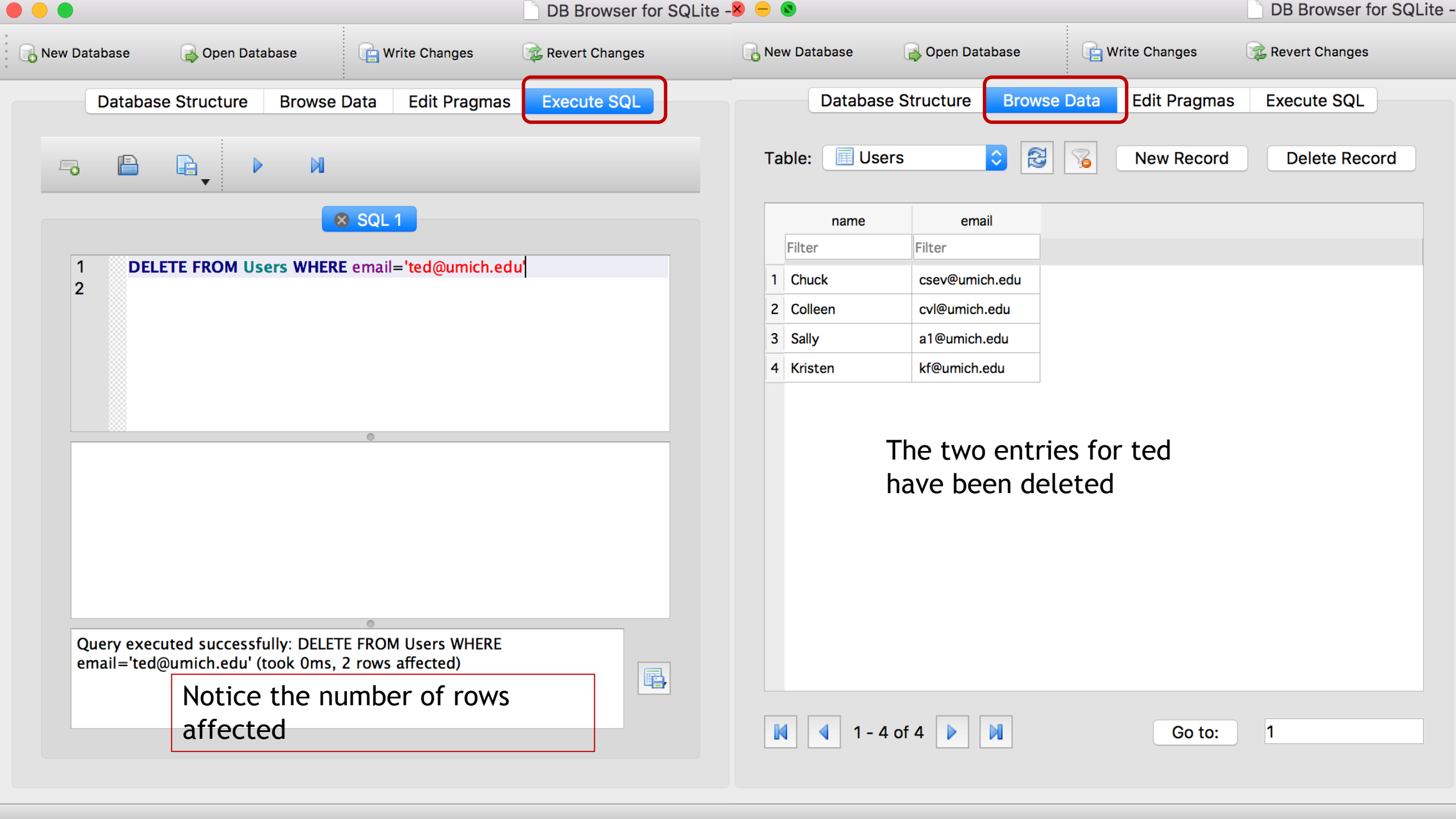


It inserted duplicate data

SQL: Delete

Deletes a row in a table based on some selection criteria

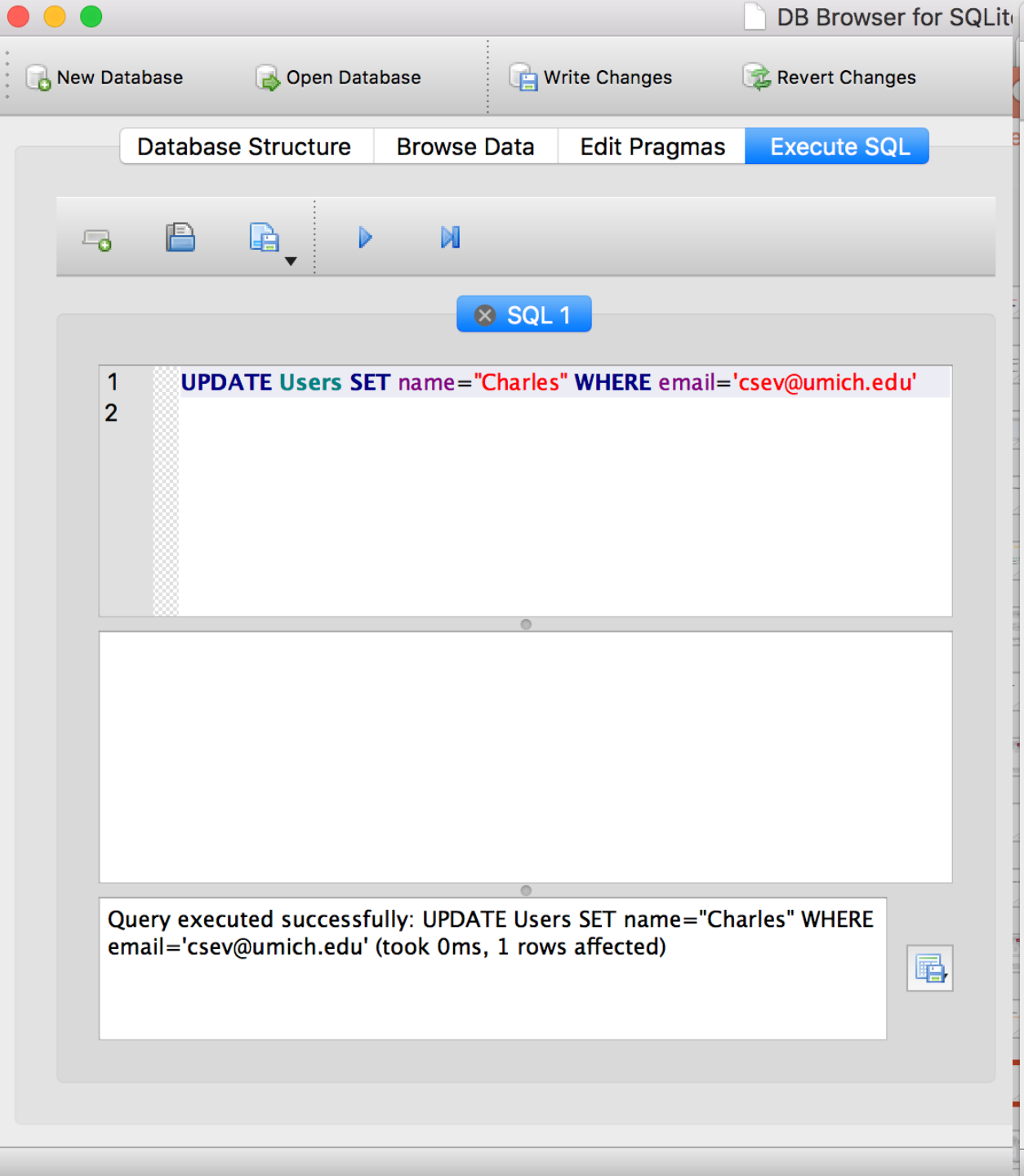
```
DELETE FROM Users WHERE
    email='ted@umich.edu'
```

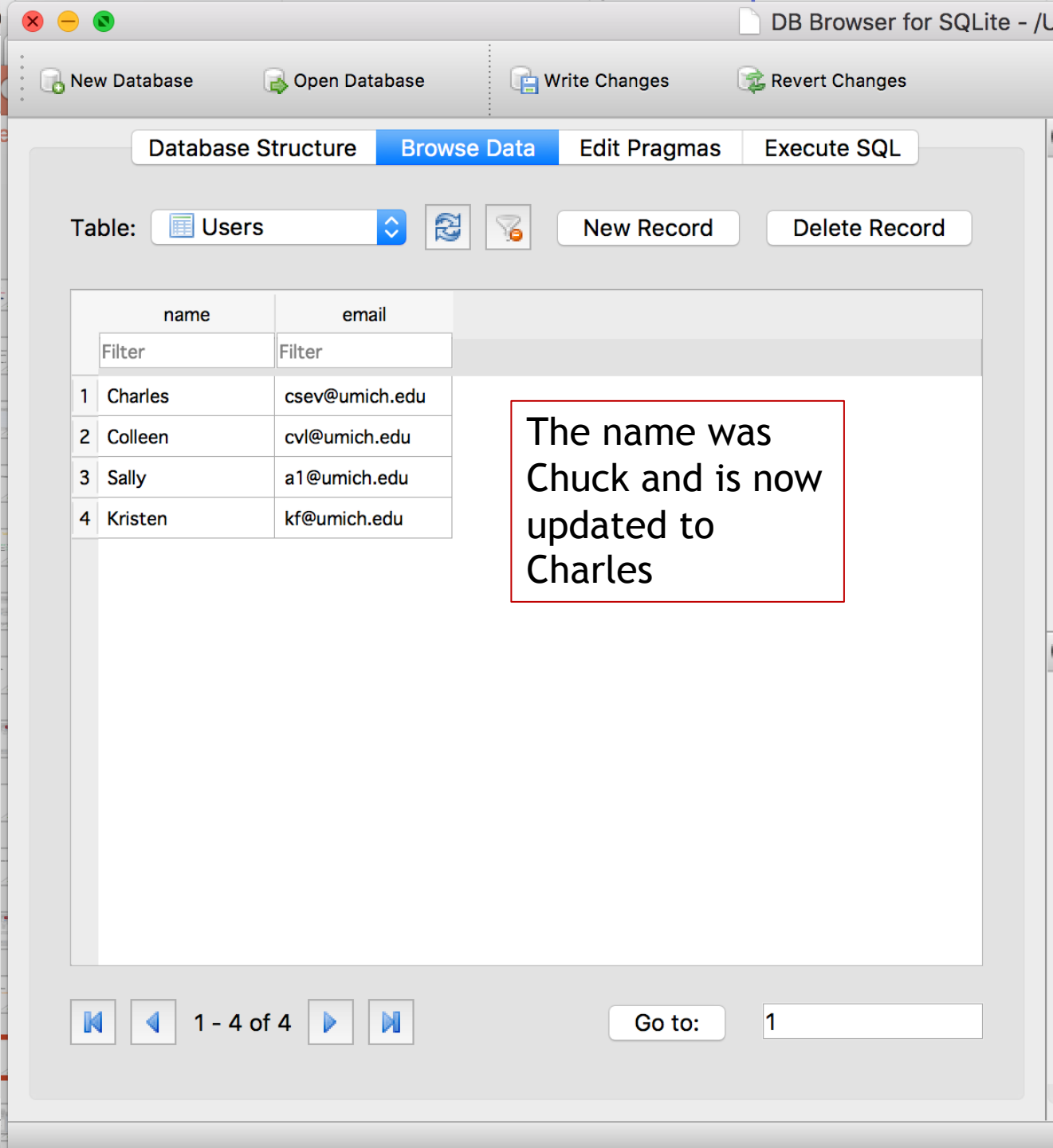

DB Browser for SQLite -
New Database
Open Database
Write Changes
Revert Changes
Database Structure
Browse Data
Edit Pragmas
Execute SQL
SQL 1
1 DELETE FROM Users WHERE email='ted@umich.edu'
2
Query executed successfully: DELETE FROM Users WHERE email='ted@umich.edu' (took 0ms, 2 rows affected)
Notice the number of rows affected
DB Browser for SQLite -
New Database
Open Database
Write Changes
Revert Changes
Database Structure
Browse Data
Edit Pragmas
Execute SQL
Table: Users
New Record
Delete Record
name
email
Filter
Filter
1 Chuck csev@umich.edu
2 Colleen cvl@umich.edu
3 Sally a1@umich.edu
4 Kristen kf@umich.edu
The two entries for ted have been deleted
1 - 4 of 4
Go to:
1

SQL: Update

Allows the updating of a field with a where clause

```
UPDATE Users SET name='Charles' WHERE
email='csev@umich.edu'
```


DB Browser for SQLit
New Database
Open Database
Write Changes
Revert Changes
Database Structure
Browse Data
Edit Pragmas
Execute SQL
SQL 1
1 UPDATE Users SET name="Charles" WHERE email='csev@umich.edu'
2
Query executed successfully: UPDATE Users SET name="Charles" WHERE email='csev@umich.edu' (took 0ms, 1 rows affected)

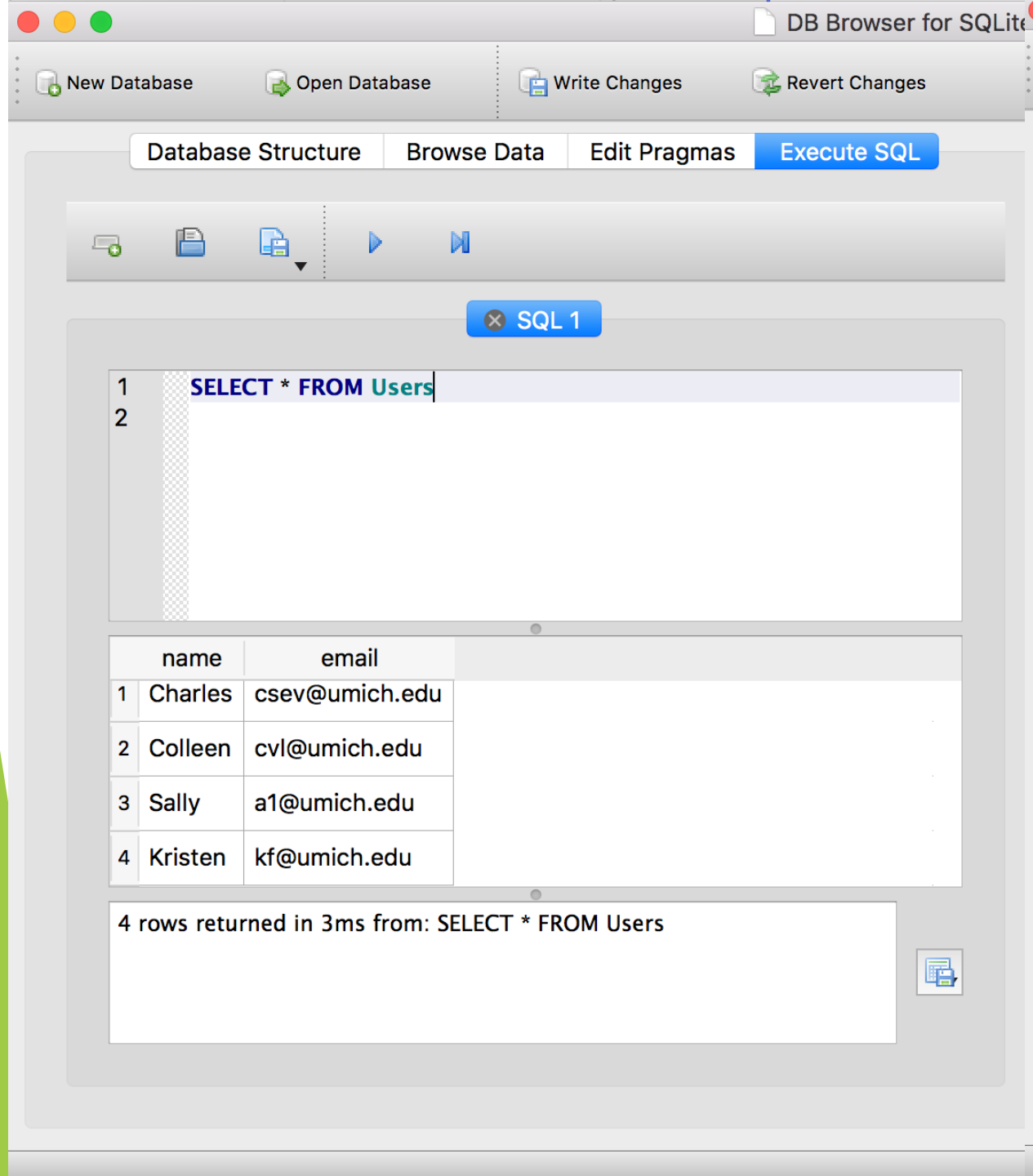

DB Browser for SQLite - /U
New Database
Open Database
Write Changes
Revert Changes
Database Structure
Browse Data
Edit Pragmas
Execute SQL
Table: Users
New Record
Delete Record
name
email
Filter
Filter
1 Charles csev@umich.edu
2 Colleen cvl@umich.edu
3 Sally a1@umich.edu
4 Kristen kf@umich.edu
The name was Chuck and is now updated to Charles
1 - 4 of 4
Go to:
1

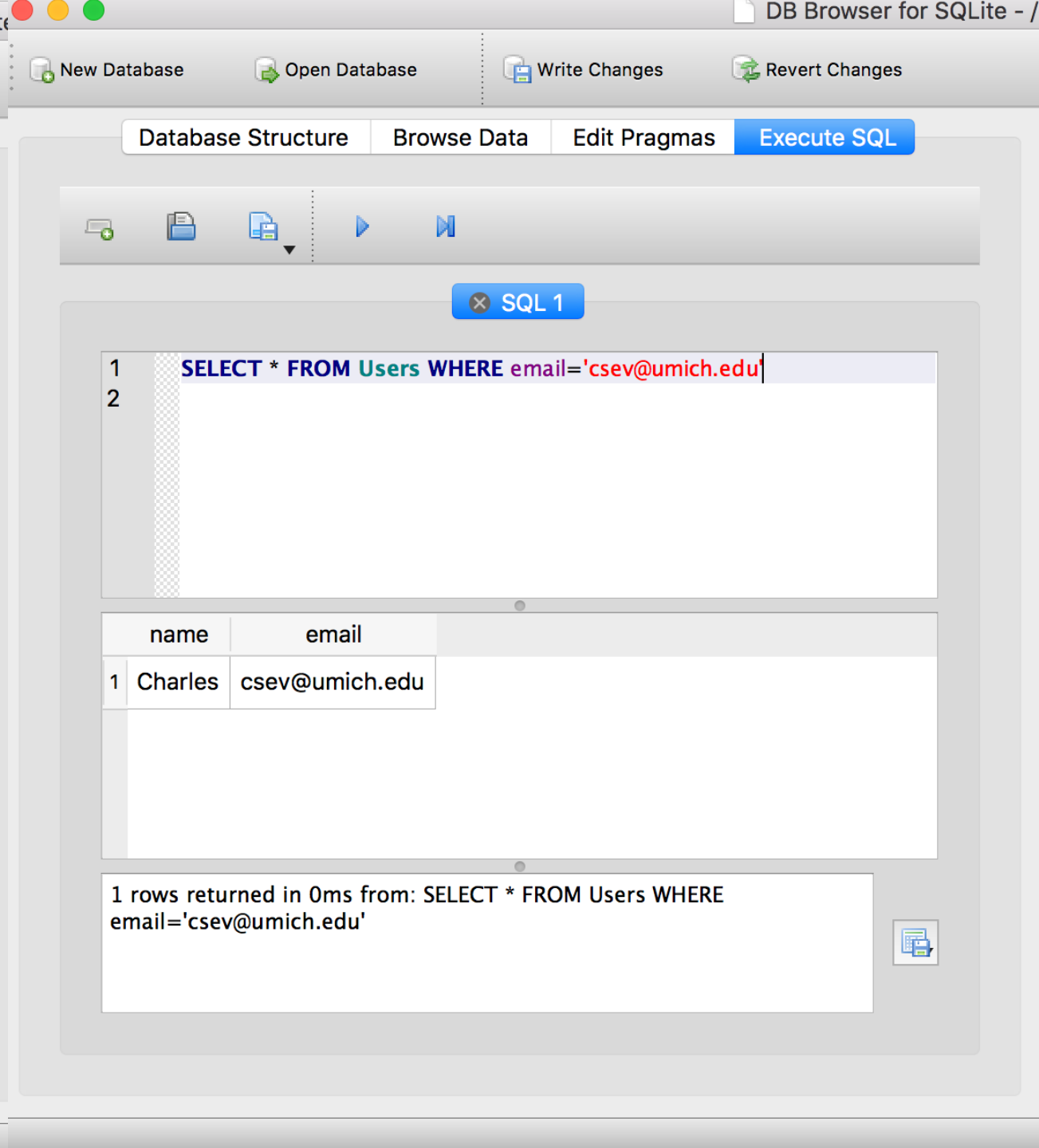
Retrieving Records:
Select

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

```
SELECT * FROM Users
```

```
SELECT * FROM Users WHERE email='csev@umich.edu'
```


DB Browser for SQLite
New Database
Open Database
Write Changes
Revert Changes
Database Structure
Browse Data
Edit Pragmas
Execute SQL
SQL 1
1 SELECT * FROM Users
2
name email
1 Charles csev@umich.edu
2 Colleen cvl@umich.edu
3 Sally a1@umich.edu
4 Kristen kf@umich.edu
4 rows returned in 3ms from: SELECT * FROM Users

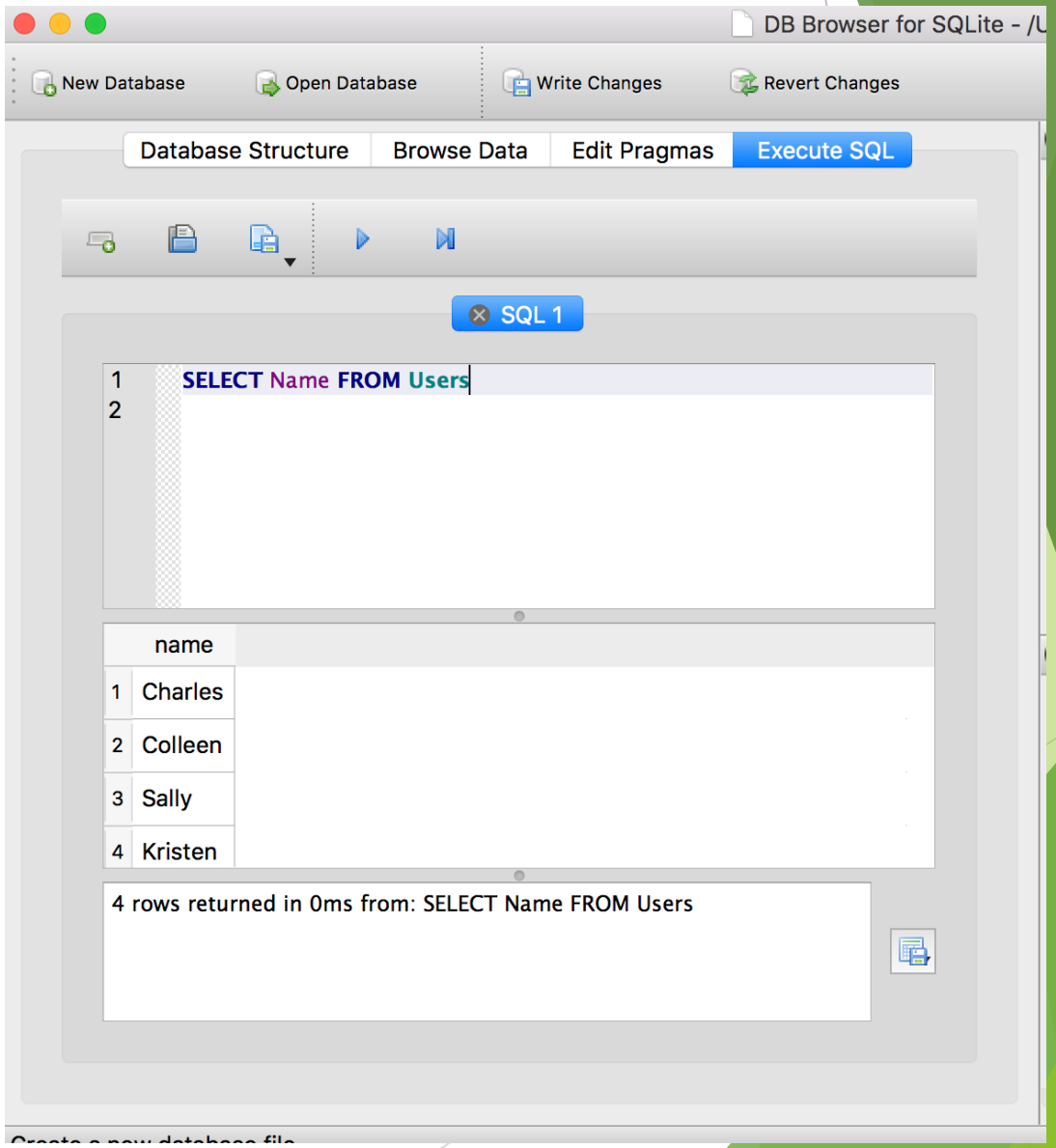

DB Browser for SQLite - /
New Database
Open Database
Write Changes
Revert Changes
Database Structure
Browse Data
Edit Pragmas
Execute SQL
SQL 1
1 SELECT * FROM Users WHERE email='csev@umich.edu'
2
name email
1 Charles csev@umich.edu
1 rows returned in 0ms from: SELECT * FROM Users WHERE email='csev@umich.edu'

Lecture Practice

- Go to the ebook and do Practice-SQL for up to 5 points

Can Limit What Fields Are Returned

- SELECT field FROM table
- SELECT field1, field2 FROM table



Working with SQL in Python

- Import sqlite
  - **import** sqlite3
- Create a connection to the database
  - Used to communicate with the database (even if on another computer)
  - conn = sqlite3.connect('music.sqlite')
- Create a cursor
  - Used to execute SQL commands and help you process the results
  - cur = conn.cursor()

Working with SQL in Python - Continued

- Use the cursor to execute SQL
  - cur.execute('DROP TABLE IF EXISTS Tracks')
- Commit the changes (if any)
  - conn.commit()
- Close the connection
  - conn.close()

Sample Python Code

select-Test1.py

```
1   import sqlite3
2
3   # connect to the database
4   conn = sqlite3.connect('/Users/barbarer/Desktop/Test1.db')
5
6   # get a cursor from the database connection
7   cur = conn.cursor()
8
9   # select the data from the User table
10  cur.execute('SELECT * FROM Users')
11  for row in cur:
12      print(row)
13
14  # close the cursor
15  cur.close()
16
```

Example Python Code

```
1  import sqlite3
2
3  conn = sqlite3.connect('/Users/barbarer/Desktop/music.sqlite')
4  cur = conn.cursor()
5
6  # delete the table if it already exists
7  cur.execute('DROP TABLE IF EXISTS Tracks')
8
9  # create the table
10 cur.execute('CREATE TABLE Tracks (title TEXT, plays INTEGER)')
11
12 # insert data into the Tracks table
13 cur.execute('INSERT INTO Tracks (title, plays) VALUES (?, ?)',
14     ('Thunderstruck', 20))
15 cur.execute('INSERT INTO Tracks (title, plays) VALUES (?, ?)',
16     ('My Way', 15))
17
18 # commit the changes
19 conn.commit()
20
21 # select and print the data
22 print('Tracks:')
23 cur.execute('SELECT title, plays FROM Tracks')
24 for row in cur:
25      print(row)
26
27 cur.close()
28
```

db-ex2.py

Drop, Create, Insert and Select

Warning!

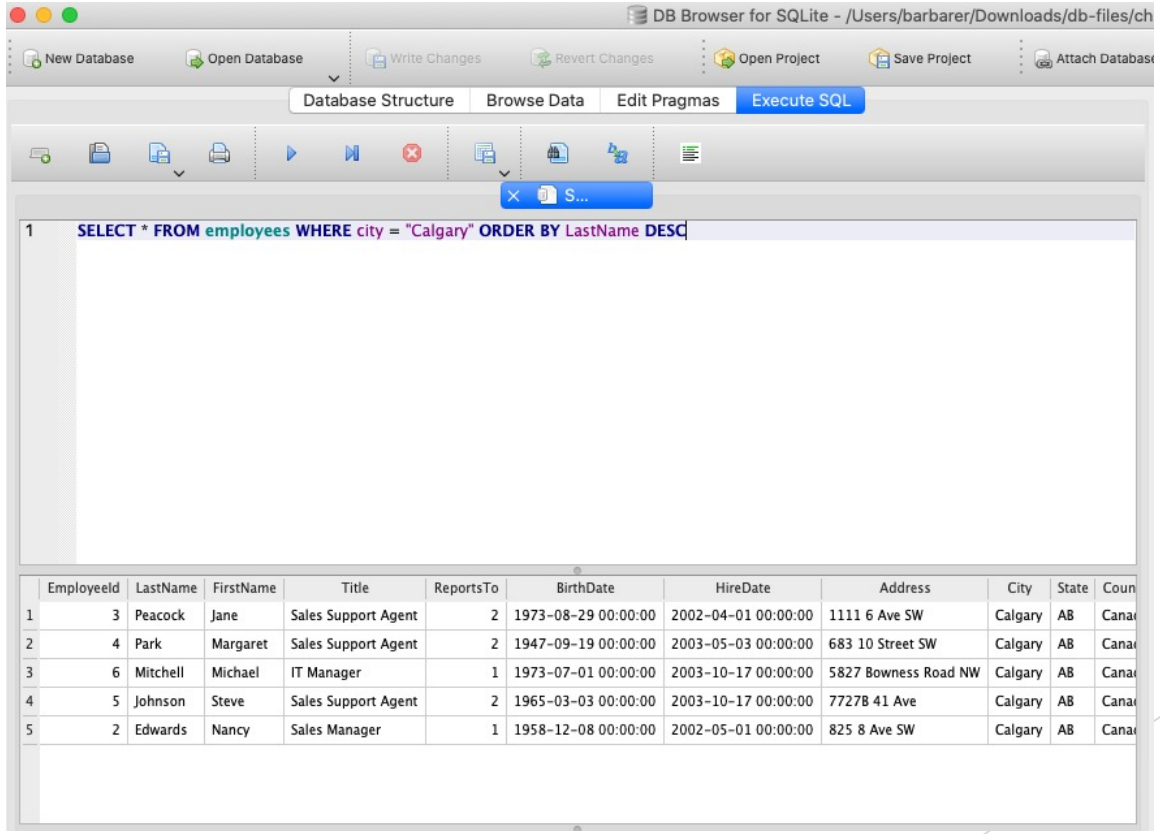
- Only drop a table if you want to delete all of the previous data
- Do not drop any tables on the final project in your code!
  - You will be adding content over time to the database

Working with the Cursor

- Can treat the cursor as an iterator and loop through the rows in it
  - for row in curr:
- cursor.fetchone()
  - Returns the first row that was returned or None
- cursor.fetchall()
  - Returns a list of the rows that were selected

Sorting the Result from a Select

- Use ORDER BY to sort the result from a SELECT
  - Use DESC to do it in descending order – see chinook.db



Count and Max

- You can get a count of the number of rows in a field (column)

```
1 SELECT COUNT(bike_number) FROM trip_data;
2
```

COUNT(bike_number)
408922

- You can get the max value for a column

```
1 SELECT MAX(bike_number) FROM trip_data;
2
```

MAX(bike_number)
w01117

Lecture Practice

- Go to the ebook and do
  - Practice-Database for up to 5 points