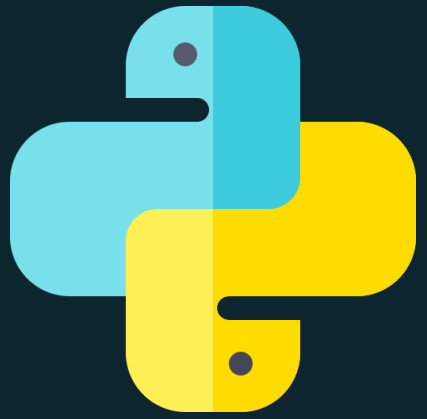
Brief Python

Python Course for Programmers



Learn Python Language for Data Science

CHAPTER 12B: DATABASE (SQL)

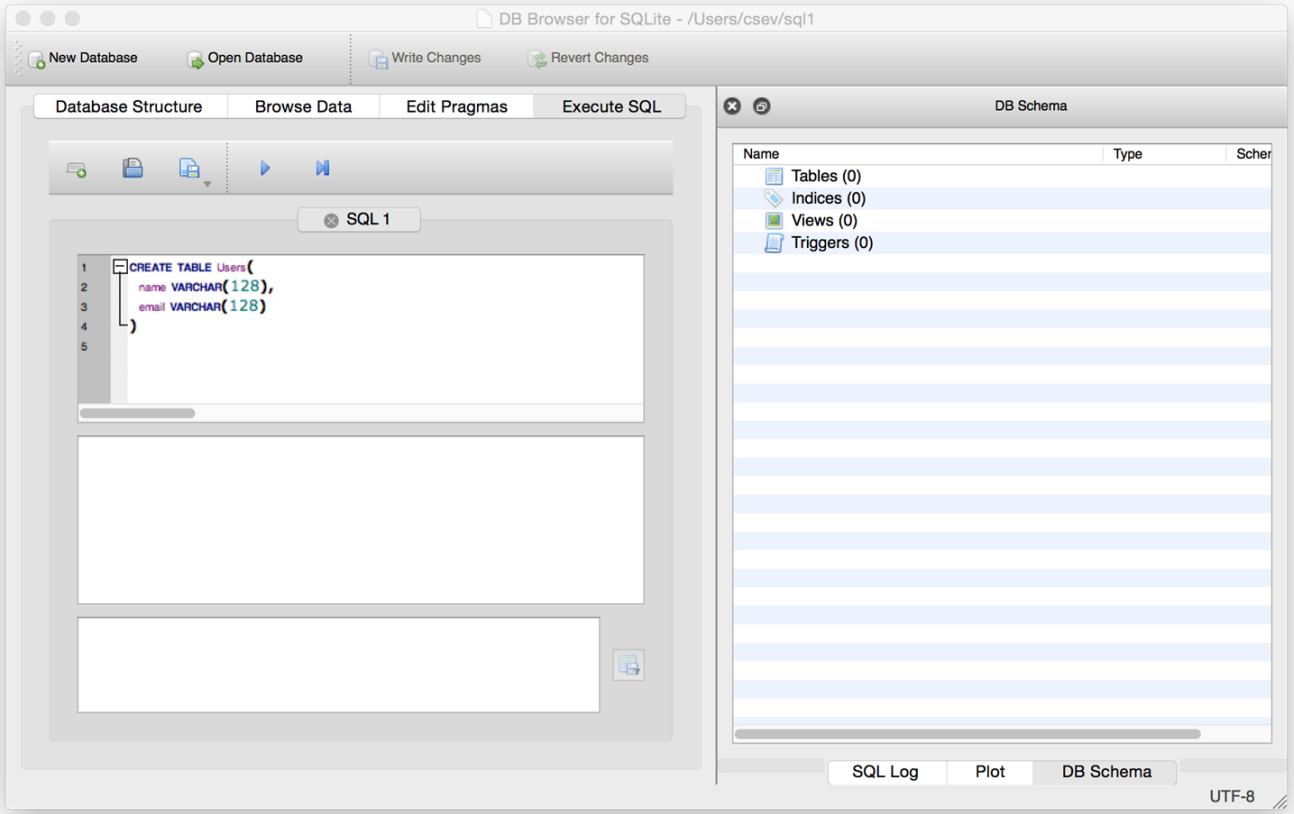
DR. ERIC CHOU

IEEE SENIOR MEMBER

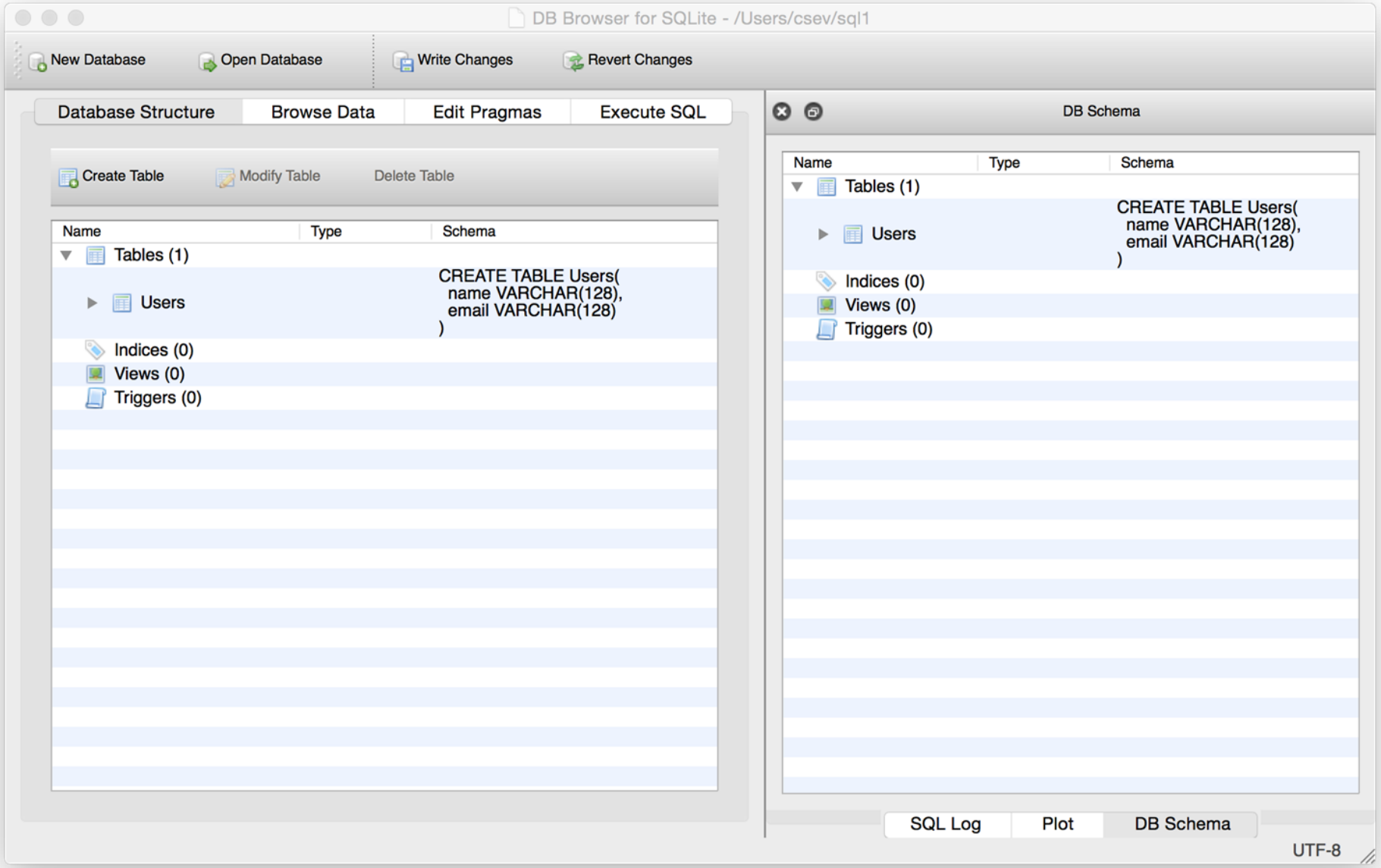
SQL Table Design

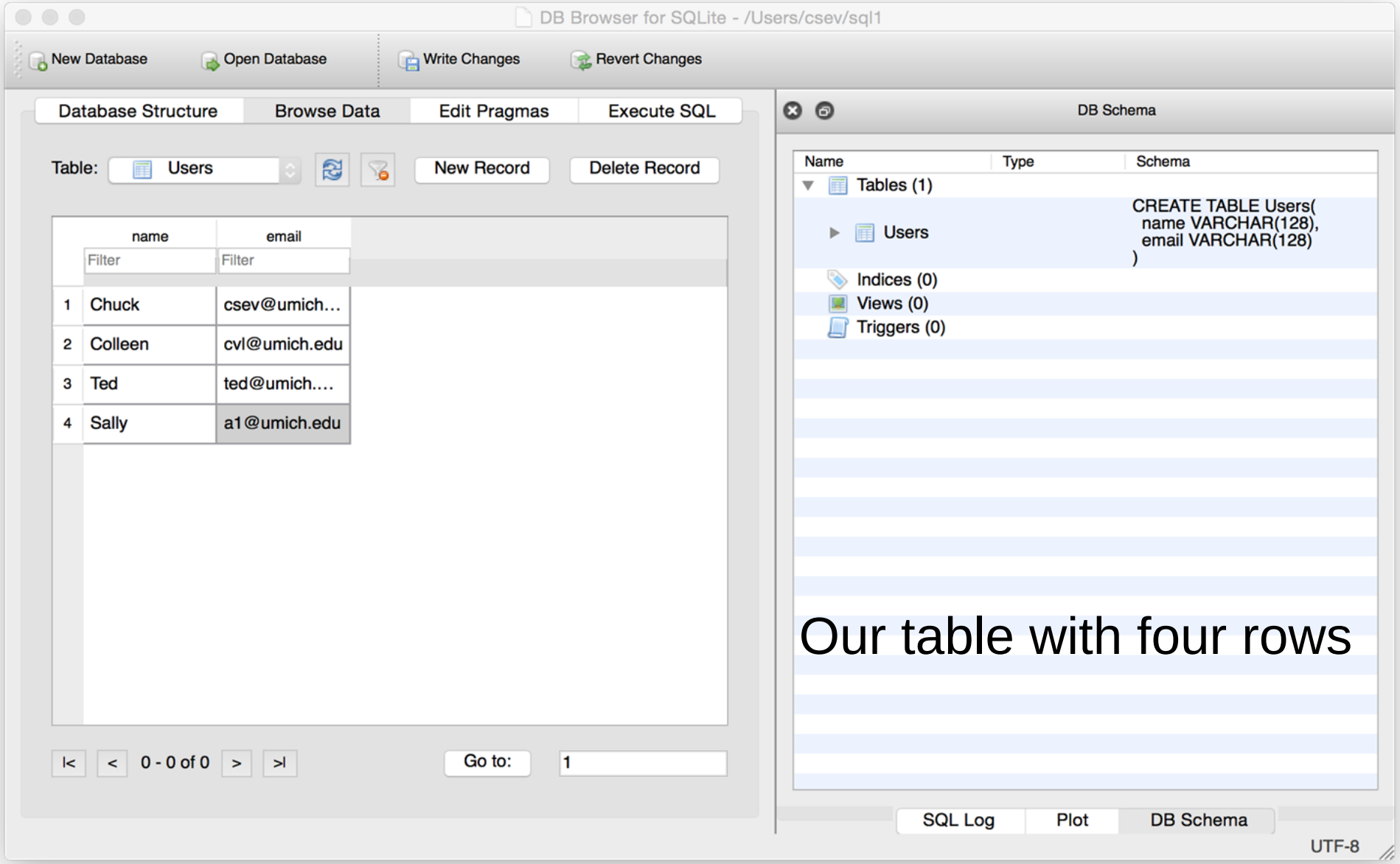
LECTURE 3

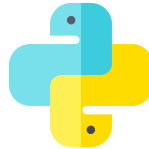
Start Simple - A Single Table




```
CREATE TABLE Users(
    name VARCHAR(128),
    email VARCHAR(128)
)
```


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 (1)
Users
CREATE TABLE Users(
name VARCHAR(128),
email VARCHAR(128)
)
Indices (0)
Views (0)
Triggers (0)
DB Schema
Name
Type
Schema
Tables (1)
Users
CREATE TABLE Users(
name VARCHAR(128),
email VARCHAR(128)
)
Indices (0)
Views (0)
Triggers (0)
SQL Log
Plot
DB Schema
UTF-8

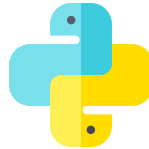

DB Browser for SQLite - /Users/csev/sql1
New Database
Open Database
Write Changes
Revert Changes
Database Structure
Browse Data
Edit Pragmas
Execute SQL
Table:
Users
New Record
Delete Record
name
email
Filter
Filter
1 Chuck csev@umich...
2 Colleen cvl@umich.edu
3 Ted ted@umich....
4 Sally a1@umich.edu
I<
<
0 - 0 of 0
>
>I
Go to:
1
DB Schema
Name
Type
Schema
Tables (1)
Users
CREATE TABLE Users(
name VARCHAR(128),
email VARCHAR(128)
)
Indices (0)
Views (0)
Triggers (0)
Our table with four rows
SQL Log
Plot
DB Schema
UTF-8



SQL

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

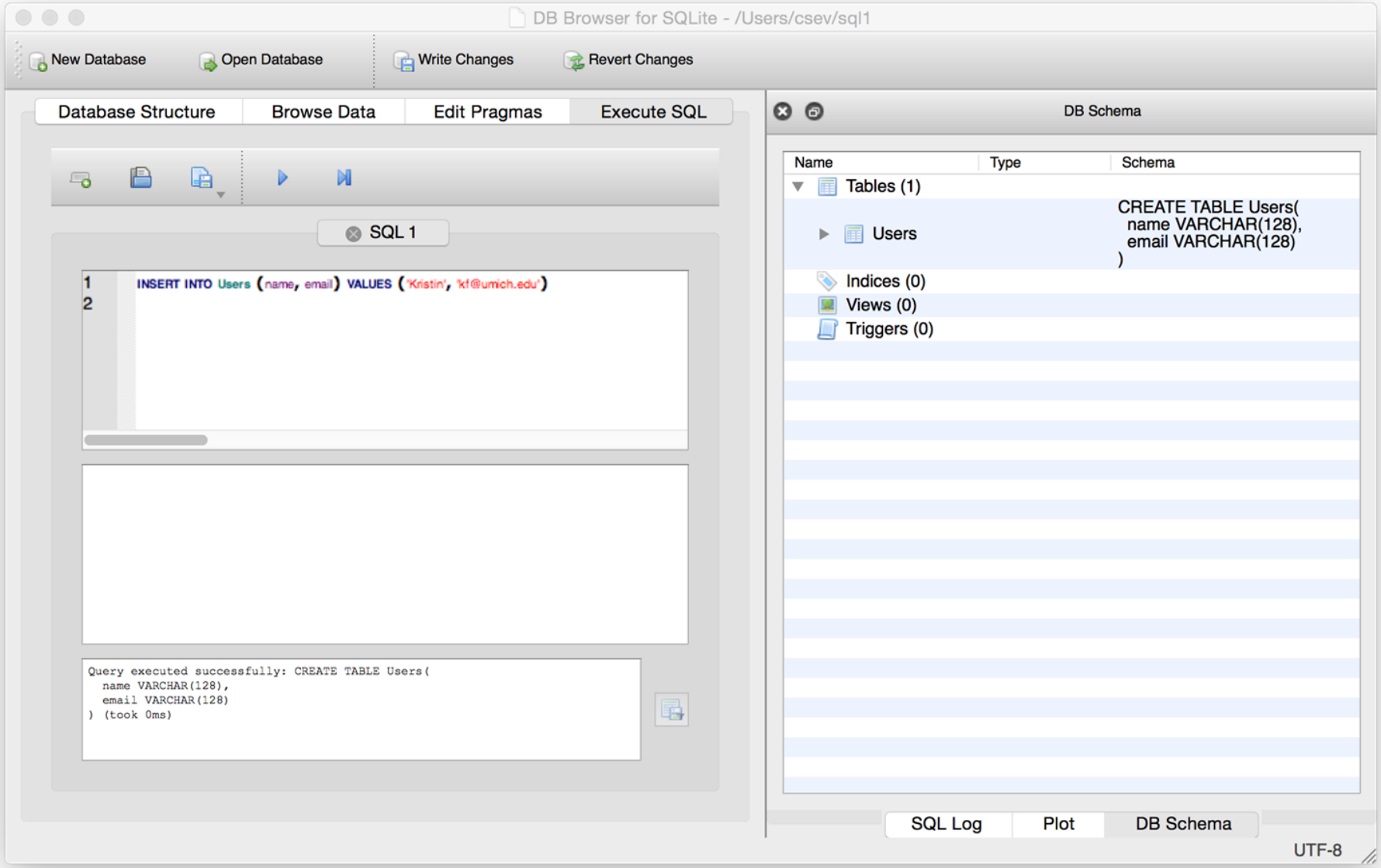
- Create data (a.k.a Insert)
- Retrieve data
- Update data
- Delete data

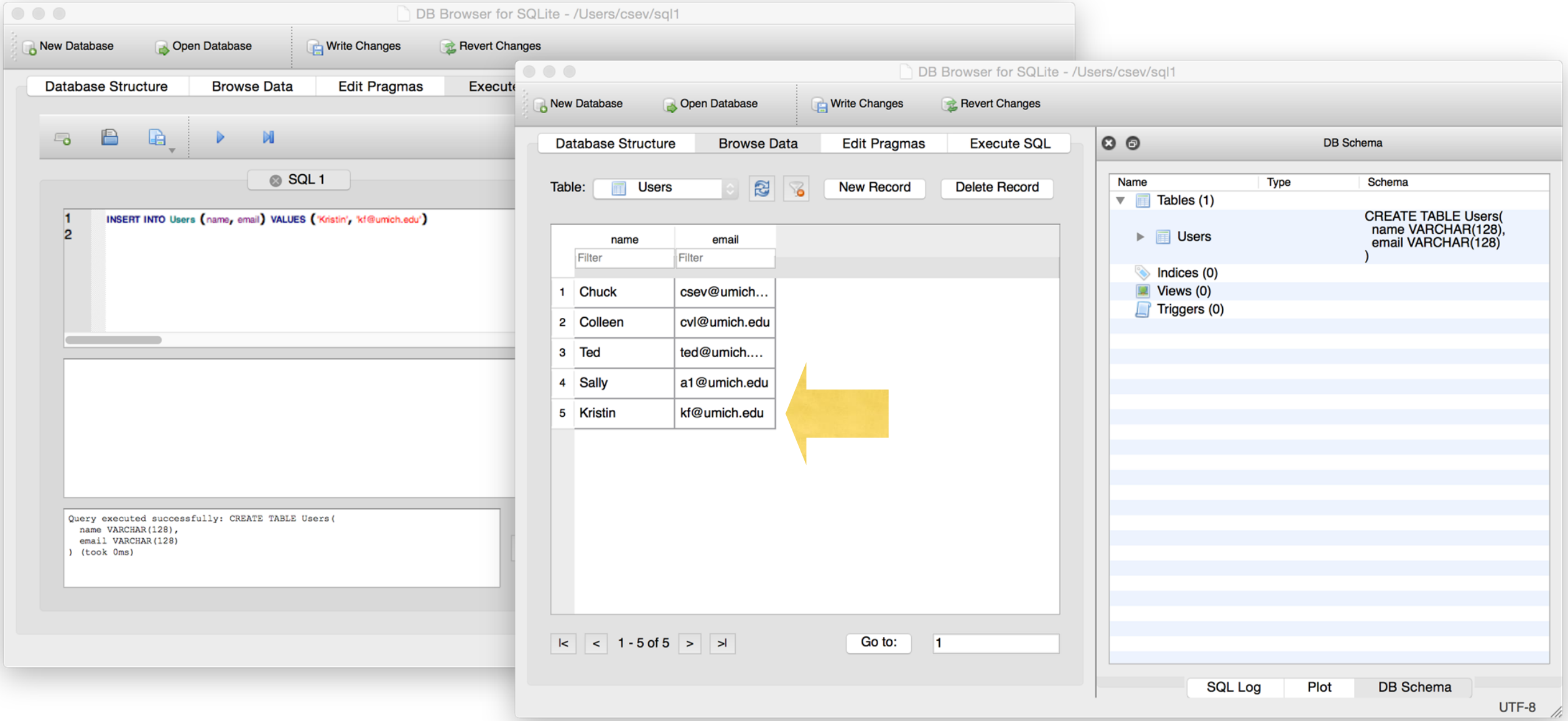


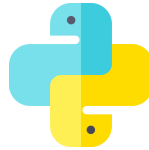
SQL: Insert

The Insert statement inserts a row into a table

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


DB Browser for SQLite - /Users/csev/sql1
New Database
Open Database
Write Changes
Revert Changes
Database Structure
Browse Data
Edit Pragmas
Execute SQL
SQL 1
1
2
INSERT INTO Users (name, email) VALUES ('Kristin', 'kf@umich.edu')
Query executed successfully: CREATE TABLE Users(
name VARCHAR(128),
email VARCHAR(128)
) (took 0ms)
DB Schema
Name
Type
Schema
Tables (1)
Users
CREATE TABLE Users(
name VARCHAR(128),
email VARCHAR(128)
)
Indices (0)
Views (0)
Triggers (0)
SQL Log
Plot
DB Schema
UTF-8

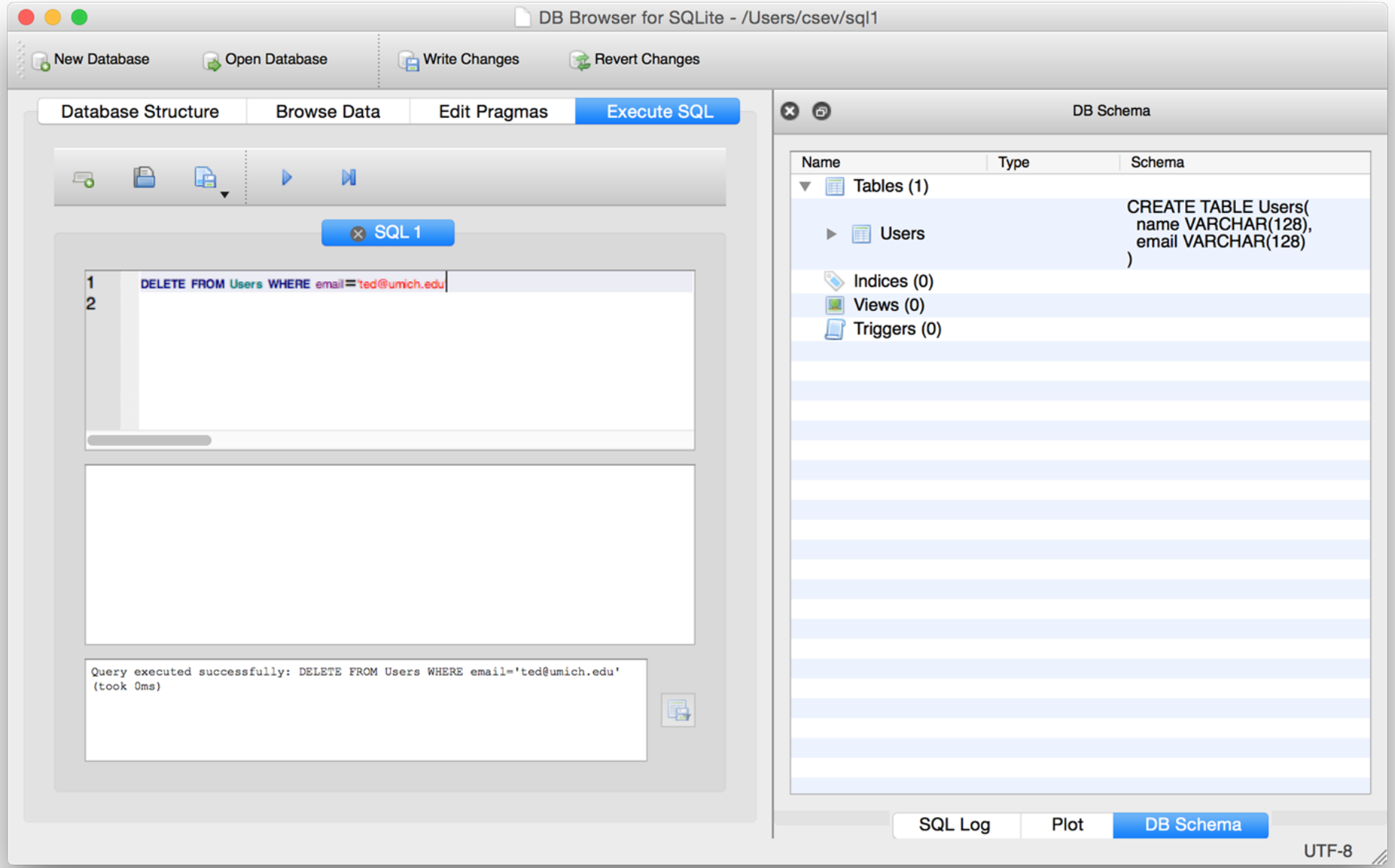

DB Browser for SQLite - /Users/csev/sql1
New Database
Open Database
Write Changes
Revert Changes
Database Structure
Browse Data
Edit Pragmas
Execute SQL
SQL 1
1 INSERT INTO Users (name, email) VALUES ('Kristin', 'kf@umich.edu')
2
Query executed successfully: CREATE TABLE Users(
name VARCHAR(128),
email VARCHAR(128)
) (took 0ms)
Table: Users
New Record
Delete Record
name
email
Filter
1 Chuck csev@umich...
2 Colleen cvl@umich.edu
3 Ted ted@umich....
4 Sally a1@umich.edu
5 Kristin kf@umich.edu
|< < 1 - 5 of 5 > >|
Go to: 1
DB Schema
Name
Type
Schema
Tables (1)
Users
CREATE TABLE Users(
name VARCHAR(128),
email VARCHAR(128)
)
Indices (0)
Views (0)
Triggers (0)
SQL Log
Plot
DB Schema
UTF-8

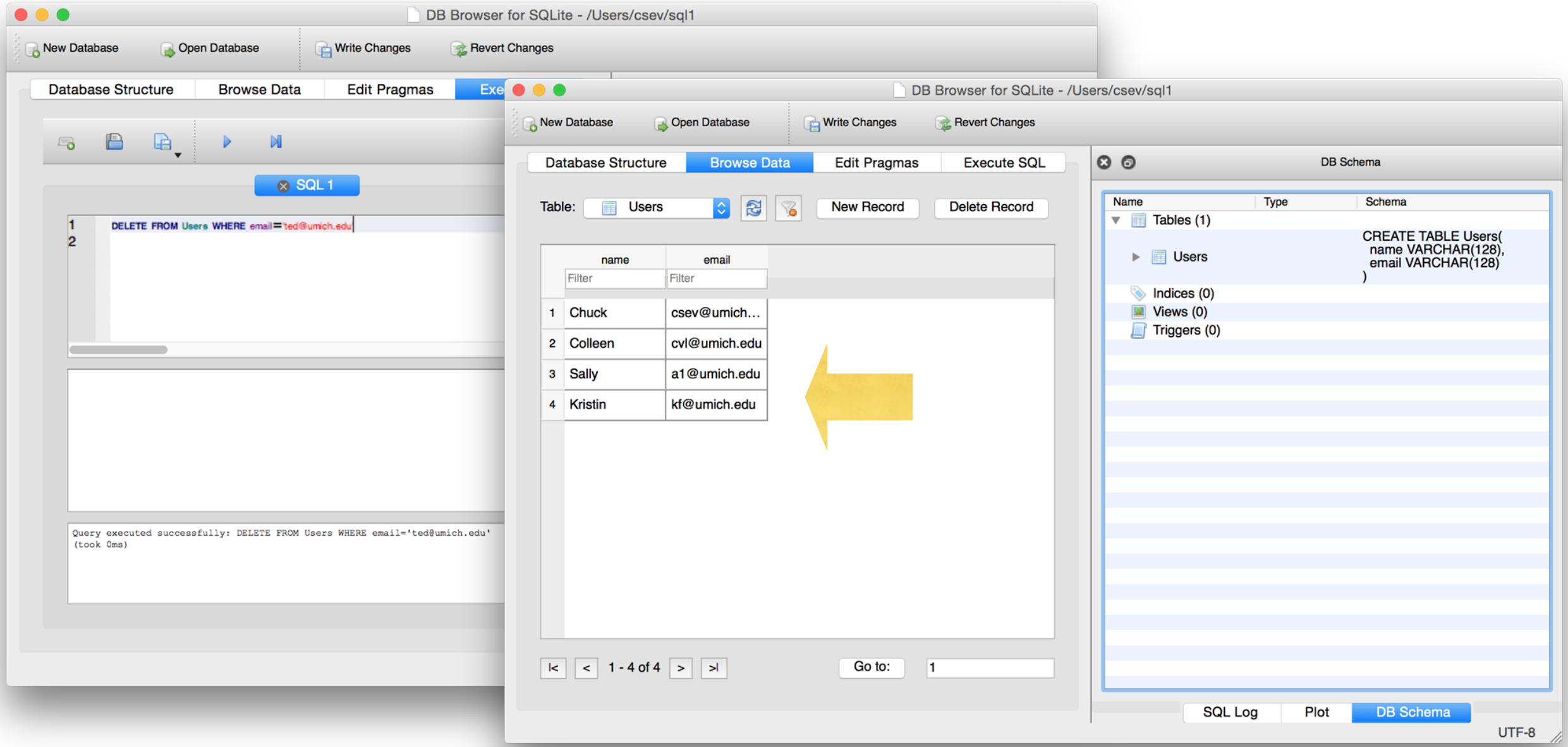


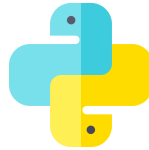
SQL: Delete

Deletes a row in a table based on selection criteria

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


DB Browser for SQLite - /Users/csev/sql1
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)
DB Schema
Name
Type
Schema
Tables (1)
Users
CREATE TABLE Users(
name VARCHAR(128),
email VARCHAR(128)
)
Indices (0)
Views (0)
Triggers (0)
SQL Log
Plot
DB Schema
UTF-8

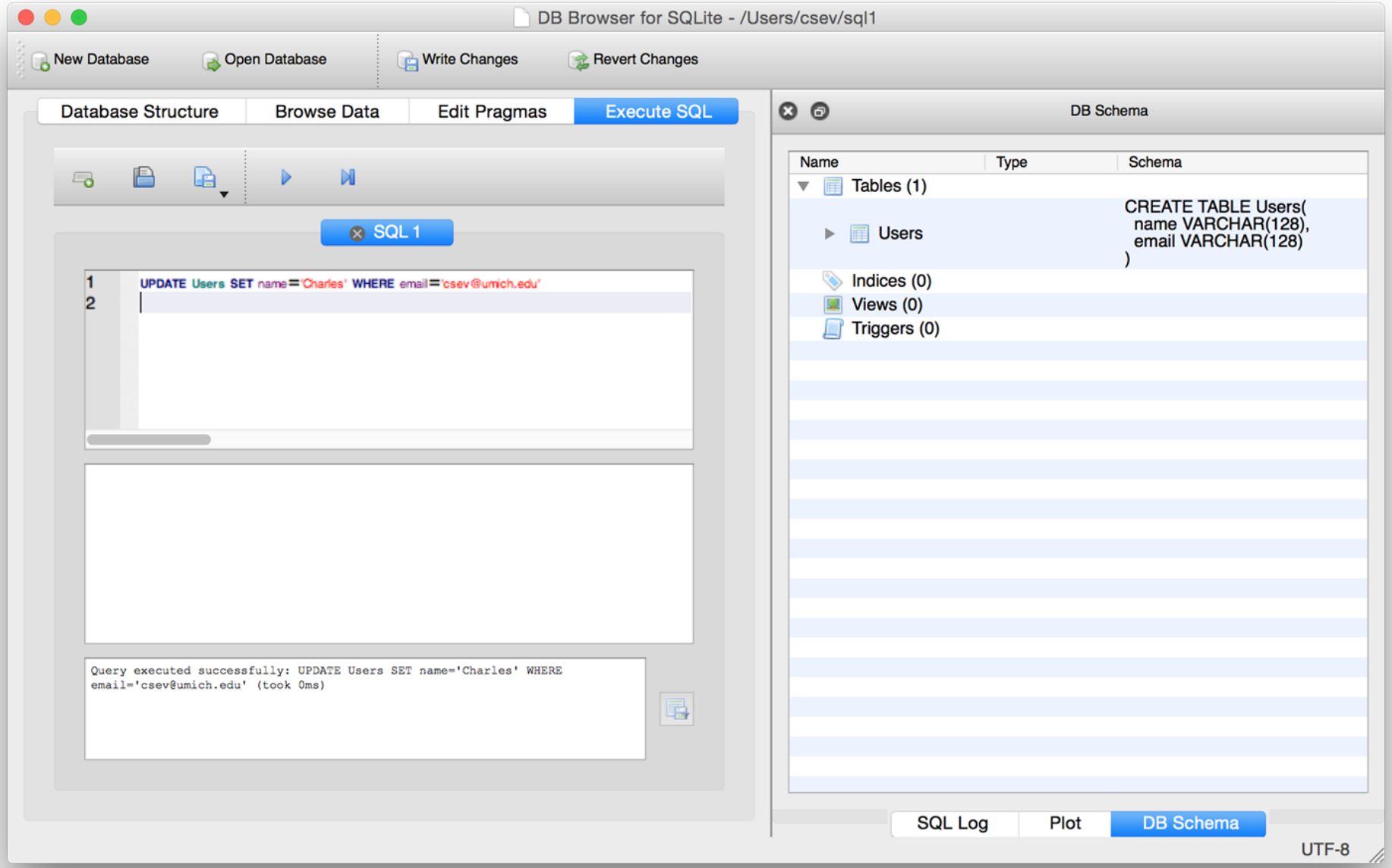

DB Browser for SQLite - /Users/csev/sql1
New Database
Open Database
Write Changes
Revert Changes
Database Structure
Browse Data
Edit Pragmas
Execute SQL
SQL 1
1 DELETE FROM Users WHERE email=ted@umich.edu
2
Query executed successfully: DELETE FROM Users WHERE email='ted@umich.edu' (took 0ms)
Table: Users
New Record
Delete Record
name
email
Filter
1 Chuck csev@umich...
2 Colleen cvl@umich.edu
3 Sally a1@umich.edu
4 Kristin kf@umich.edu
1 - 4 of 4
Go to: 1
DB Schema
Name
Type
Schema
Tables (1)
Users
CREATE TABLE Users(name VARCHAR(128), email VARCHAR(128))
Indices (0)
Views (0)
Triggers (0)
SQL Log
Plot
DB Schema
UTF-8

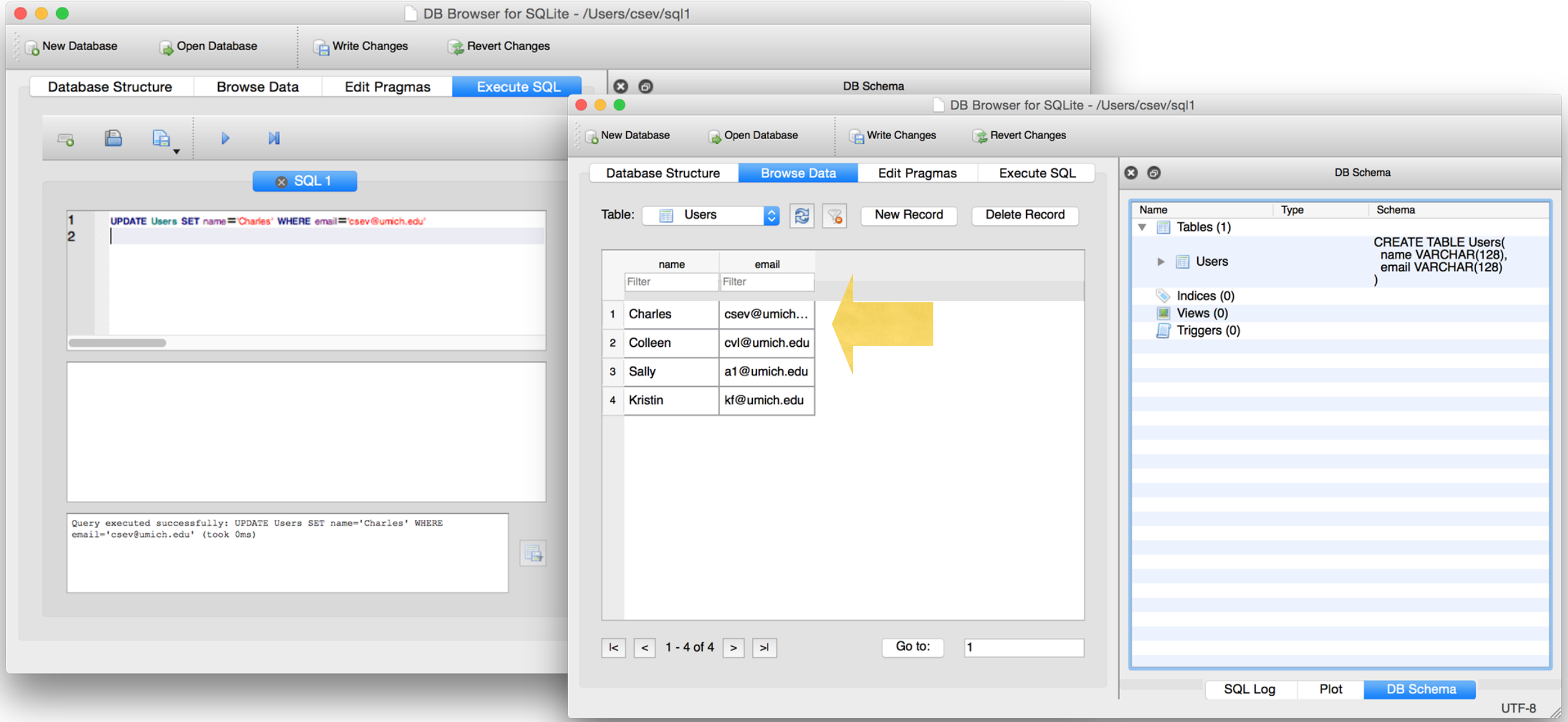


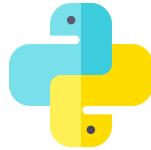
SQL: Update

Allows the updating of a field with a where clause

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


DB Browser for SQLite - /Users/csev/sql1
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)
DB Schema
Name
Type
Schema
Tables (1)
Users
CREATE TABLE Users(name VARCHAR(128), email VARCHAR(128))
Indices (0)
Views (0)
Triggers (0)
SQL Log
Plot
DB Schema
UTF-8



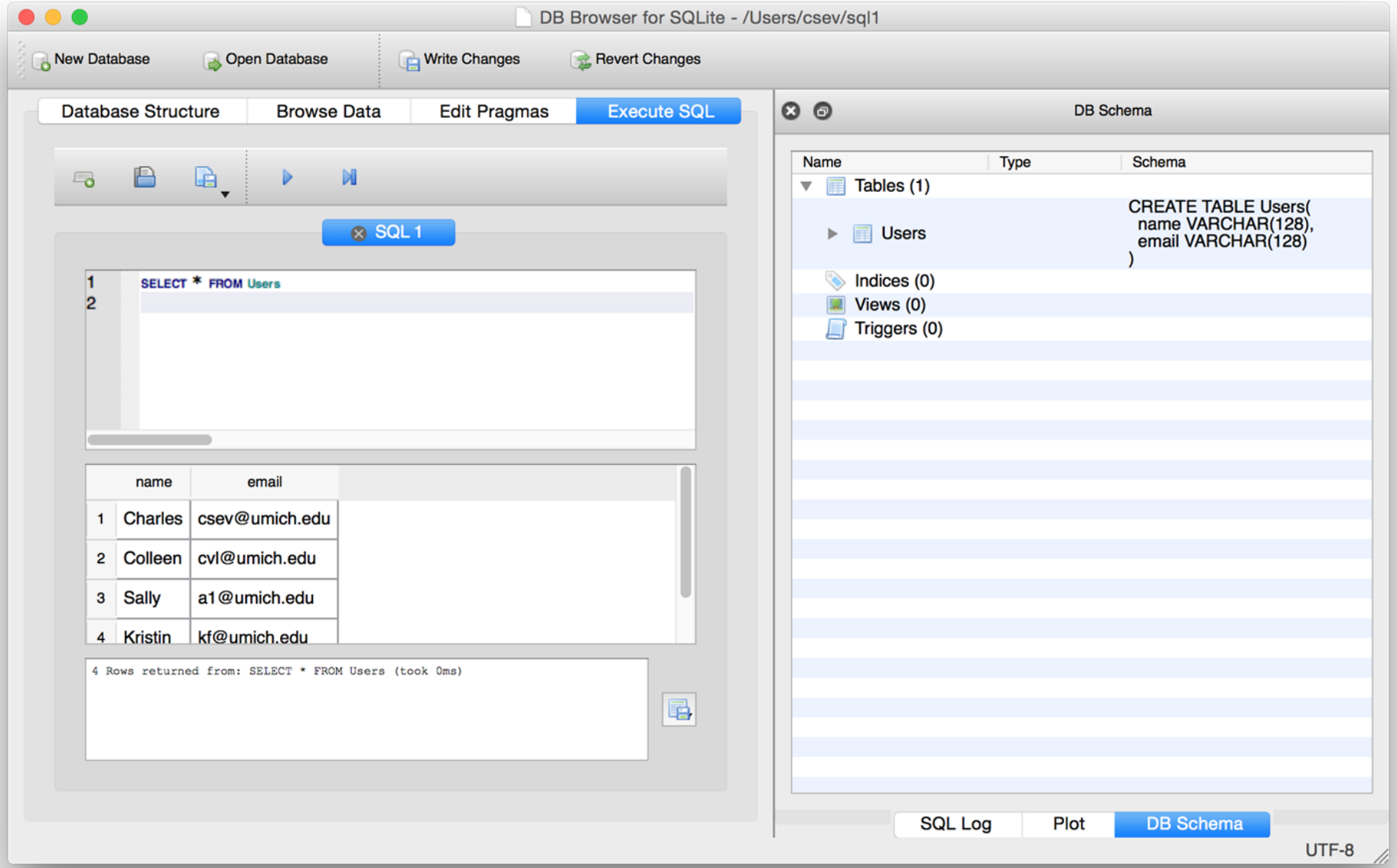


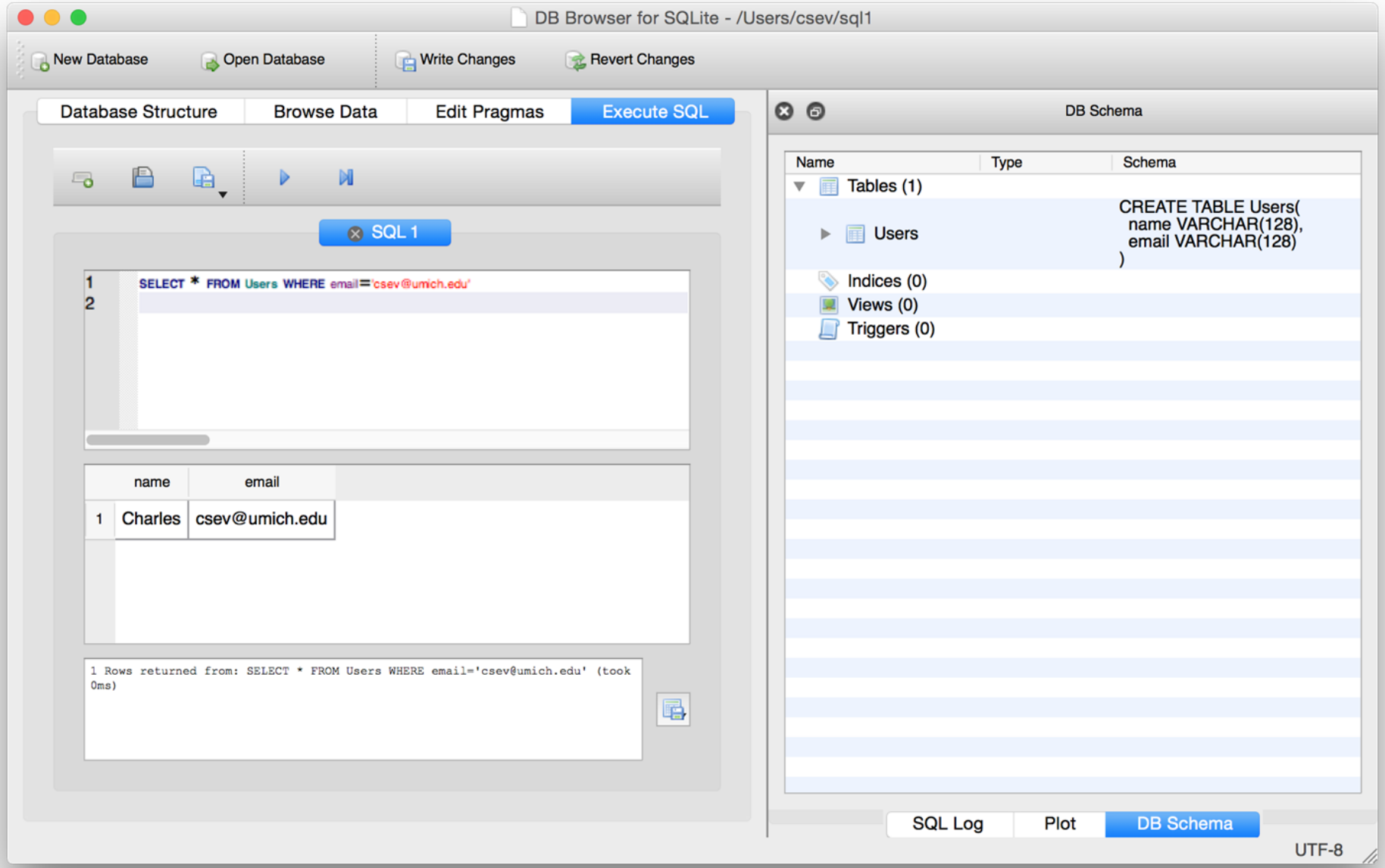
Retrieving Records: Select

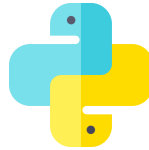
The select statement retrieves a group of records - you can either retrieve all the records or a subset of the records with a WHERE clause

```
SELECT * FROM Users
```

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


DB Browser for SQLite - /Users/csev/sql1
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 Kristin kf@umich.edu
4 Rows returned from: SELECT * FROM Users (took 0ms)
DB Schema
Name Type Schema
Tables (1)
Users
CREATE TABLE Users(
name VARCHAR(128),
email VARCHAR(128)
)
Indices (0)
Views (0)
Triggers (0)
SQL Log
Plot
DB Schema
UTF-8


DB Browser for SQLite - /Users/csev/sql1
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 from: SELECT * FROM Users WHERE email='csev@umich.edu' (took 0ms)
DB Schema
Name
Type
Schema
Tables (1)
Users
CREATE TABLE Users(name VARCHAR(128), email VARCHAR(128))
Indices (0)
Views (0)
Triggers (0)
SQL Log
Plot
DB Schema
UTF-8

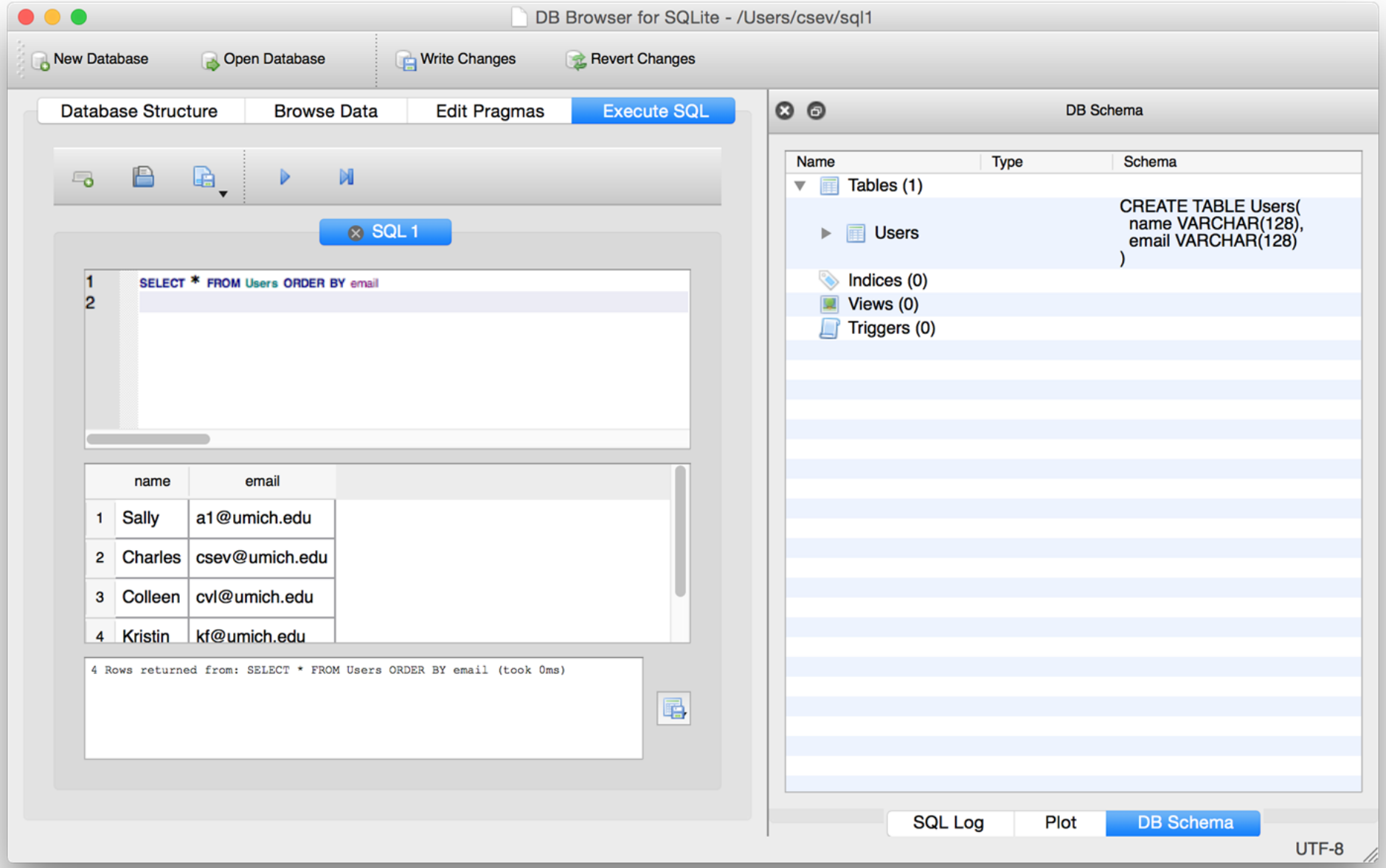


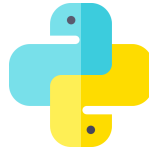
Sorting with ORDER BY

- You can add an ORDER BY clause to SELECT statements to get the results sorted in ascending or descending order

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

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


DB Browser for SQLite - /Users/csev/sql1
New Database
Open Database
Write Changes
Revert Changes
Database Structure
Browse Data
Edit Pragmas
Execute SQL
SQL 1
1 SELECT * FROM Users ORDER BY email
2
name email
1 Sally a1@umich.edu
2 Charles csev@umich.edu
3 Colleen cvl@umich.edu
4 Kristin kf@umich.edu
4 Rows returned from: SELECT * FROM Users ORDER BY email (took 0ms)
DB Schema
Name Type Schema
Tables (1)
Users
CREATE TABLE Users(
name VARCHAR(128),
email VARCHAR(128)
)
Indices (0)
Views (0)
Triggers (0)
SQL Log
Plot
DB Schema
UTF-8

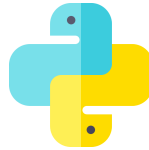


SQL Summary

```
INSERT INTO Users (name, email) VALUES ('Kristin', '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
```

Data Models

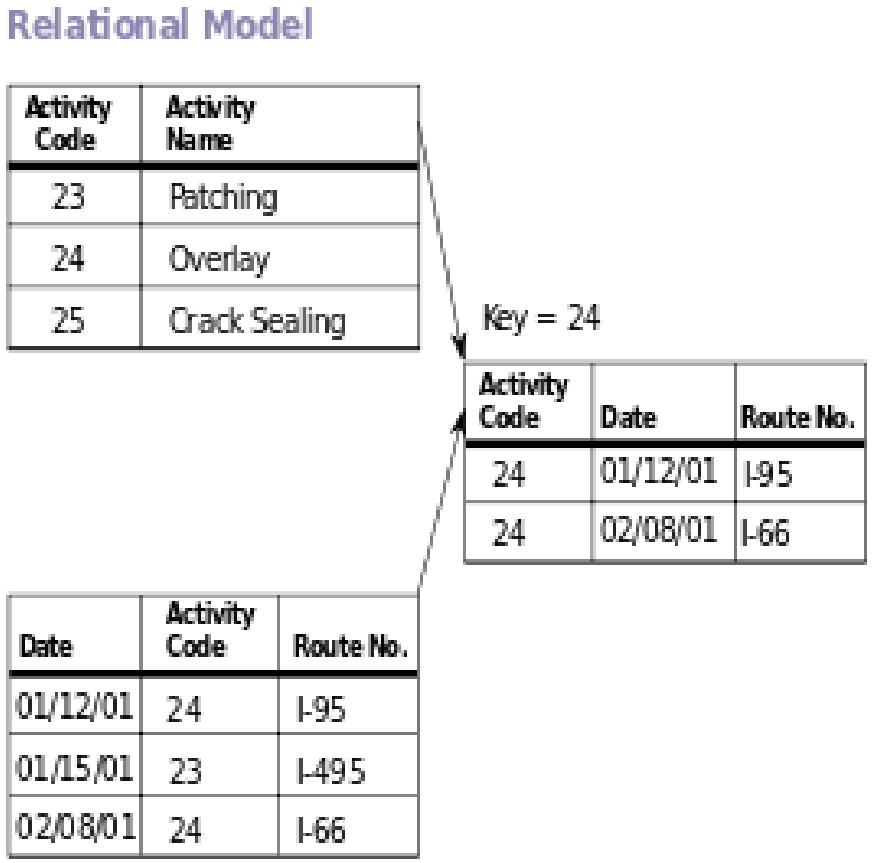
LECTURE 4

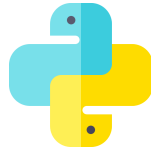


From the Use of Tools to Programming

- Tables pretty much look like big fast programmable spreadsheets with rows, columns, and commands
- The power comes when we have more than one table and we can exploit the relationships between the tables

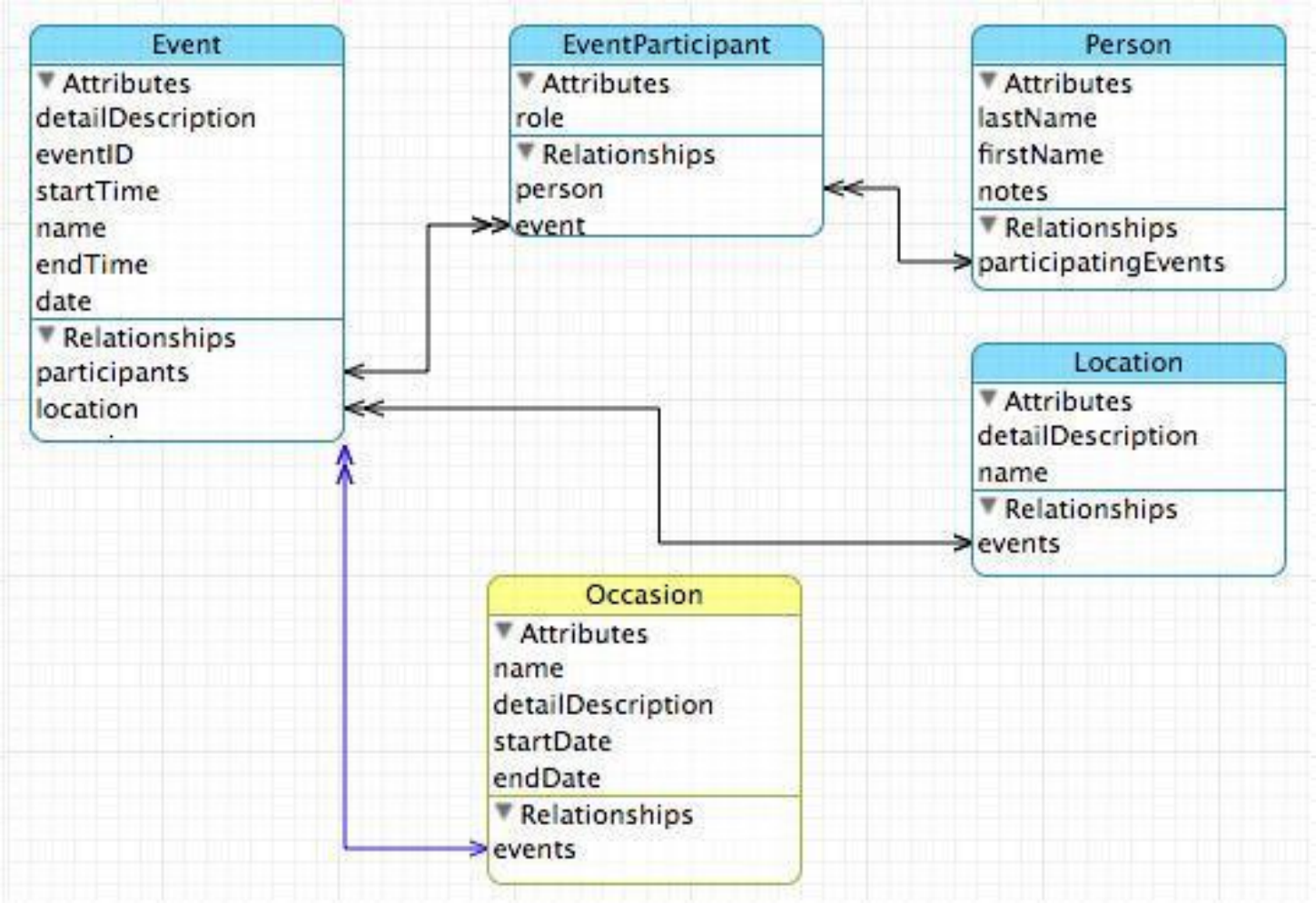
Complex Data Models and Relationships

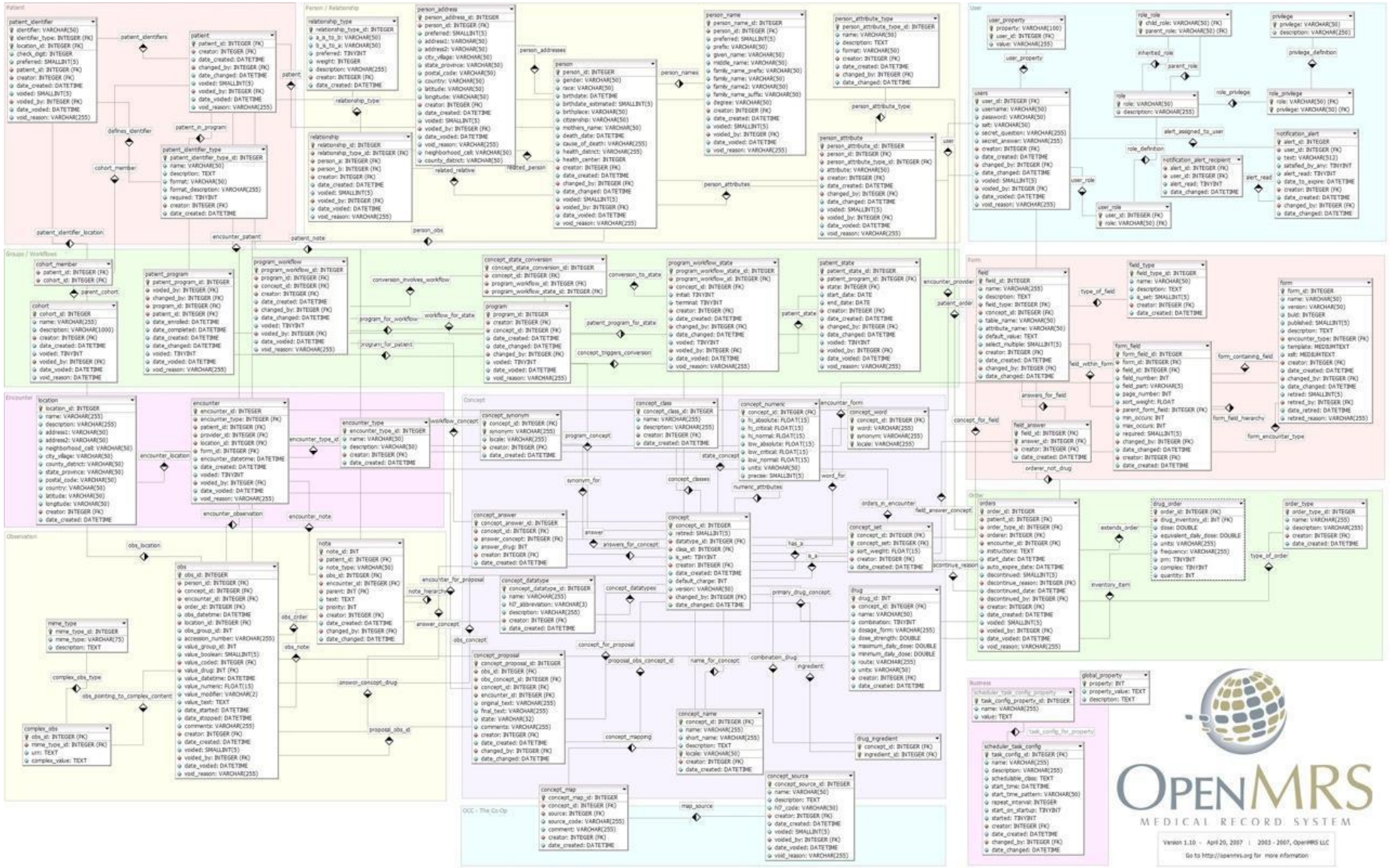


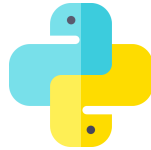


Database Design

- Database design is an art form of its own with particular skills and experience
- Our goal is to avoid the really bad mistakes and design clean and easily understood databases
- Others may performance tune things later
- Database design starts with a picture...


Event
Attributes
detailDescription
eventID
startTime
name
endTime
date
Relationships
participants
location
EventParticipant
Attributes
role
Relationships
person
event
Person
Attributes
lastName
firstName
notes
Relationships
participatingEvents
Location
Attributes
detailDescription
name
Relationships
events
Occasion
Attributes
name
detailDescription
startDate
endDate
Relationships
events





Building a Data Model

- Drawing a picture of the data objects for our application and then figuring out how to represent the objects and their relationships
- Basic Rule: Don't put the same string data in twice - use a relationship instead
- When there is one thing in the "real world" there should be one copy of that thing in the database

Hells Bells	5:13	AC/DC	Who Made Who	Rock	★★★★★	61
Shake Your Foundations	3:54	AC/DC	Who Made Who	Rock	★★★★★	70
Chase the Ace	3:01	AC/DC	Who Made Who	Rock		56
For Those About To Rock (We ...	5:54	AC/DC	Who Made Who	Rock	★★★★★	61
Dúlamán	3:43	Altan	Natural Wonders M...	New Age		31
Rode Across the Desert	4:10	America	Greatest Hits	Easy Listen...	★★★★★	23
Now You Are Gone	3:08	America	Greatest Hits	Easy Listen...	★★★★★	18
Tin Man	3:30	America	Greatest Hits	Easy Listen...	★★★★★	23
Sister Golden Hair	3:22	America	Greatest Hits	Easy Listen...	★★★★★	24
Track 01	4:22	Billy Price	Danger Zone	Blues/R&B	★★★★★	26
Track 02	2:45	Billy Price	Danger Zone	Blues/R&B	★★★★★	18
Track 03	3:26	Billy Price	Danger Zone	Blues/R&B	★★★★★	22
Track 04	4:17	Billy Price	Danger Zone	Blues/R&B	★★★★★	18
Track 05	3:50	Billy Price	Danger Zone	Blues/R&B	★★★★★	21
War Pigs/Luke's Wall	7:58	Black Sabbath	Paranoid	Metal	★★★★★	25
Paranoid	2:53	Black Sabbath	Paranoid	Metal	★★★★★	22
Planet Caravan	4:35	Black Sabbath	Paranoid	Metal	★★★★★	25
Iron Man	5:59	Black Sabbath	Paranoid	Metal	★★★★★	26
Electric Funeral	4:53	Black Sabbath	Paranoid	Metal	★★★★★	22
Hand of Doom	7:10	Black Sabbath	Paranoid	Metal	★★★★★	23
Rat Salad	2:30	Black Sabbath	Paranoid	Metal	★★★★★	31
Jack the Stripper/Fairies Wear ...	6:14	Black Sabbath	Paranoid	Metal	★★★★★	24
Bomb Squad (TECH)	3:28	Brent	Brent's Album			1
clay techno	4:36	Brent	Brent's Album			2
Heavy	3:08	Brent	Brent's Album			1
Hi metal man	4:20	Brent	Brent's Album			1
Mistro	2:58	Brent	Brent's Album			1

For each “piece of info”…

- Is the column an object or an attribute of another object?
- Once we define objects, we need to define the relationships between objects

Len

Album

Genre

Artist

Rating

Track

Count

Hells Bells	5:13	AC/DC	Who Made Who	Rock	★★★★★	61
Shake Your Foundations	3:54	AC/DC	Who Made Who	Rock	★★★★★	70
Chase the Ace	3:01	AC/DC	Who Made Who	Rock		56
For Those About To Rock (We …	5:54	AC/DC	Who Made Who	Rock	★★★★★	61
Dúlamán	3:43	Altan	Natural Wonders M…	New Age		31
Rode Across the Desert	4:10	America	Greatest Hits	Easy Listen…	★★★★★	23
Now You Are Gone	3:08	America	Greatest Hits	Easy Listen…	★★★★★	18

Track

Album

Artist

Genre

Rating

Len

Count

Hells Bells	5:13	AC/DC	Who Made Who	Rock	★★★★★	61
Shake Your Foundations	3:54	AC/DC	Who Made Who	Rock	★★★★★	70
Chase the Ace	3:01	AC/DC	Who Made Who	Rock		56
For Those About To Rock (We …	5:54	AC/DC	Who Made Who	Rock	★★★★★	61
Dúlamán	3:43	Altan	Natural Wonders M…	New Age		31
Rode Across the Desert	4:10	America	Greatest Hits	Easy Listen…	★★★★★	23
Now You Are Gone	3:08	America	Greatest Hits	Easy Listen…	★★★★★	18

Track

Album

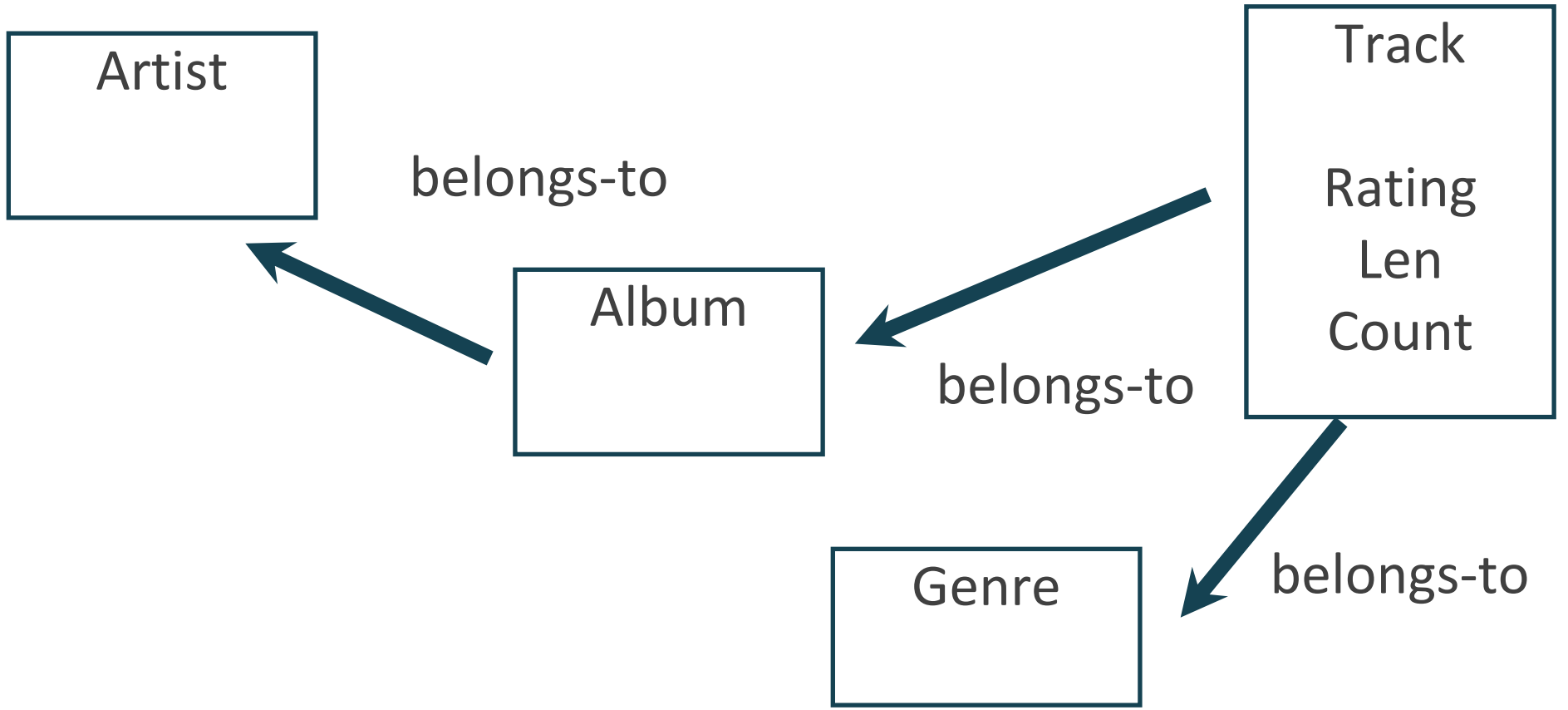
Artist

Genre

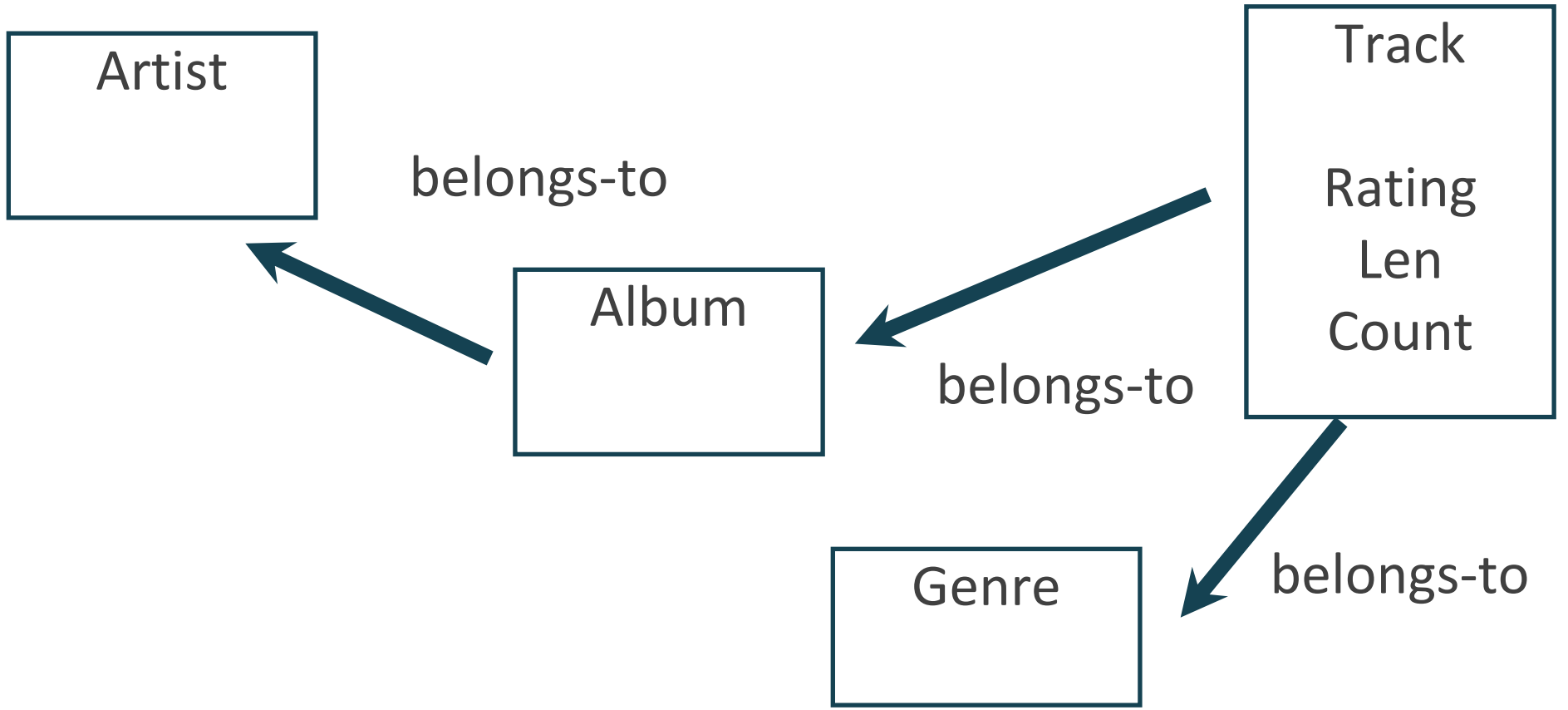
Rating

Len

Count



Hells Bells	5:13	AC/DC	Who Made Who	Rock	★★★★★	61
Shake Your Foundations	3:54	AC/DC	Who Made Who	Rock	★★★★★	70
Chase the Ace	3:01	AC/DC	Who Made Who	Rock		56
For Those About To Rock (We ...	5:54	AC/DC	Who Made Who	Rock	★★★★★	61
Dúlamán	3:43	Altan	Natural Wonders M...	New Age		31
Rode Across the Desert	4:10	America	Greatest Hits	Easy Listen...	★★★★★	23
Now You Are Gone	3:08	America	Greatest Hits	Easy Listen...	★★★★★	18



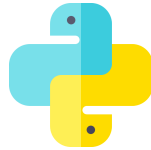
Hells Bells	5:13	AC/DC	Who Made Who	Rock	★★★★★	61
Shake Your Foundations	3:54	AC/DC	Who Made Who	Rock	★★★★★	70
Chase the Ace	3:01	AC/DC	Who Made Who	Rock		56
For Those About To Rock (We ...	5:54	AC/DC	Who Made Who	Rock	★★★★★	61
Dúlamán	3:43	Altan	Natural Wonders M...	New Age		31
Rode Across the Desert	4:10	America	Greatest Hits	Easy Listen...	★★★★★	23
Now You Are Gone	3:08	America	Greatest Hits	Easy Listen...	★★★★★	18

Relational Database

LECTURE 5

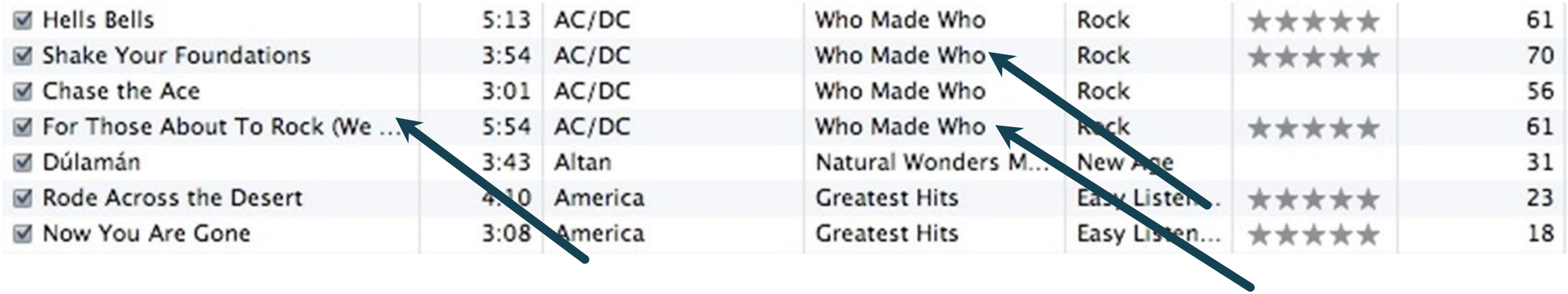
Representing Relationships in a Database

ACTIVITY



Database Normalization (3NF)

- There is *tons* of database theory - way too much to understand without excessive predicate calculus
- Do not replicate data - reference data - point at data
- Use integers for keys and for references
- Add a special “key” column to each table which we will make references to. By convention, many programmers call this column “id”



Hells Bells	5:13	AC/DC	Who Made Who	Rock	★★★★★	61
Shake Your Foundations	3:54	AC/DC	Who Made Who	Rock	★★★★★	70
Chase the Ace	3:01	AC/DC	Who Made Who	Rock		56
For Those About To Rock (We ...	5:54	AC/DC	Who Made Who	Rock	★★★★★	61
Dúlamán	3:43	Altan	Natural Wonders M...	New Age		31
Rode Across the Desert	4:10	America	Greatest Hits	Easy Listen...	★★★★★	23
Now You Are Gone	3:08	America	Greatest Hits	Easy Listen...	★★★★★	18

- We want to keep track of which band is the “creator” of each music track...
- What album does this song “belong to”??
- Which album is this song related to?

Integer Reference Pattern

We use integers to reference rows in another table

Artist

id	name
Filter	Filter
1	Led Zepplin
2	AC/DC

Album

id	artist_id	title
Filter	Filter	Filter
1	2	Who Made Who
2	1	IV

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
...

Key Rules

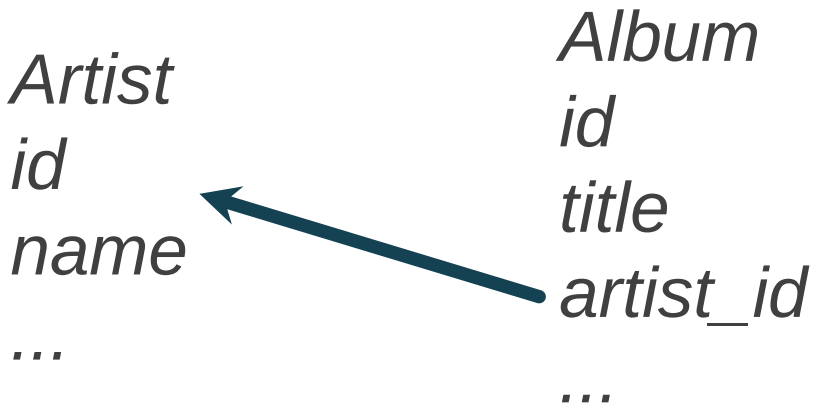
Best practices

- Never use your logical key as the primary key
- Logical keys can and do change, albeit slowly
- Relationships that are based on matching string fields are less efficient than integers

User
id
login
password
name
email
created_at
modified_at
login_at

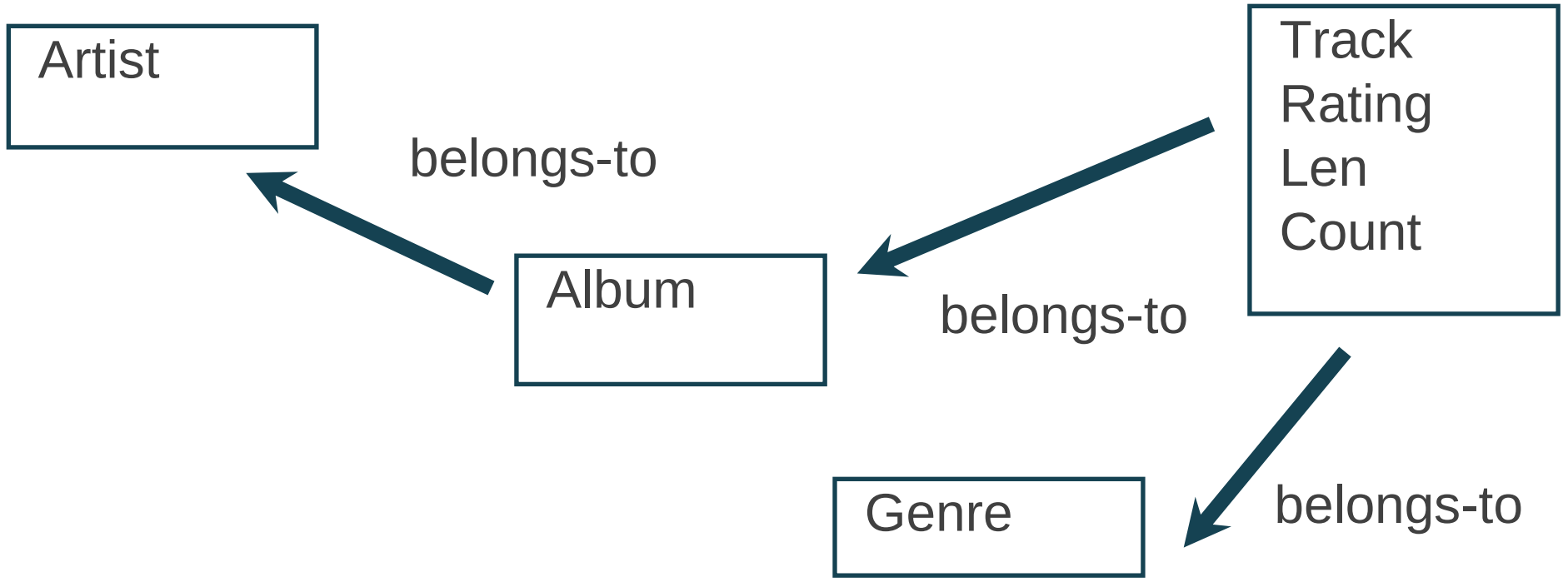
Foreign Keys

- A foreign key is when a table has a column that contains a key which points to the primary key of another table.
- When all primary keys are integers, then all foreign keys are integers - this is good - very good

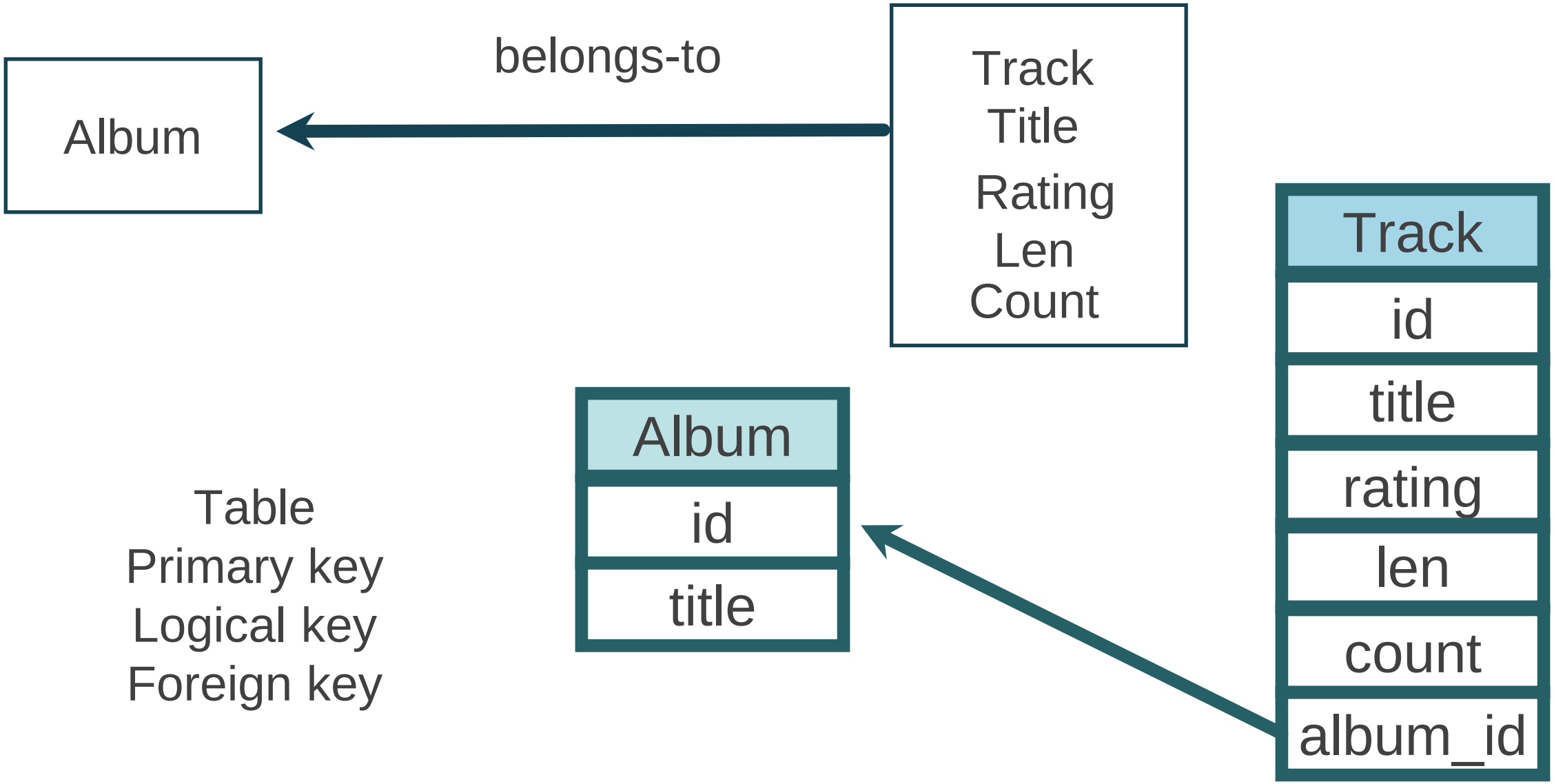


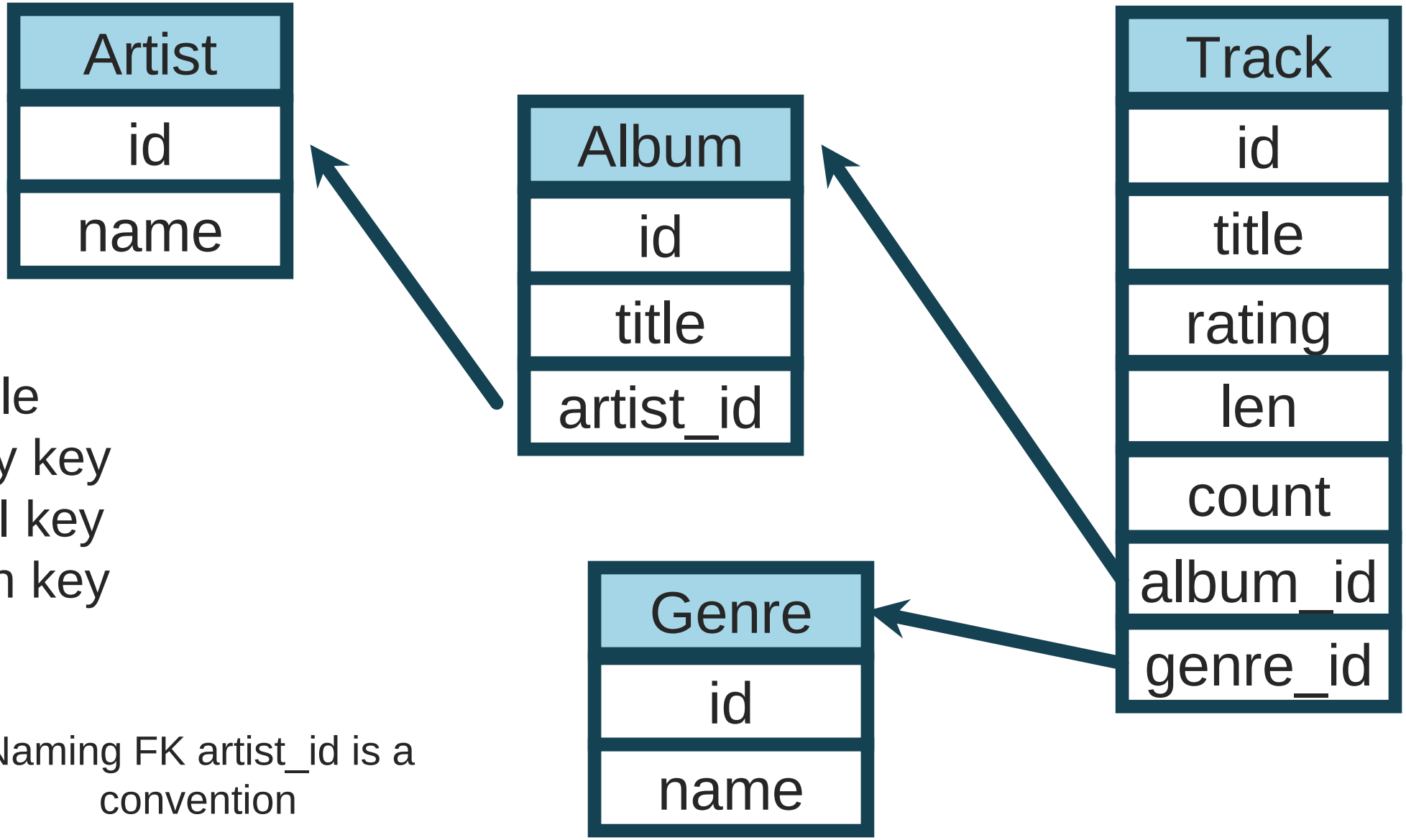
Relationship Building (in tables)

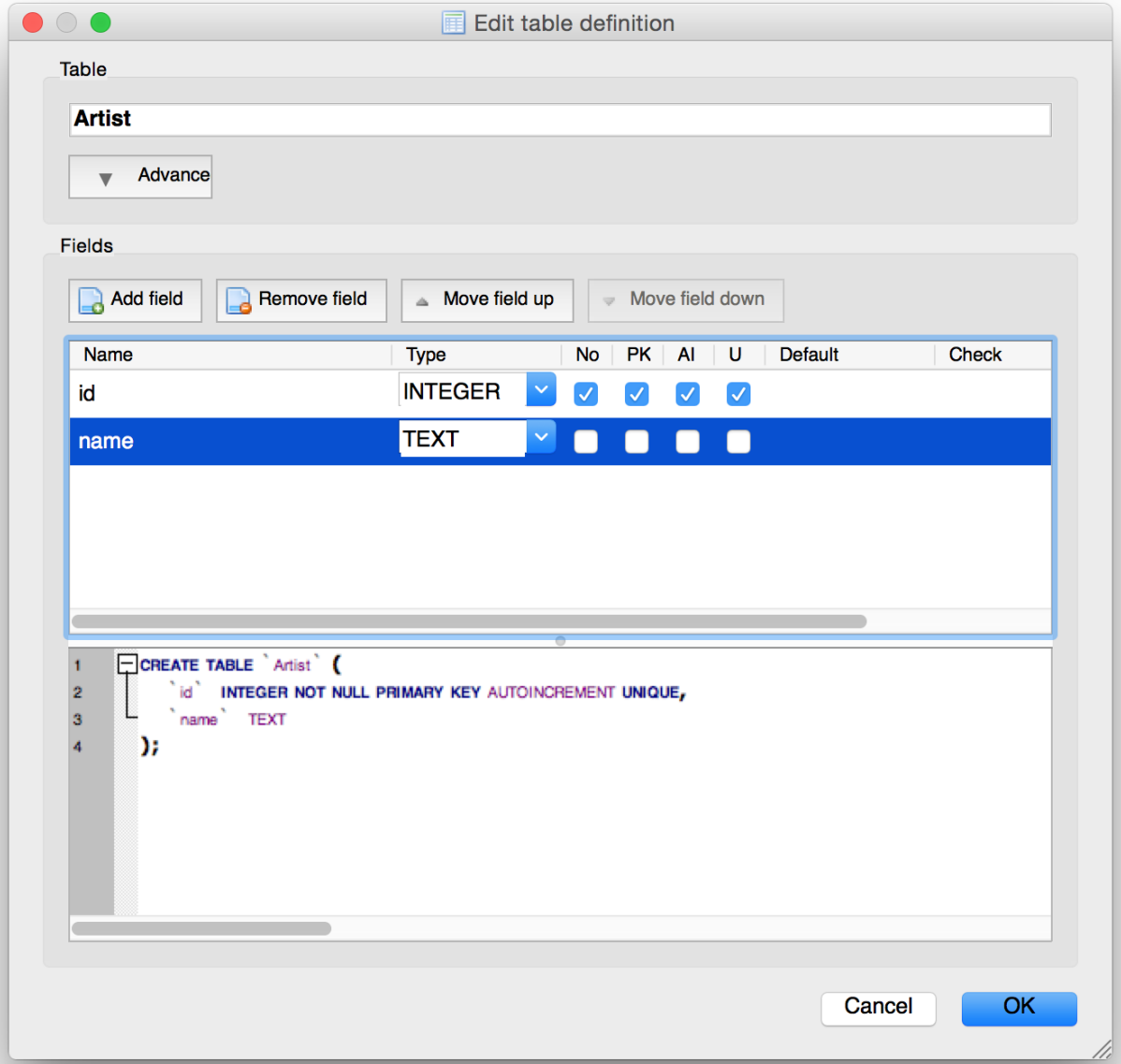
ACTIVITY

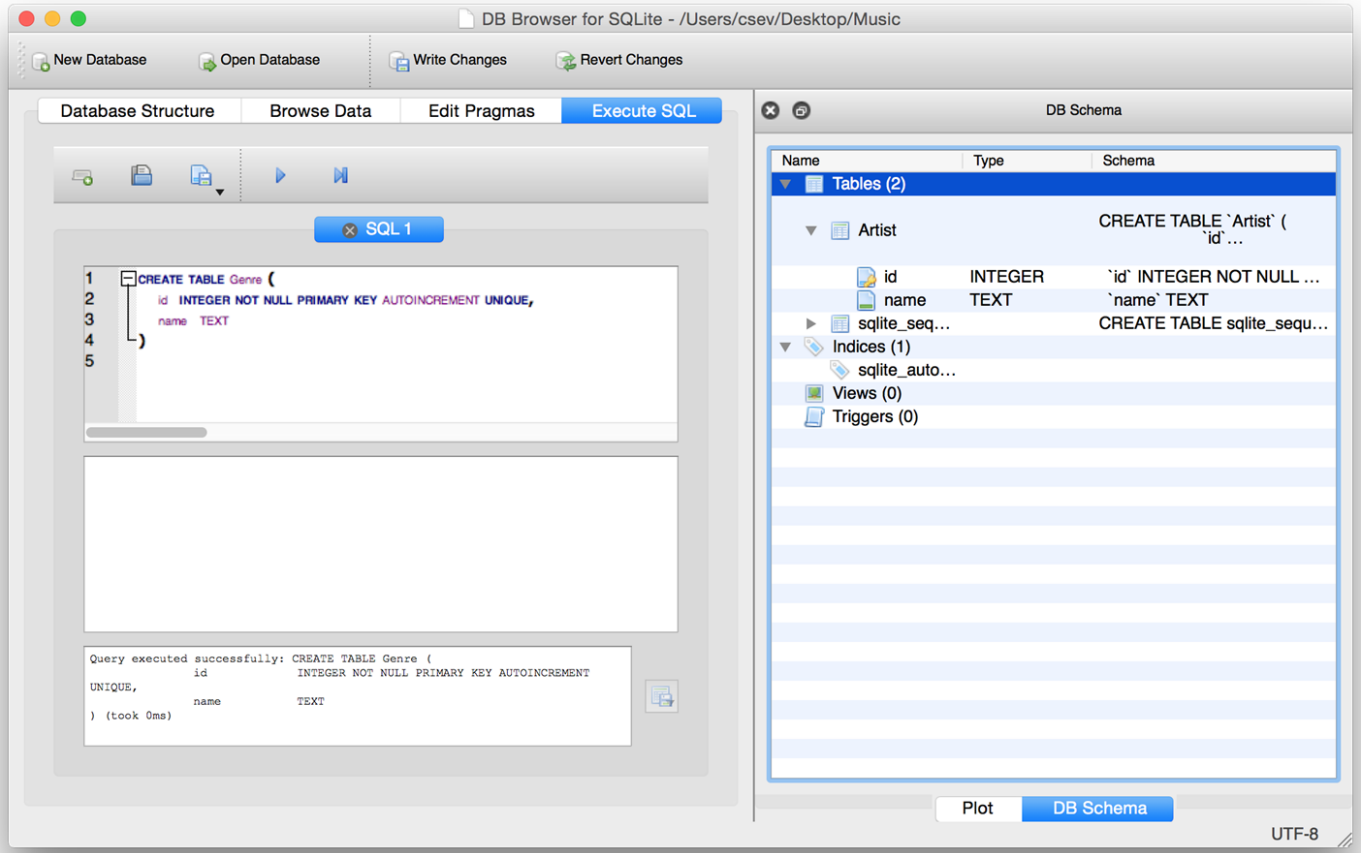


Hells Bells	5:13	AC/DC	Who Made Who	Rock	★★★★★	61
Shake Your Foundations	3:54	AC/DC	Who Made Who	Rock	★★★★★	70
Chase the Ace	3:01	AC/DC	Who Made Who	Rock		56
For Those About To Rock (We ...	5:54	AC/DC	Who Made Who	Rock	★★★★★	61
Dúlamán	3:43	Altan	Natural Wonders M...	New Age		31
Rode Across the Desert	4:10	America	Greatest Hits	Easy Listen...	★★★★★	23
Now You Are Gone	3:08	America	Greatest Hits	Easy Listen...	★★★★★	18


Album
belongs-to
Track
Title
Rating
Len
Count
Track
id
title
rating
len
count
album_id
Album
id
title
Table
Primary key
Logical key
Foreign key

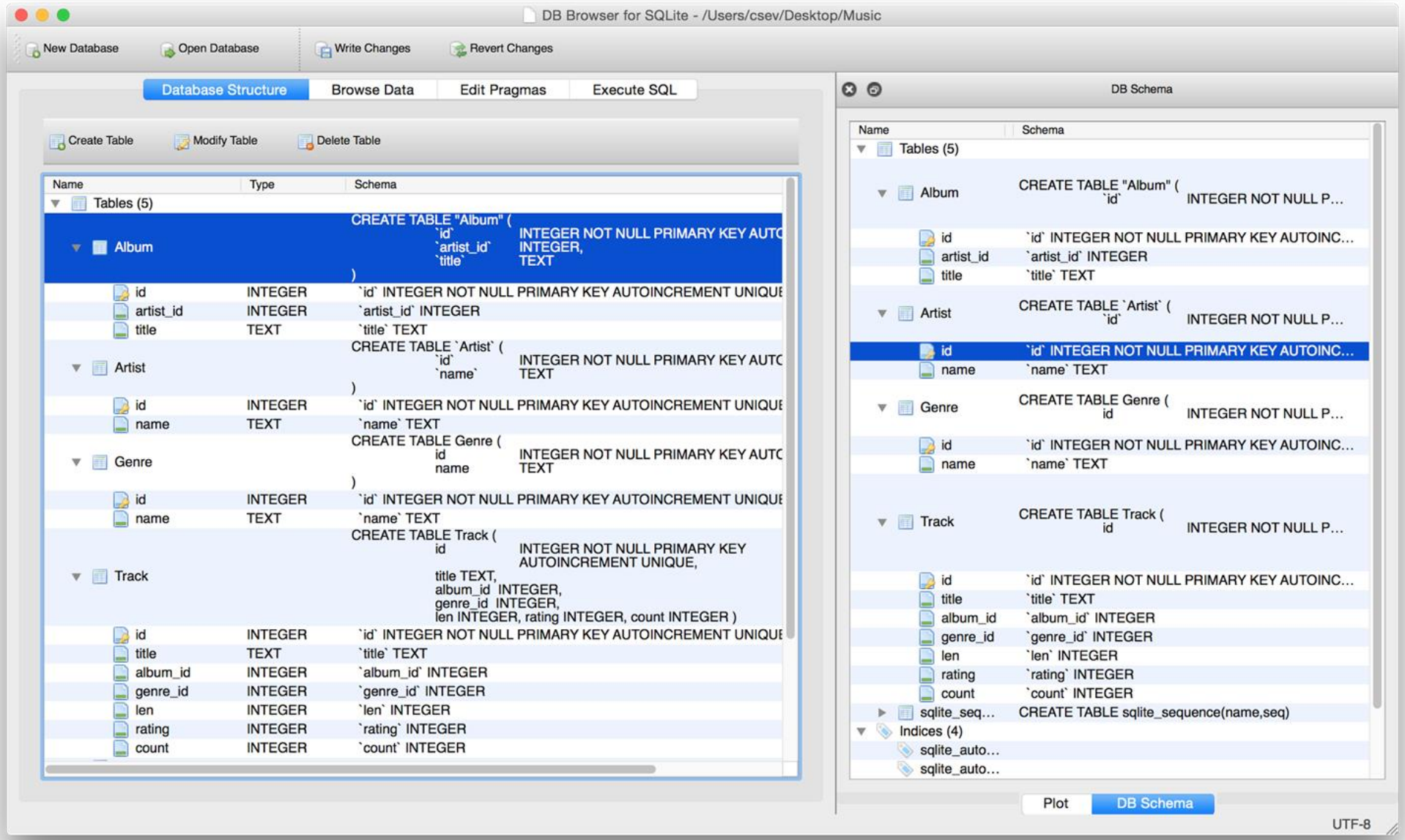

Artist
id
name
Album
id
title
artist_id
Track
id
title
rating
len
count
album_id
genre_id
Genre
id
name
Table
Primary key
Logical key
Foreign key
Naming FK artist_id is a convention

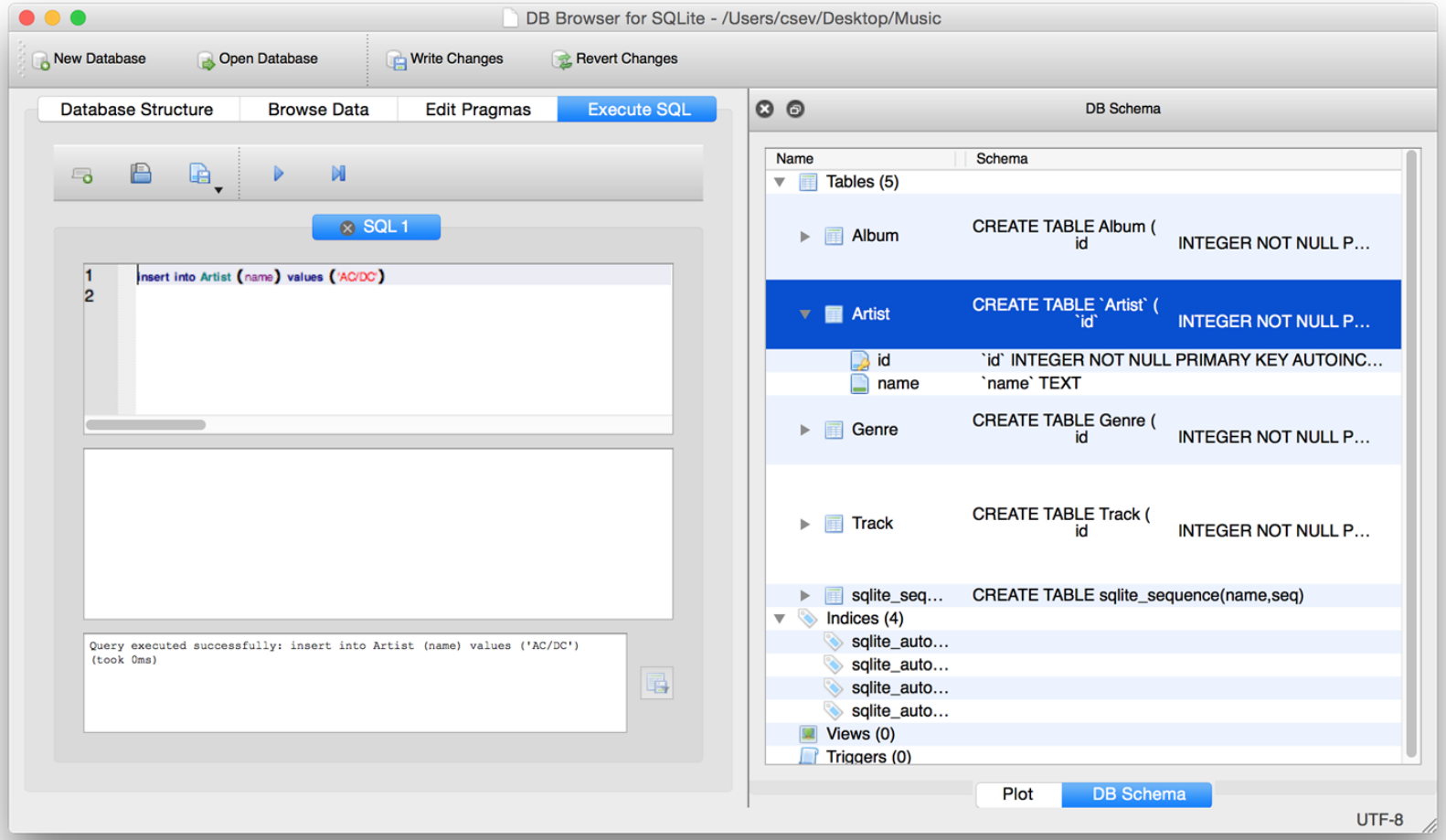




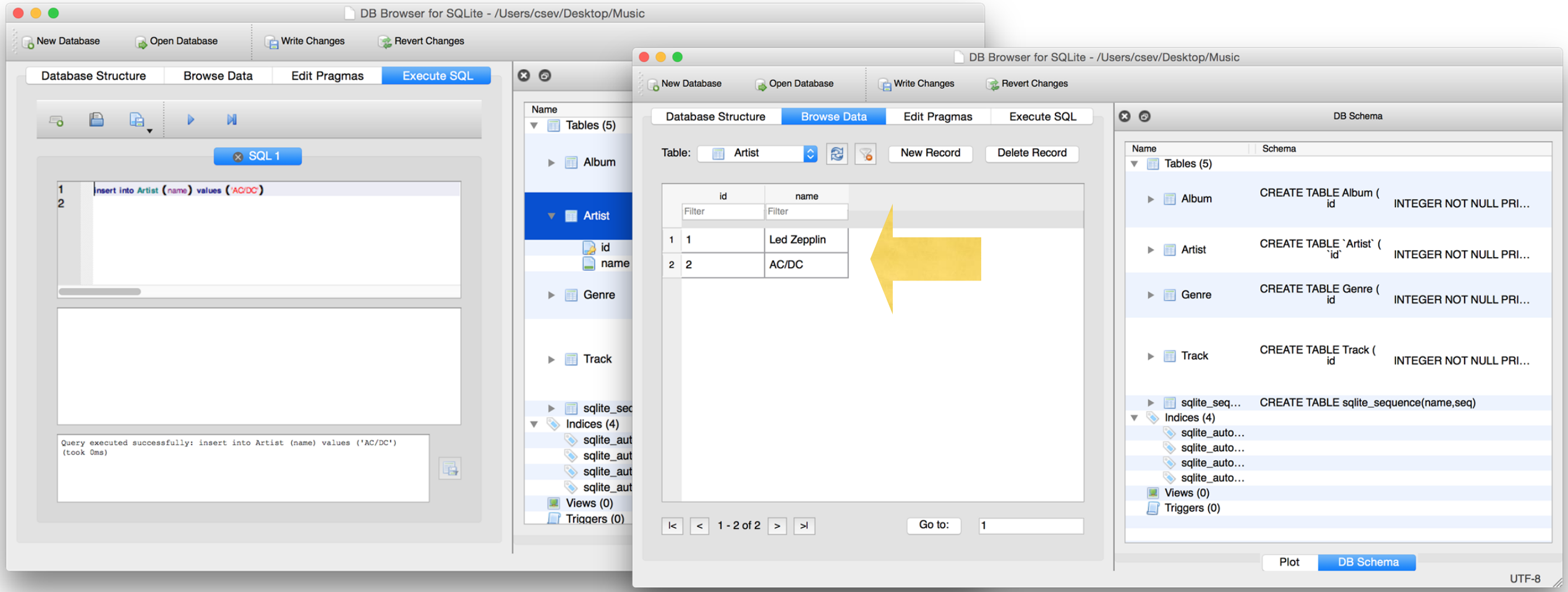

```
CREATE TABLE Genre (
    id      INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
    name    TEXT
)
```

```
CREATE TABLE Album (
        id      INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
        artist_id  INTEGER,
        title   TEXT
)


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

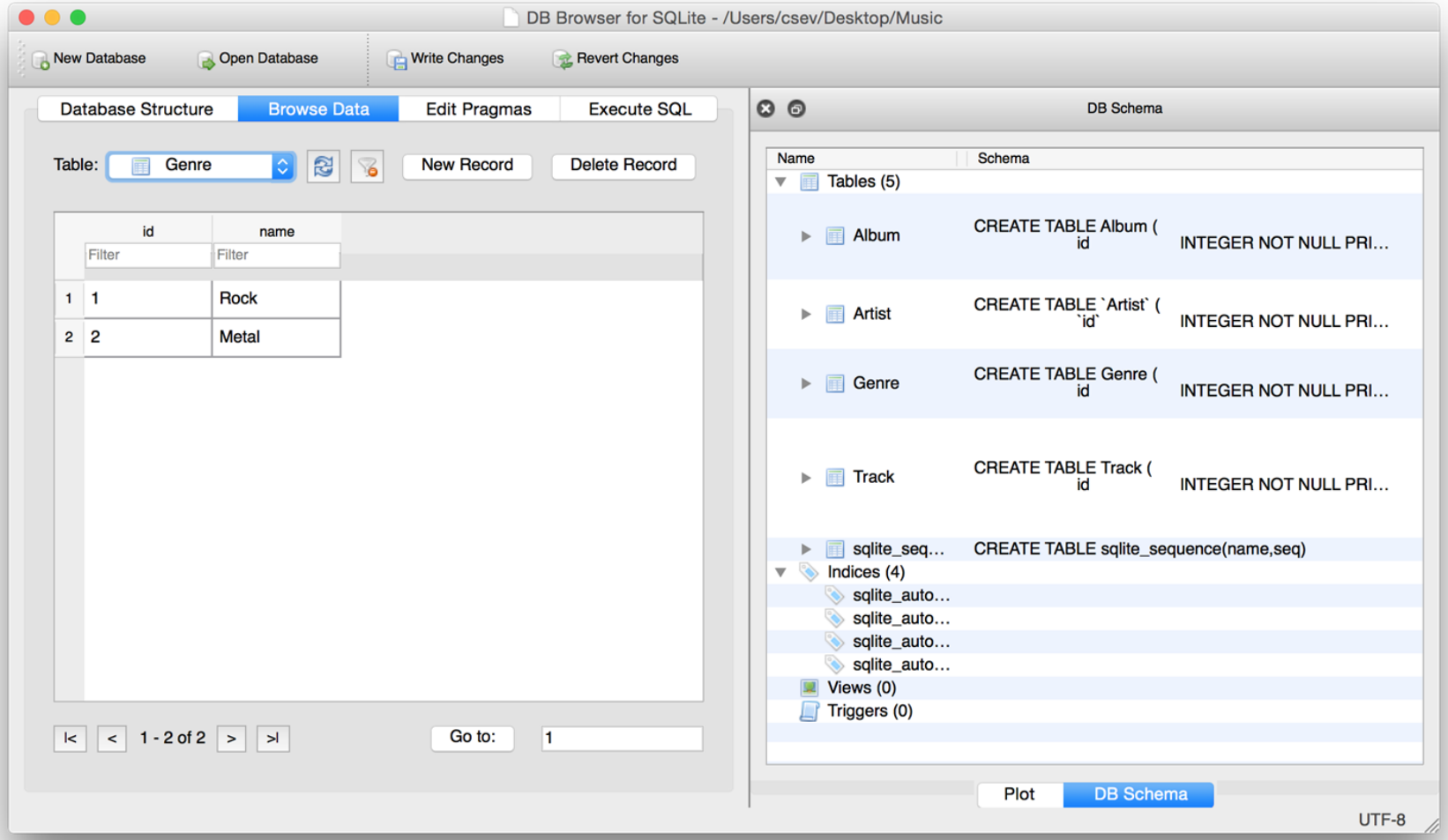




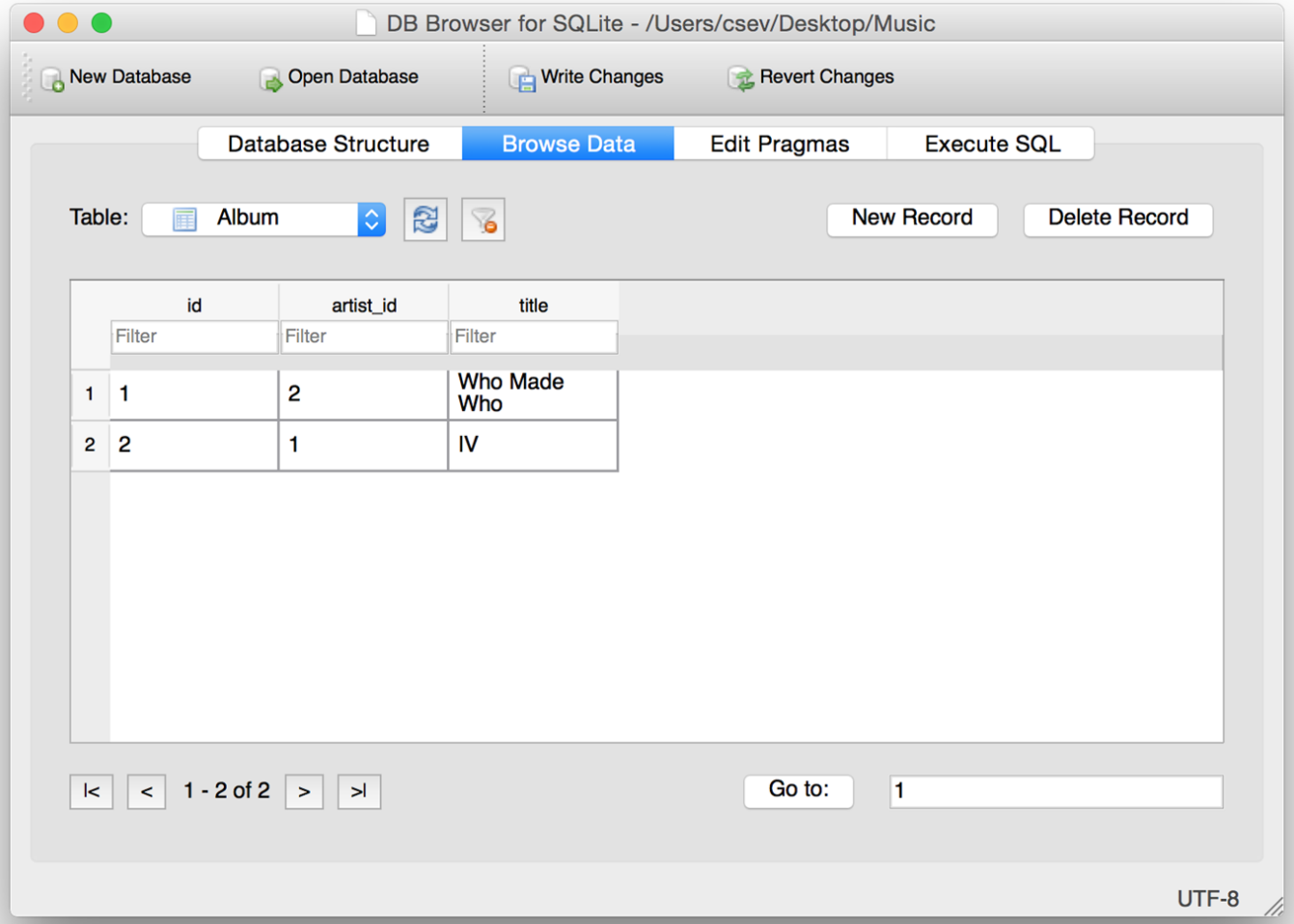
insert into Artist (name) values ('Led Zepplin')
insert into Artist (name) values ('AC/DC')



insert into Artist (name) values ('Led Zepplin')
insert into Artist (name) values ('AC/DC')



insert into Genre (name) values ('Rock')
insert into Genre (name) values ('Metal')



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
	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	Black Dog	2	1	297	5	0
2	2	Stairway	2	1	482	5	0
3	3	About to Rock	1	2	313	5	0
4	4	Who Made Who	1	2	207	5	0

Track

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

Album

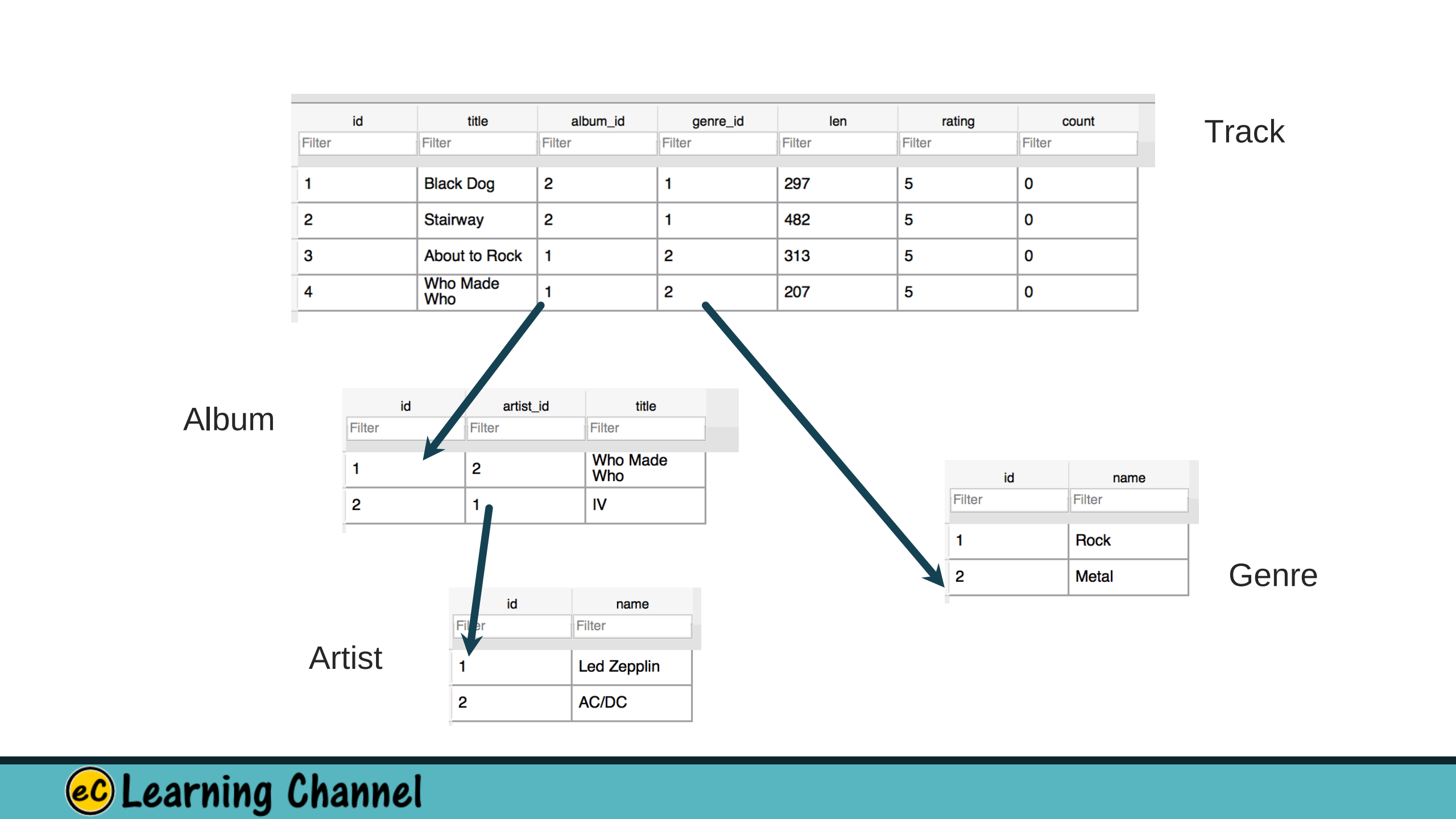
id	artist_id	title
1	2	Who Made Who
2	1	IV

Artist

id	name
1	Led Zepplin
2	AC/DC

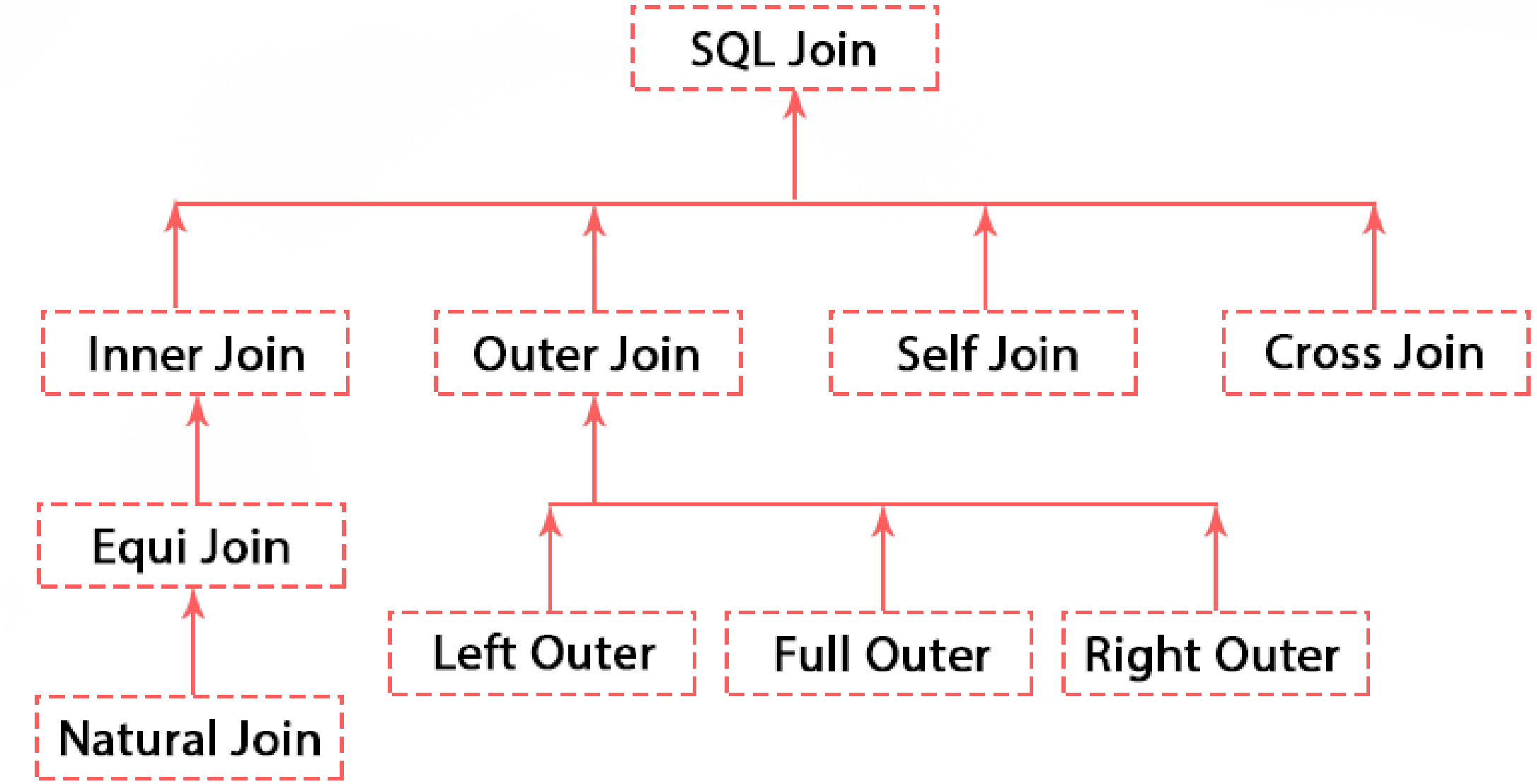
id	name
1	Rock
2	Metal

Genre



Using Join Across Tables

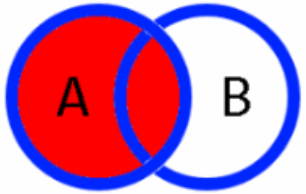
ACTIVITY



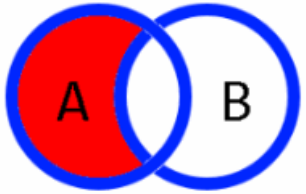
(SQL JOIN TYPES)

SQL JOINS

LEFT OUTER JOIN

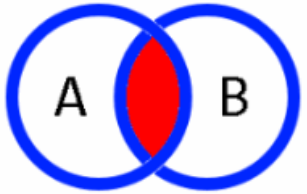



```
SELECT *
FROM TableA a
LEFT JOIN TableB b
    ON a.KEY = b.KEY
```



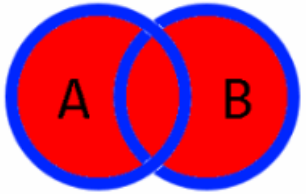

```
SELECT *
FROM TableA a
LEFT JOIN TableB b
    ON a.KEY = b.KEY
WHERE b.KEY IS NULL
```

INNER JOIN

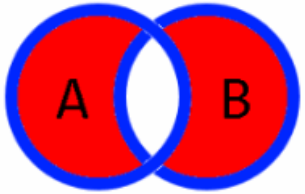



```
SELECT *
FROM TableA a
INNER JOIN TableB b
    ON a.KEY = b.KEY
```

FULL OUTER JOIN

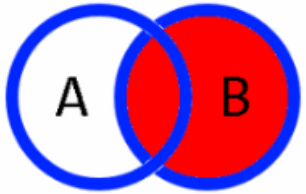



```
SELECT *
FROM TableA a
FULL OUTER JOIN TableB b
    ON a.KEY = b.KEY
```

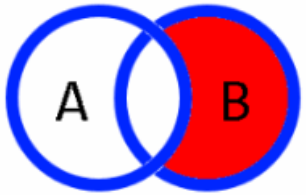



```
SELECT *
FROM TableA a
FULL OUTER JOIN TableB b
    ON a.KEY = b.KEY
WHERE a.KEY IS NULL
    OR b.KEY IS NULL
```

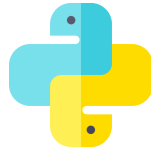
RIGHT OUTER JOIN




```
SELECT *
FROM TableA a
RIGHT JOIN TableB b
    ON a.KEY = b.KEY
```

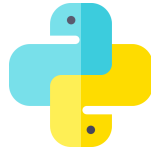



```
SELECT *
FROM TableA a
RIGHT JOIN TableB b
    ON a.KEY = b.KEY
WHERE a.KEY IS NULL
```



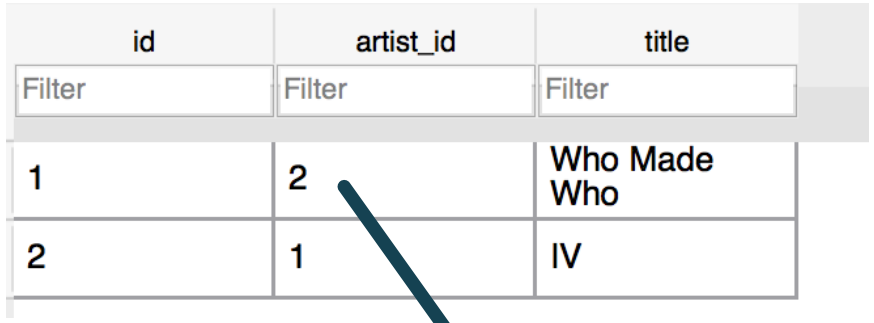
Relational Power

- By removing the replicated data and replacing it with references to a single copy of each bit of data we build a "web" of information that the relational database can read through very quickly - even for very large amounts of data
- Often when you want some data it comes from a number of tables linked by these foreign keys



The JOIN Operation

- The JOIN operation links across several tables as part of a select operation
- You must tell the JOIN how to use the keys that make the connection between the tables using an ON clause



Album

Artist

	title	name
1	Who Made Who	AC/DC
2	IV	Led Zepplin

id	name
Filter	Filter
1	Led Zepplin
2	AC/DC

```
select Album.title, Artist.name from Album join Artist on Album.artist_id = Artist.id
```

What we want to see

The tables that hold the data

How the tables are linked

id	artist_id	title
Filter	Filter	Filter
1	2	Who Made Who
2	1	IV

id	name
Filter	Filter
1	Led Zepplin
2	AC/DC

	title	artist_id	id	name
1	Who Made Who	2	2	AC/DC
2	IV	1	1	Led Zepplin

select Album.title, Album.artist_id, Artist.id,Artist.name
from Album join Artist on Album.artist_id = Artist.id

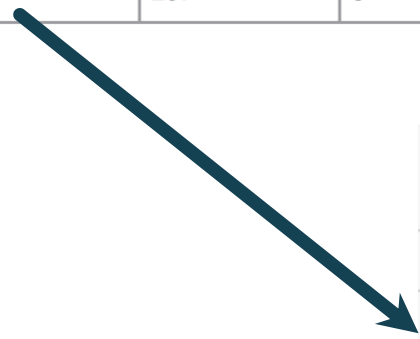
	title	genre_id	id	name
1	Black Dog	1	1	Rock
2	Black Dog	1	2	Metal
3	Stairway	1	1	Rock
4	Stairway	1	2	Metal
5	About to Rock	2	1	Rock
6	About to Rock	2	2	Metal
7	Who Made Who	2	1	Rock
8	Who Made Who	2	2	Metal

```
SELECT Track.title,
    Track.genre_id,
    Genre.id, Genre.name
FROM Track JOIN Genre
```

Joining two tables without an ON clause gives all possible combinations of rows.

	title	name
1	Black Dog	Rock
2	Stairway	Rock
3	About to Rock	Metal
4	Who Made Who	Metal

id	title	album_id	genre_id	len	rating	count
Filter	Filter	Filter	Filter	Filter	Filter	Filter
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



id	name
Filter	Filter
1	Rock
2	Metal

select Track.title, Genre.name from Track join Genre on Track.genre_id = Genre.id

What we want to see

The tables that hold the data

How the tables are linked

```
select Track.title, Artist.name, Album.title, Genre.name from Track join
Genre join Album join Artist on Track.genre_id = Genre.id and
Track.album_id = Album.id and Album.artist_id = Artist.id
```

	title	name	title	name
1	Black Dog	Led Zepplin	IV	Rock
2	Stairway	Led Zepplin	IV	Rock
3	About to Rock	AC/DC	Who Made Who	Metal
4	Who Made Who	AC/DC	Who Made Who	Metal

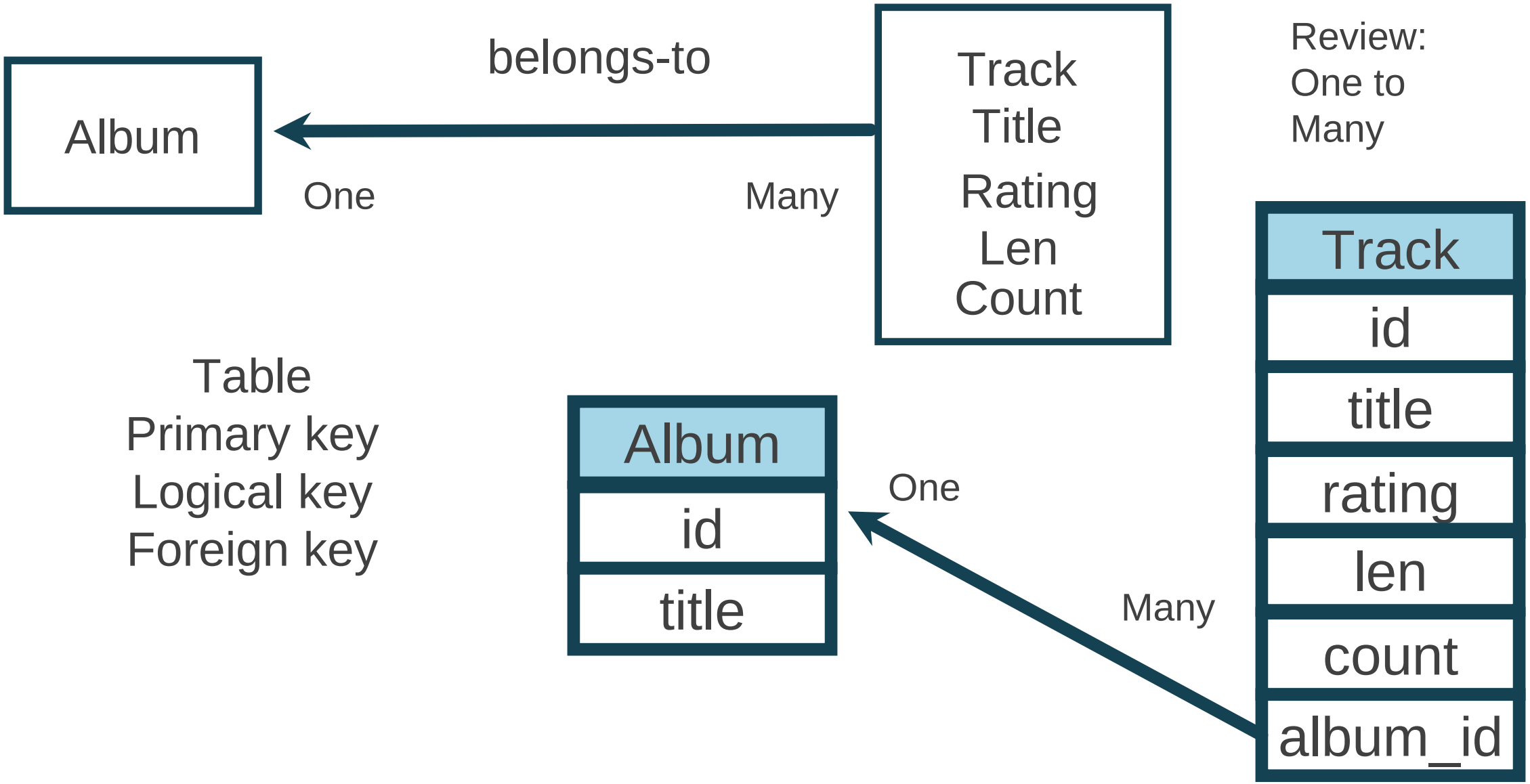
- What we want to see
- The tables which hold the data
- How the tables are linked

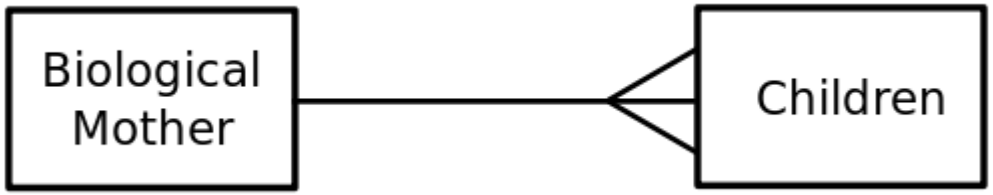
Hells Bells	5:13	AC/DC	Who Made Who	Rock	★★★★★		61
Shake Your Foundations	3:54	AC/DC	Who Made Who	Rock	★★★★★		70
Chase the Ace	3:01	AC/DC	Who Made Who	Rock			56
For Those About To Rock (We ...	5:54	AC/DC	Who Made Who	Rock	★★★★★		61
Dúlamán	3:43	Altan	Natural Wonders M...	New Age			31
Rode Across the Desert	4:10	America	Greatest Hits	Easy Listen...	★★★★★		23
Now You Are Gone	3:08	America	Greatest Hits	Easy Listen...	★★★★★		18
Tin Man	3:30	America	Greatest Hits	Easy Listen...	★★★★★		23
Sister Golden Hair	3:22	America	Greatest Hits	Easy Listen...	★★★★★		24
Track 01	4:22	Billy Price	Danger Zone	Blues/R&B	★★★★★		26
Track 02	2:45	Billy Price	Da				
Track 03	3:26	Billy Price	Da				
Track 04	4:17	Billy Price	Da				
Track 05	3:50	Billy Price	Da				
War Pigs/Luke's Wall	7:58	Black Sabbath	Pa				
Paranoid	2:53	Black Sabbath	Pa				
Planet Caravan	4:35	Black Sabbath	Pa				
Iron Man	5:59	Black Sabbath	Pa				
Electric Funeral	4:53	Black Sabbath	Pa				
Hand of Doom	7:10	Black Sabbath	Pa				
Rat Salad	2:30	Black Sabbath	Pa				
Jack the Stripper/Fairies Wear ...	6:14	Black Sabbath	Pa				
Bomb Squad (TECH)	3:28	Brent	Br				
clay techno	4:36	Brent	Br				
Heavy	3:08	Brent	Br				
Hi metal man	4:20	Brent	Brent's Album				1
Mistro	2:58	Brent	Brent's Album				1

	title	name	title	name
1	Black Dog	Led Zepplin	IV	Rock
2	Stairway	Led Zepplin	IV	Rock
3	About to Rock	AC/DC	Who Made Who	Metal
4	Who Made Who	AC/DC	Who Made Who	Metal

Many-To-Many Relationships

ACTIVITY


Album
belongs-to
Track
Title
Rating
Len
Count
One
Many
Review:
One to
Many
Table
Primary key
Logical key
Foreign key
Album
id
title
One
Many
Track
id
title
rating
len
count
album_id



id	name
Filter	Filter
1	Rock
2	Metal

One

One

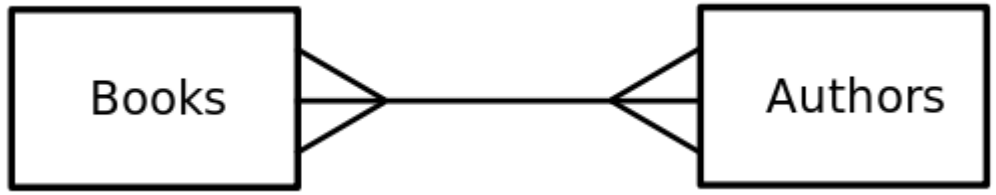
Many

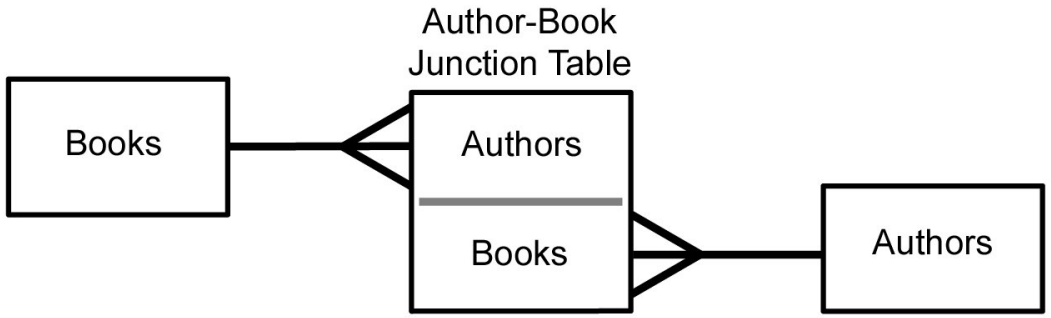
Many

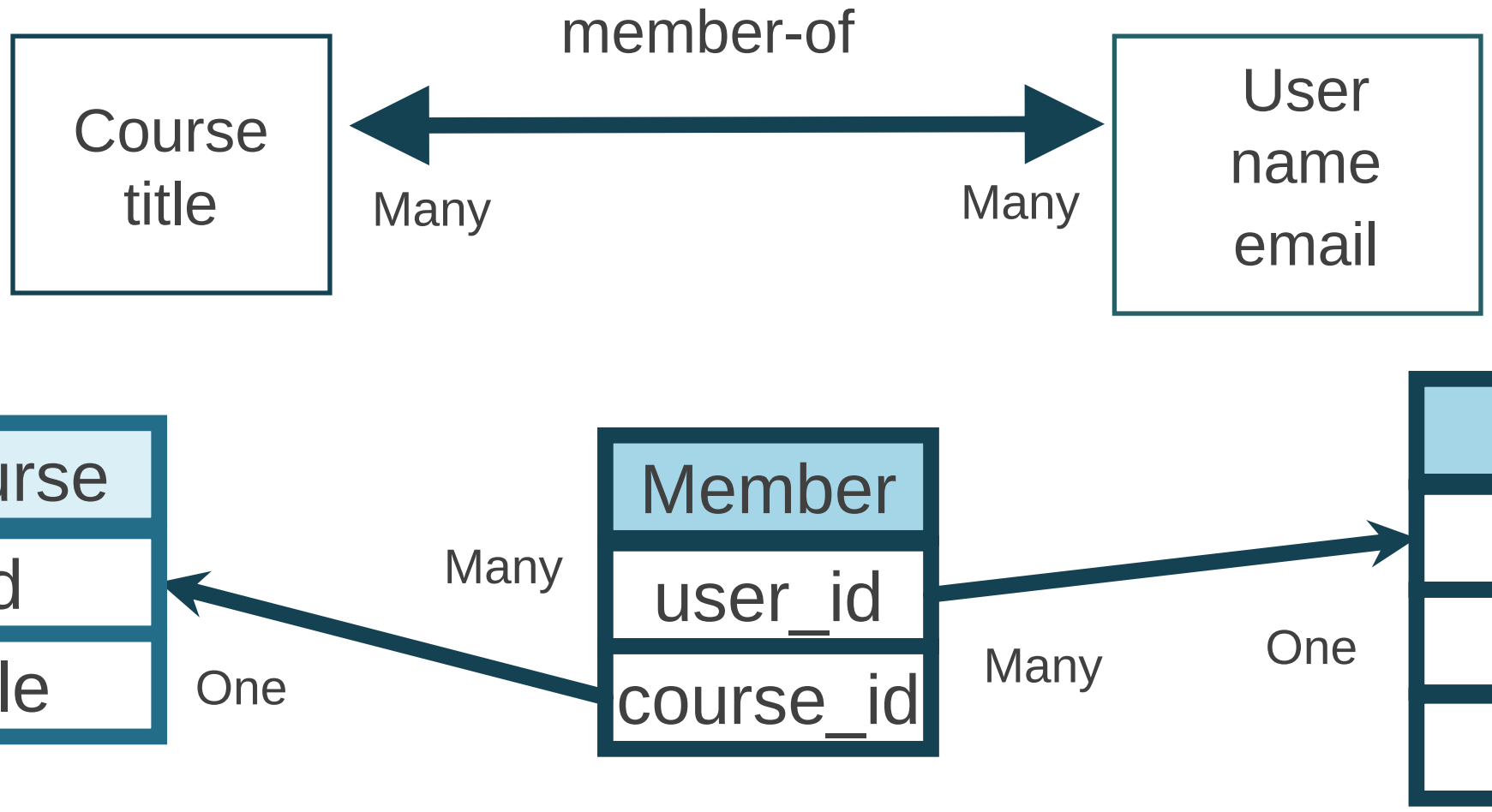
id	title	album_id	genre_id	len	rating	count
Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	Black Dog	2	1	297	5	0
2	Stairway	2	1	482	5	0
3	About to Rock	1	2	313	5	0
4	Who Made Who	1	2	207	5	0

Many to Many

- Sometimes we need to model a relationship that is many-to-many
- We need to add a "connection" table with two foreign keys
- There is usually no separate primary key

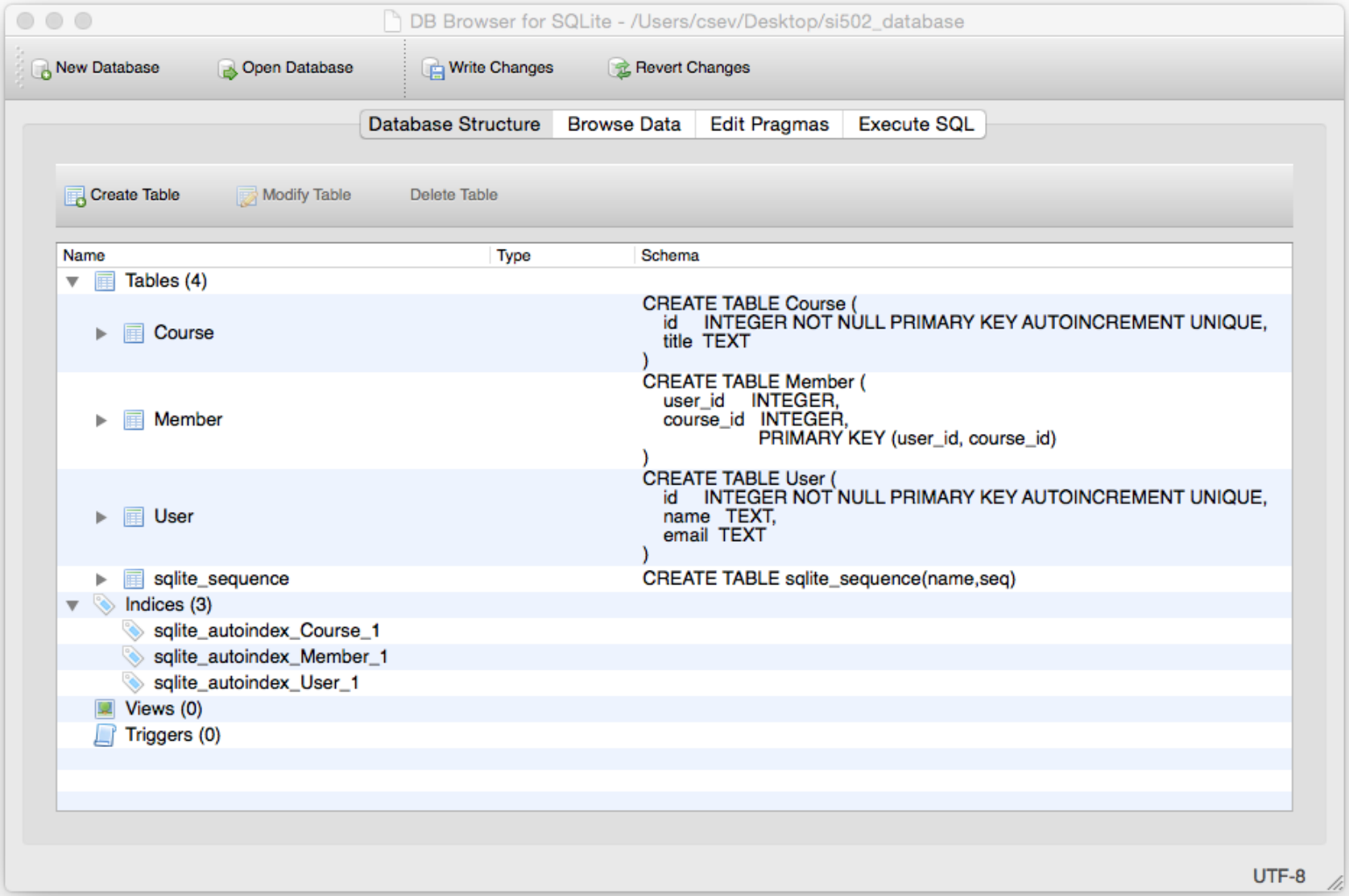


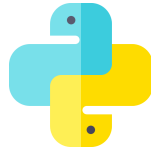



member-of
Course
title
Many
Many
User
name
email
Course
id
title
One
Many
Member
user_id
course_id
Many
One
User
id
name
email

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

CREATE TABLE Course (
    id      INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
    title   TEXT UNIQUE
)

CREATE TABLE Member (
    user_id     INTEGER,
    course_id   INTEGER,
        role        INTEGER,
    PRIMARY KEY (user_id, course_id)
)
```

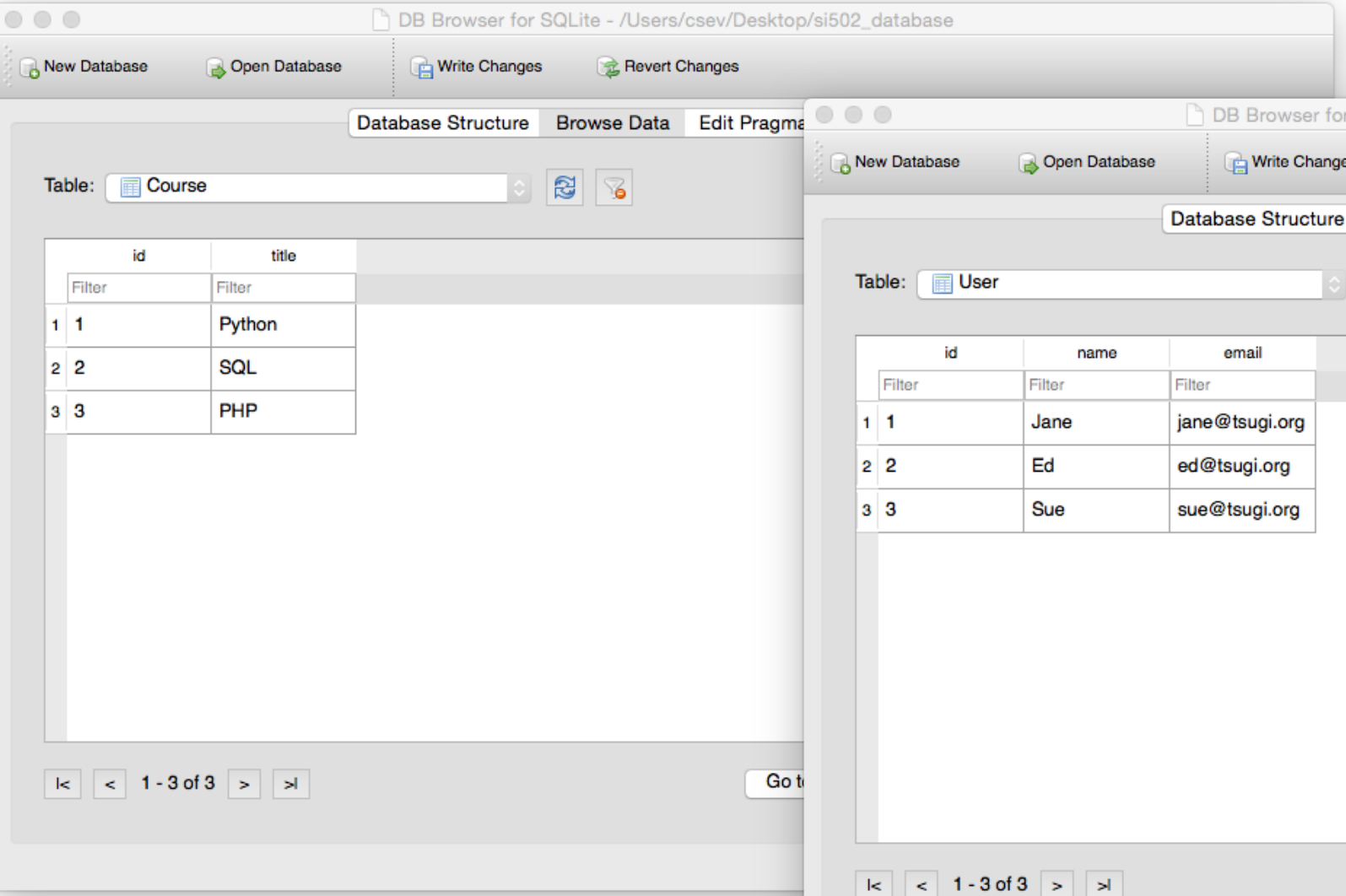
Start with a Fresh Database

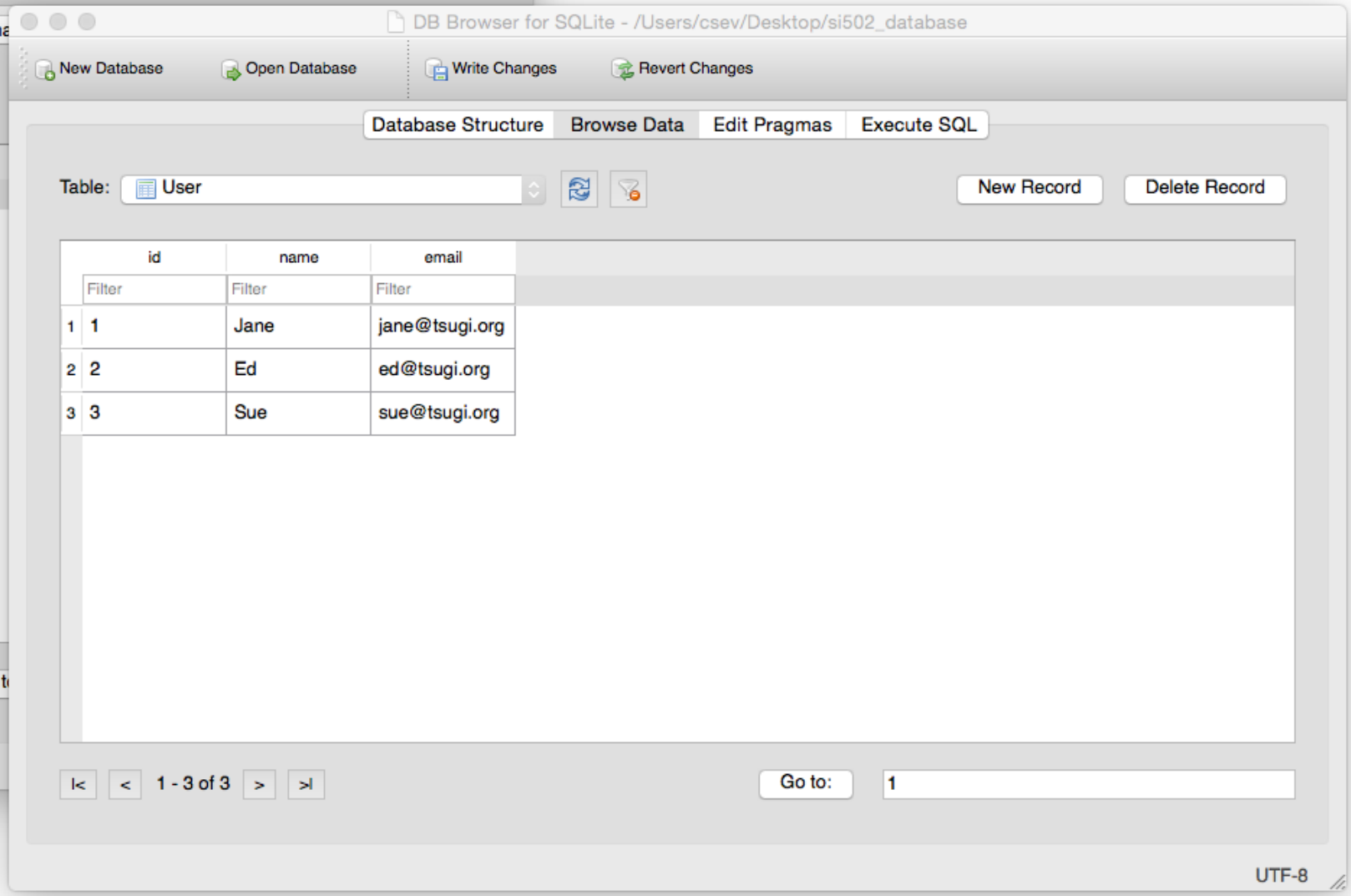

DB Browser for SQLite - /Users/csev/Desktop/si502_database
New Database
Open Database
Write Changes
Revert Changes
Database Structure
Browse Data
Edit Pragmas
Execute SQL
Create Table
Modify Table
Delete Table
Name
Type
Schema
Tables (4)
Course
CREATE TABLE Course (
id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
title TEXT
)
Member
CREATE TABLE Member (
user_id INTEGER,
course_id INTEGER,
PRIMARY KEY (user_id, course_id)
)
User
CREATE TABLE User (
id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
name TEXT,
email TEXT
)
sqlite_sequence
CREATE TABLE sqlite_sequence(name,seq)
Indices (3)
sqlite_autoindex_Course_1
sqlite_autoindex_Member_1
sqlite_autoindex_User_1
Views (0)
Triggers (0)
UTF-8



Insert Users and Courses

```
INSERT INTO User (name, email) VALUES ('Jane', 'jane@tsugi.org');
INSERT INTO User (name, email) VALUES ('Ed', 'ed@tsugi.org');
INSERT INTO User (name, email) VALUES ('Sue', 'sue@tsugi.org');

INSERT INTO Course (title) VALUES ('Python');
INSERT INTO Course (title) VALUES ('SQL');
INSERT INTO Course (title) VALUES ('PHP');
```

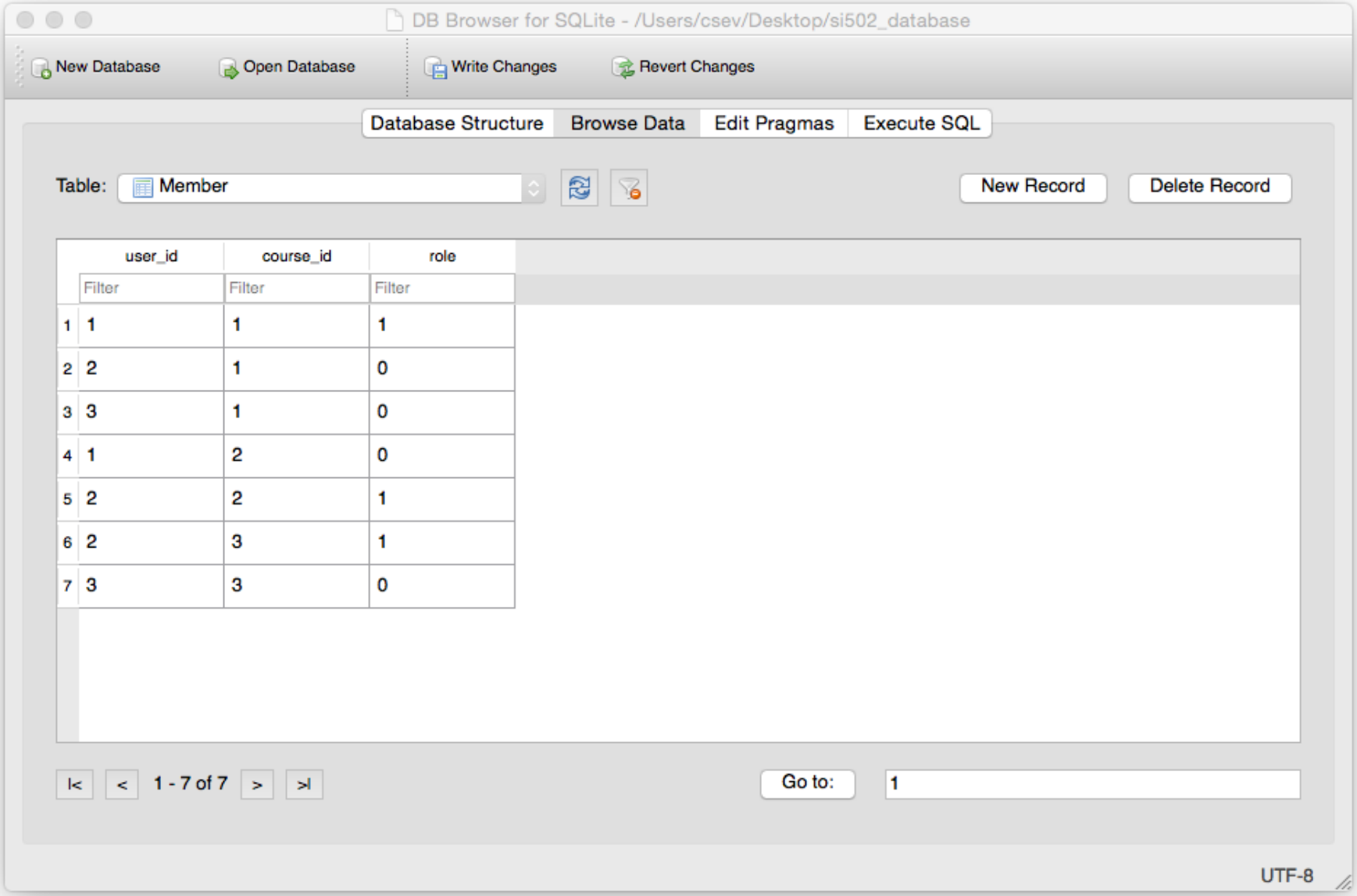

DB Browser for SQLite - /Users/csev/Desktop/si502_database
New Database
Open Database
Write Changes
Revert Changes
Database Structure
Browse Data
Edit Pragma
Table:
Course
id
title
Filter
Filter
1 1 Python
2 2 SQL
3 3 PHP
I<
<
1 - 3 of 3
>
>I


DB Browser for SQLite - /Users/csev/Desktop/si502_database
New Database
Open Database
Write Changes
Revert Changes
Database Structure
Browse Data
Edit Pragmas
Execute SQL
Table:
User
New Record
Delete Record
id
name
email
Filter
Filter
Filter
1 1 Jane jane@tsugi.org
2 2 Ed ed@tsugi.org
3 3 Sue sue@tsugi.org
I<
<
1 - 3 of 3
>
>I
Go to:
1
UTF-8

id	name	email
Filter	Filter	Filter
1	Jane	jane@tsugi.org
2	Ed	ed@tsugi.org
3	Sue	sue@tsugi.org

id	title
Filter	Filter
1	Python
2	SQL
3	PHP

```
INSERT INTO Member (user_id, course_id, role) VALUES (1, 1, 1);
INSERT INTO Member (user_id, course_id, role) VALUES (2, 1, 0);
INSERT INTO Member (user_id, course_id, role) VALUES (3, 1, 0);

INSERT INTO Member (user_id, course_id, role) VALUES (1, 2, 0);
INSERT INTO Member (user_id, course_id, role) VALUES (2, 2, 1);

INSERT INTO Member (user_id, course_id, role) VALUES (2, 3, 1);
INSERT INTO Member (user_id, course_id, role) VALUES (3, 3, 0);
```


DB Browser for SQLite - /Users/csev/Desktop/si502_database
New Database
Open Database
Write Changes
Revert Changes
Database Structure
Browse Data
Edit Pragmas
Execute SQL
Table:
Member
New Record
Delete Record
user_id
course_id
role
Filter
Filter
Filter
1 1 1 1
2 2 1 0
3 3 1 0
4 1 2 0
5 2 2 1
6 2 3 1
7 3 3 0
|<
<
1 - 7 of 7
>
>|
Go to:
1
UTF-8

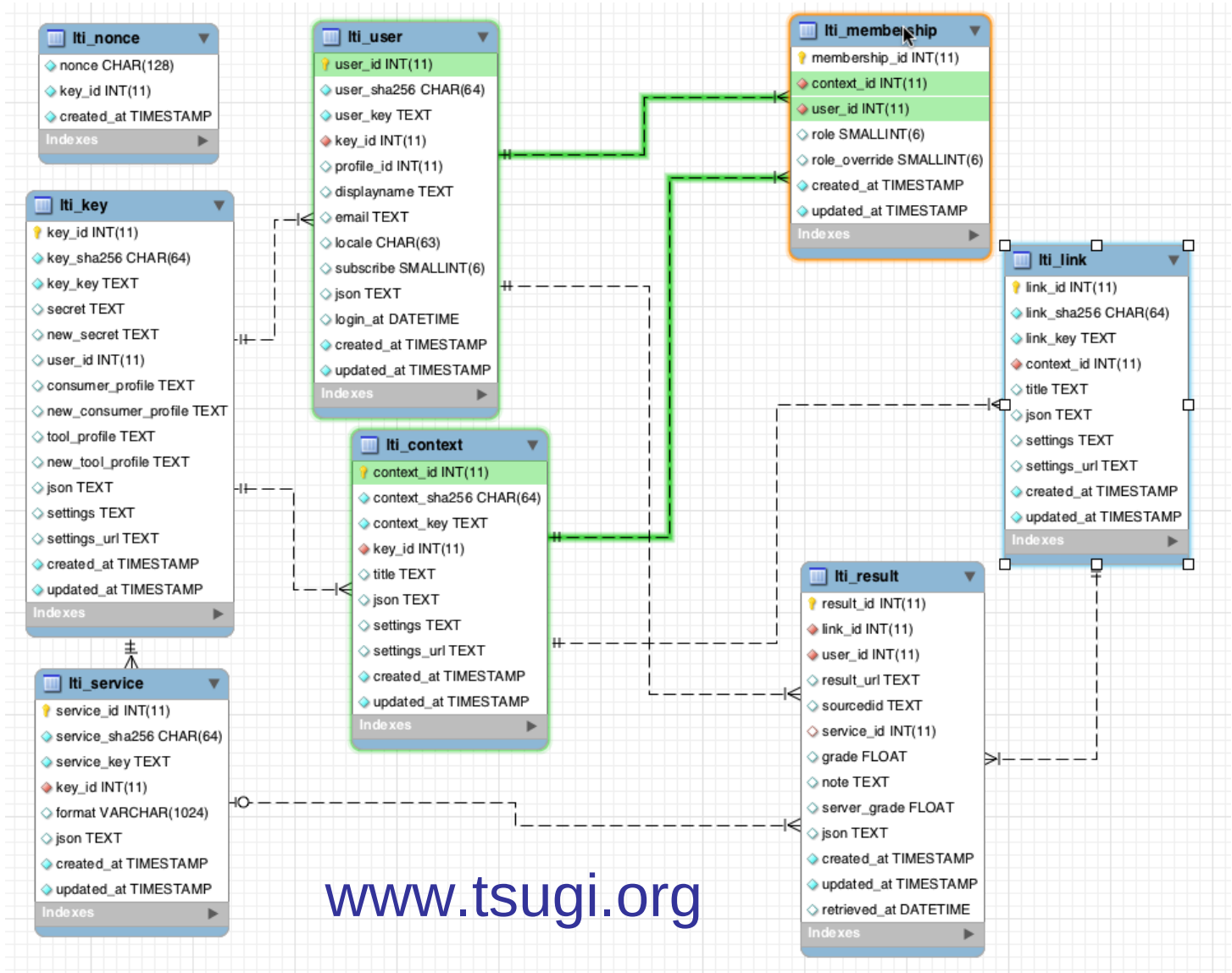
id	name	email
Filter	Filter	Filter
1	Jane	jane@tsugi.org
2	Ed	ed@tsugi.org
3	Sue	sue@tsugi.org

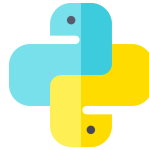
user_id	course_id	role
Filter	Filter	Filter
1	1	1
2	1	0
3	1	0
1	2	0
2	2	1
2	3	1
3	3	0

id	title
Filter	Filter
1	Python
2	SQL
3	PHP

```
SELECT User.name, Member.role, Course.title
FROM User JOIN Member JOIN Course
ON Member.user_id = User.id AND
Member.course_id = Course.id
ORDER BY Course.title, Member.role DESC, User.name
```

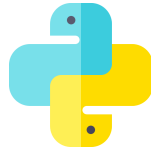
	name	role	title
2	Sue	0	PHP
3	Jane	1	Python
4	Ed	0	Python
5	Sue	0	Python
6	Ed	1	SQL


lti_nonce
nonce CHAR(128)
key_id INT(11)
created_at TIMESTAMP
Indexes
lti_user
user_id INT(11)
user_sha256 CHAR(64)
user_key TEXT
key_id INT(11)
profile_id INT(11)
displayname TEXT
email TEXT
locale CHAR(63)
subscribe SMALLINT(6)
json TEXT
login_at DATETIME
created_at TIMESTAMP
updated_at TIMESTAMP
Indexes
lti_membership
membership_id INT(11)
context_id INT(11)
user_id INT(11)
role SMALLINT(6)
role_override SMALLINT(6)
created_at TIMESTAMP
updated_at TIMESTAMP
Indexes
lti_key
key_id INT(11)
key_sha256 CHAR(64)
key_key TEXT
secret TEXT
new_secret TEXT
user_id INT(11)
consumer_profile TEXT
new_consumer_profile TEXT
tool_profile TEXT
new_tool_profile TEXT
json TEXT
settings TEXT
settings_url TEXT
created_at TIMESTAMP
updated_at TIMESTAMP
Indexes
lti_context
context_id INT(11)
context_sha256 CHAR(64)
context_key TEXT
key_id INT(11)
title TEXT
json TEXT
settings TEXT
settings_url TEXT
created_at TIMESTAMP
updated_at TIMESTAMP
Indexes
lti_link
link_id INT(11)
link_sha256 CHAR(64)
link_key TEXT
context_id INT(11)
title TEXT
json TEXT
settings TEXT
settings_url TEXT
created_at TIMESTAMP
updated_at TIMESTAMP
Indexes
lti_result
result_id INT(11)
link_id INT(11)
user_id INT(11)
result_url TEXT
sourcedid TEXT
service_id INT(11)
grade FLOAT
note TEXT
server_grade FLOAT
json TEXT
created_at TIMESTAMP
updated_at TIMESTAMP
retrieved_at DATETIME
Indexes
lti_service
service_id INT(11)
service_sha256 CHAR(64)
service_key TEXT
key_id INT(11)
format VARCHAR(1024)
json TEXT
created_at TIMESTAMP
updated_at TIMESTAMP
Indexes
www.tsugi.org



Complexity Enables Speed

- Complexity makes speed possible and allows you to get very fast results as the data size grows
- By normalizing the data and linking it with integer keys, the overall amount of data which the relational database must scan is far lower than if the data were simply flattened out
- It might seem like a tradeoff - spend some time designing your database so it continues to be fast when your application is a success



Additional SQL Topics

- Indexes improve access performance for things like string fields
- Constraints on data - (cannot be NULL, etc..)
- Transactions - allow SQL operations to be grouped and done as a unit