https://ukrainer.net/en/

https://www.instagram.com/ukrainer_net/

DSCI 510
Principles of Programming for Data Science

Jose-Luis Ambite

Research Team Leader, Information Sciences Institute
Associate Research Professor, Department of Computer Science

University of Southern California

Databases and SQL
Chapter 15 in textbook

"DB" Classes

- DSCI-551 Foundations of Data Management
- DSCI-550 Data Science at Scale
- DSCI-558/CSCI-563 Building Knowledge Graphs
- CSCI-585 Database Systems
- CSCI-548 Information Integration on the Web

Why Databases?

When you have lots of data (i. e., too much to store in main memory), you need to:

1) organize your data

2) be able to find what you need quickly

Database

- Logically coherent collection of data with inherent meaning
- Represents some aspect of the real world
- Miniworld or universe of discourse
  - Logic: FOL Model
- Built for a specific purpose
- Maintains metadata (data about data): describes data characteristics and relationships

[based on Elmasri & Navathe, 2011]

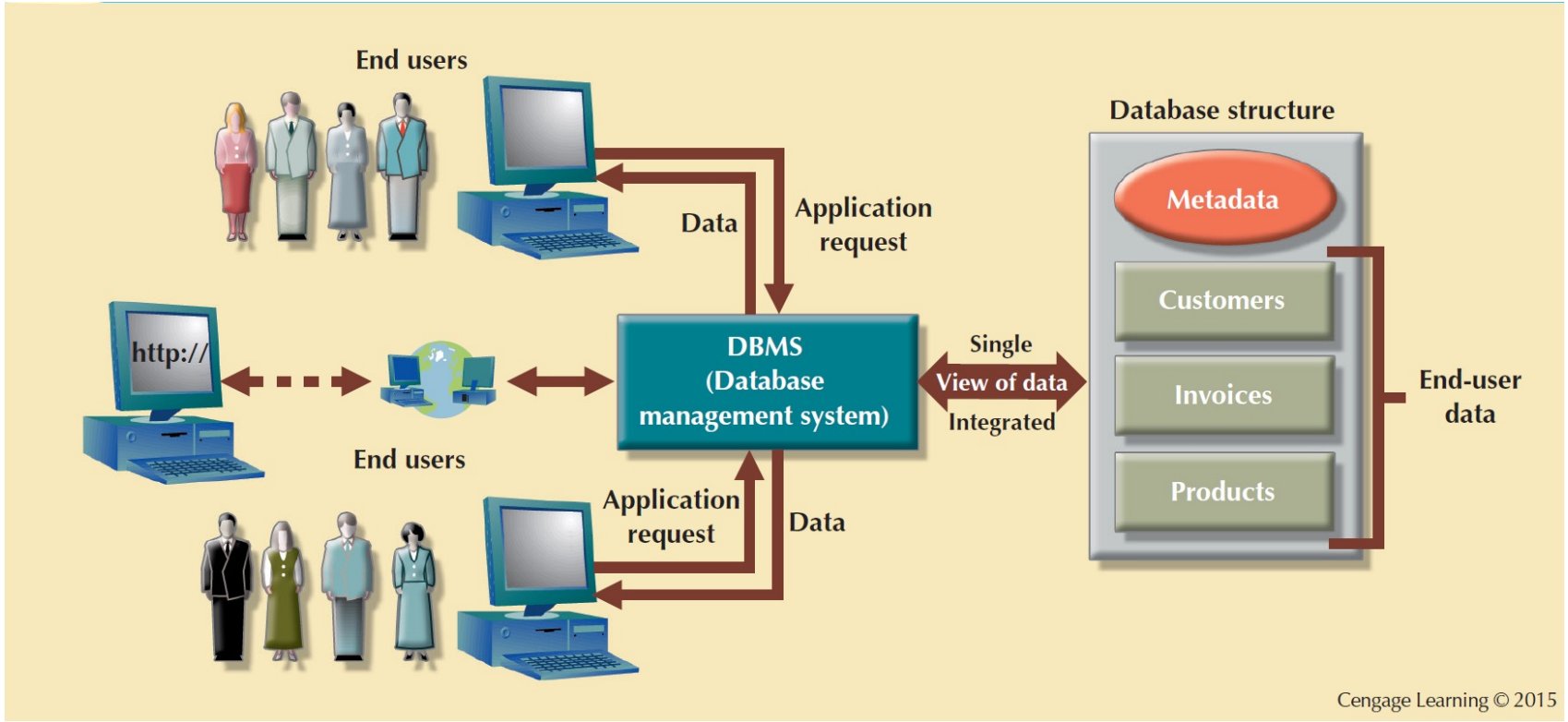
Database Management System (DBMS)

Collection of programs that:

- Manage the database structure
- Evaluate queries efficiently against the data
- Control access to data stored in the database
- Manage concurrent access/modifications by multiple users

[based on Elmasri & Navathe, 2011]

DBMS Manages Interaction between End User and Database



DBMS Functions

Data dictionary management

- **Data dictionary**: Stores definitions of the data elements and their relationships

Data storage management

- **Performance tuning**: Ensures efficient performance of the database in terms of storage and access speed

Data transformation and presentation

- Transforms entered data to conform to required data structures

Security management

- Enforces user security and data privacy

DBMS Functions

Multiuser access control

- Sophisticated algorithms ensure that multiple users can access the database concurrently without compromising its integrity

Backup and recovery management

- Enables recovery of the database after a failure

Data integrity management

- Minimizes redundancy and maximizes consistency

DBMS Functions

Database access languages and application programming interfaces

- **Query language**: Lets the user specify what must be done without having to specify how
- **Structured Query Language (SQL)**: De facto query language and data access standard supported by the majority of DBMS vendors

Database communication interfaces

- Accept end-user requests via multiple, different network environments

Relational Databases

Relational databases model data by storing rows and columns in tables. The power of the relational database lies in its ability to efficiently retrieve data from those tables and in particular where there are multiple tables and the relationships between those tables involved in the query.

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

Terminology

- Database - contains many tables
- Relation (or table) - contains tuples and attributes
- Tuple (or row) - a set of fields that generally represents an “object” (entity) like a person or a music track
- Attribute (also column or field) - one of possibly many elements of data corresponding to the object represented by the row

Relation, Database (Discrete Math)

Given k sets (domains) D_1, D_2, … D_k,

the cartesian product of these sets $D_1 \times D_2 \times \ldots \times D_k$

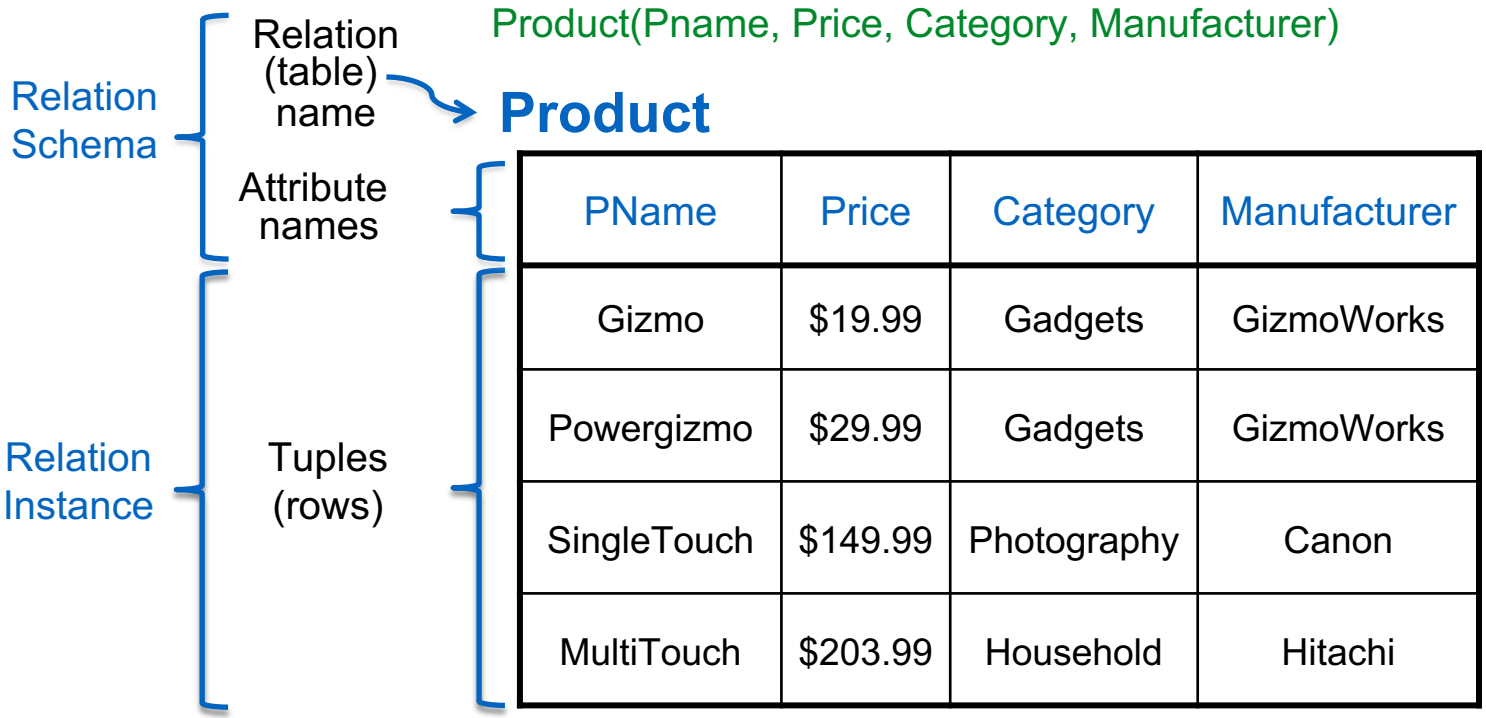
is the set of all k-tuples $(d_1, d_2, \ldots, d_k)$ such that $d_i \in D_i$ for $1 \leq i \leq k$

A k-ary relation R is a subset of the cartesian product of k sets, i.e.,

$R \subseteq D_1 \times D_2 \times \ldots \times D_k$

A database is a set of relations

Relational schema, instance



[based on slide by Doan, Halevy, Ives]

STUDENT

Name	Student_number	Class	Major
Smith	17	1	CS
Brown	8	2	CS

COURSE

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	07	King
92	CS1310	Fall	07	Anderson
102	CS3320	Spring	08	Knuth
112	MATH2410	Fall	08	Chang
119	CS1310	Fall	08	Anderson
135	CS3380	Fall	08	Stone

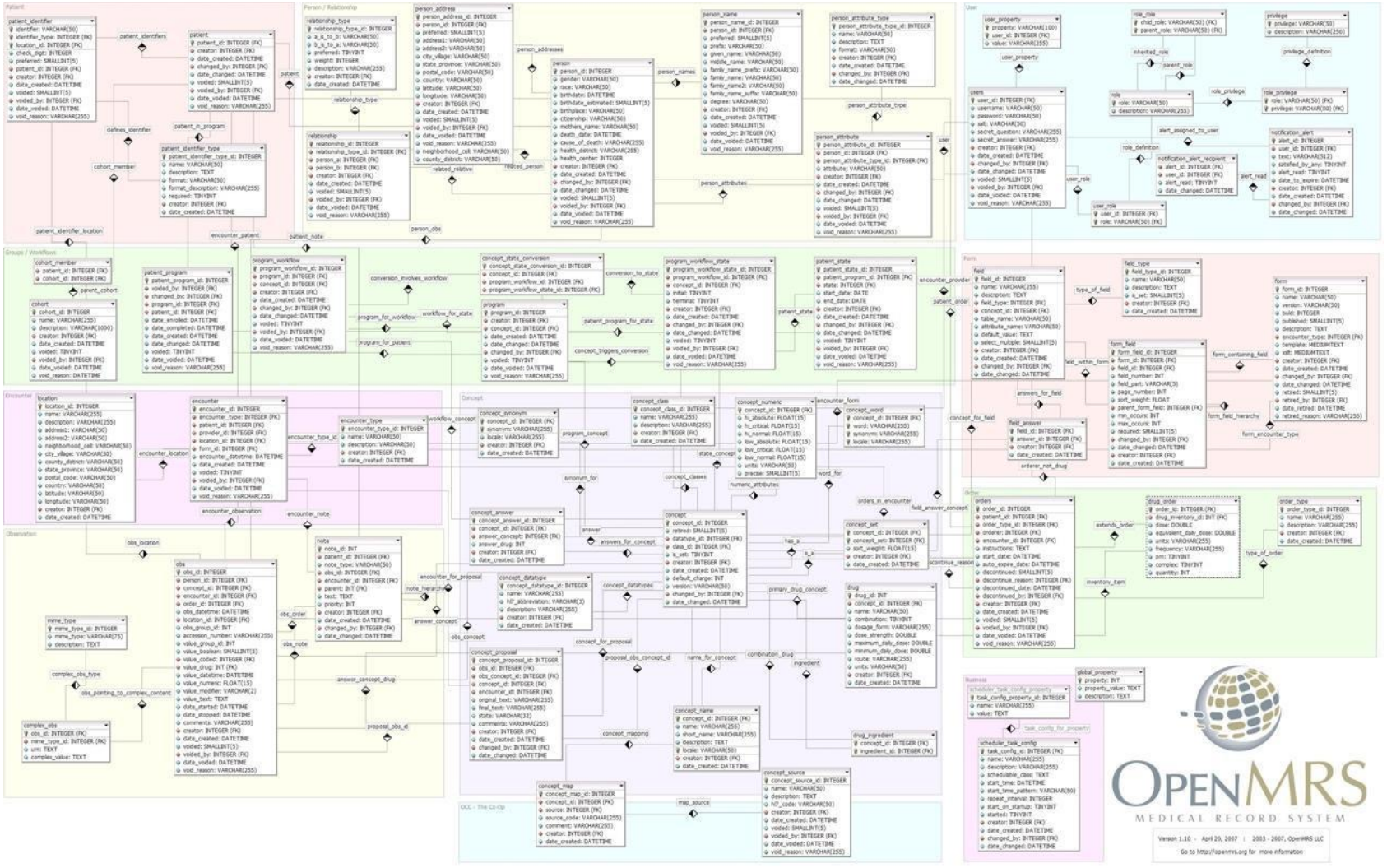
GRADE_REPORT

Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

Figure 1.2
A database that stores student and course information.

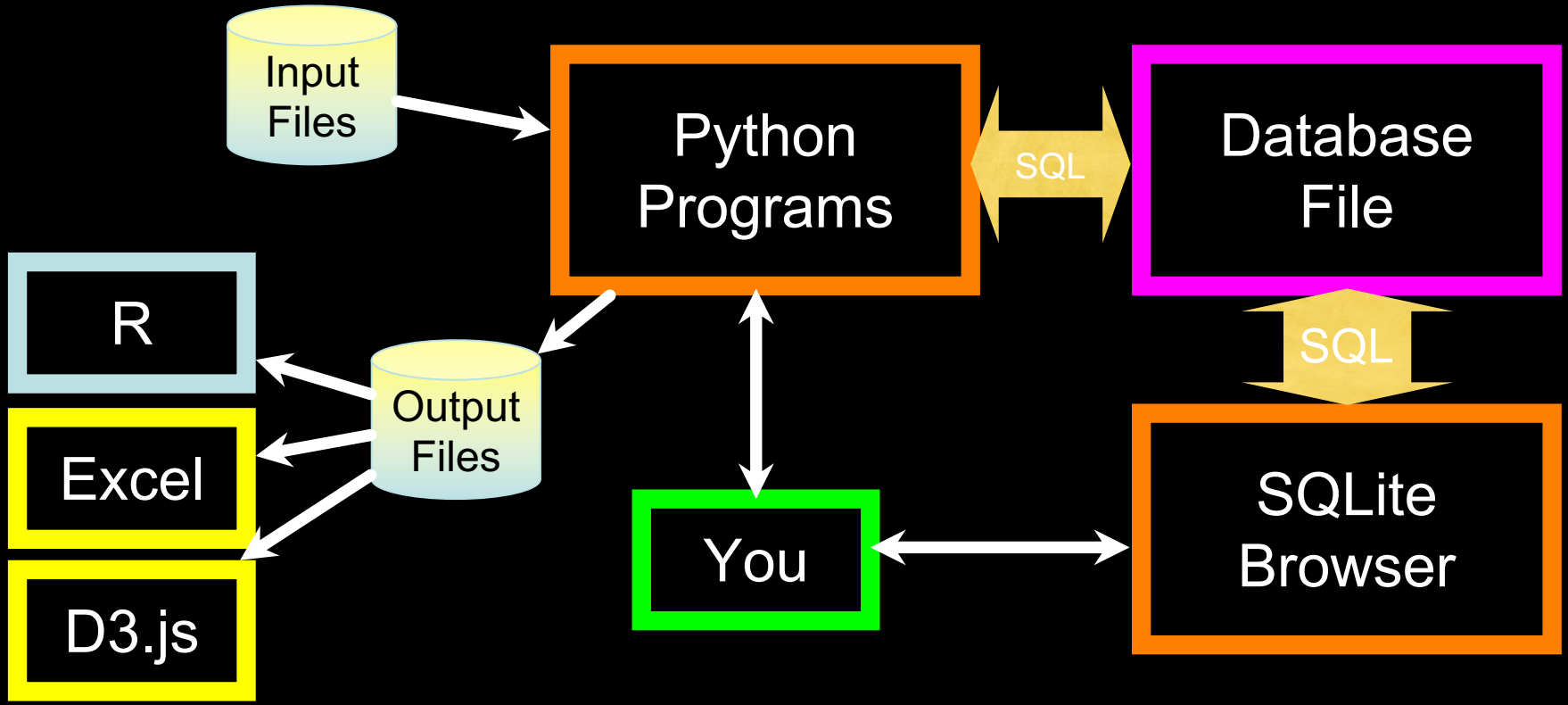

Patient
Person / Relationship
User
Groups / Workflows
Form
Encounter
Concept
Order
Observation
Business
OCC - The Co-Op
OpenMRS
MEDICAL RECORD SYSTEM
Version 1.10 - April 20, 2007 | 2003 - 2007, OpenMRS LLC
Go to http://openmrs.org for more information

SQL

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

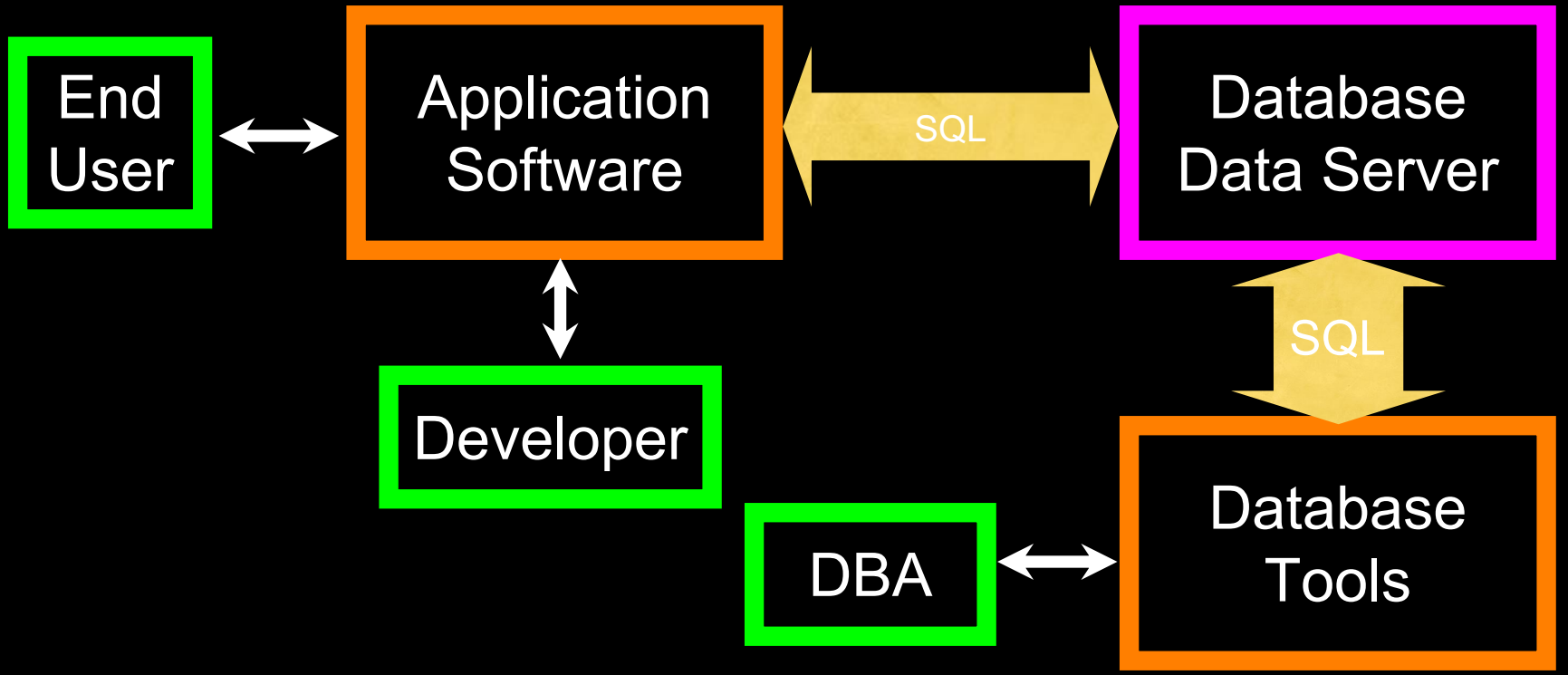
- Create data (a.k.a. Insert)
- Retrieve data
- Update data
- Delete data

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


Input Files
Python Programs
SQL
Database File
R
Excel
D3.js
Output Files
You
SQL
SQLite Browser

Major Web Applications w/ Databases

- Application Developer - Builds the logic for the application, the look and feel of the application - monitors the application for problems
- Database Administrator - Monitors and adjusts the database as the program runs in production
- Often both people participate in the building of the “Data model”



DBA = Database Administrator

Database Administrator

A database administrator (DBA) is a person responsible for the design, implementation, maintenance, and repair of an organization's database. The role includes the development and design of database strategies, monitoring and improving database performance and capacity, and planning for future expansion requirements. They may also plan, coordinate, and implement security measures to safeguard the database.

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

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, PostgreSQL (aka Postgres), ...

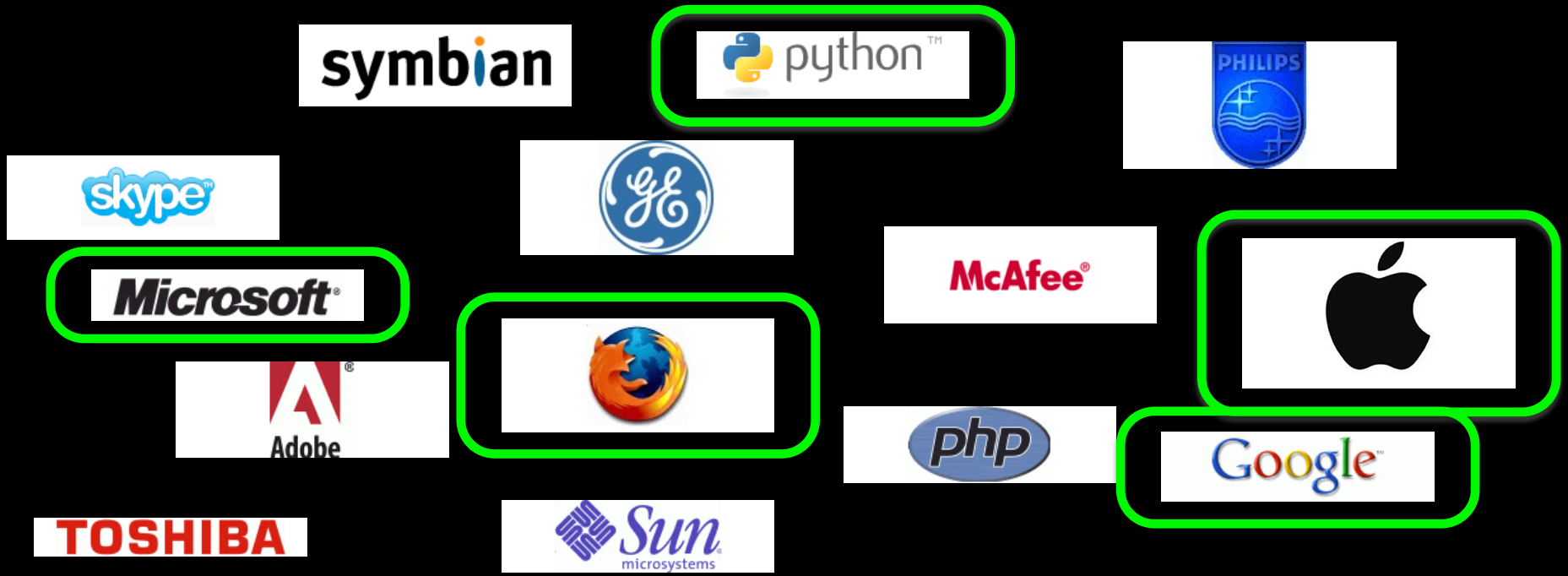
We will be using this

But for anything more serious you should probably use PostgreSQL or MySQL

SQLite

- SQLite is a very popular database - it is free and fast and small
- SQLite Browser allows us to directly manipulate SQLite files
  - http://sqlitebrowser.org/
- SQLite is embedded in Python and a number of other languages

SQLite Is in Lots of Software...



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

sqlitebrowser.org

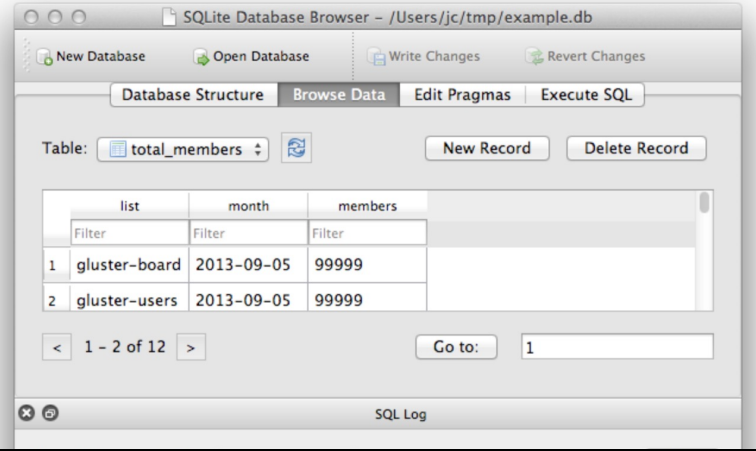
DB Browser for SQLite

About **Download** Blog Docs GitHub Gitter Stats Patreon DBHub.io

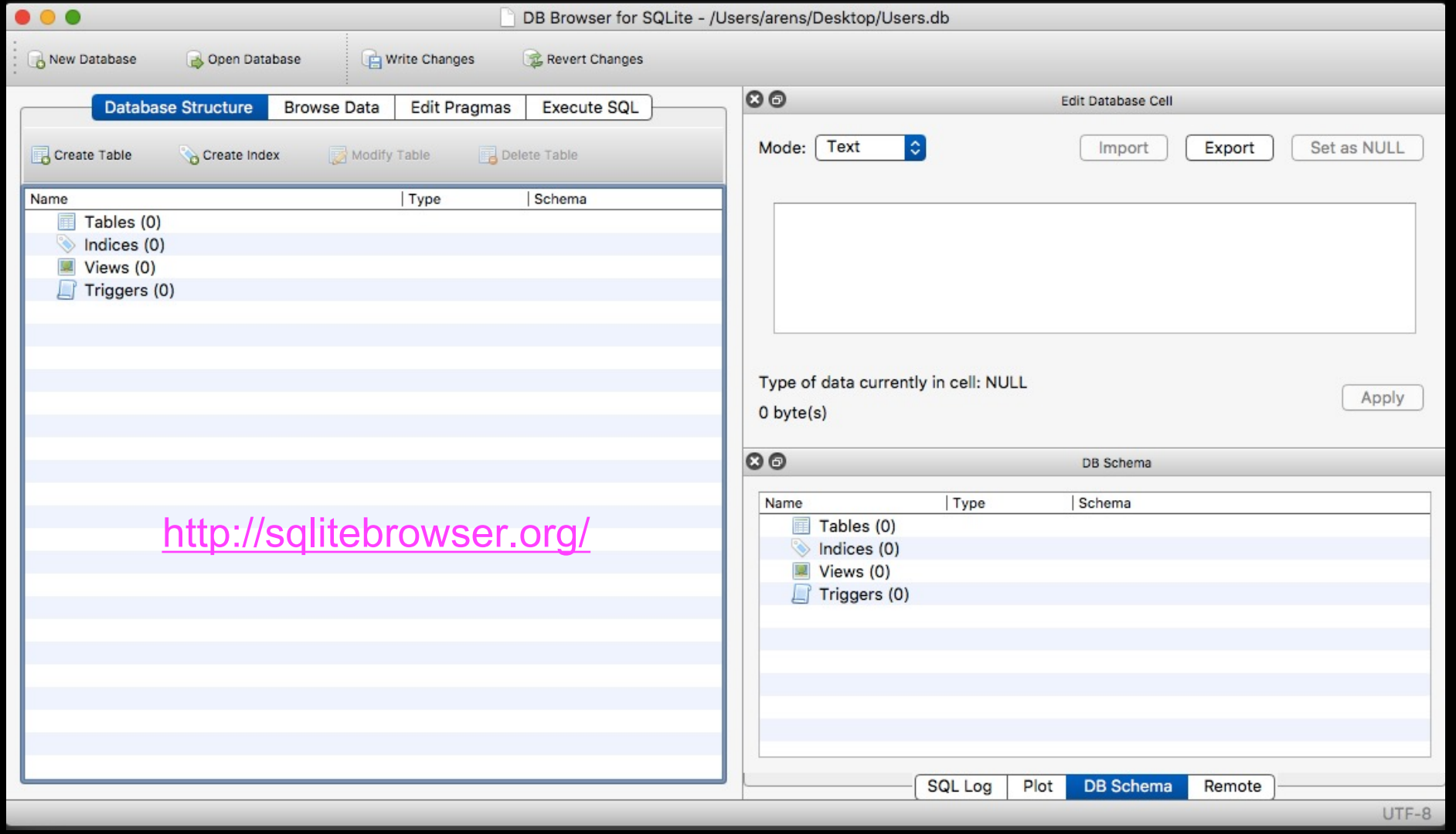
DB Browser for SQLite

The Official home of the DB Browser for SQLite

Screenshot



http://sqlitebrowser.org/


DB Browser for SQLite - /Users/arens/Desktop/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 (0)
Indices (0)
Views (0)
Triggers (0)
http://sqlitebrowser.org/
Edit Database Cell
Mode:
Text
Import
Export
Set as NULL
Type of data currently in cell: NULL
0 byte(s)
Apply
DB Schema
Name
Type
Schema
Tables (0)
Indices (0)
Views (0)
Triggers (0)
SQL Log
Plot
DB Schema
Remote
UTF-8

Lets Make a Database

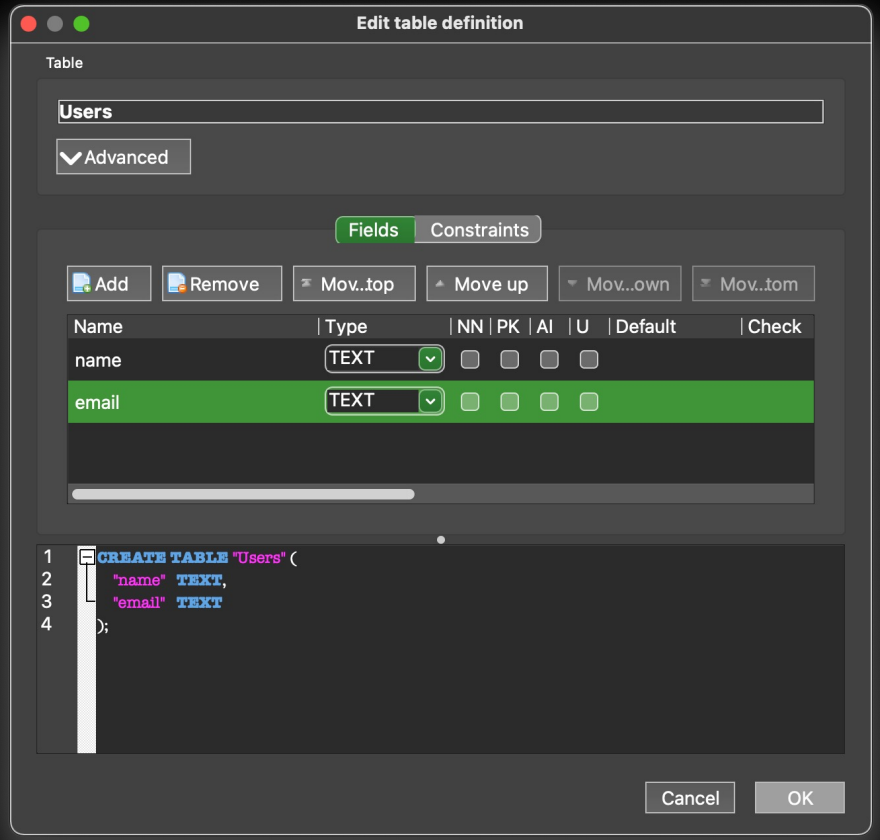
https://www.py4e.com/lectures3/Pythonlearn-15-Database-Handout.txt

Go to week12-Database-Handout.txt
Show in SQLite browser

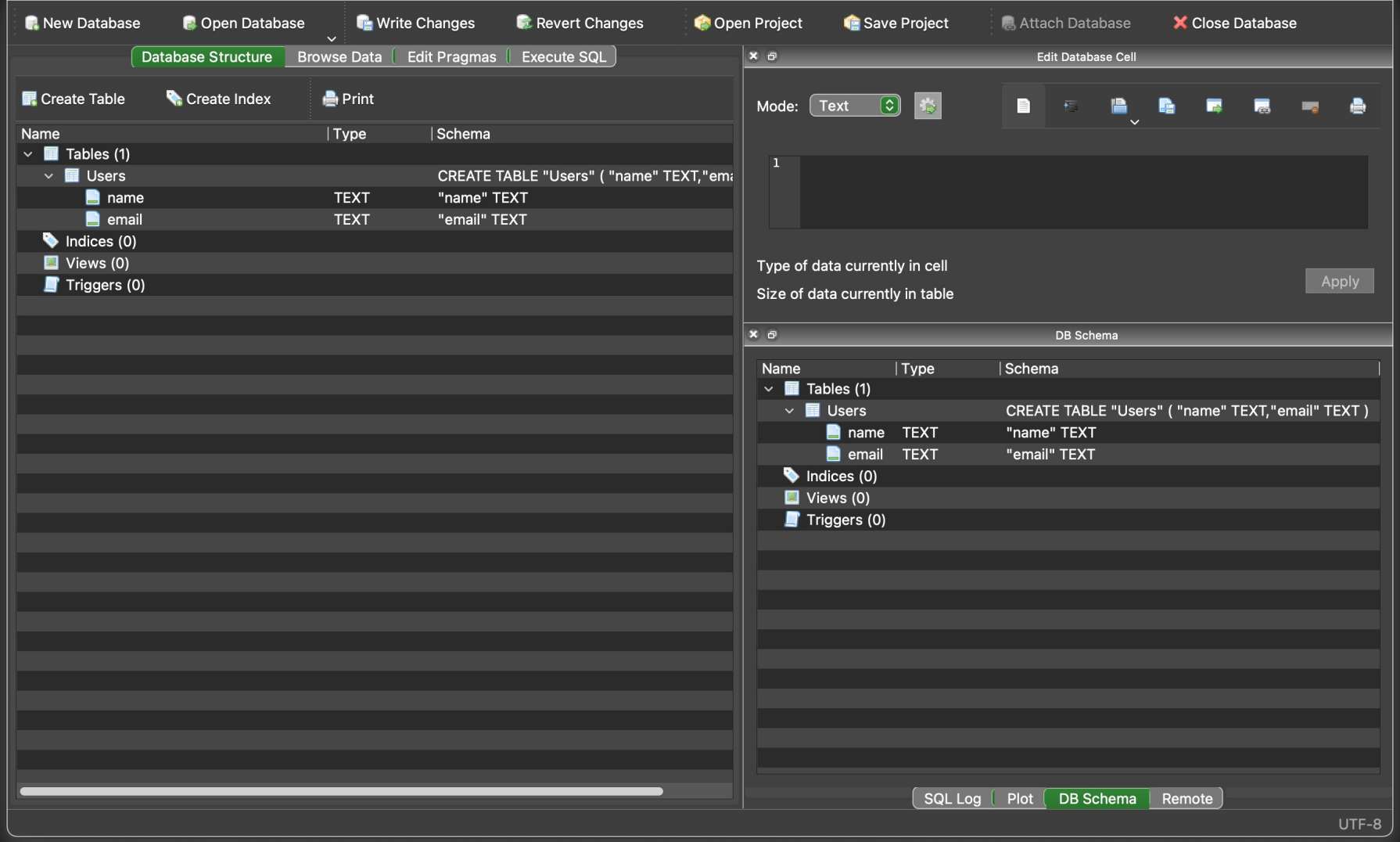
Please Note!

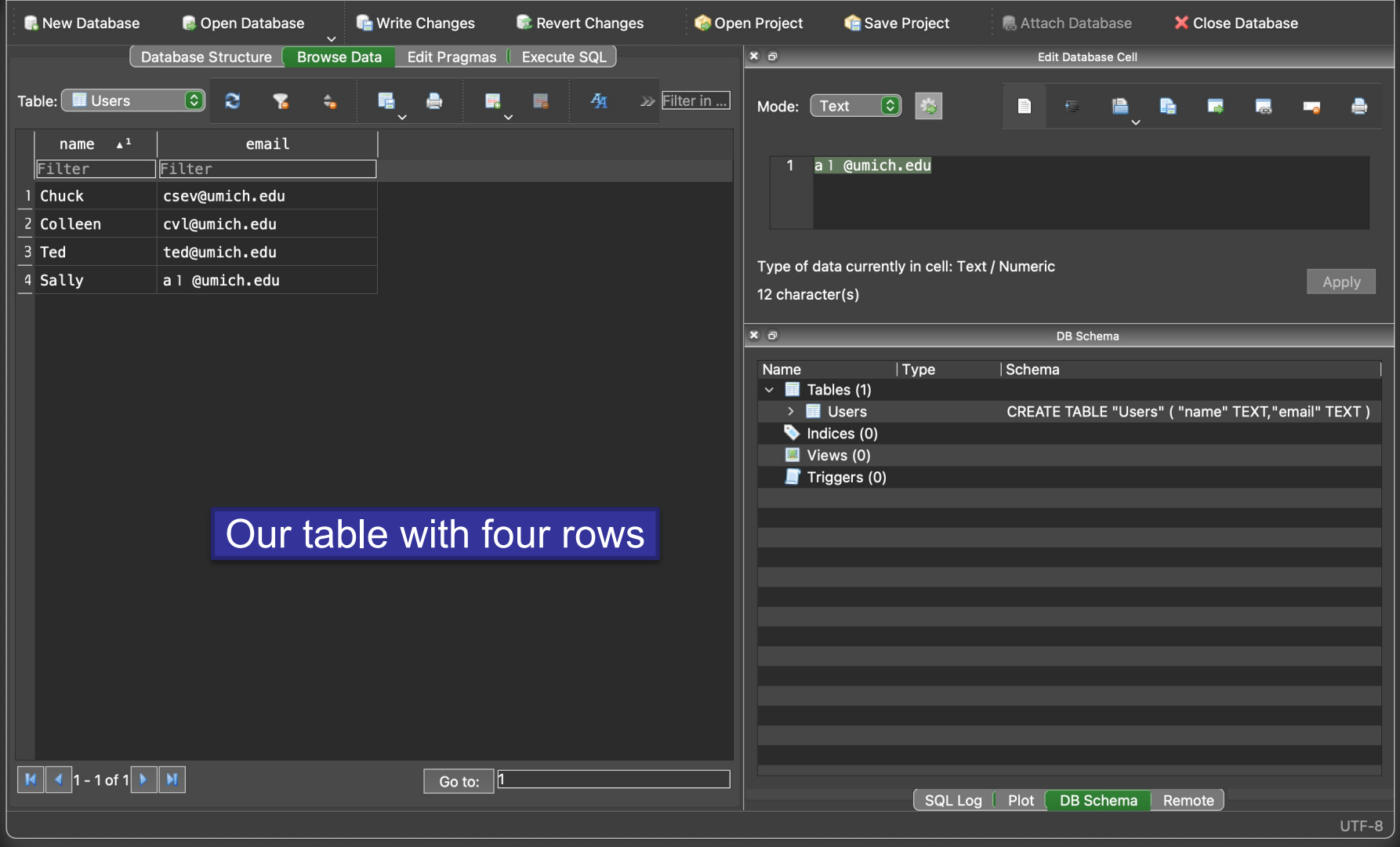
The following slides describe what we can do with the DB Browser application. It's not practical to constantly shift back and forth between my slides and the actual application.
When you get home, download and open the DB Browser application and go through all the steps I did to make sure you understand **what** I did and **how to use** the database application.

Start Simple - A Single Table




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


New Database
Open Database
Write Changes
Revert Changes
Open Project
Save Project
Attach Database
Close Database
Database Structure
Browse Data
Edit Pragmas
Execute SQL
Create Table
Create Index
Print
Name
Type
Schema
Tables (1)
Users
CREATE TABLE "Users" ("name" TEXT,"ema
name
TEXT
"name" TEXT
email
TEXT
"email" TEXT
Indices (0)
Views (0)
Triggers (0)
Edit Database Cell
Mode:
Text
1
Type of data currently in cell
Size of data currently in table
Apply
DB Schema
Name
Type
Schema
Tables (1)
Users
CREATE TABLE "Users" ("name" TEXT,"email" TEXT)
name
TEXT
"name" TEXT
email
TEXT
"email" TEXT
Indices (0)
Views (0)
Triggers (0)
SQL Log
Plot
DB Schema
Remote
UTF-8


New Database
Open Database
Write Changes
Revert Changes
Open Project
Save Project
Attach Database
Close Database
Database Structure
Browse Data
Edit Pragmas
Execute SQL
Table: Users
Filter in ...
name
email
Filter
Filter
1 Chuck csev@umich.edu
2 Colleen cvl@umich.edu
3 Ted ted@umich.edu
4 Sally a l @umich.edu
Our table with four rows
1 - 1 of 1
Go to: 1
Edit Database Cell
Mode: Text
1 a l @umich.edu
Type of data currently in cell: Text / Numeric
12 character(s)
Apply
DB Schema
Name
Type
Schema
Tables (1)
Users
CREATE TABLE "Users" ("name" TEXT,"email" TEXT)
Indices (0)
Views (0)
Triggers (0)
SQL Log
Plot
DB Schema
Remote
UTF-8

SQL

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

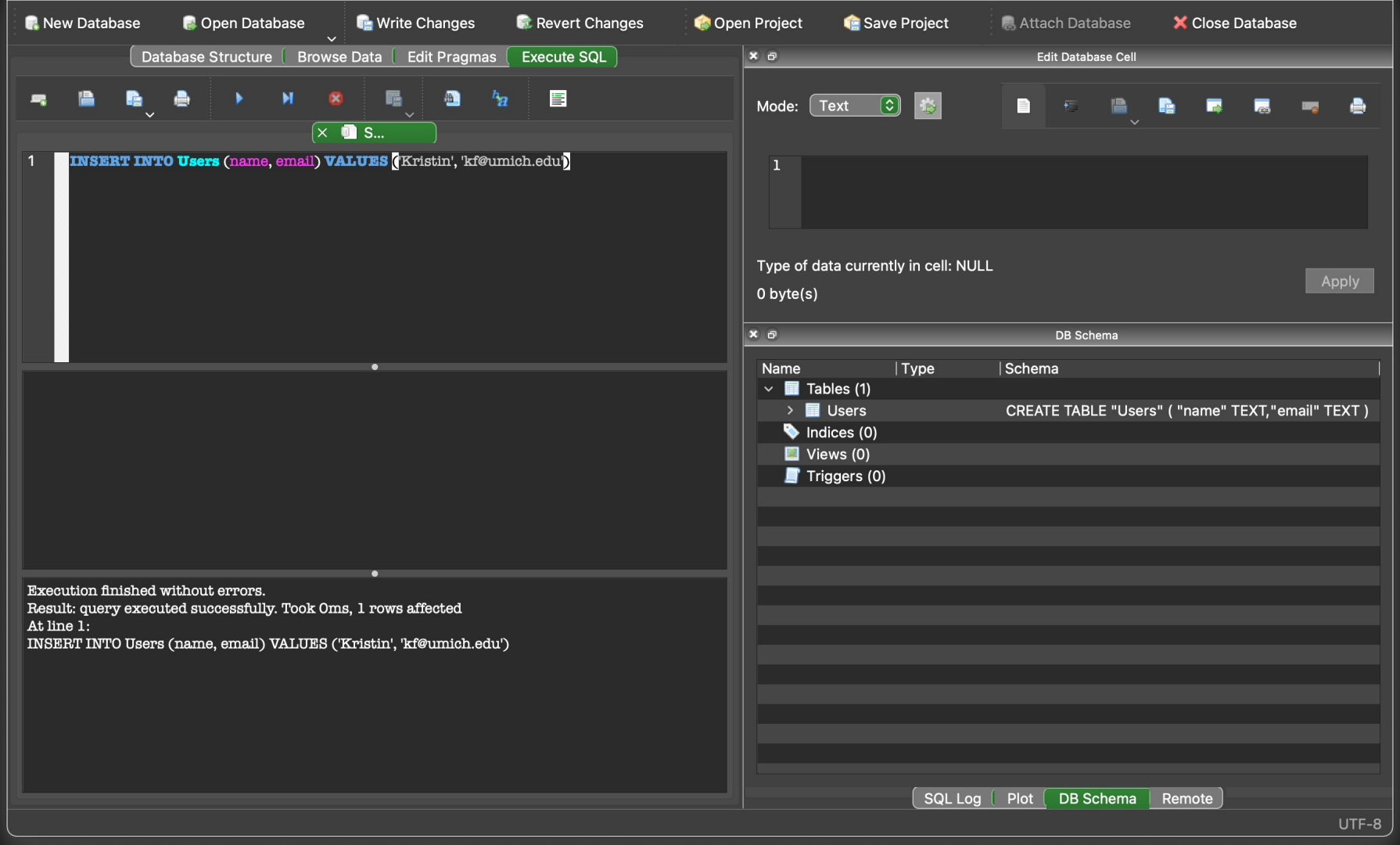
- Create data (a.k.a Insert)
- Retrieve data
- Update data
- Delete data

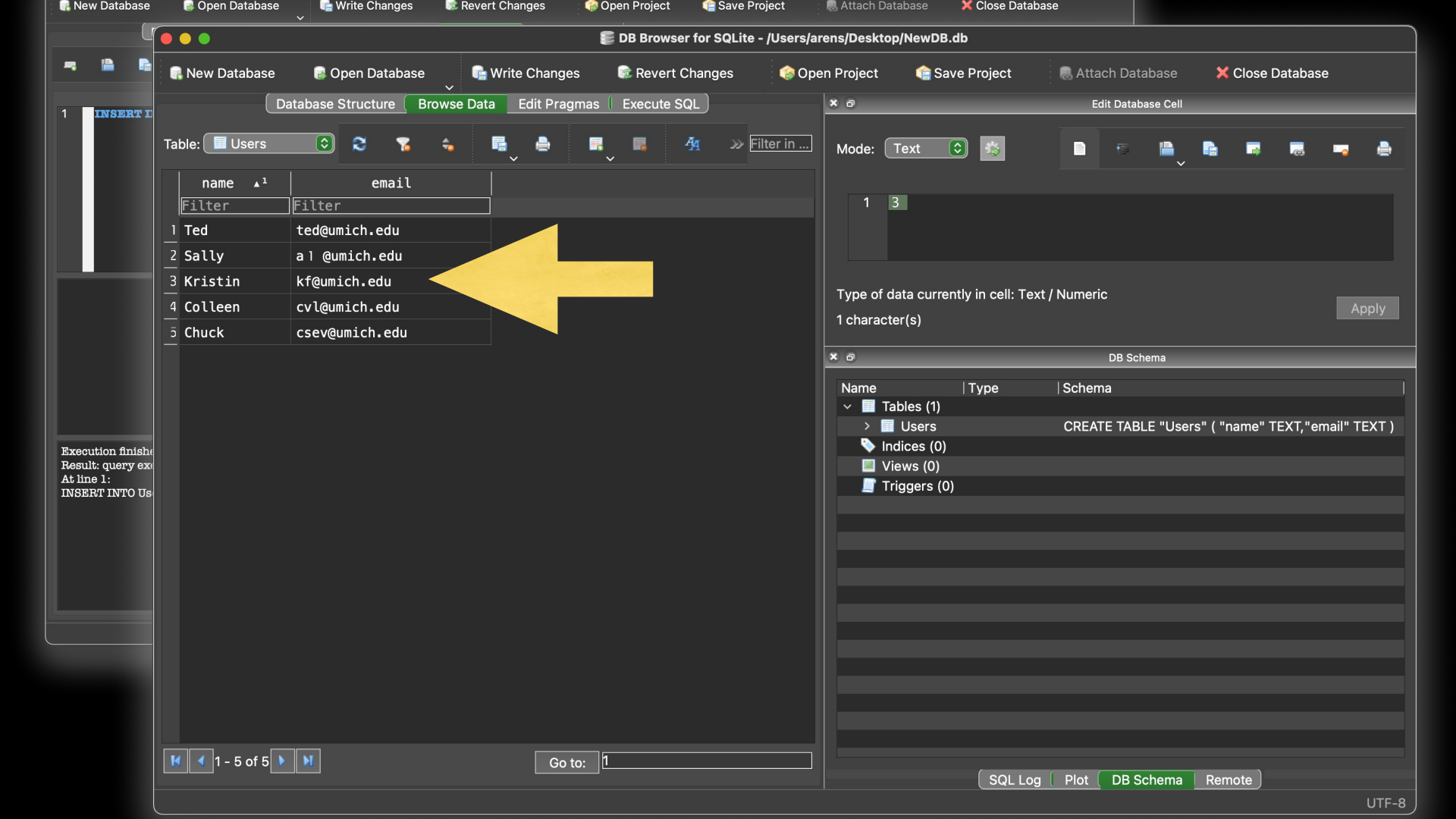
http://en.wikipedia.org/wiki/SQL

SQL: Insert

The Insert statement inserts a row into a table

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

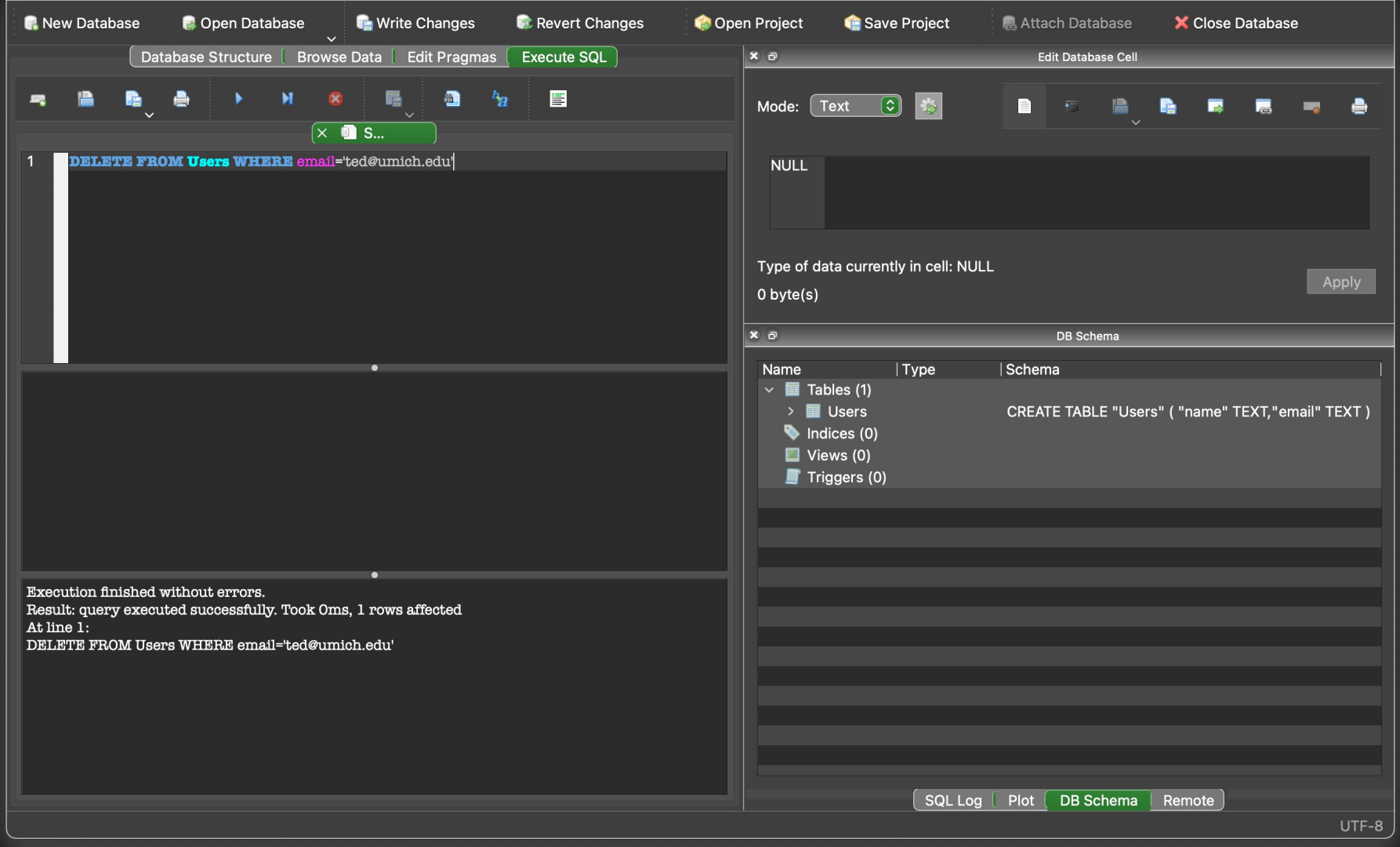

New Database
Open Database
Write Changes
Revert Changes
Open Project
Save Project
Attach Database
Close Database
Database Structure
Browse Data
Edit Pragmas
Execute SQL
S...
1
INSERT INTO Users (name, email) VALUES ('Kristin', 'kf@umich.edu')
Execution finished without errors.
Result: query executed successfully. Took 0ms, 1 rows affected
At line 1:
INSERT INTO Users (name, email) VALUES ('Kristin', 'kf@umich.edu')
Edit Database Cell
Mode:
Text
1
Type of data currently in cell: NULL
0 byte(s)
Apply
DB Schema
Name
Type
Schema
Tables (1)
Users
CREATE TABLE "Users" ("name" TEXT,"email" TEXT)
Indices (0)
Views (0)
Triggers (0)
SQL Log
Plot
DB Schema
Remote
UTF-8

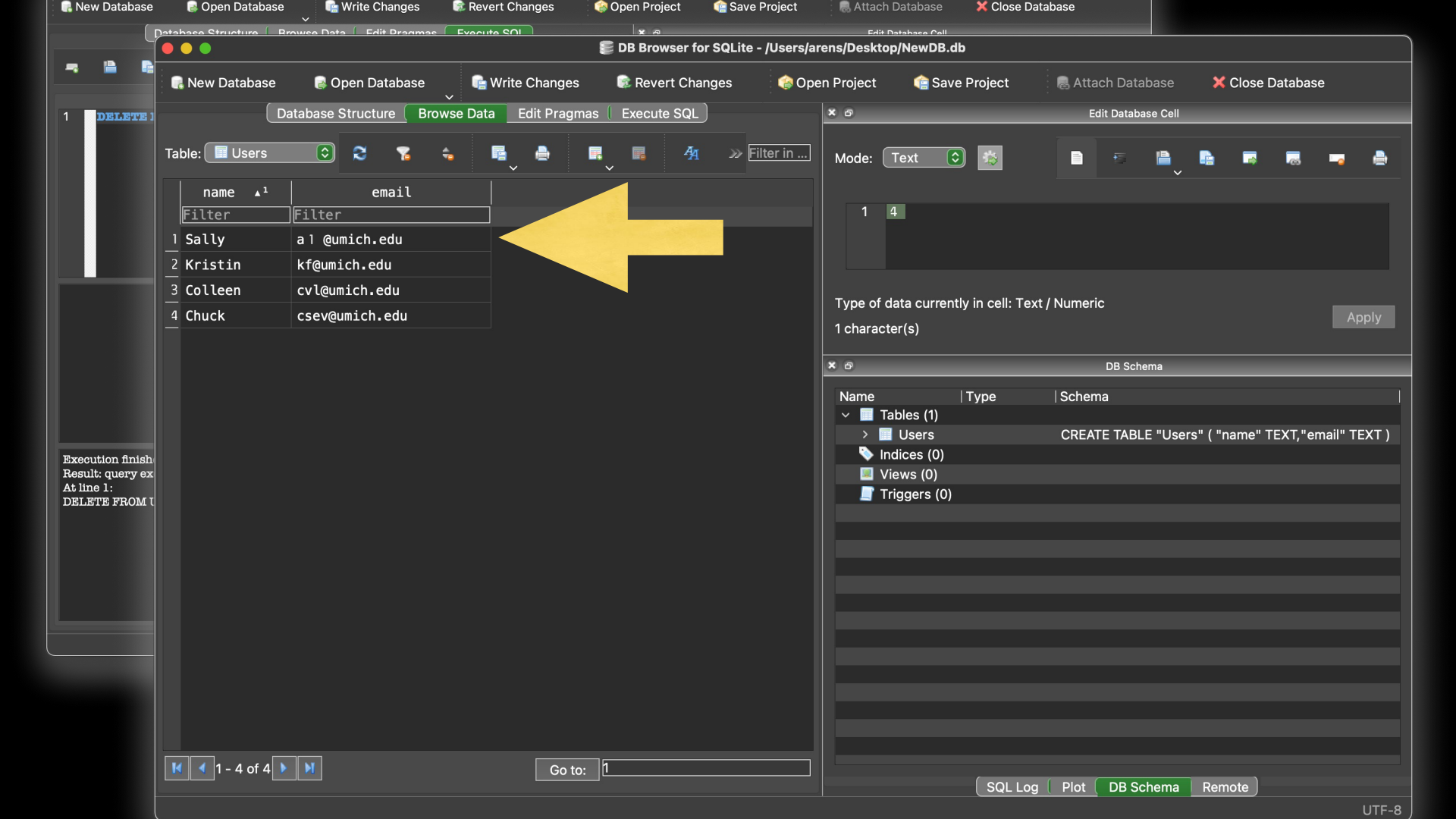

DB Browser for SQLite - /Users/arens/Desktop/NewDB.db
New Database
Open Database
Write Changes
Revert Changes
Open Project
Save Project
Attach Database
Close Database
Database Structure
Browse Data
Edit Pragmas
Execute SQL
Table: Users
Filter in ...
name
email
Filter
1 Ted ted@umich.edu
2 Sally a l @umich.edu
3 Kristin kf@umich.edu
4 Colleen cvl@umich.edu
5 Chuck csev@umich.edu
1 - 5 of 5
Go to: 1
Edit Database Cell
Mode: Text
1 3
Type of data currently in cell: Text / Numeric
1 character(s)
Apply
DB Schema
Name
Type
Schema
Tables (1)
Users
CREATE TABLE "Users" ("name" TEXT,"email" TEXT)
Indices (0)
Views (0)
Triggers (0)
SQL Log
Plot
DB Schema
Remote
UTF-8

SQL: Delete

Deletes a row in a table based on selection criteria

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

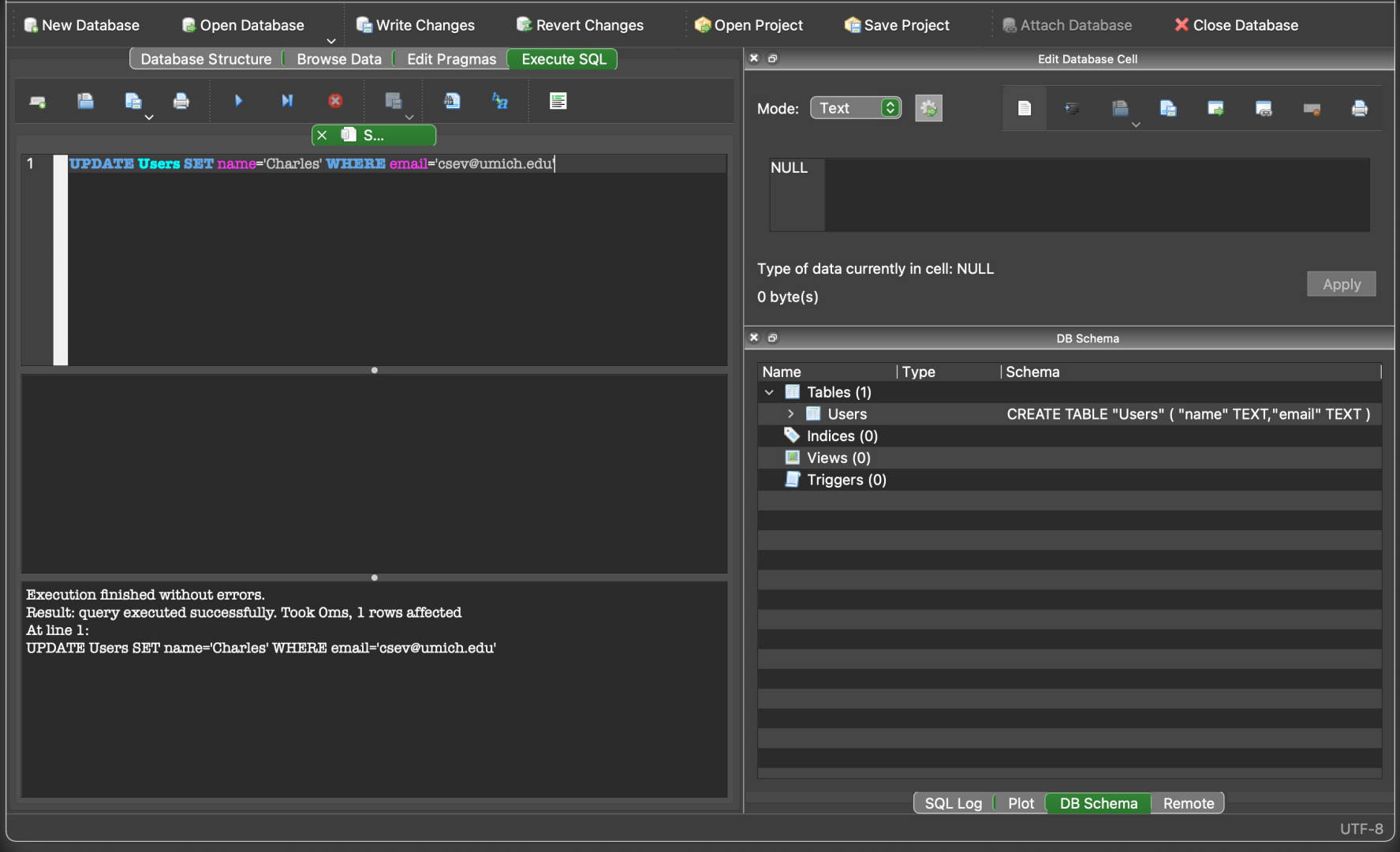

New Database
Open Database
Write Changes
Revert Changes
Open Project
Save Project
Attach Database
Close Database
Database Structure
Browse Data
Edit Pragmas
Execute SQL
S...
1
DELETE FROM Users WHERE email='ted@umich.edu'
Execution finished without errors.
Result: query executed successfully. Took 0ms, 1 rows affected
At line 1:
DELETE FROM Users WHERE email='ted@umich.edu'
Edit Database Cell
Mode:
Text
NULL
Type of data currently in cell: NULL
0 byte(s)
Apply
DB Schema
Name
Type
Schema
Tables (1)
Users
CREATE TABLE "Users" ("name" TEXT,"email" TEXT)
Indices (0)
Views (0)
Triggers (0)
SQL Log
Plot
DB Schema
Remote
UTF-8

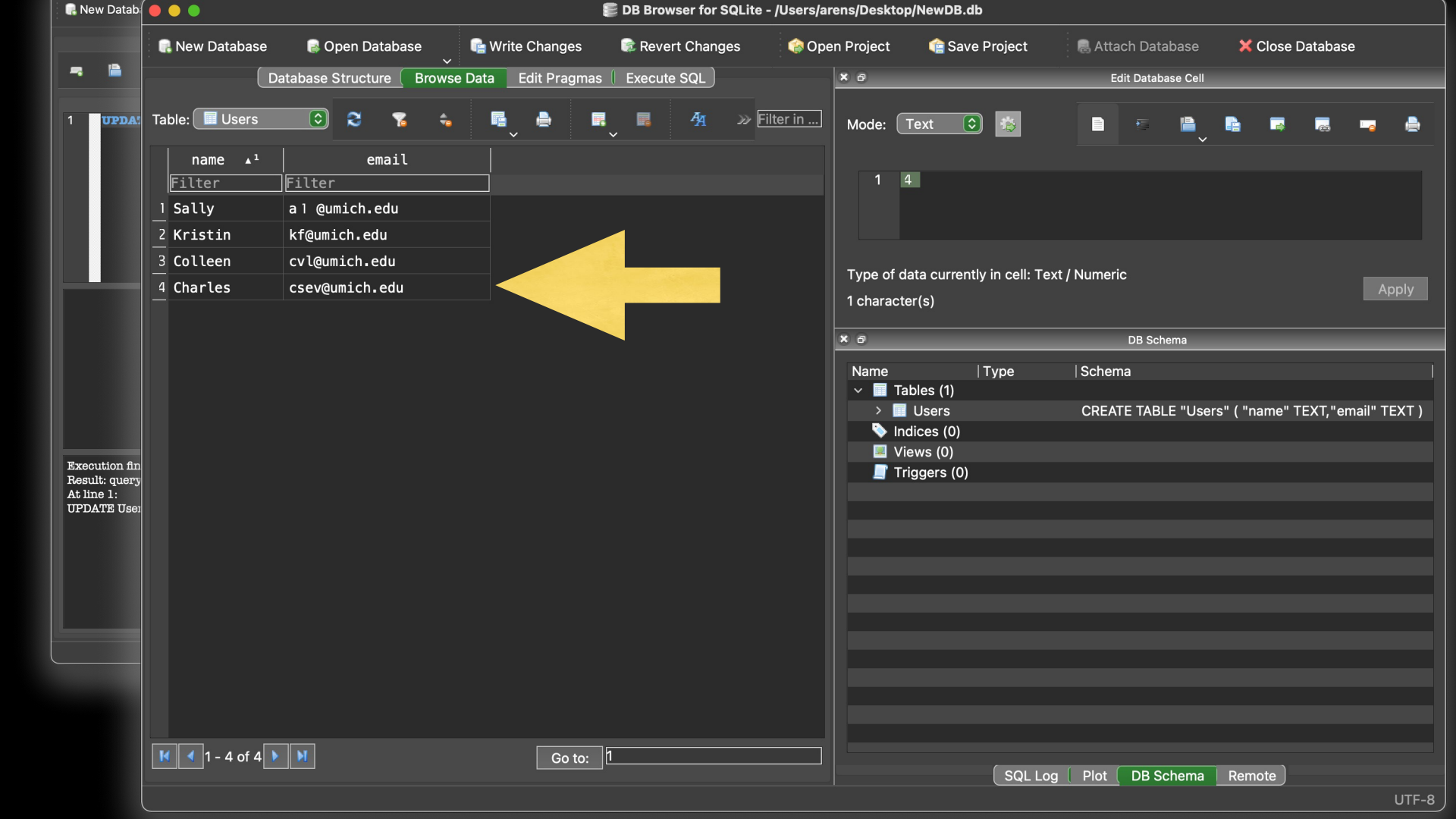

DB Browser for SQLite - /Users/arens/Desktop/NewDB.db
New Database
Open Database
Write Changes
Revert Changes
Open Project
Save Project
Attach Database
Close Database
Database Structure
Browse Data
Edit Pragmas
Execute SQL
Table: Users
Filter in ...
name
email
Filter
1 Sally a l @umich.edu
2 Kristin kf@umich.edu
3 Colleen cvl@umich.edu
4 Chuck csev@umich.edu
1 - 4 of 4
Go to: 1
Edit Database Cell
Mode: Text
1 4
Type of data currently in cell: Text / Numeric
1 character(s)
Apply
DB Schema
Name
Type
Schema
Tables (1)
Users
CREATE TABLE "Users" ("name" TEXT,"email" TEXT)
Indices (0)
Views (0)
Triggers (0)
SQL Log
Plot
DB Schema
Remote
UTF-8

SQL: Update

Allows the updating of a field with a where clause

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


New Database
Open Database
Write Changes
Revert Changes
Open Project
Save Project
Attach Database
Close Database
Database Structure
Browse Data
Edit Pragmas
Execute SQL
S...
1
UPDATE Users SET name='Charles' WHERE email='csev@umich.edu'
Execution finished without errors.
Result: query executed successfully. Took 0ms, 1 rows affected
At line 1:
UPDATE Users SET name='Charles' WHERE email='csev@umich.edu'
Edit Database Cell
Mode:
Text
NULL
Type of data currently in cell: NULL
0 byte(s)
Apply
DB Schema
Name
Type
Schema
Tables (1)
Users
CREATE TABLE "Users" ("name" TEXT,"email" TEXT)
Indices (0)
Views (0)
Triggers (0)
SQL Log
Plot
DB Schema
Remote
UTF-8


DB Browser for SQLite - /Users/arens/Desktop/NewDB.db
New Database
Open Database
Write Changes
Revert Changes
Open Project
Save Project
Attach Database
Close Database
Database Structure
Browse Data
Edit Pragmas
Execute SQL
Table: Users
Filter in ...
name
email
Filter
Filter
1 Sally a l @umich.edu
2 Kristin kf@umich.edu
3 Colleen cvl@umich.edu
4 Charles csev@umich.edu
1 - 4 of 4
Go to: 1
Edit Database Cell
Mode: Text
1 4
Type of data currently in cell: Text / Numeric
1 character(s)
Apply
DB Schema
Name
Type
Schema
Tables (1)
Users
CREATE TABLE "Users" ("name" TEXT,"email" TEXT)
Indices (0)
Views (0)
Triggers (0)
SQL Log
Plot
DB Schema
Remote
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

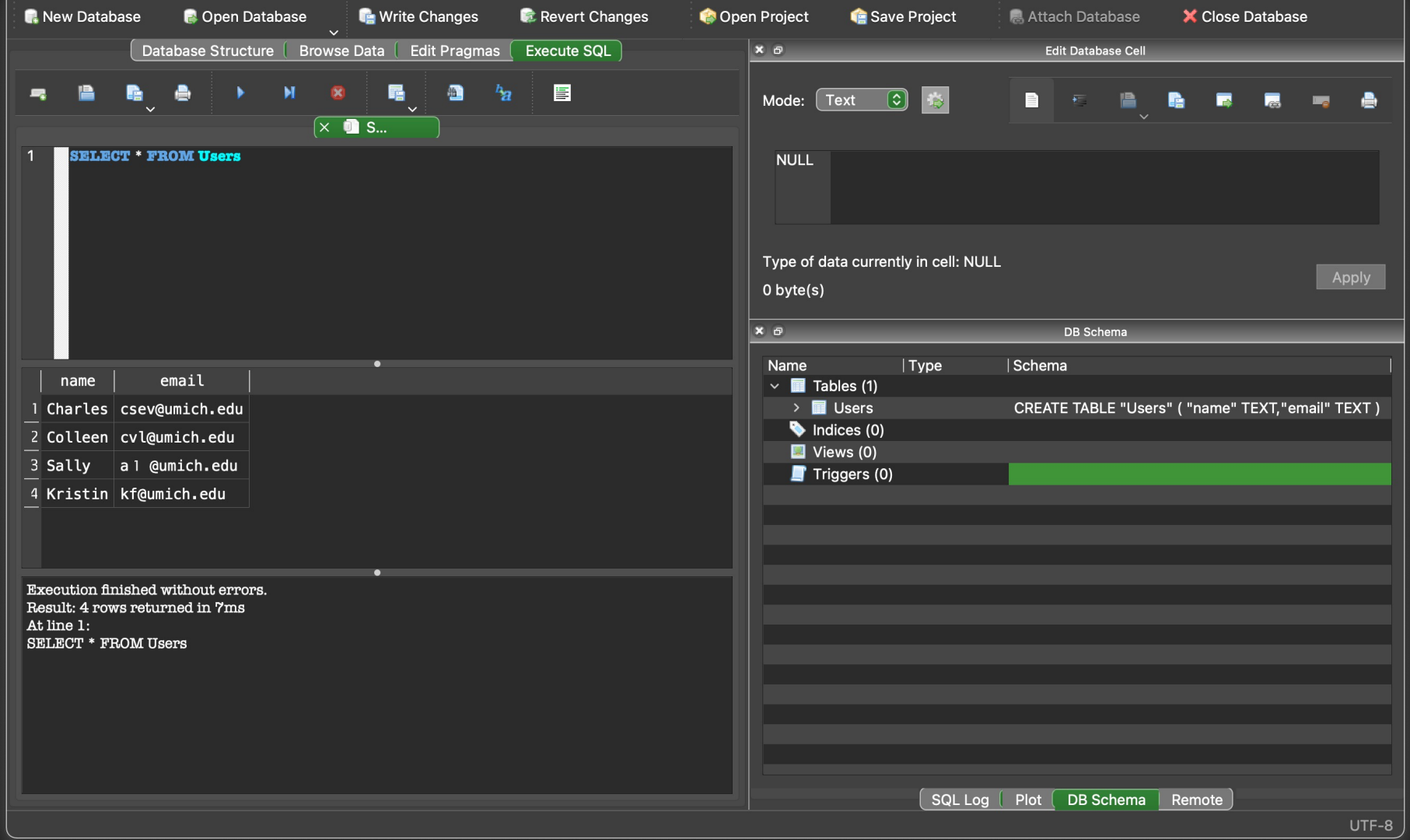
```
SELECT * FROM Users
```

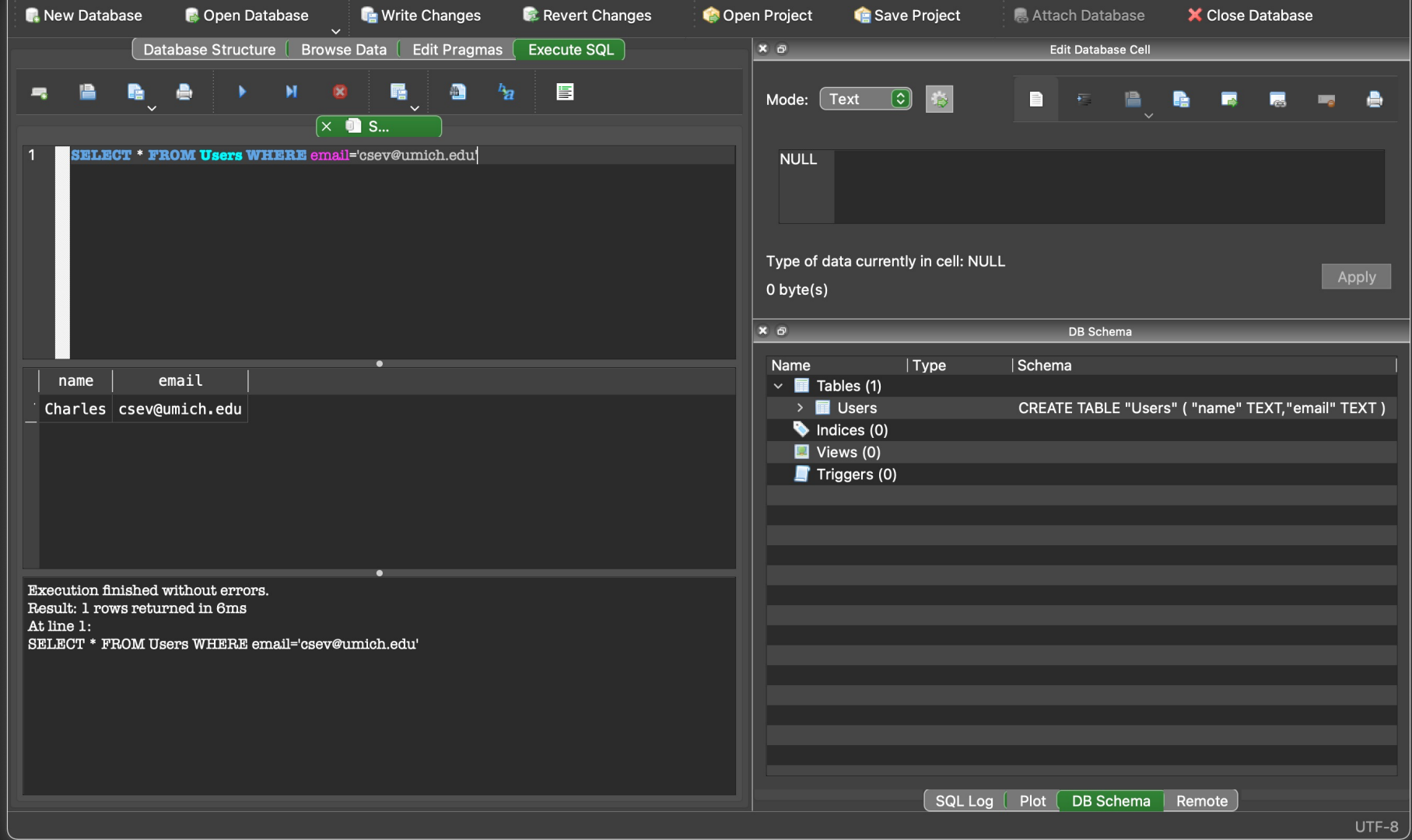
* means retrieve all the columns

or just list the columns you want to get

```
SELECT email FROM Users
```

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


New Database
Open Database
Write Changes
Revert Changes
Open Project
Save Project
Attach Database
Close Database
Database Structure
Browse Data
Edit Pragmas
Execute SQL
S...
1
SELECT * FROM Users
name
email
1 Charles csev@umich.edu
2 Colleen cvl@umich.edu
3 Sally a1 @umich.edu
4 Kristin kf@umich.edu
Execution finished without errors.
Result: 4 rows returned in 7ms
At line 1:
SELECT * FROM Users
Edit Database Cell
Mode:
Text
NULL
Type of data currently in cell: NULL
0 byte(s)
Apply
DB Schema
Name
Type
Schema
Tables (1)
Users
CREATE TABLE "Users" ("name" TEXT,"email" TEXT)
Indices (0)
Views (0)
Triggers (0)
SQL Log
Plot
DB Schema
Remote
UTF-8

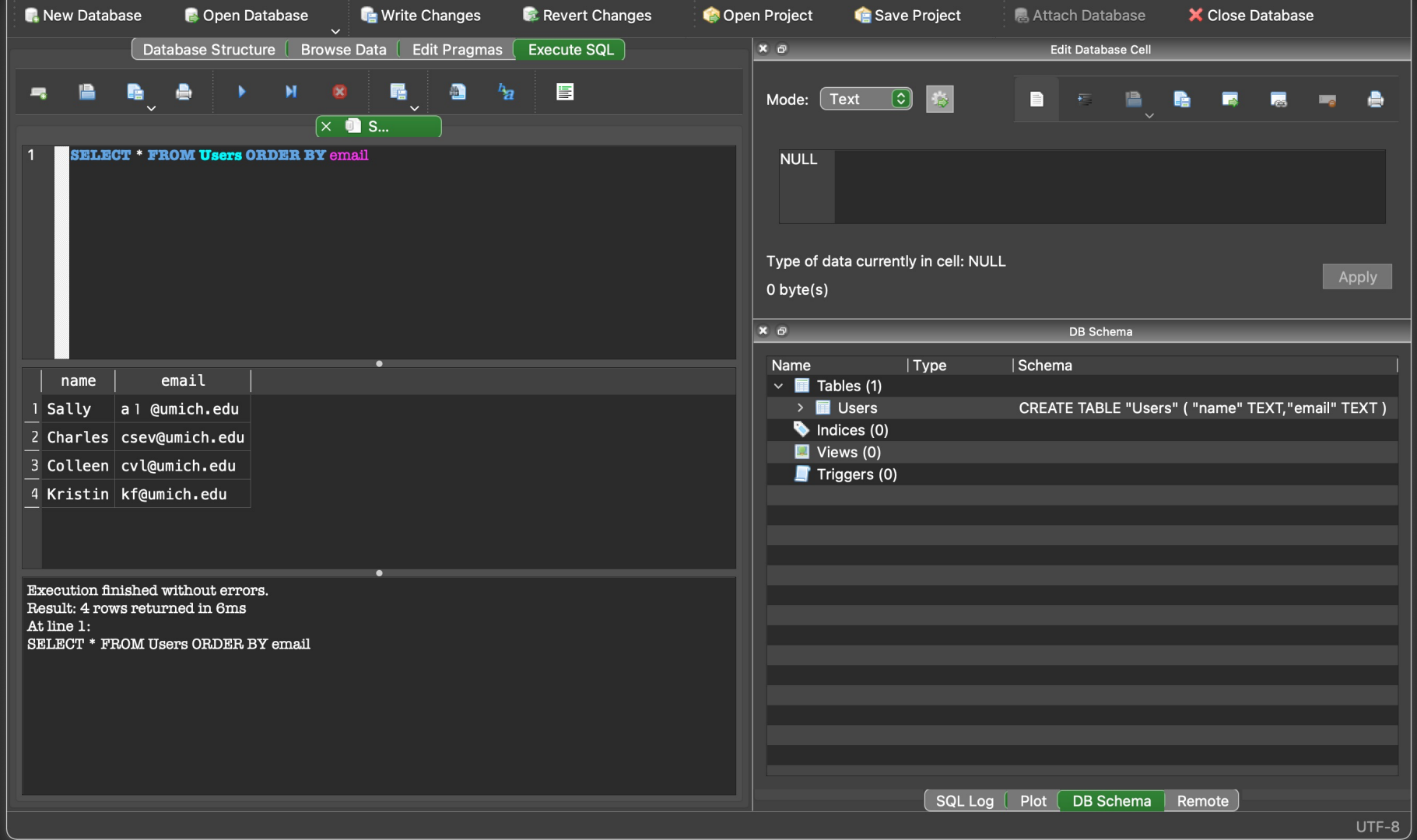

New Database
Open Database
Write Changes
Revert Changes
Open Project
Save Project
Attach Database
Close Database
Database Structure
Browse Data
Edit Pragmas
Execute SQL
S...
1
SELECT * FROM Users WHERE email='csev@umich.edu'
name
email
Charles
csev@umich.edu
Execution finished without errors.
Result: 1 rows returned in 6ms
At line 1:
SELECT * FROM Users WHERE email='csev@umich.edu'
Edit Database Cell
Mode:
Text
NULL
Type of data currently in cell: NULL
0 byte(s)
Apply
DB Schema
Name
Type
Schema
Tables (1)
Users
CREATE TABLE "Users" ("name" TEXT,"email" TEXT)
Indices (0)
Views (0)
Triggers (0)
SQL Log
Plot
DB Schema
Remote
UTF-8

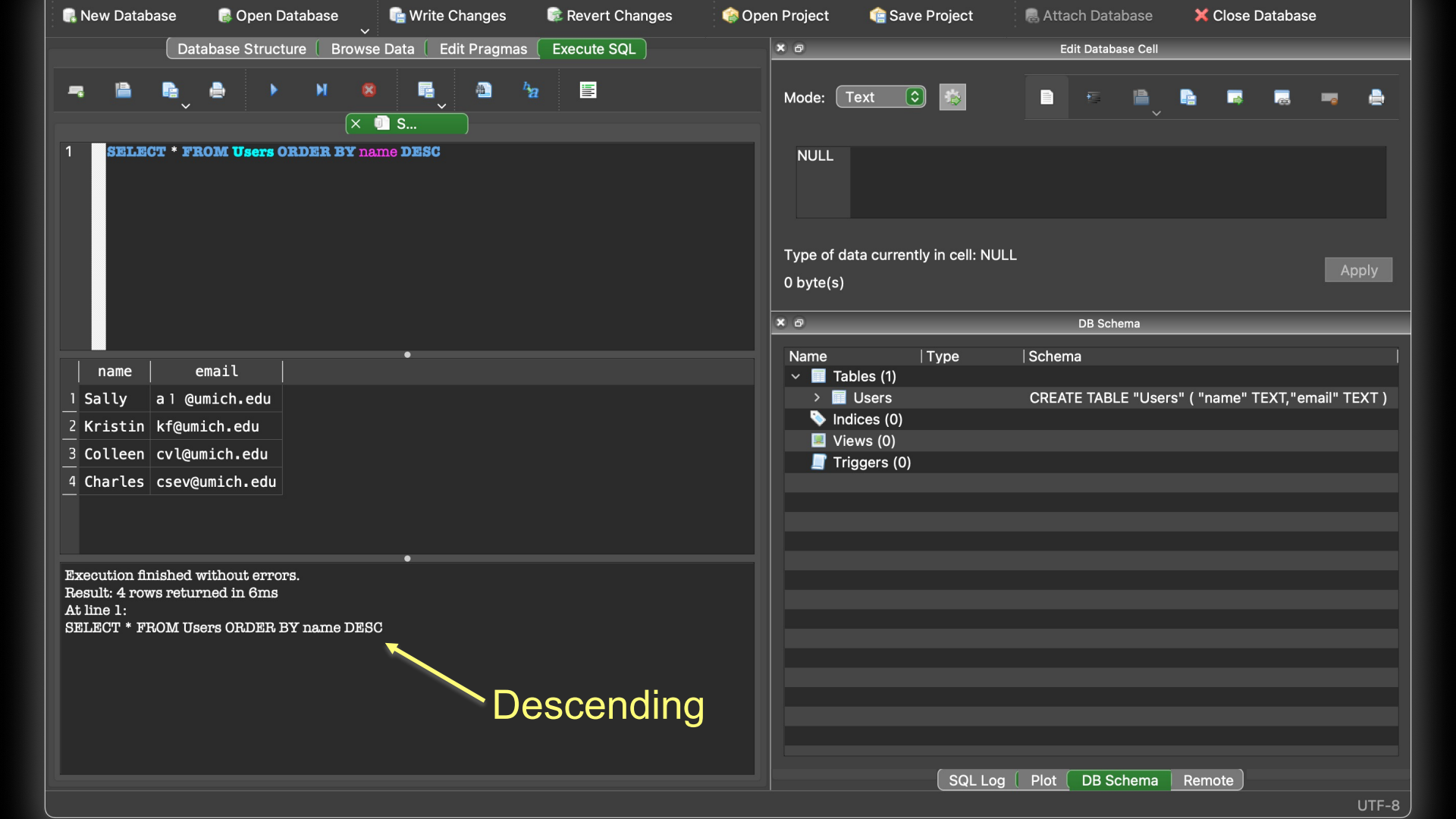
Sorting with ORDER BY

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
```


New Database
Open Database
Write Changes
Revert Changes
Open Project
Save Project
Attach Database
Close Database
Database Structure
Browse Data
Edit Pragmas
Execute SQL
S...
1 SELECT * FROM Users ORDER BY email
name email
1 Sally a l @umich.edu
2 Charles csev@umich.edu
3 Colleen cvl@umich.edu
4 Kristin kf@umich.edu
Execution finished without errors.
Result: 4 rows returned in 6ms
At line 1:
SELECT * FROM Users ORDER BY email
Edit Database Cell
Mode: Text
NULL
Type of data currently in cell: NULL
0 byte(s)
Apply
DB Schema
Name Type Schema
Tables (1)
Users CREATE TABLE "Users" ("name" TEXT,"email" TEXT)
Indices (0)
Views (0)
Triggers (0)
SQL Log
Plot
DB Schema
Remote
UTF-8


New Database
Open Database
Write Changes
Revert Changes
Open Project
Save Project
Attach Database
Close Database
Database Structure
Browse Data
Edit Pragmas
Execute SQL
S...
1
SELECT * FROM Users ORDER BY name DESC
name
email
1 Sally a l @umich.edu
2 Kristin kf@umich.edu
3 Colleen cvl@umich.edu
4 Charles csev@umich.edu
Execution finished without errors.
Result: 4 rows returned in 6ms
At line 1:
SELECT * FROM Users ORDER BY name DESC
Descending
Edit Database Cell
Mode:
Text
NULL
Type of data currently in cell: NULL
0 byte(s)
Apply
DB Schema
Name
Type
Schema
Tables (1)
Users
CREATE TABLE "Users" ("name" TEXT,"email" TEXT)
Indices (0)
Views (0)
Triggers (0)
SQL Log
Plot
DB Schema
Remote
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 name, email FROM Users

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

SELECT * FROM Users ORDER BY email
```

Creating a Database in Python

```
import sqlite3

# create a connection to a database (file)
# If database file does not exist, sqlite3 creates it for you
conn = sqlite3.connect('emaildb.sqlite')  # emaildb.sqlite is a filename

# A database cursor is a mechanism that enables traversal over the records in a database.
# It allows retrieval, addition and removal of database records.
# It's an iterator.
cur = conn.cursor()

cur.execute('DROP TABLE IF EXISTS Counts')
cur.execute('CREATE TABLE Counts (email TEXT, count INTEGER)')

with open('mbox-short.txt') as f:
    for line in f:
        if not line.startswith('From: '): continue
        email = line.split()[1]
        cur.execute(f"SELECT count FROM Counts WHERE email = '{email}'")
        row = cur.fetchone()
        if row is None:
            cur.execute(f"INSERT INTO Counts (email, count) VALUES ('{email}', 1)")
        else:
            cur.execute(f"UPDATE Counts SET count = count + 1 WHERE email = '{email}'")
        conn.commit()

query = 'SELECT email, count FROM Counts ORDER BY count DESC LIMIT 10'

# cur.execute(query) returns an iterator over the query results
for row in cur.execute(query):
    print(f'{row[0]:32s} {row[1]:4d}')

cur.close() # we are done sending commands to the DB for now
conn.close() # we are completely done with the DB in this program
```

Counting emails by sender
Go to week12-email-db.ipynb

This is not too exciting (so far)

- 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

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

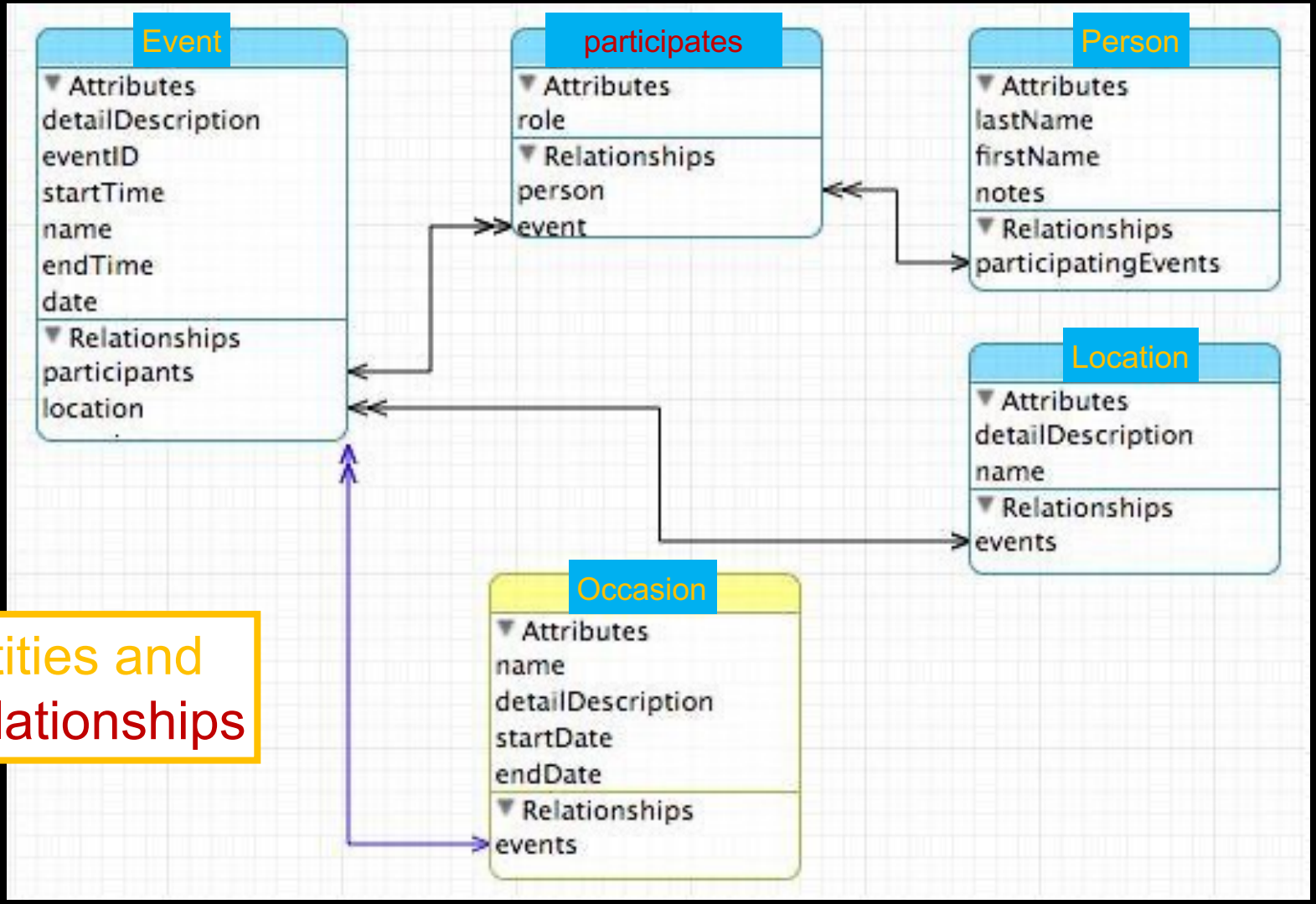
Database Design

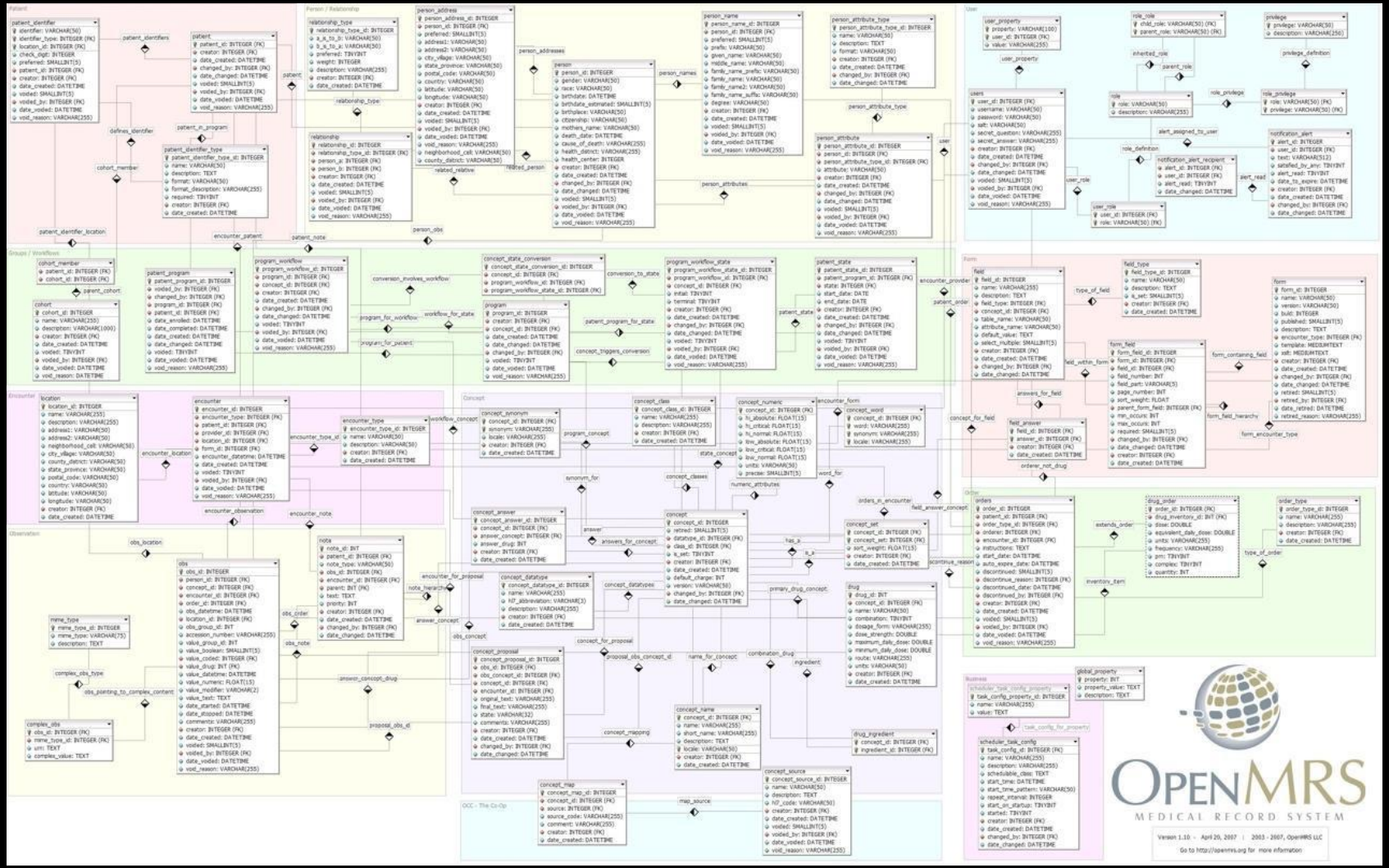
- Database design is an art form of its own with particular skills and experience
- Our goal is to avoid the really bad mistakes and design clean and easily understood databases
- Others may performance tune things later
- Database design starts with a picture...

Building an (Entity-Relationship) Data Model

- Think of the entities (and their attributes) and the relationships among entities that exist in your domain/application
- Draw a picture of the entities and their relationships
- Usually entities are "nouns" and relationships "verbs"
- These entities and relationships turn into tables

- Basic Rule:
  - Don’t put the same piece of 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


Event
Attributes
detailDescription
eventID
startTime
name
endTime
date
Relationships
participants
location
participates
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
Entities and
Relationships


OpenMRS
MEDICAL RECORD SYSTEM
Version 1.10 · April 20, 2007 | 2003 - 2007, OpenMRS LLC
Go to http://openmrs.org for more information

Track	Len	Artist	Album	Genre	Rating	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
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
- Avoid duplication

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

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

Track

Album

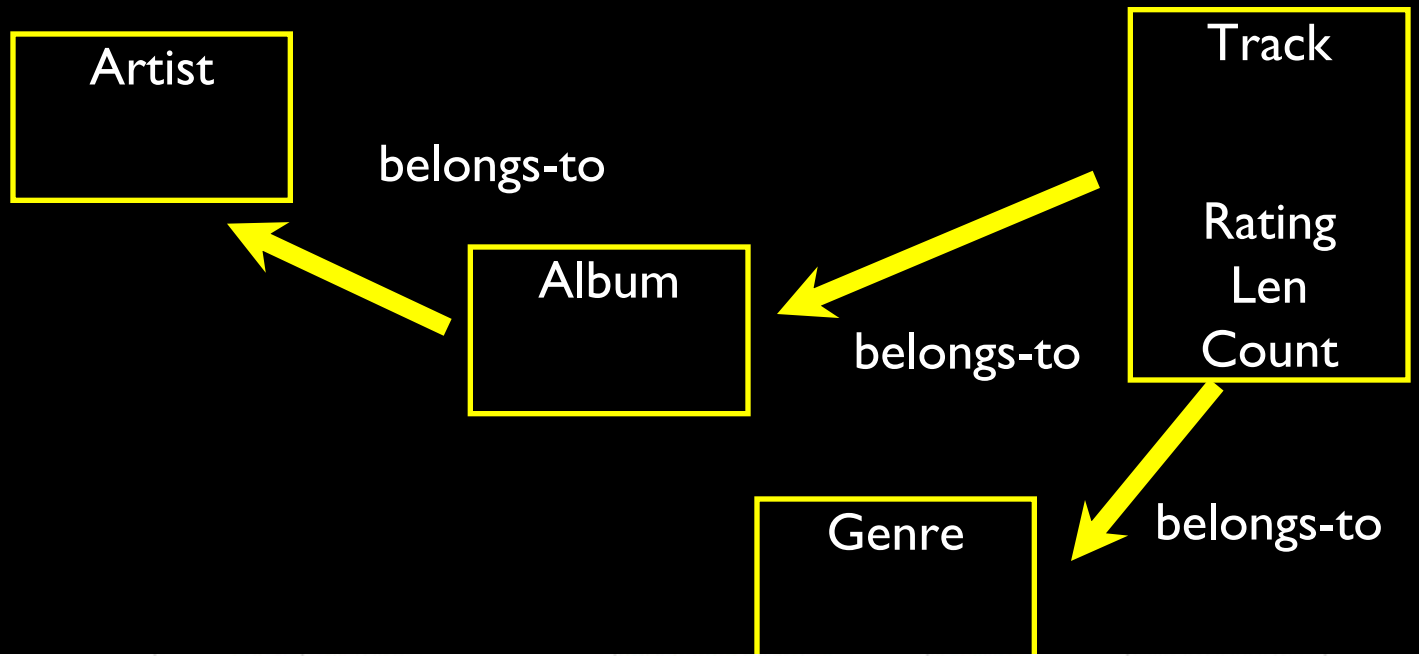
Artist

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

Representing Relationships in a Database

Database Normalization

- 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”

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

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

Key Rules

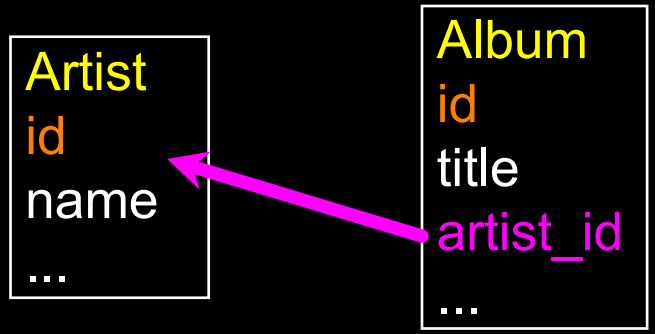
Best practices

- 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

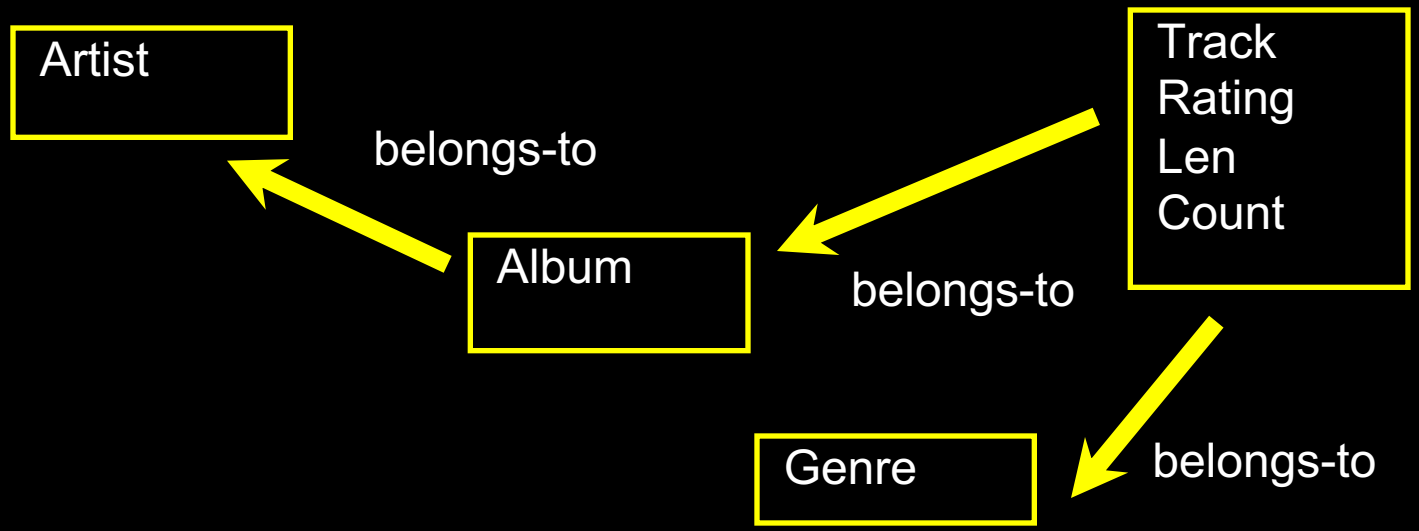
User
id
login
password
name
email
created_at
modified_at
login_at

Foreign Keys

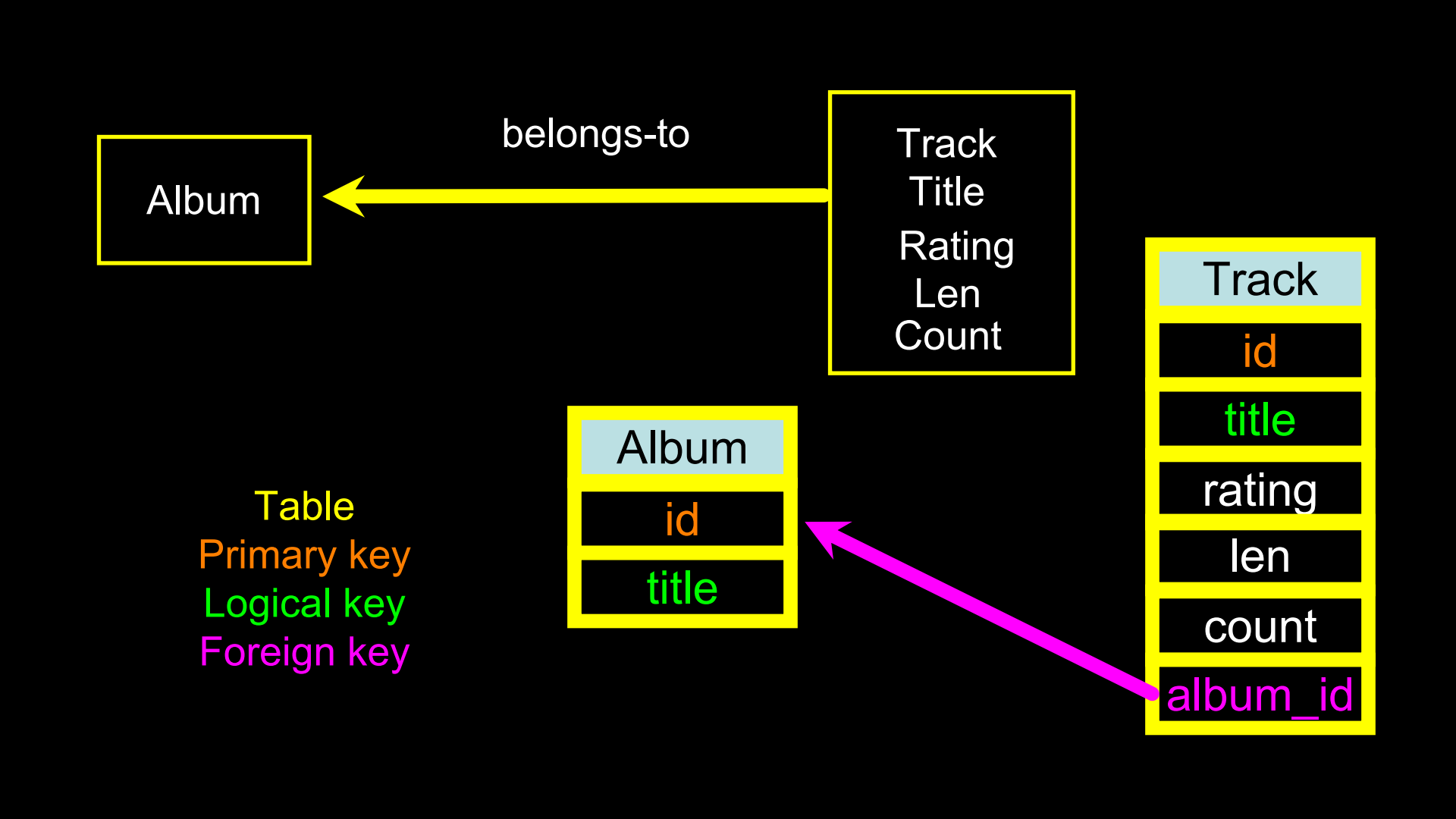
- A foreign key is when a table has a column that contains a key which points to the primary key of another table.
- When all primary keys are integers, then all foreign keys are integers - this is good - very good
- Good naming convention:
  <other_table>_id e.g., artist_id

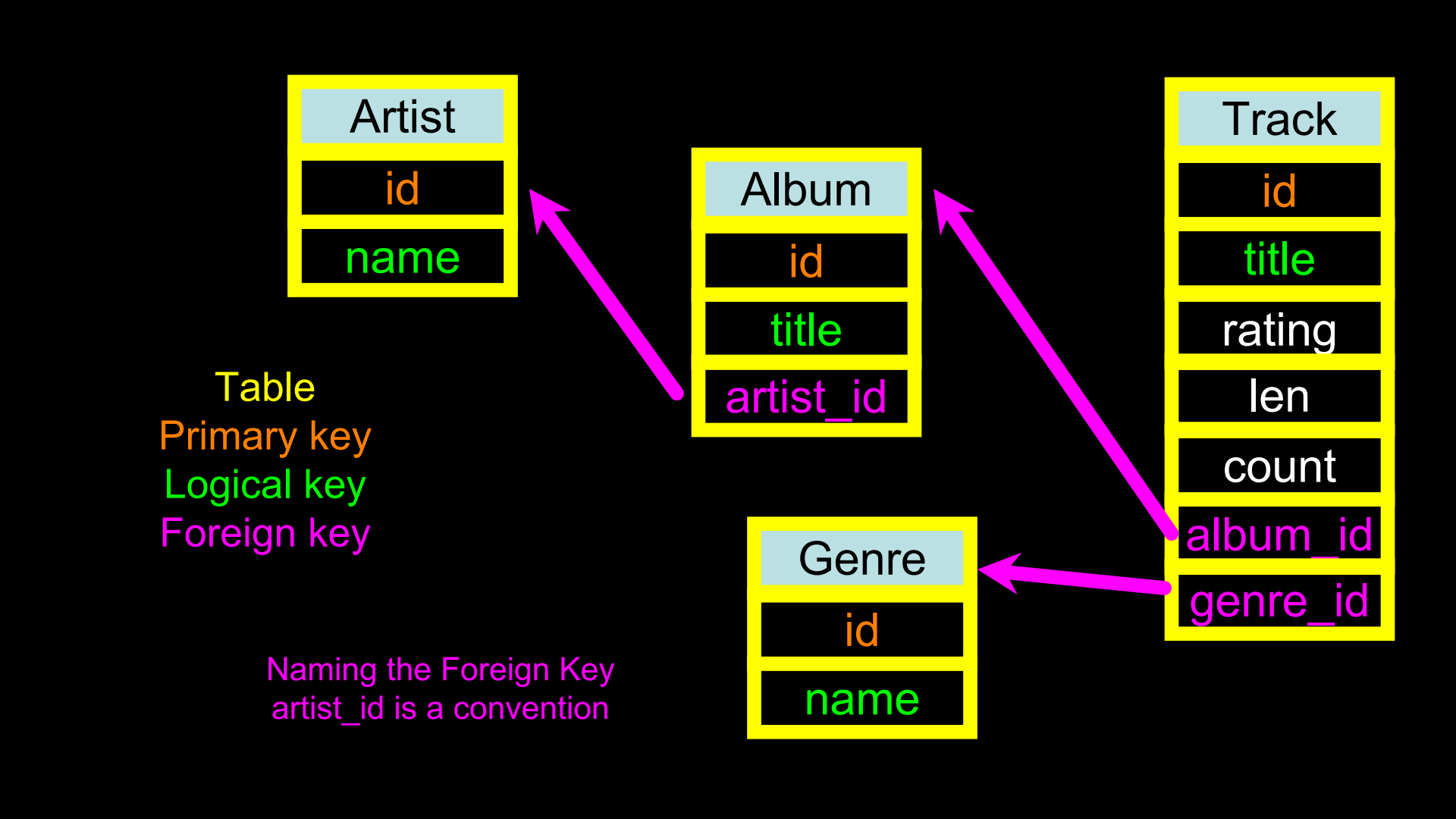


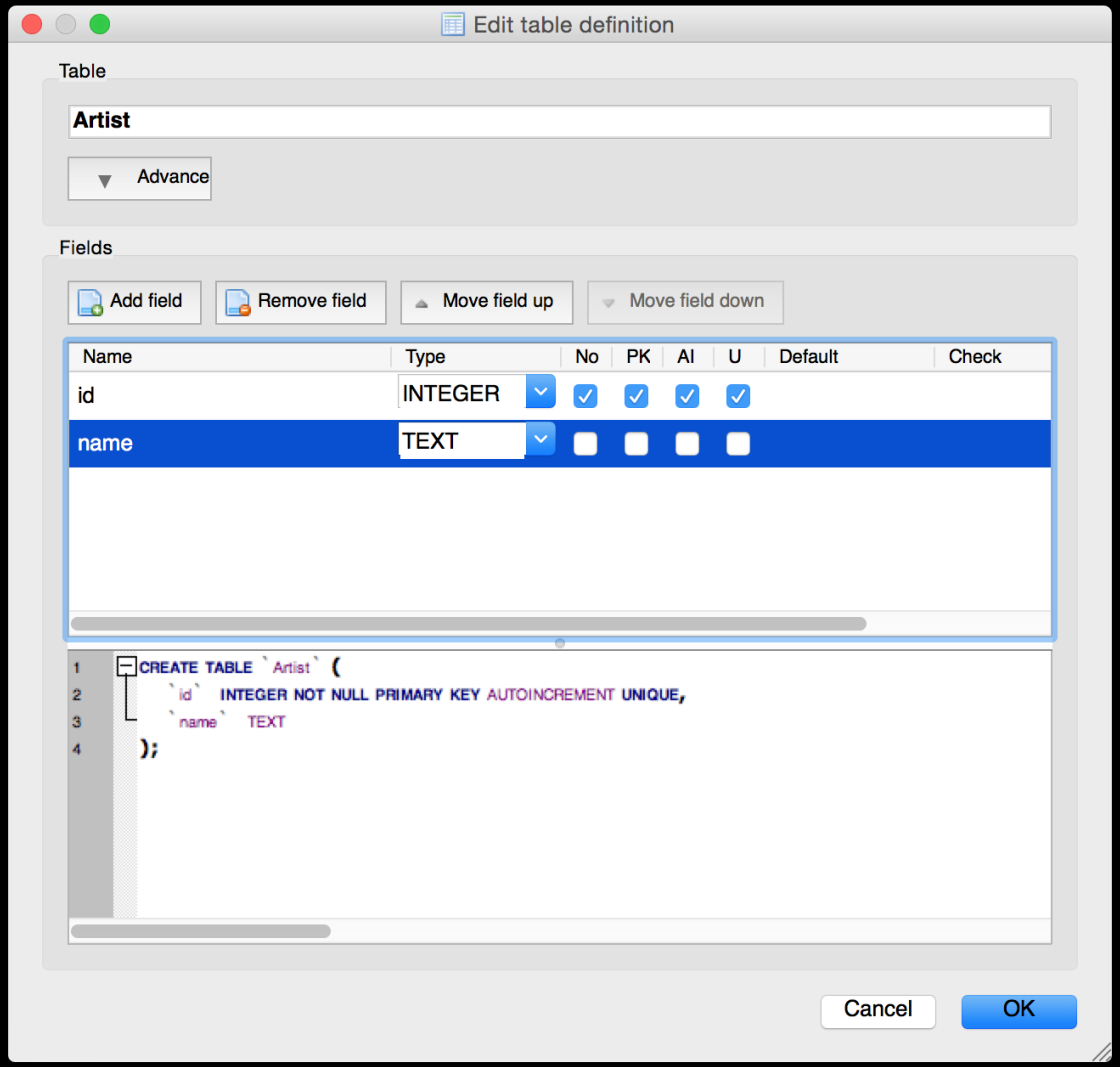
Relationship Building (in tables)



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

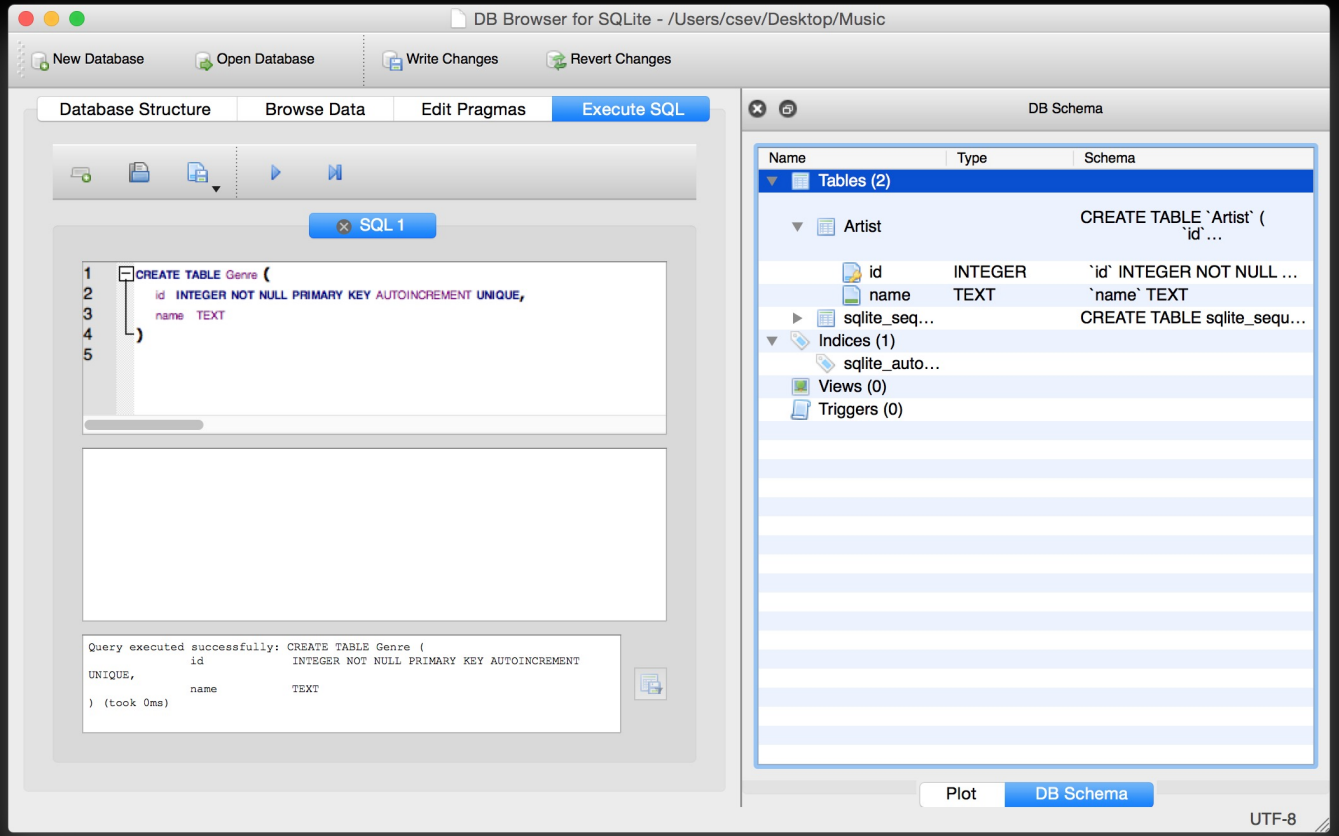

Album
belongs-to
Track
Title
Rating
Len
Count
Track
id
title
rating
len
count
album_id
Album
id
title
Table
Primary key
Logical key
Foreign key


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 the Foreign Key
artist_id is a convention



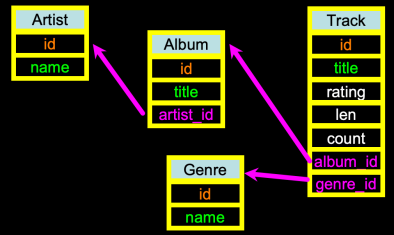
Id is the key:

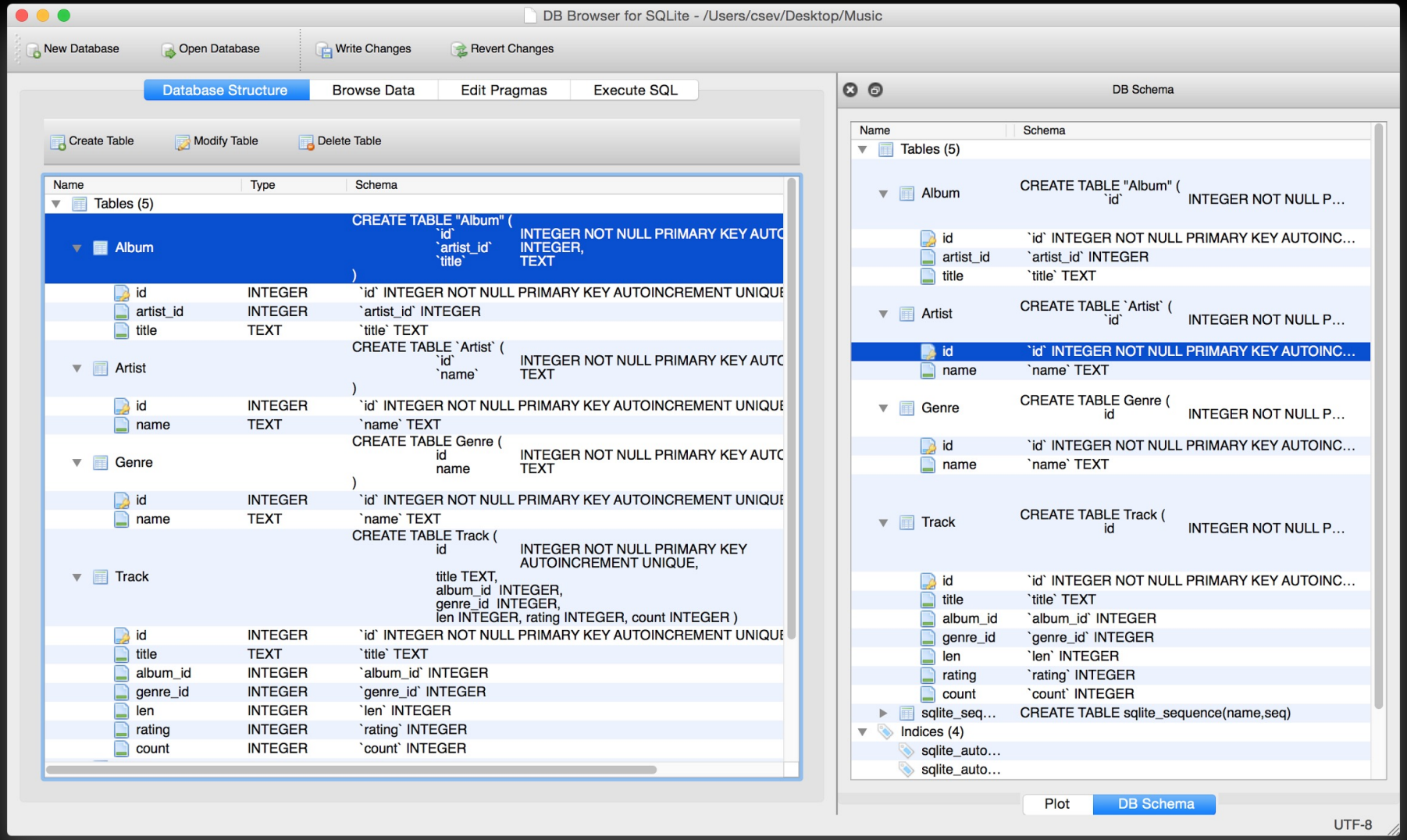
- No: NOT NULL
- PK: Primary Key
- AI: Auto-incremented
- U: Unique values

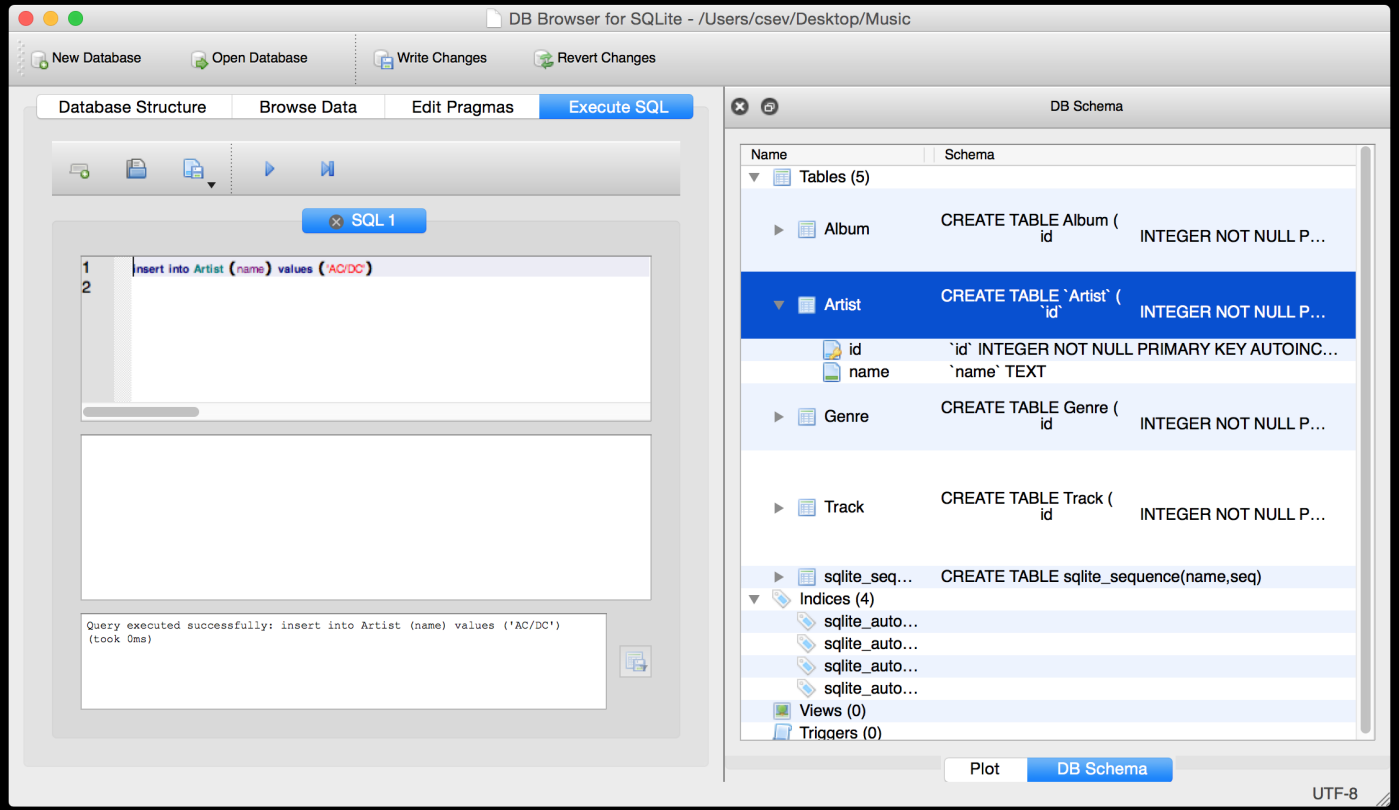



```
CREATE TABLE Genre (
    id      INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
    name    TEXT
)
```

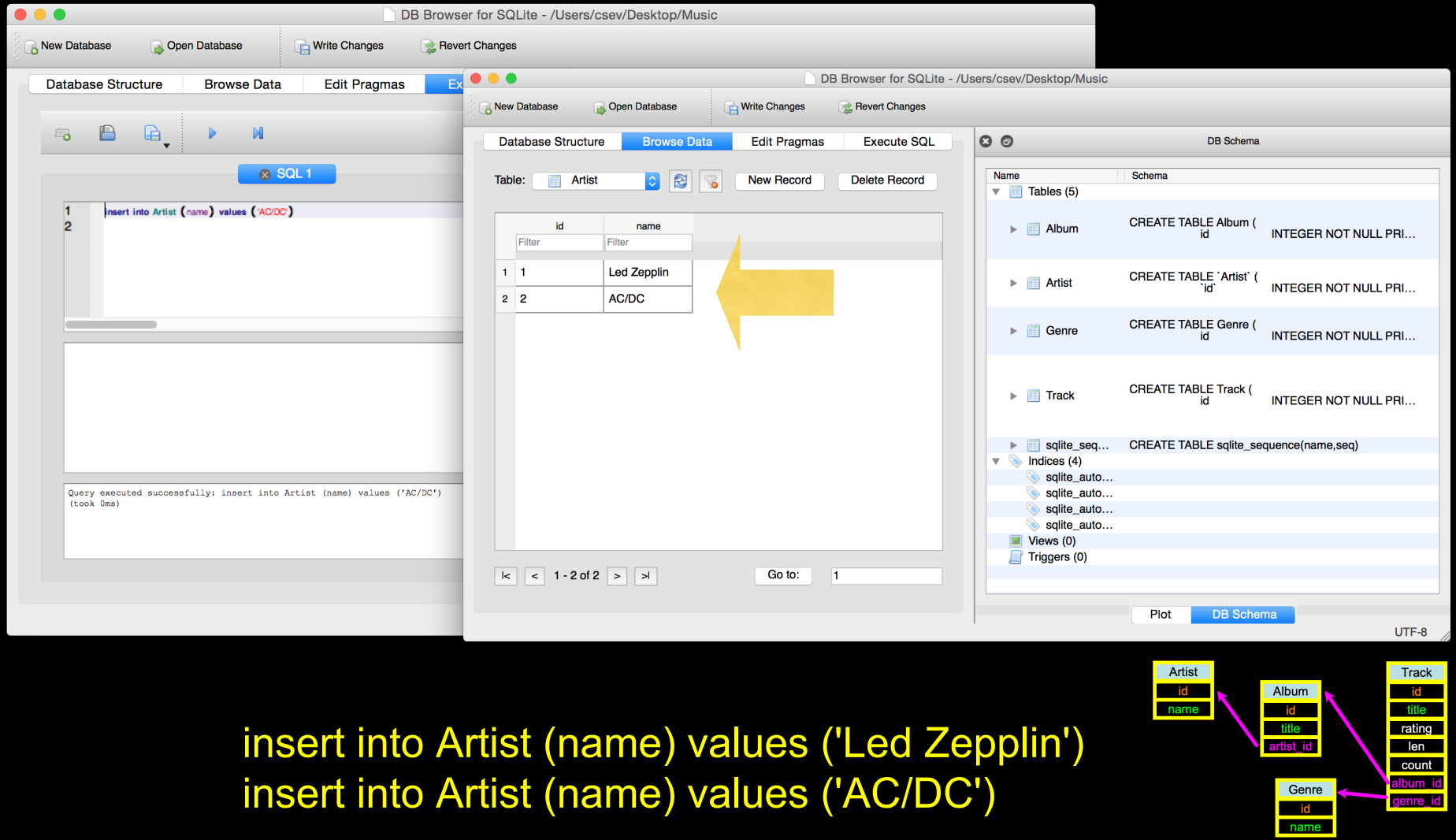
```
CREATE TABLE Album (
        id      INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
        artist_id  INTEGER,
        title   TEXT
)


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
)
```



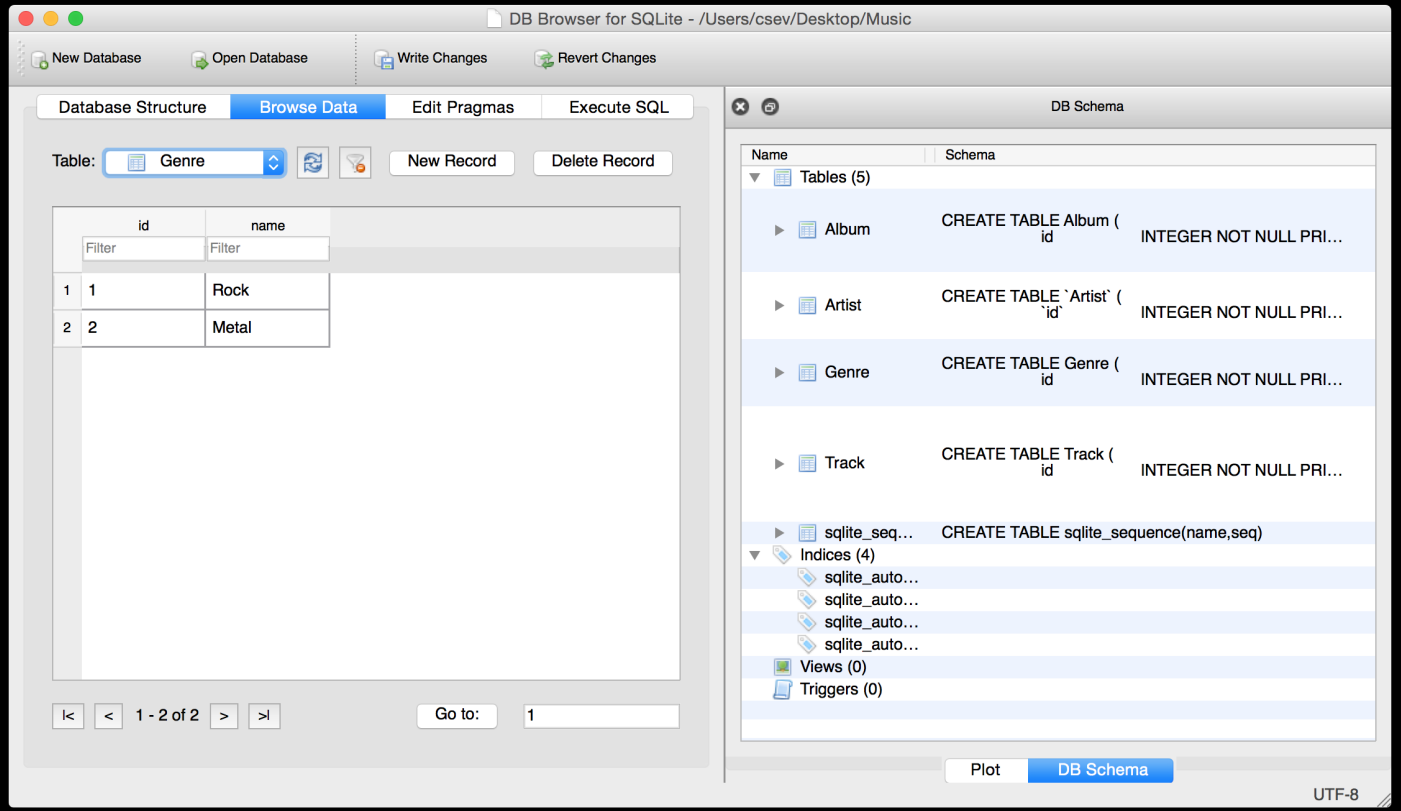

DB Browser for SQLite - /Users/csev/Desktop/Music
New Database
Open Database
Write Changes
Revert Changes
Database Structure
Browse Data
Edit Pragmas
Execute SQL
Create Table
Modify Table
Delete Table
Name
Type
Schema
Tables (5)
Album
CREATE TABLE "Album" (
`id` INTEGER NOT NULL PRIMARY KEY AUTO
`artist_id` INTEGER,
`title` TEXT
)
id INTEGER `id` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE
artist_id INTEGER `artist_id` INTEGER
title TEXT `title` TEXT
Artist
CREATE TABLE `Artist` (
`id` INTEGER NOT NULL PRIMARY KEY AUTO
`name` TEXT
)
id INTEGER `id` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE
name TEXT `name` TEXT
Genre
CREATE TABLE Genre (
id INTEGER NOT NULL PRIMARY KEY AUTO
name TEXT
)
id INTEGER `id` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE
name TEXT `name` TEXT
Track
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)
id INTEGER `id` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE
title TEXT `title` TEXT
album_id INTEGER `album_id` INTEGER
genre_id INTEGER `genre_id` INTEGER
len INTEGER `len` INTEGER
rating INTEGER `rating` INTEGER
count INTEGER `count` INTEGER
DB Schema
Name
Schema
Tables (5)
Album
CREATE TABLE "Album" (
`id` INTEGER NOT NULL P…
id `id` INTEGER NOT NULL PRIMARY KEY AUTOINC…
artist_id `artist_id` INTEGER
title `title` TEXT
Artist
CREATE TABLE `Artist` (
`id` INTEGER NOT NULL P…
id `id` INTEGER NOT NULL PRIMARY KEY AUTOINC…
name `name` TEXT
Genre
CREATE TABLE Genre (
id INTEGER NOT NULL P…
id `id` INTEGER NOT NULL PRIMARY KEY AUTOINC…
name `name` TEXT
Track
CREATE TABLE Track (
id INTEGER NOT NULL P…
id `id` INTEGER NOT NULL PRIMARY KEY AUTOINC…
title `title` TEXT
album_id `album_id` INTEGER
genre_id `genre_id` INTEGER
len `len` INTEGER
rating `rating` INTEGER
count `count` INTEGER
sqlite_seq…
CREATE TABLE sqlite_sequence(name,seq)
Indices (4)
sqlite_auto…
sqlite_auto…
Plot
DB Schema
UTF-8

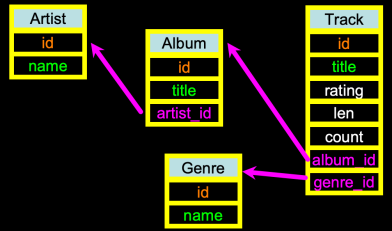


insert into Artist (name) values ('Led Zepplin')
insert into Artist (name) values ('AC/DC')

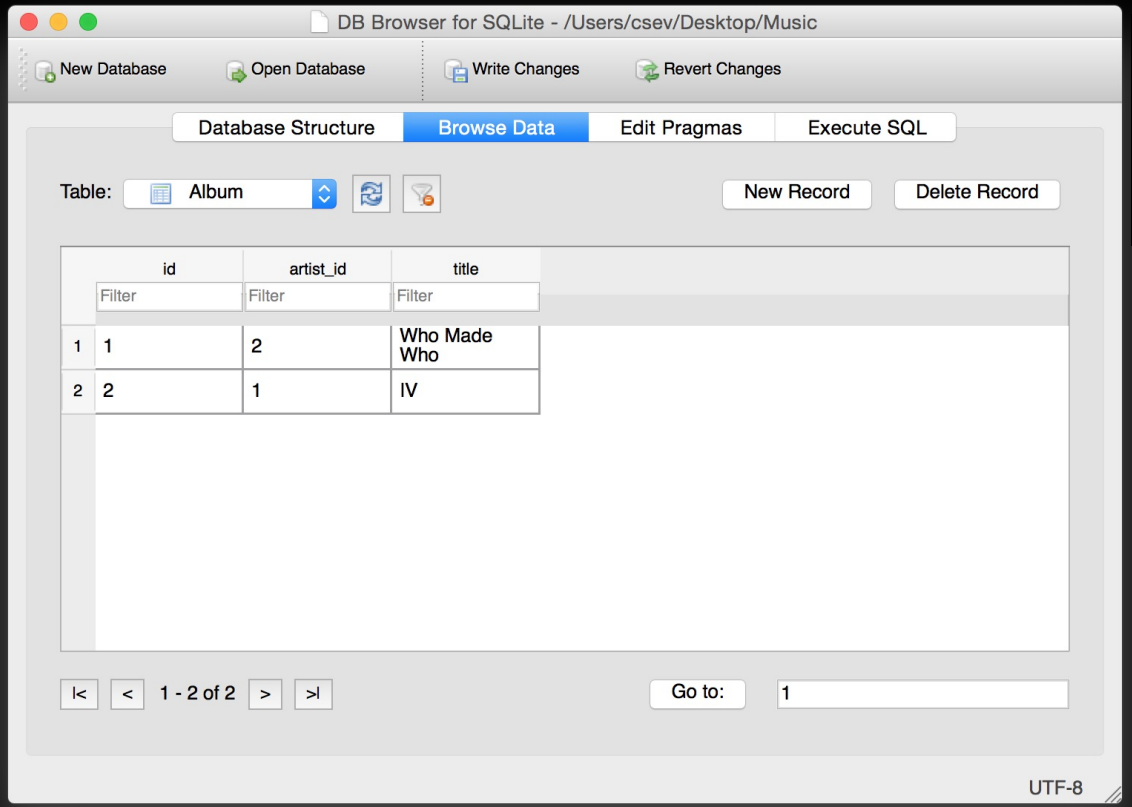


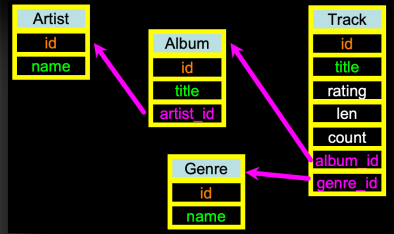
insert into Artist (name) values ('Led Zepplin')
insert into Artist (name) values ('AC/DC')



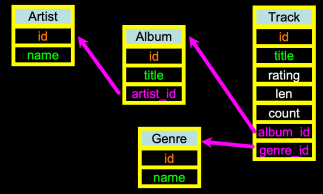


insert into Genre (name) values ('Rock')
insert into Genre (name) values ('Metal')





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
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

Album

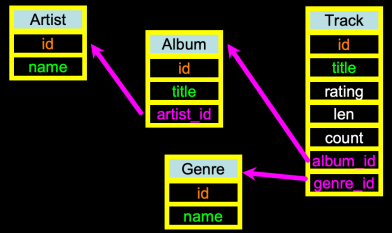
id	artist_id	title
Filter	Filter	Filter
1	2	Who Made Who
2	1	IV

Artist

id	name
Filter	Filter
1	Led Zepplin
2	AC/DC

Genre

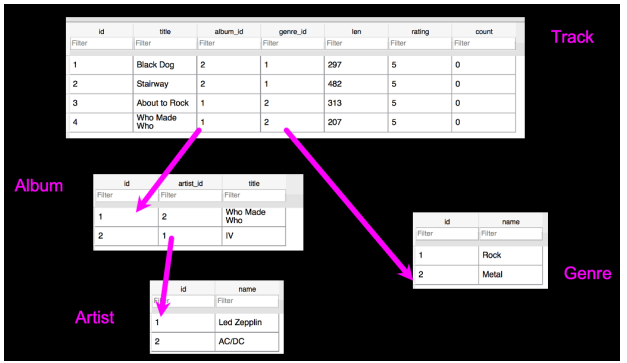
id	name
Filter	Filter
1	Rock
2	Metal



Declaring a Foreign Key

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

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

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

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

Database will enforce that *artist_id* corresponds to a valid *id* in table *Artist*.



Relate Data Across Tables Using Join

http://en.wikipedia.org/wiki/Join_(SQL)

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

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

Album

Artist

id	name
Filter	Filter
1	Led Zepplin
2	AC/DC

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

```
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 (aka the cartesian product)

	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

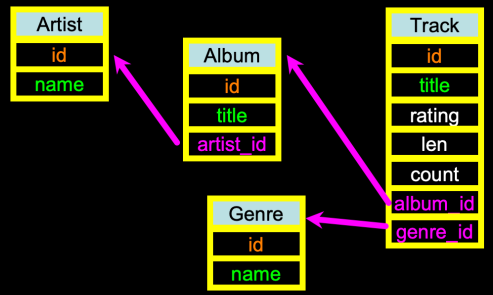
```
select Track.title, Genre.name from Track join Genre on Track.genre_id = Genre.id
```

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

Alternative Notations for Joins

```
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
```

What we want to see

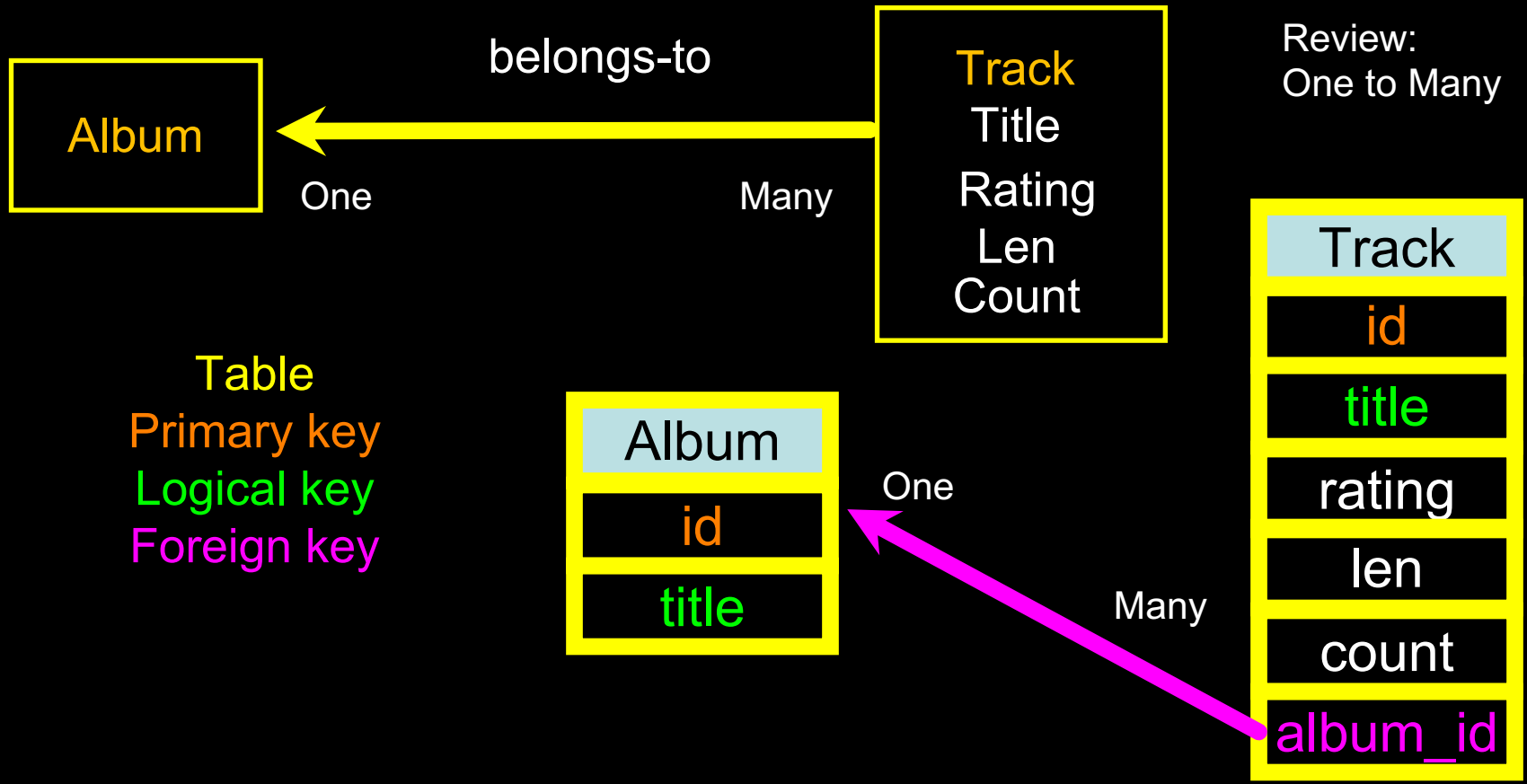
The tables which hold the data

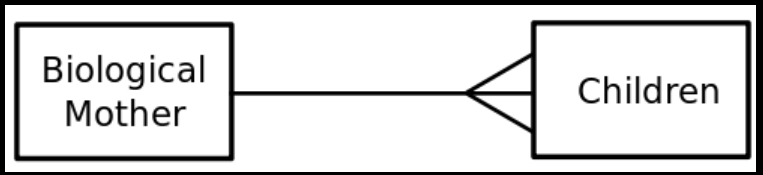
How the tables are linked

```
SELECT Track.title, Artist.name, Album.title, Genre.name
FROM Track, Genre, Album, Artist
WHERE  Track.genre_id = Genre.id and
           Track.album_id = Album.id and
           Album.artist_id = Artist.id
```

Many-To-Many Relationships

https://en.wikipedia.org/wiki/Many-to-many_(data_model)


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
https://en.wikipedia.org/wiki/One-to-many_(data_model)



id	name
Filter	Filter
1	Rock
2	Metal

One

One Many

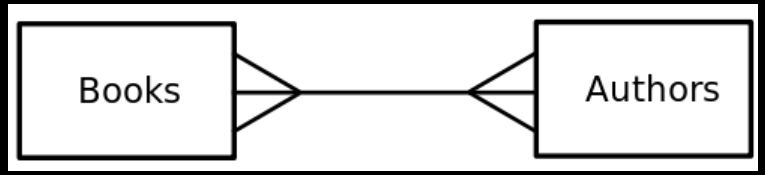
Many

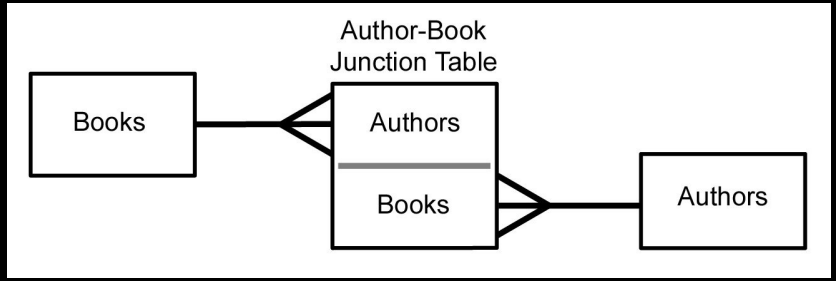
id	title	album_id	genre_id	len	rating	count
Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	Black Dog	2	1	297	5	0
2	Stairway	2	1	482	5	0
3	About to Rock	1	2	313	5	0
4	Who Made Who	1	2	207	5	0

https://en.wikipedia.org/wiki/One-to-many_(data_model)

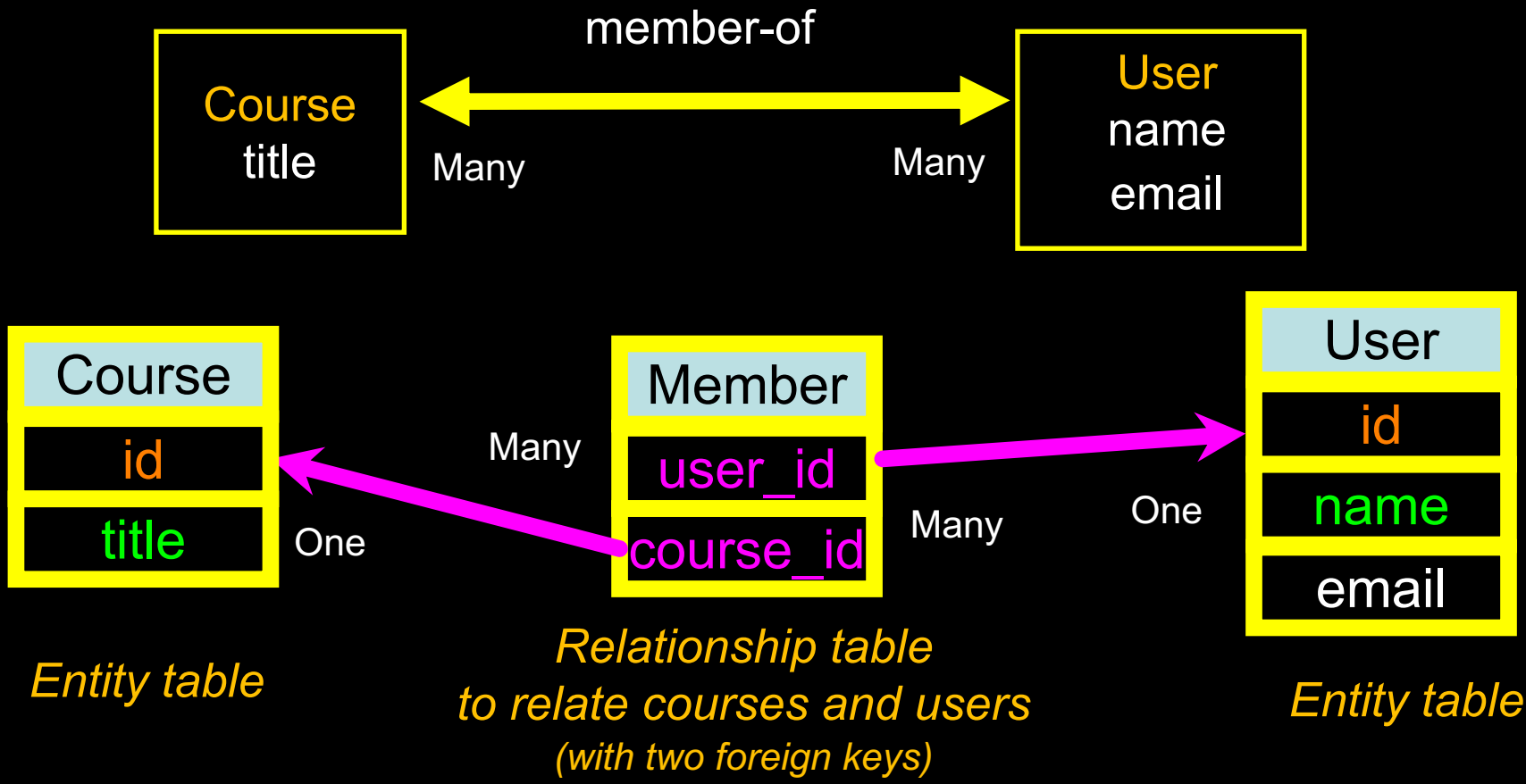
Many to Many

- 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





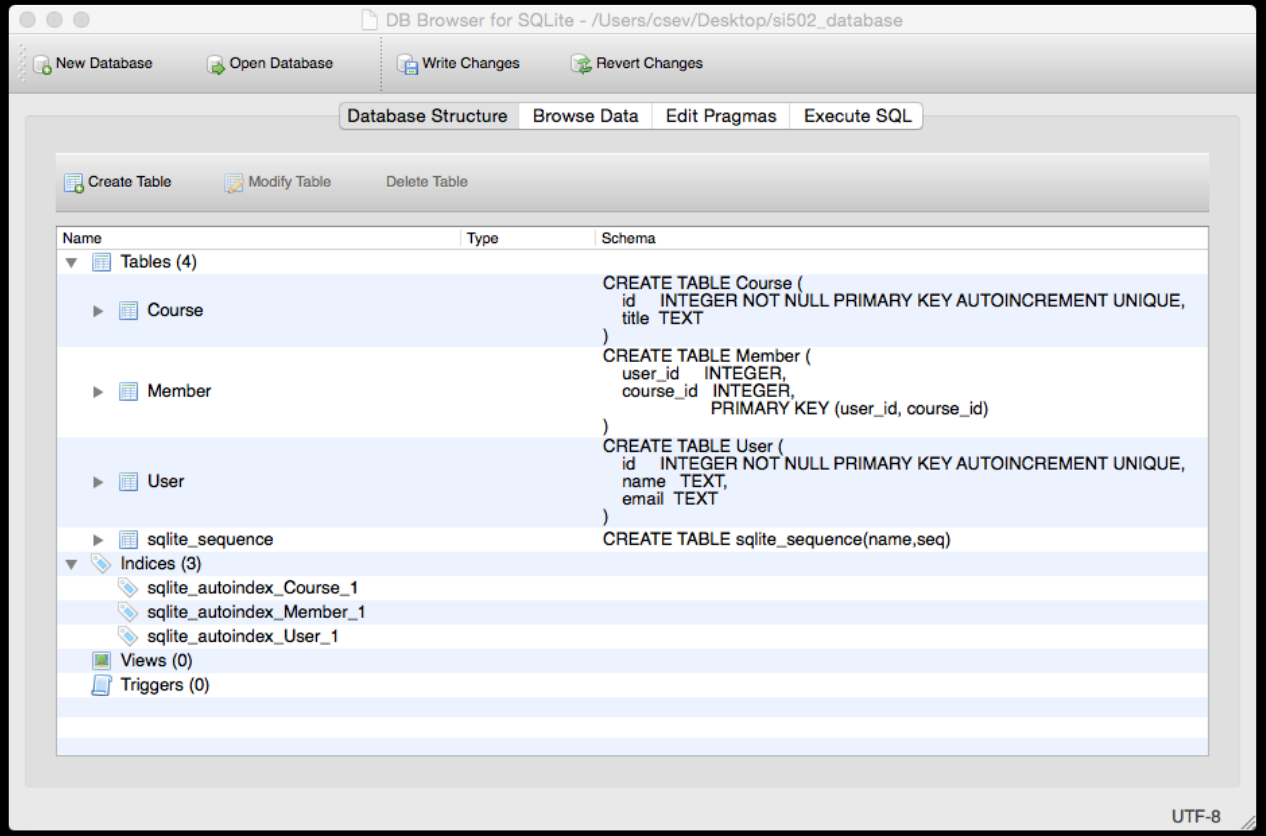
https://en.wikipedia.org/wiki/Many-to-many_(data_model)



https://en.wikipedia.org/wiki/Many-to-many_(data_model)

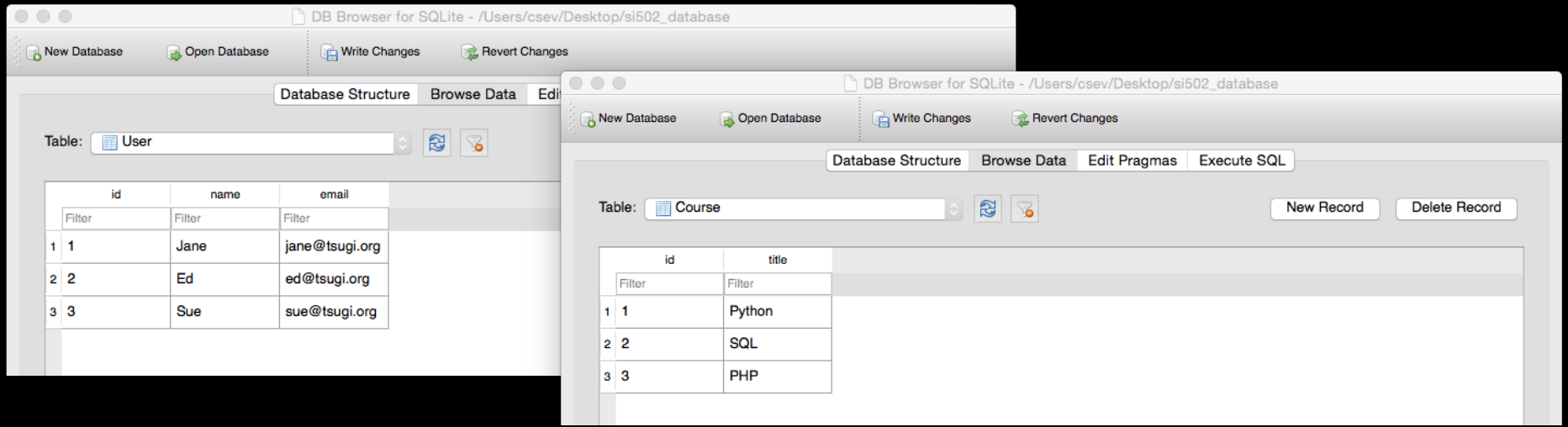
```
CREATE TABLE User (
    id      INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
    name    TEXT UNIQUE,
    email   TEXT
)

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)
)
```

In addition to the foreign keys of the entities that the connection (relationship) table links, it can have additional attributes, like role


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');
```



User

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

Course

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);
```

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

Run Code

Jupyter Notebook:
week11-multi-tables-db.ipynb

Go to SQLite:
Run Agents-customers-orders-queries.sql

SQL Data Definition Commands

COMMAND OR OPTION	DESCRIPTION
CREATE SCHEMA AUTHORIZATION	Creates a database schema
CREATE TABLE	Creates a new table in the user's database schema
NOT NULL	Ensures that a column will not have null values
UNIQUE	Ensures that a column will not have duplicate values
PRIMARY KEY	Defines a primary key for a table
FOREIGN KEY	Defines a foreign key for a table
DEFAULT	Defines a default value for a column (when no value is given)
CHECK	Validates data in an attribute
CREATE INDEX	Creates an index for a table
CREATE VIEW	Creates a dynamic subset of rows and columns from one or more tables (see Chapter 8, Advanced SQL)
ALTER TABLE	Modifies a table's definition (adds, modifies, or deletes attributes or constraints)
CREATE TABLE AS	Creates a new table based on a query in the user's database schema
DROP TABLE	Permanently deletes a table (and its data)
DROP INDEX	Permanently deletes an index
DROP VIEW	Permanently deletes a view

SQL Data Manipulation Commands

COMMAND OR OPTION	DESCRIPTION
INSERT	Inserts row(s) into a table
SELECT	Selects attributes from rows in one or more tables or views
WHERE	Restricts the selection of rows based on a conditional expression
GROUP BY	Groups the selected rows based on one or more attributes
HAVING	Restricts the selection of grouped rows based on a condition
ORDER BY	Orders the selected rows based on one or more attributes
UPDATE	Modifies an attribute's values in one or more table's rows
DELETE	Deletes one or more rows from a table
COMMIT	Permanently saves data changes
ROLLBACK	Restores data to their original values
Comparison operators	
=, <, >, <=, >=, <>	Used in conditional expressions
Logical operators	
AND/OR/NOT	Used in conditional expressions
Special operators	Used in conditional expressions
BETWEEN	Checks whether an attribute value is within a range
IS NULL	Checks whether an attribute value is null
LIKE	Checks whether an attribute value matches a given string pattern
IN	Checks whether an attribute value matches any value within a value list
EXISTS	Checks whether a subquery returns any rows
DISTINCT	Limits values to unique values
Aggregate functions	Used with SELECT to return mathematical summaries on columns
COUNT	Returns the number of rows with non-null values for a given column
MIN	Returns the minimum attribute value found in a given column
MAX	Returns the maximum attribute value found in a given column
SUM	Returns the sum of all values for a given column
AVG	Returns the average of all values for a given column

Relational Modeling 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 (Indexes) improve data access/query performance
- Constraints on data - (cannot be NULL, etc..)
- Transactions - allow SQL operations to be grouped and done as a unit

Indices (Indexes)

Purpose: Speed up data retrieval

Require some time for indexing, more space, but give great query speedups
(I once created a table with 3 billion rows in Postgres, worked great after indexing!)

```
CREATE TABLE contacts (
    first_name text NOT NULL,
    last_name text NOT NULL,
    email text NOT NULL
);

CREATE INDEX idx_contacts_email
    ON contacts (email)
```

Faster due to index:

```
SELECT * FROM contacts WHERE email = 'ulf@isi.edu'
```

Indexes (multi-column)

```
CREATE TABLE contacts (
    first_name text NOT NULL,
    last_name text NOT NULL,
    email text NOT NULL
);

CREATE UNIQUE INDEX idx_contacts_name
    ON contacts (first_name, last_name)
```

Faster due to index:

```
SELECT * FROM contacts
WHERE first_name = 'Ulf' AND last_name = 'Hermjakob'
```

CREATE UNIQUE INDEX means that duplicate index entries are not allowed. Any attempt to insert a duplicate entry will result in an error.

https://www.sqlite.org/lang_createindex.html

Transactions

Purpose: Transactions ensure data integrity
An important strength of a database system

Transactions of SQLite (and other DBMSs) **are ACID compliant**

- **Atomic:** Transaction cannot be broken down into smaller parts.
  All operations in a transaction must be completed (else abort).
- **Consistent:** Change DB from one valid state to another
- **Isolation:** Pending transaction must be invisible to other transactions.
  Data used during transaction cannot be used by another transaction
- **Durable:** A committed transaction must be permanent
  - even in the face of power failures or system crashes

Transaction Example: Bank Transfer

```
BEGIN TRANSACTION;

UPDATE accounts
   SET balance = balance - 1000
 WHERE account_no = 100;

UPDATE accounts
   SET balance = balance + 1000
 WHERE account_no = 200;

INSERT INTO account_changes(account_no,flag,amount,changed_at)
VALUES(100,'-',1000,datetime('now'));

INSERT INTO account_changes(account_no,flag,amount,changed_at)
VALUES(200,'+',1000,datetime('now'));

COMMIT
```

Use **ROLLBACK** command to rollback uncommitted transactions.

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 much be scanned when doing complex operations across large amounts of data
- Database and SQL design is a bit of an art form