Python Objects

Charles Severance

Python for Everybody
www.py4e.com

Warning

- This lecture is very much about definitions and mechanics for objects
- This lecture is a lot more about “how it works” and less about “how you use it”
- You won’t get the entire picture until this is all looked at in the context of a real problem
- So please suspend disbelief and learn technique for the next 40 or so slides..

5. Data Structures

This chapter describes some things you've learned about already in more detail, and adds some new things as well.

5.1. More on Lists

The list data type has some more methods. Here are all of the methods of list objects:

`list.`**`append`**(*x*)

Add an item to the end of the list. Equivalent to `a[len(a):] = [x]`.

`list.`**`extend`**(*L*)

Extend the list by appending all the items in the given list. Equivalent to `a[len(a):] = L`.

`list.`**`insert`**(*i*, *x*)

Insert an item at a given position. The first argument is the index of the element before which to insert, so `a.insert(0, x)` inserts at the front of the list, and `a.insert(len(a), x)` is equivalent to `a.append(x)`.

`list.`**`remove`**(*x*)

Remove the first item from the list whose value is *x*. It is an error if there is no such item.

`list.`**`pop`**([*i*])

Remove the item at the given position in the list, and return it. If no index is specified, `a.pop()` removes and returns the last item in the list. (The square brackets around the *i* in the method signature denote that the parameter is optional, not that you should type square brackets at that position. You will see this notation frequently in the Python Library Reference.)

https://docs.python.org/3/tutorial/datastructures.html

12.6. sqlite3 — DB-API 2.0 interface for SQLite databases

Source code: Lib/sqlite3/

SQLite is a C library that provides a lightweight disk-based database that doesn't require a separate server process and allows accessing the database using a nonstandard variant of the SQL query language. Some applications can use SQLite for internal data storage. It's also possible to prototype an application using SQLite and then port the code to a larger database such as PostgreSQL or Oracle.

The sqlite3 module was written by Gerhard Häring. It provides a SQL interface compliant with the DB-API 2.0 specification described by PEP 249.

To use the module, you must first create a `Connection` object that represents the database. Here the data will be stored in the `example.db` file:

```
import sqlite3
conn = sqlite3.connect('example.db')
```

You can also supply the special name `:memory:` to create a database in RAM.

Once you have a `Connection`, you can create a `Cursor` object and call its `execute()` method to perform SQL commands:

```
c = conn.cursor()

# Create table
c.execute('''CREATE TABLE stocks
             (date text, trans text, symbol text, qty real, price real)''')
```

https://docs.python.org/3/library/sqlite3.html

Lets Start with Programs

```
inp = input('Europe floor?')
usf = int(inp) + 1
print('US floor', usf)
```

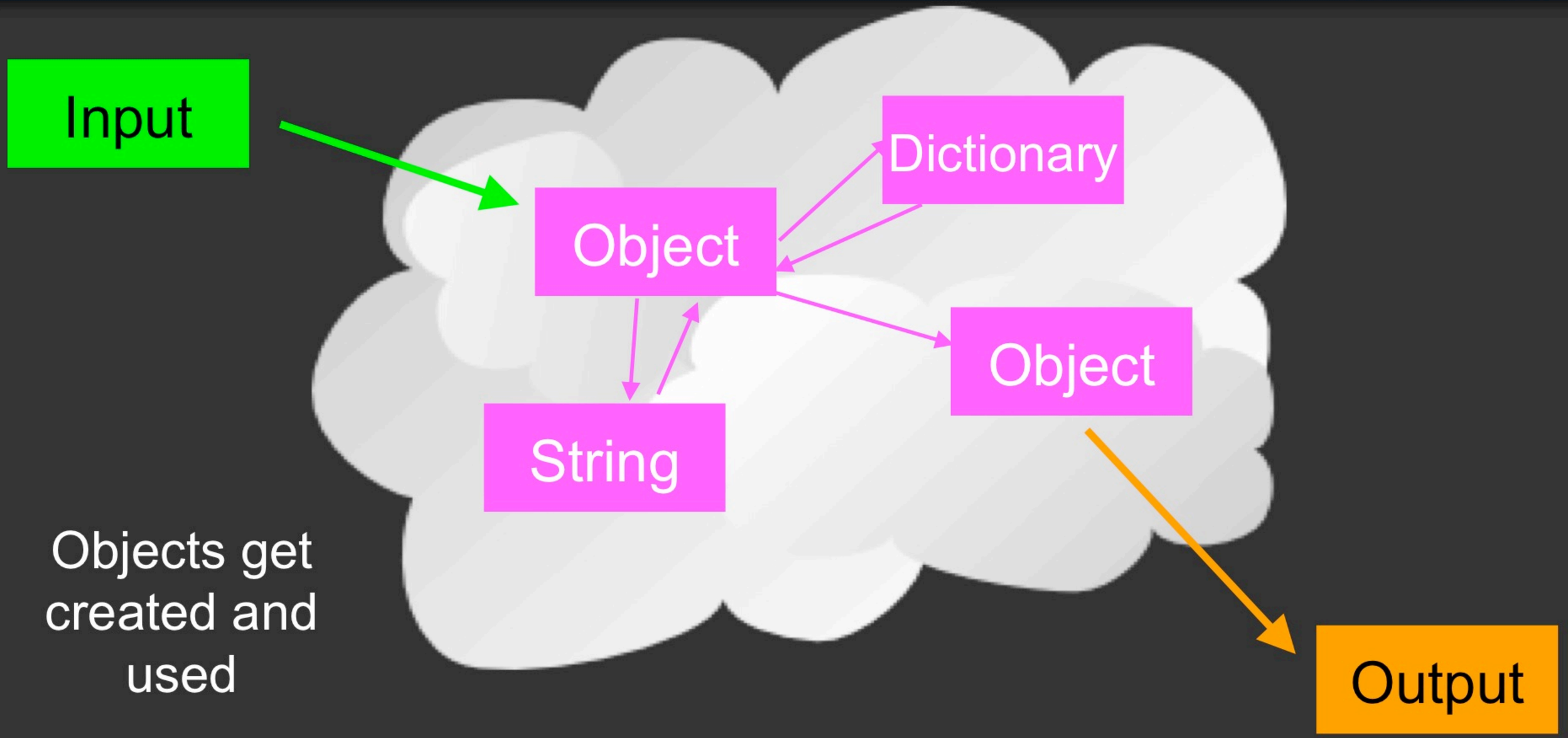
Europe floor? 0
US floor 1

Object Oriented

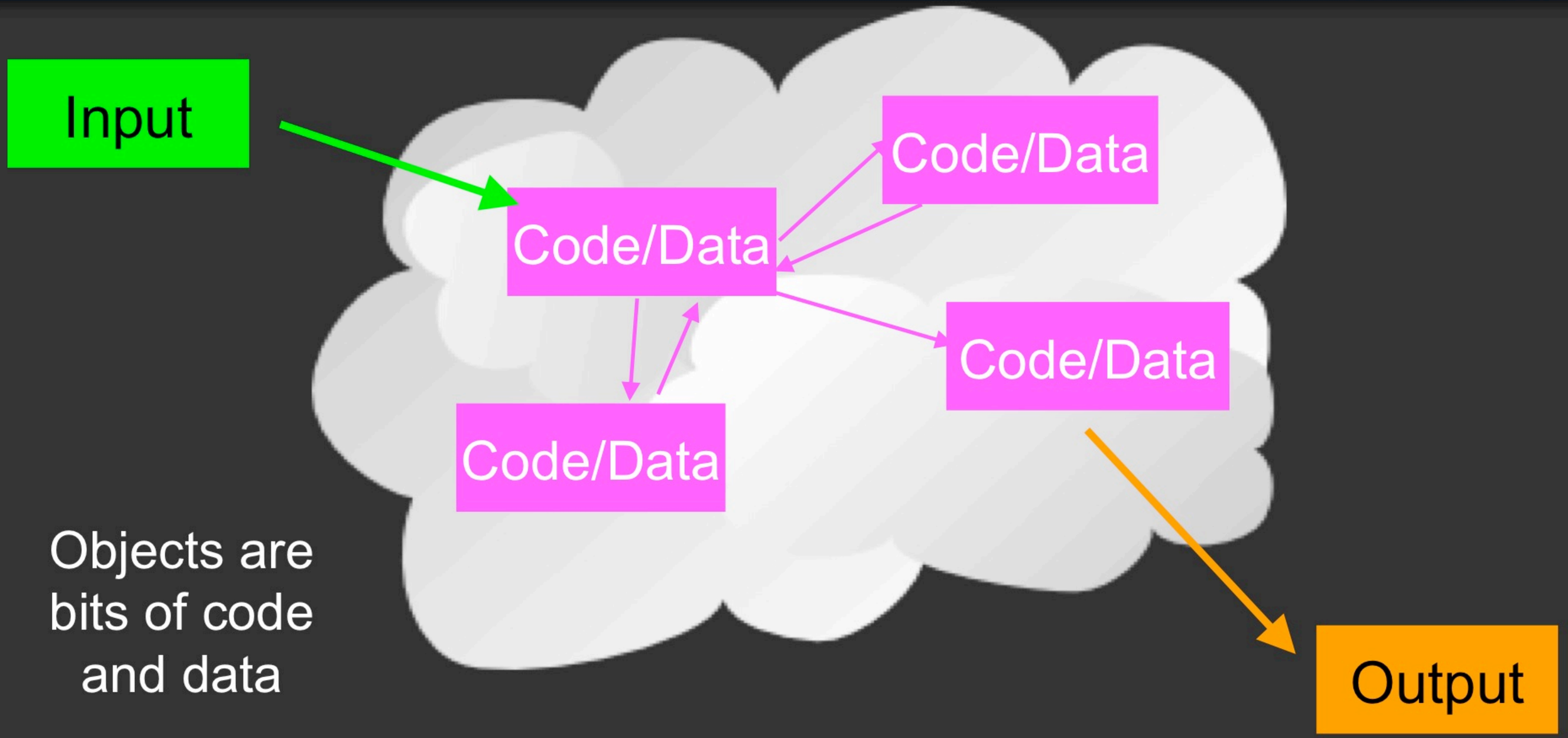
- A program is made up of many cooperating objects
- Instead of being the “whole program” - each object is a little “island” within the program and cooperatively working with other objects.
- A program is made up of one or more objects working together - objects make use of each other’s capabilities

Object

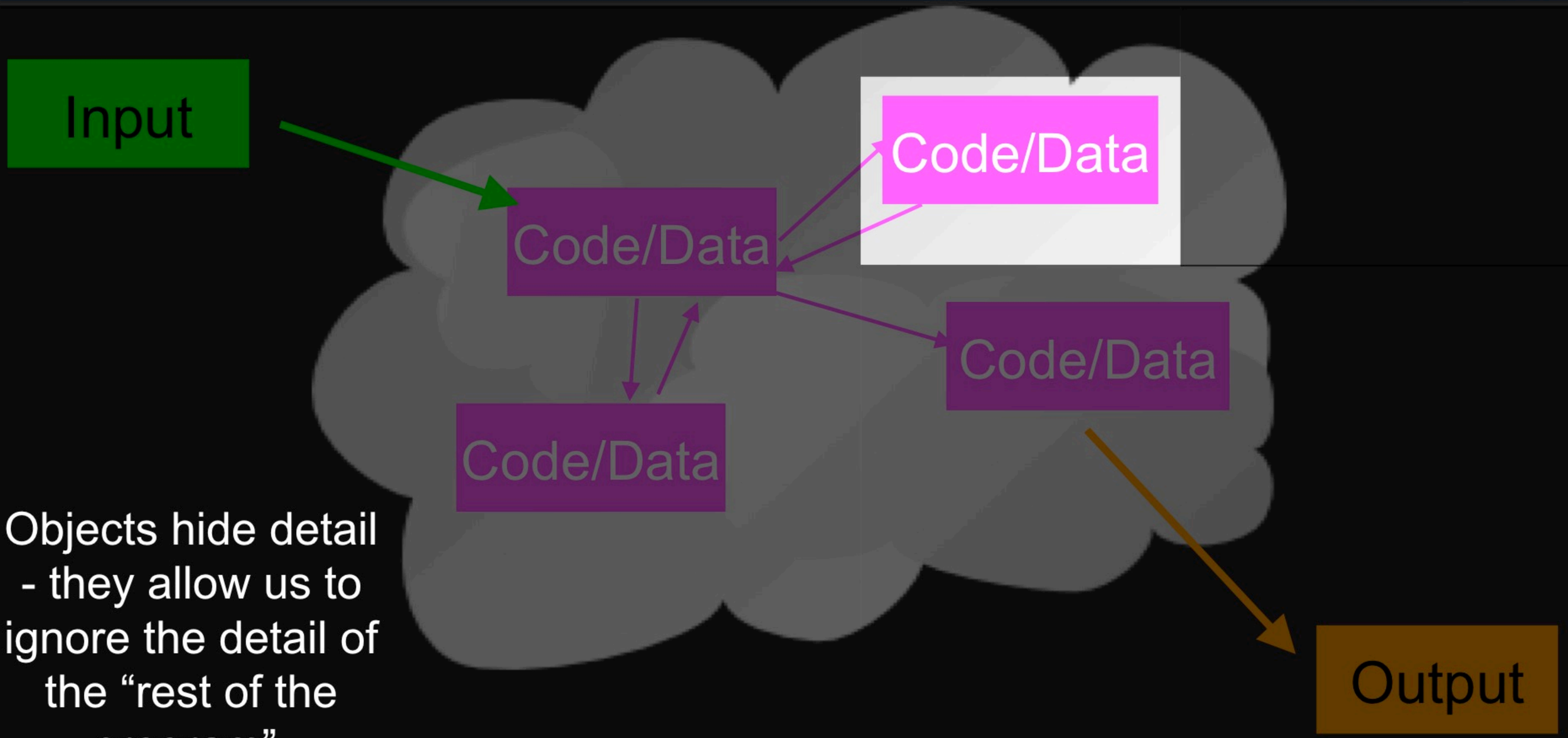
- An Object is a bit of self-contained Code and Data
- A key aspect of the Object approach is to break the problem into smaller understandable parts (divide and conquer)
- Objects have boundaries that allow us to ignore un-needed detail
- We have been using objects all along: String Objects, Integer Objects, Dictionary Objects, List Objects...



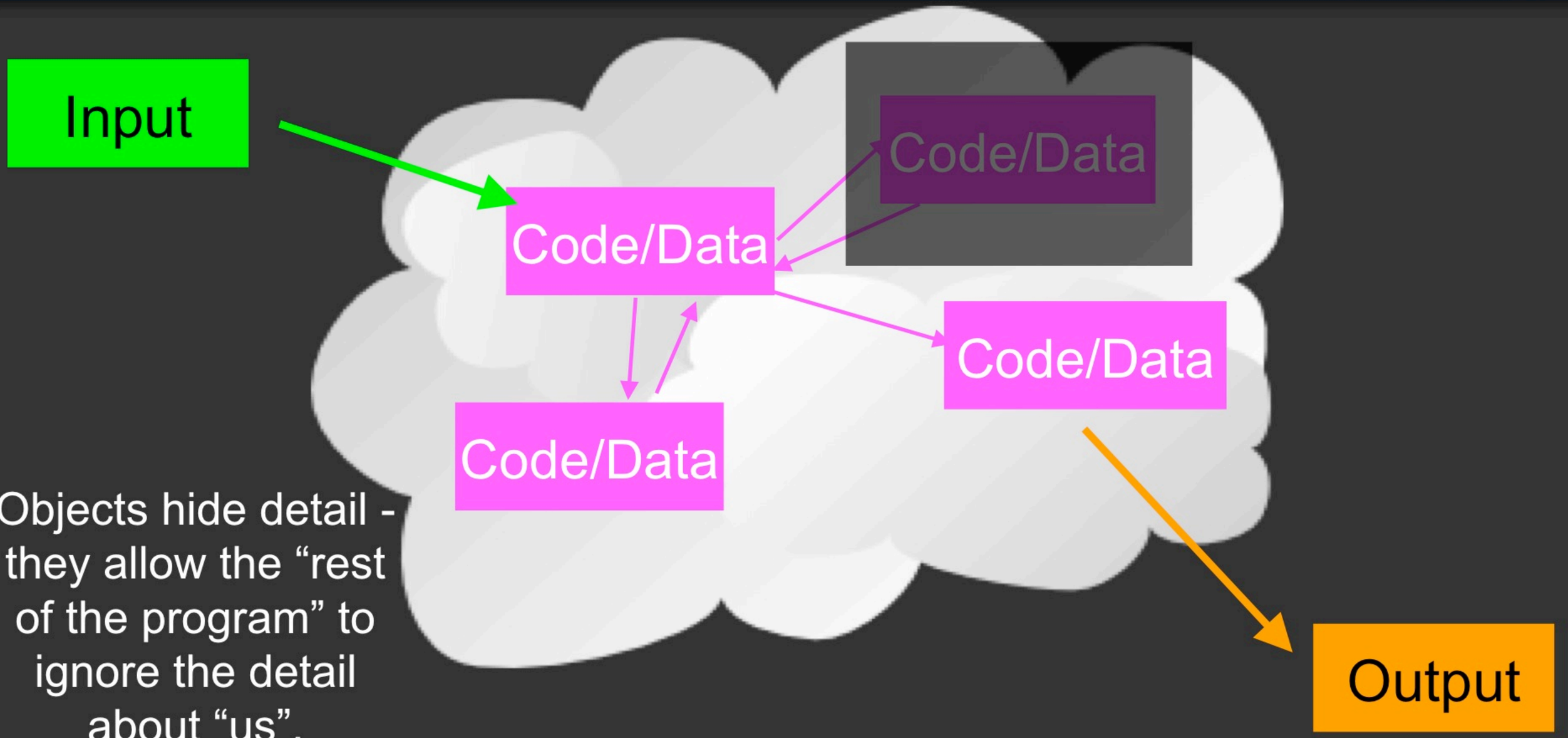
Objects get created and used



Objects are bits of code and data



Objects hide detail - they allow us to ignore the detail of the “rest of the program”.



Objects hide detail - they allow the "rest of the program" to ignore the detail about "us".

Definitions

Class - a template

Method or Message - A defined capability of a class

Field or attribute- A bit of data in a class

Object or Instance - A particular instance of a class

Terminology: Class

Defines the abstract characteristics of a thing (object), including the thing's characteristics (its attributes, fields or properties) and the thing's behaviors (the things it can do, or methods, operations or features). One might say that a class is a blueprint or factory that describes the nature of something. For example, the class Dog would consist of traits shared by all dogs, such as breed and fur color (characteristics), and the ability to bark and sit (behaviors).

http://en.wikipedia.org/wiki/Object-oriented_programming

Terminology: Instance

One can have an instance of a class or a particular object. The instance is the actual object created at runtime. In programmer jargon, the Lassie object is an instance of the Dog class. The set of values of the attributes of a particular object is called its state. The object consists of state and the behavior that's defined in the object's class.

Object and Instance are often used interchangeably.

http://en.wikipedia.org/wiki/Object-oriented_programming

Terminology: Method

An object's abilities. In language, methods are verbs. Lassie, being a Dog, has the ability to bark. So bark() is one of Lassie's methods. She may have other methods as well, for example sit() or eat() or walk() or save_timmy(). Within the program, using a method usually affects only one particular object; all Dogs can bark, but you need only one particular dog to do the barking

Method and Message are often used interchangeably.

http://en.wikipedia.org/wiki/Object-oriented_programming

Some Python Objects

```
>>> x = 'abc'
>>> type(x)
<class 'str'>
>>> type(2.5)
<class 'float'>
>>> type(2)
<class 'int'>
>>> y = list()
>>> type(y)
<class 'list'>
>>> z = dict()
>>> type(z)
<class 'dict'>
```

```
>>> dir(x)
[ … 'capitalize', 'casefold', 'center', 'count',
'encode', 'endswith', 'expandtabs', 'find',
'format', … 'lower', 'lstrip', 'maketrans',
'partition', 'replace', 'rfind', 'rindex', 'rjust',
'rpartition', 'rsplit', 'rstrip', 'split',
'splitlines', 'startswith', 'strip', 'swapcase',
'title', 'translate', 'upper', 'zfill']
>>> dir(y)
[… 'append', 'clear', 'copy', 'count', 'extend',
'index', 'insert', 'pop', 'remove', 'reverse',
'sort']
>>> dir(z)
[…, 'clear', 'copy', 'fromkeys', 'get', 'items',
'keys', 'pop', 'popitem', 'setdefault', 'update',
'values']
```

A Sample Class

Acknowledgements / Contributions

Initial Development: Charles Severance, University of Michigan School of Information

… Insert new Contributors here

...

Additional Source Information

A Sample Class

class is a reserved word.

Each PartyAnimal object has a bit of code.

Tell the an object to run the party() code within it.

```
class PartyAnimal:
  x = 0

  def party(self) :
    self.x = self.x + 1
    print("So far",self.x)

an = PartyAnimal()

an.party()
an.party()
an.party()
```

This is the template for making PartyAnimal objects.

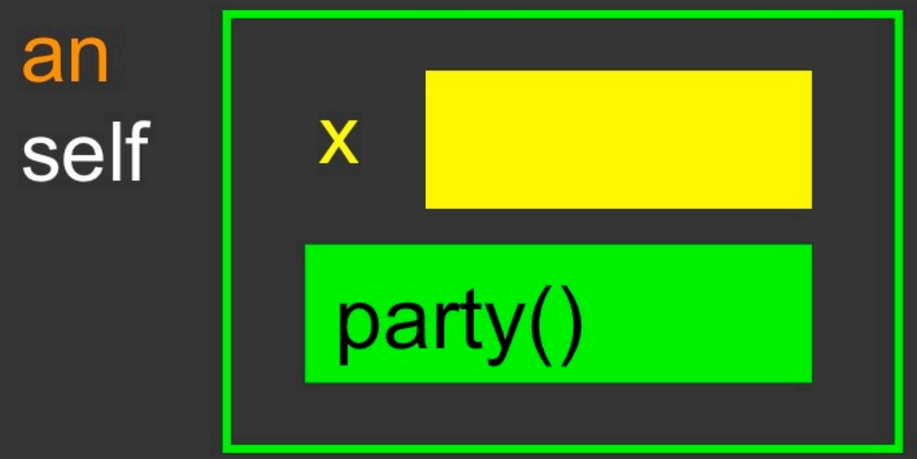
Each PartyAnimal object has a bit of data.

Construct a PartyAnimal object and store in an variable

PartyAnimal.party(an)

```
class PartyAnimal:
  x = 0

  def party(self) :
    self.x = self.x + 1
    print("So far",self.x)

an = PartyAnimal()

an.party()
an.party()
an.party()
```

```
$ python party1.py
So far 1
So far 2
So far 3
```



Playing with dir() and type()

- The dir() command lists capabilities
- Ignore the ones with underscores - these are used by Python itself
- The rest are real operations that the object can perform
- It is like type() - it tells us something *about* a variable

```
>>> y = list()
>>> type(y)
<class 'list'>
>>> dir(x)
['__add__', '__class__',
'__contains__', '__delattr__',
'__delitem__', '__delslice__',
'__doc__', … '__setitem__',
'__setslice__', '__str__',
'append', 'clear', 'copy', 'count',
'extend', 'index', 'insert', 'pop',
'remove', 'reverse', 'sort']
>>>
```

Try dir() with a String

```
>>> x = 'Hello there'
>>> dir(x)
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__',
'__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__',
'__getnewargs__', '__gt__', '__hash__', '__init__', '__iter__', '__le__',
'__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__',
'__sizeof__', '__str__', '__subclasshook__', 'capitalize', 'casefold',
'center', 'count', 'encode', 'endswith', 'expandtabs', 'find', 'format',
'format_map', 'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit',
'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle',
'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition',
'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip',
'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title',
'translate', 'upper', 'zfill']
```

```
class PartyAnimal:
    x = 0

    def party(self) :
      self.x = self.x + 1
      print("So far",self.x)

an = PartyAnimal()

print("Type", type(an))
print("Dir ", dir(an))
```

We can use dir() to find the “capabilities” of our newly created class.

```
$ python party3.py
Type <class '__main__.PartyAnimal'>
Dir  ['__class__', ... 'party', 'x']
```

Object Lifecycle

http://en.wikipedia.org/wiki/Constructor_(computer_science)

Acknowledgements / Contributions

Initial Development: Charles Severance, University of Michigan School of Information

… Insert new Contributors here

...

Additional Source Information

Object Lifecycle

http://en.wikipedia.org/wiki/Constructor_(computer_science)

Object Lifecycle

- Objects are created, used and discarded
- We have special blocks of code (methods) that get called
    - At the moment of creation (constructor)
    - At the moment of destruction (destructor)
- Constructors are used a lot
- Destructors are seldom used

Constructor

- The primary purpose of the constructor is to set up some instance variables to have the proper initial values when the object is created

```
class PartyAnimal:
    x = 0

    def __init__(self):
      print('I am constructed')

    def party(self) :
      self.x = self.x + 1
      print('So far',self.x)

    def __del__(self):
      print('I am destructed', self.x)

an = PartyAnimal()
an.party()
an.party()
an = 42
print('an contains',an)
```

```
$ python party4.py
I am constructed
So far 1
So far 2
I am destructed 2
an contains 42
```

The constructor and destructor are optional. The constructor is typically used to set up variables. The destructor is seldom used.

Constructor

- In object oriented programming, a constructor in a class is a special block of statements called when an object is created

http://en.wikipedia.org/wiki/Constructor_(computer_science)

Many Instances

- We can create lots of objects - the class is the template for the object
- We can store each distinct object in its own variable
- We call this having multiple instances of the same class
- Each instance has its own copy of the instance variables

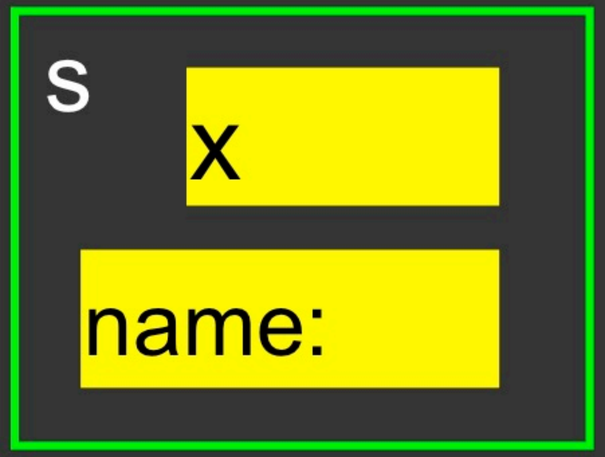
```
class PartyAnimal:
    x = 0
    name = ""
    def __init__(self, z):
      self.name = z
      print(self.name,"constructed")

    def party(self) :
      self.x = self.x + 1
      print(self.name,"party count",self.x)

s = PartyAnimal("Sally")
s.party()

j = PartyAnimal("Jim")
j.party()
s.party()
```

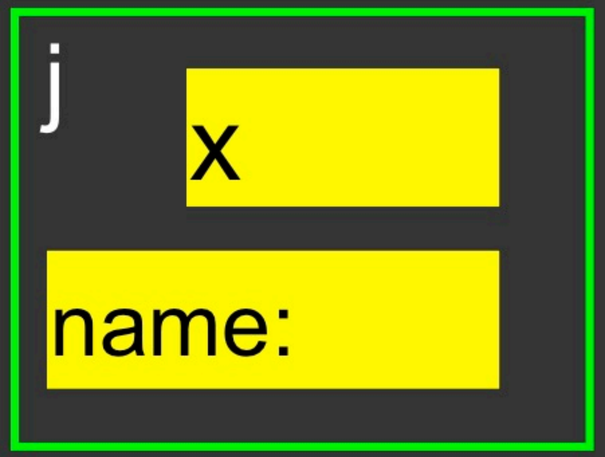
Constructors can have additional parameters. These can be used to set up instance variables for the particular instance of the class (i.e., for the particular object).

party5.py

```
class PartyAnimal:
    x = 0
    name = ""
    def __init__(self, z):
      self.name = z
      print(self.name,"constructed")

    def party(self) :
      self.x = self.x + 1
      print(self.name,"party count",self.x)

s = PartyAnimal("Sally")
s.party()

j = PartyAnimal("Jim")
j.party()
s.party()
```

We have two independent instances.





Inheritance

http://www.ibiblio.org/g2swap/byteofpython/read/inheritance.html

Acknowledgements / Contributions

Initial Development: Charles Severance, University of Michigan School of Information

… Insert new Contributors here

...

Additional Source Information

Inheritance

- When we make a new class - we can reuse an existing class and inherit all the capabilities of an existing class and then add our own little bit to make our new class
- Another form of store and reuse
- Write once - reuse many times
- The new class (child) has all the capabilities of the old class (parent) - and then some more

Terminology: Inheritance

'Subclasses' are more specialized versions of a class, which inherit attributes and behaviors from their parent classes, and can introduce their own.

http://en.wikipedia.org/wiki/Object-oriented_programming

```
class PartyAnimal:
    x = 0
    name = ""
    def __init__(self, nam):
      self.name = nam
      print(self.name,"constructed")

    def party(self) :
      self.x = self.x + 1
      print(self.name,"party count",self.x)

class FootballFan(PartyAnimal):
    points = 0
    def touchdown(self):
       self.points = self.points + 7
       self.party()
       print(self.name,"points",self.points)
```

```
s = PartyAnimal("Sally")
s.party()

j = FootballFan("Jim")
j.party()
j.touchdown()
```

FootballFan is a class which extends PartyAnimal. It has all the capabilities of PartyAnimal and more.

```
class PartyAnimal:
    x = 0
    name = ""
    def __init__(self, nam):
      self.name = nam
      print(self.name,"constructed")

    def party(self) :
      self.x = self.x + 1
      print(self.name,"party count",self.x)

class FootballFan(PartyAnimal):
    points = 0
    def touchdown(self):
       self.points = self.points + 7
       self.party()
       print(self.name,"points",self.points)
```

```
s = PartyAnimal("Sally")
s.party()

j = FootballFan("Jim")
j.party()
j.touchdown()
```

```
class PartyAnimal:
    x = 0
    name = ""
    def __init__(self, nam):
      self.name = nam
      print(self.name,"constructed")

    def party(self) :
      self.x = self.x + 1
      print(self.name,"party count",self.x)

class FootballFan(PartyAnimal):
    points = 0
    def touchdown(self):
       self.points = self.points + 7
       self.party()
       print(self.name,"points",self.points)
```

```
s = PartyAnimal("Sally")
s.party()

j = FootballFan("Jim")
j.party()
j.touchdown()
```

Definitions

Class - a template

Attribute – A variable within a class

Method - A function within a class

Object - A particular instance of a class

Constructor – Code that runs when an object is created

Inheritance - The ability to extend a class to make a new class.

Summary

- Object Oriented programming is a very structured approach to code reuse.
- We can group data and functionality together and create many independent instances of a class

Acknowledgements / Contributions

Initial Development: Charles Severance, University of Michigan School of Information

… Insert new Contributors here

...

Additional Source Information

Relational Databases and SQLite

Charles Severance

Python for Informatics: Exploring Information
www.pythonlearn.com

SQLite Browser

http://sqlitebrowser.org/

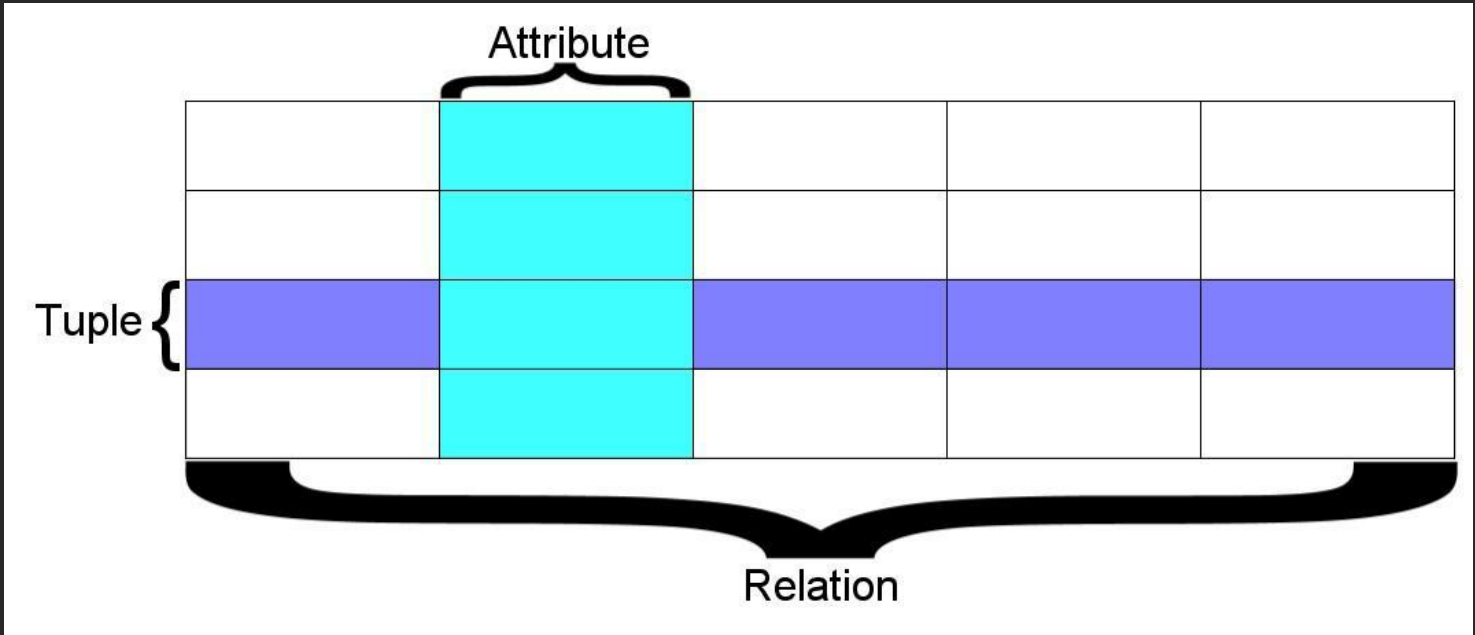
Relational Databases

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.

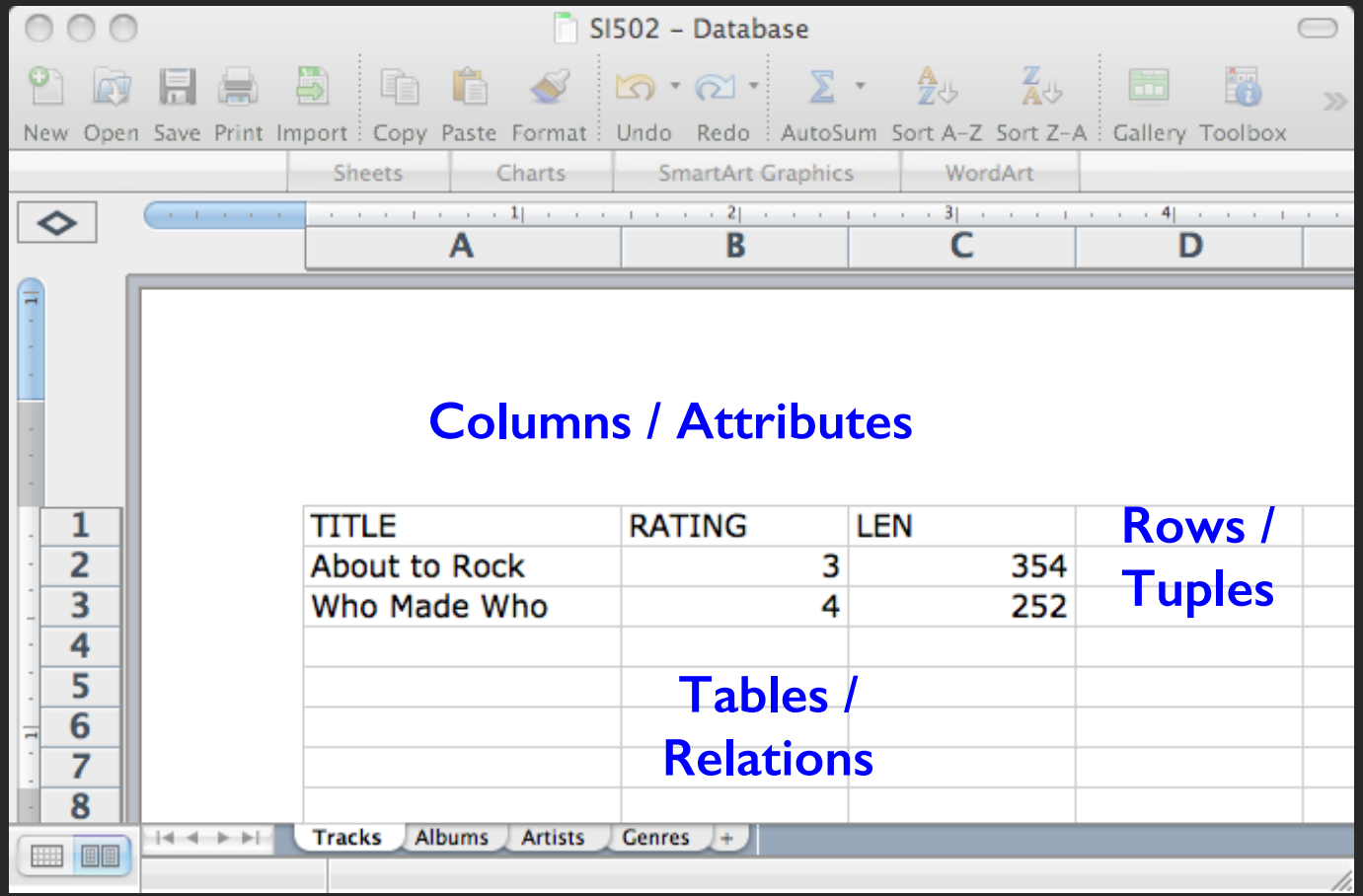
http://en.wikipedia.org/wiki/Relational_database

Terminology

- Database - contains many tables
- Relation (or table) - contains tuples and attributes
- Tuple (or row) - a set of fields that generally represents an "object" 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



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


SI502 – Database
New Open Save Print Import Copy Paste Format Undo Redo AutoSum Sort A-Z Sort Z-A Gallery Toolbox
Sheets Charts SmartArt Graphics WordArt
A B C D
1 2 3 4 5 6 7 8
Columns / Attributes
TITLE RATING LEN
About to Rock 3 354
Who Made Who 4 252
Rows / Tuples
Tables / Relations
Tracks Albums Artists Genres

SQL

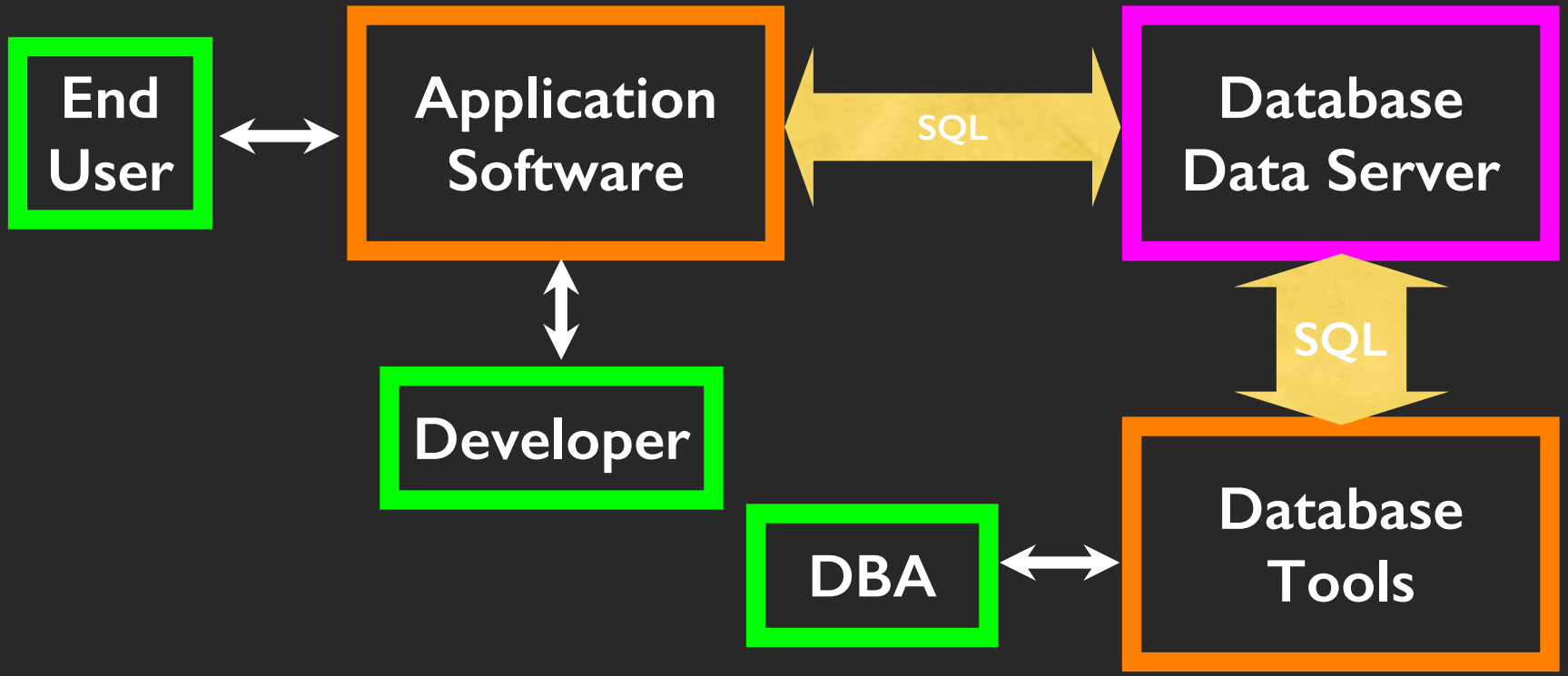
- **Structured Query Language** is the language we use to issue commands to the database
  - Create a table
  - Retrieve some data
  - Insert data
  - Delete data

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

Two Roles in Large Projects

- 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"

Large Project Structure

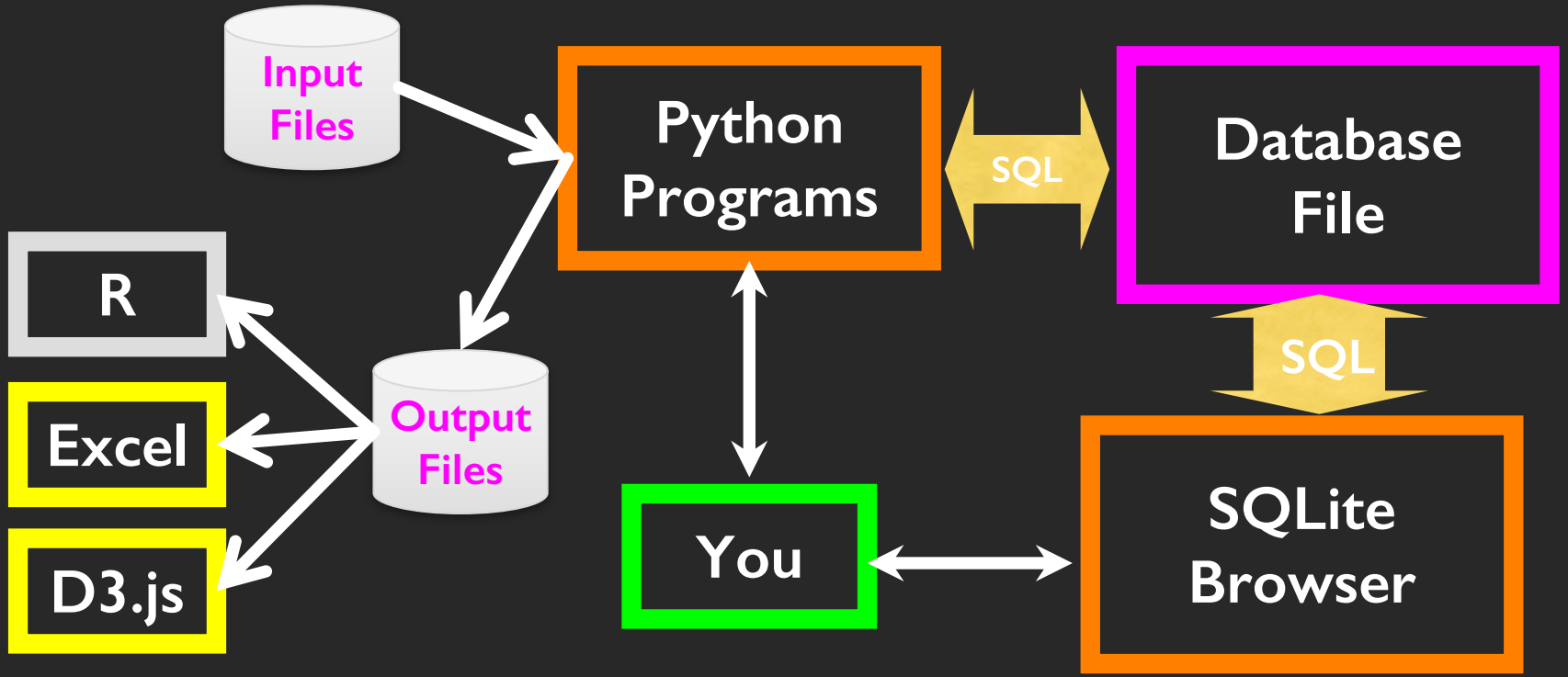


Database Administrator (dba)

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

Data Analysis Structure



Database Model

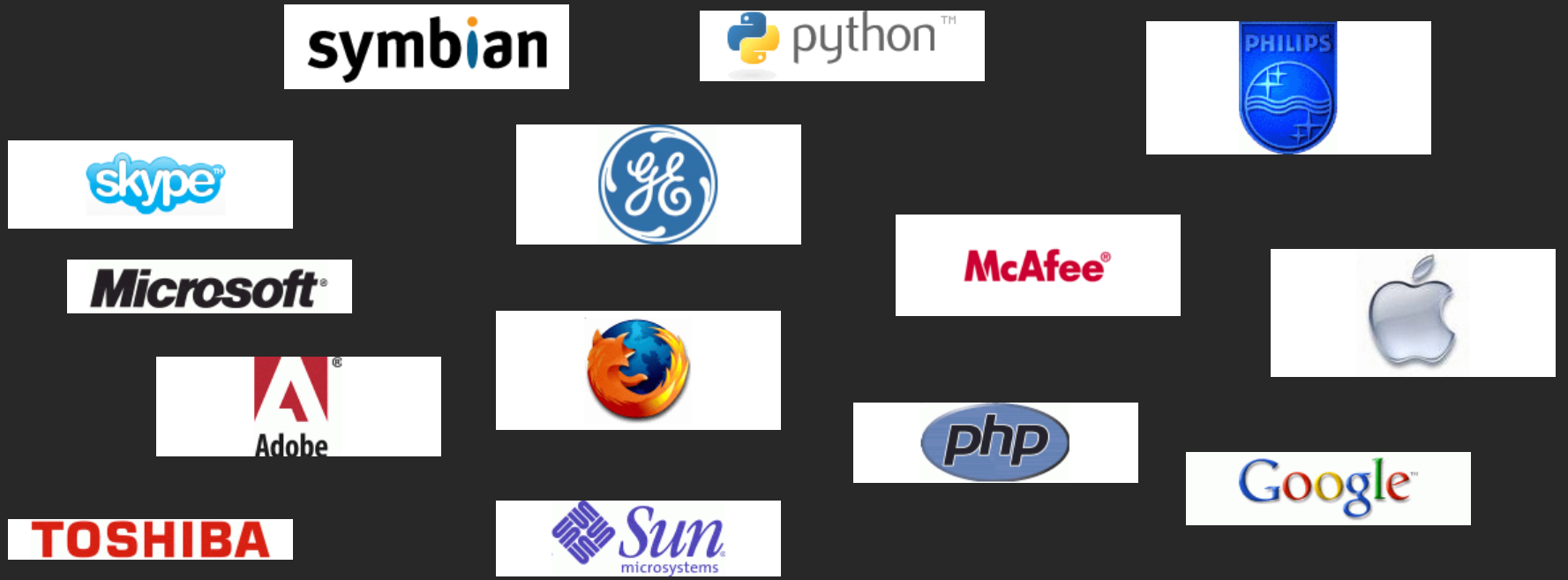
A database model or database schema is the structure or format of a database, described in a formal language supported by the database management system, In other words, a "database model" is the application of a data model when used in conjunction with a database management system.

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

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, Postgress, ...

SQLite is in lots of software...



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

Writing SQL – Making a Database

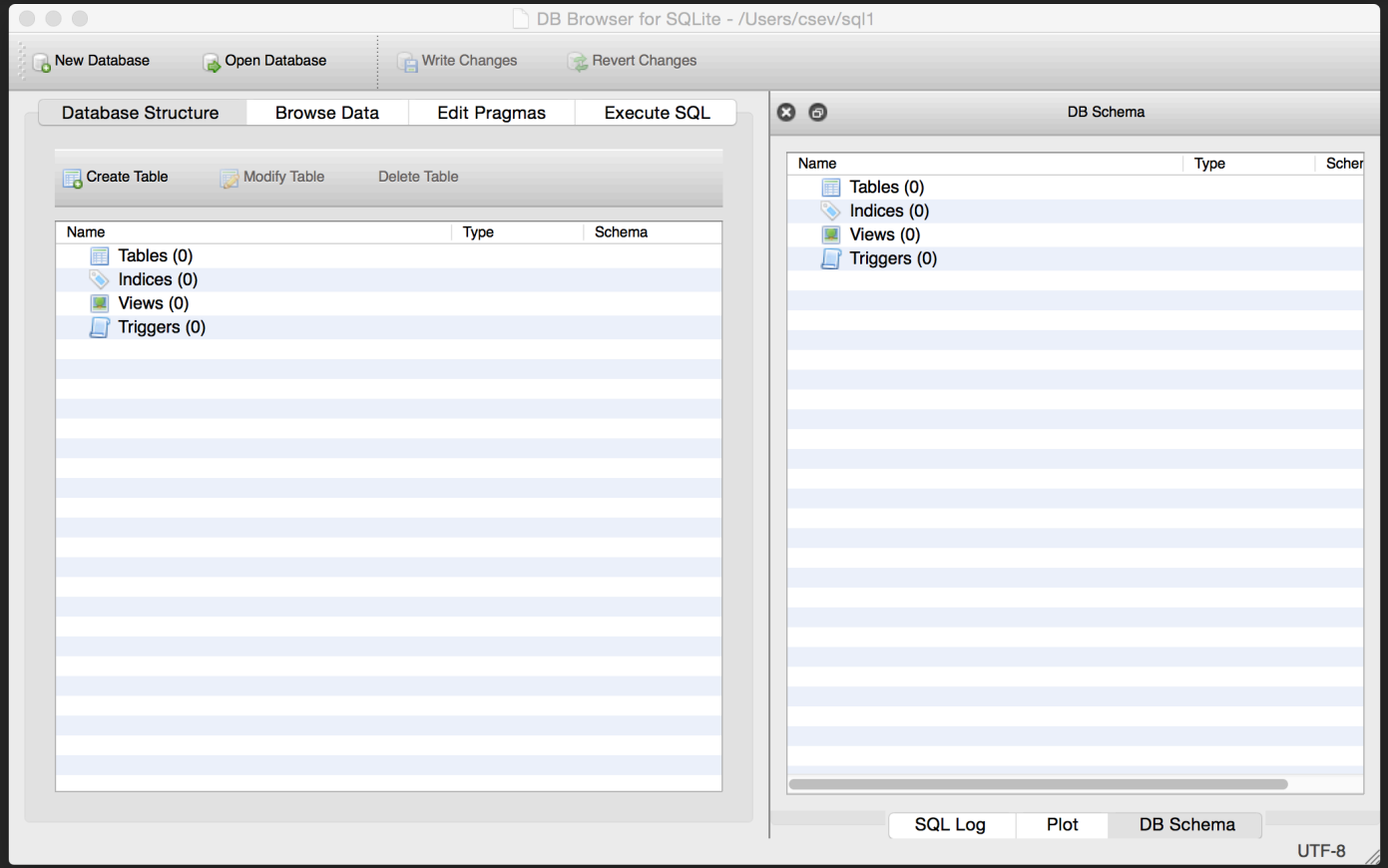
SQLite Browser

- 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/

There is also a Firefox plugin to manipulate SQLite database

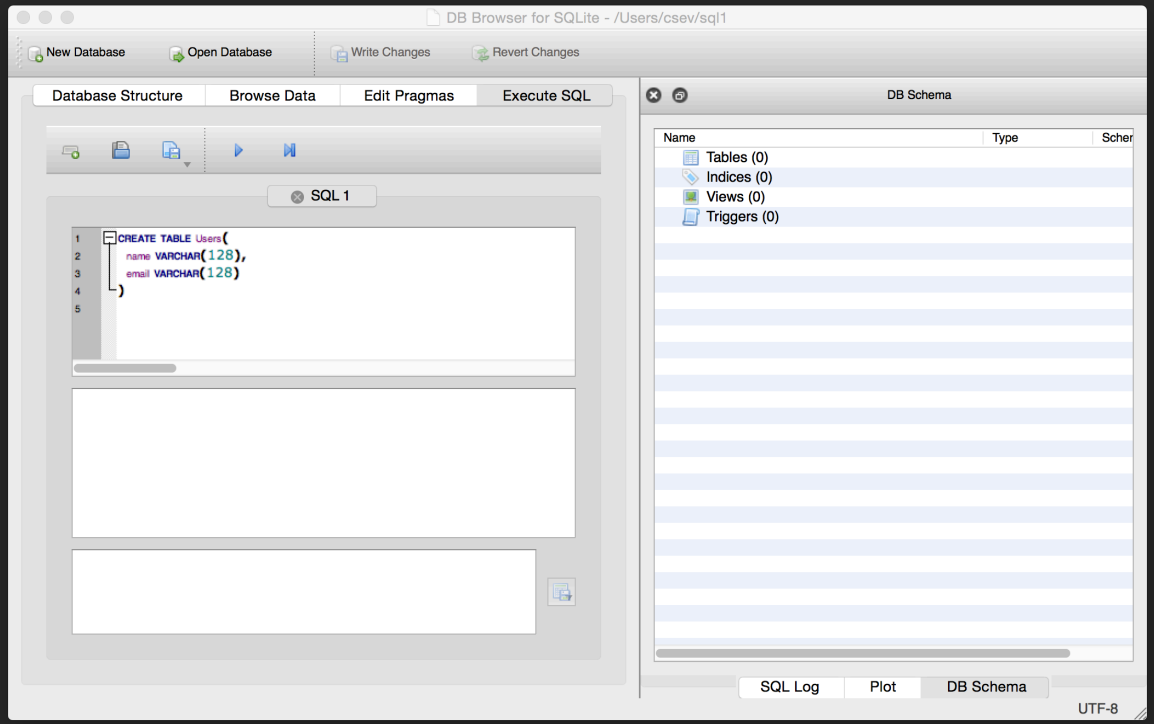
- https://addons.mozilla.org/en-US/firefox/addon/sqlite-manager/

SQLite is embedded in Python and a number of other languages

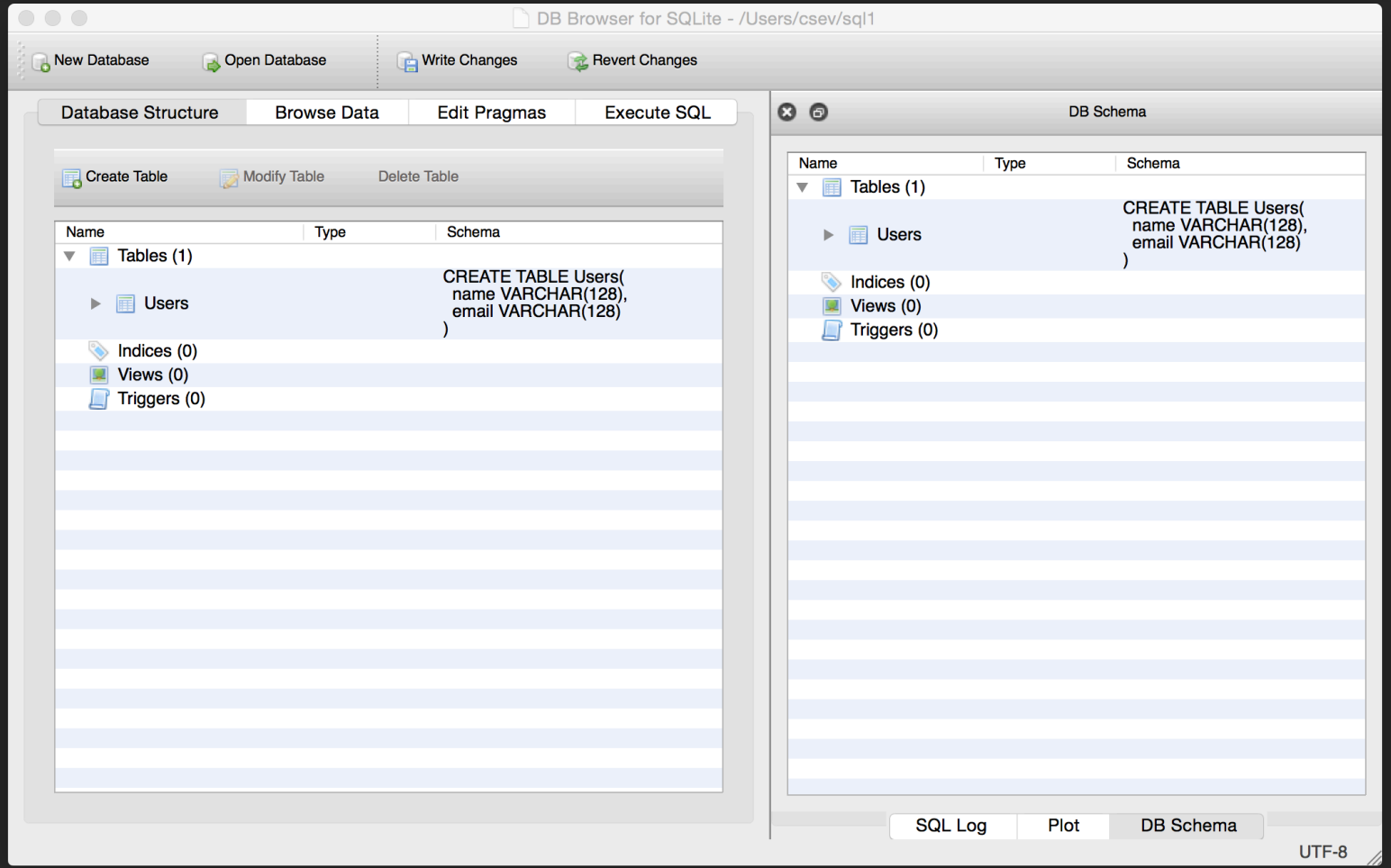


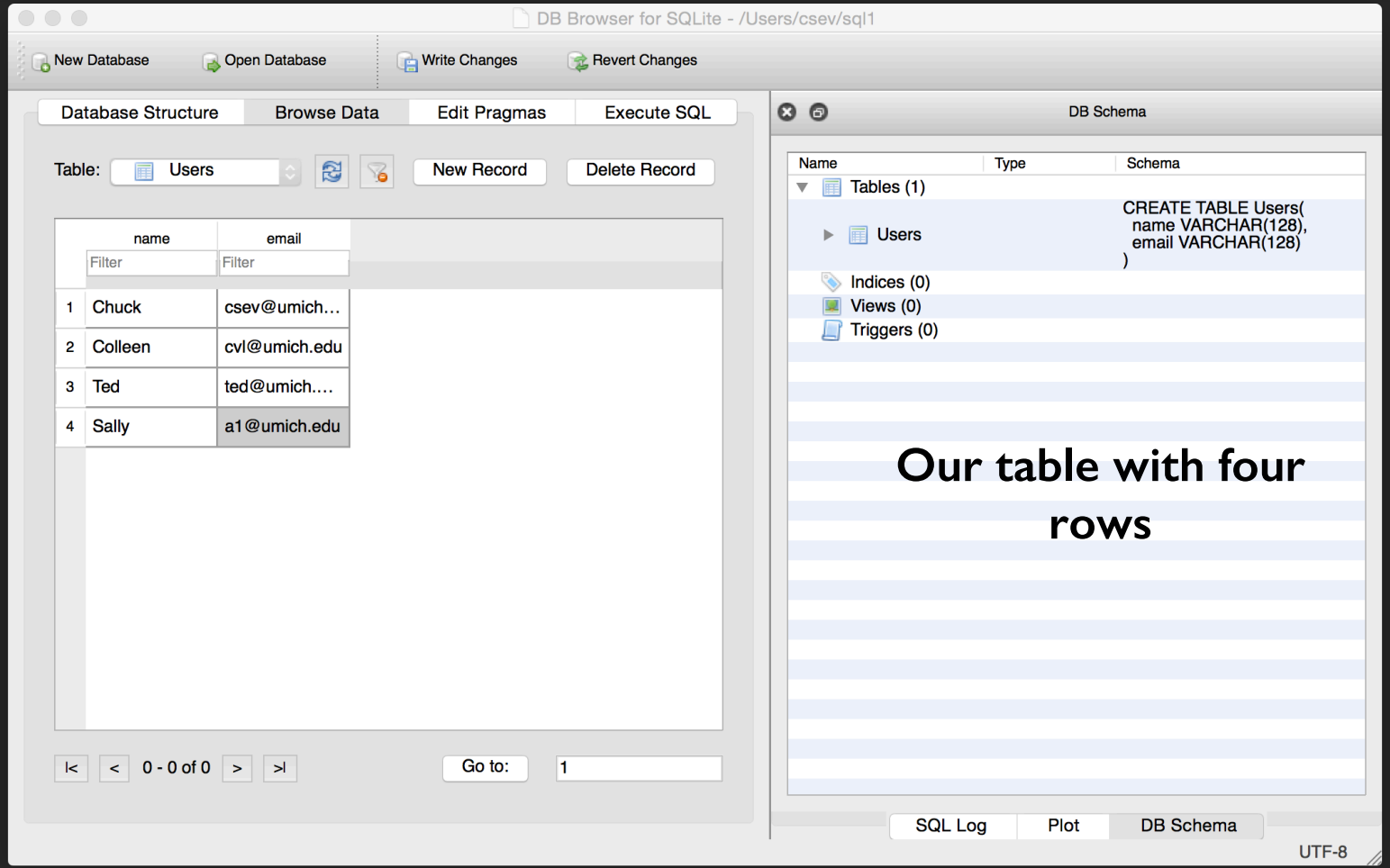
http://sqlitebrowser.org/

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
|<
<
0 - 0 of 0
>
>|
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

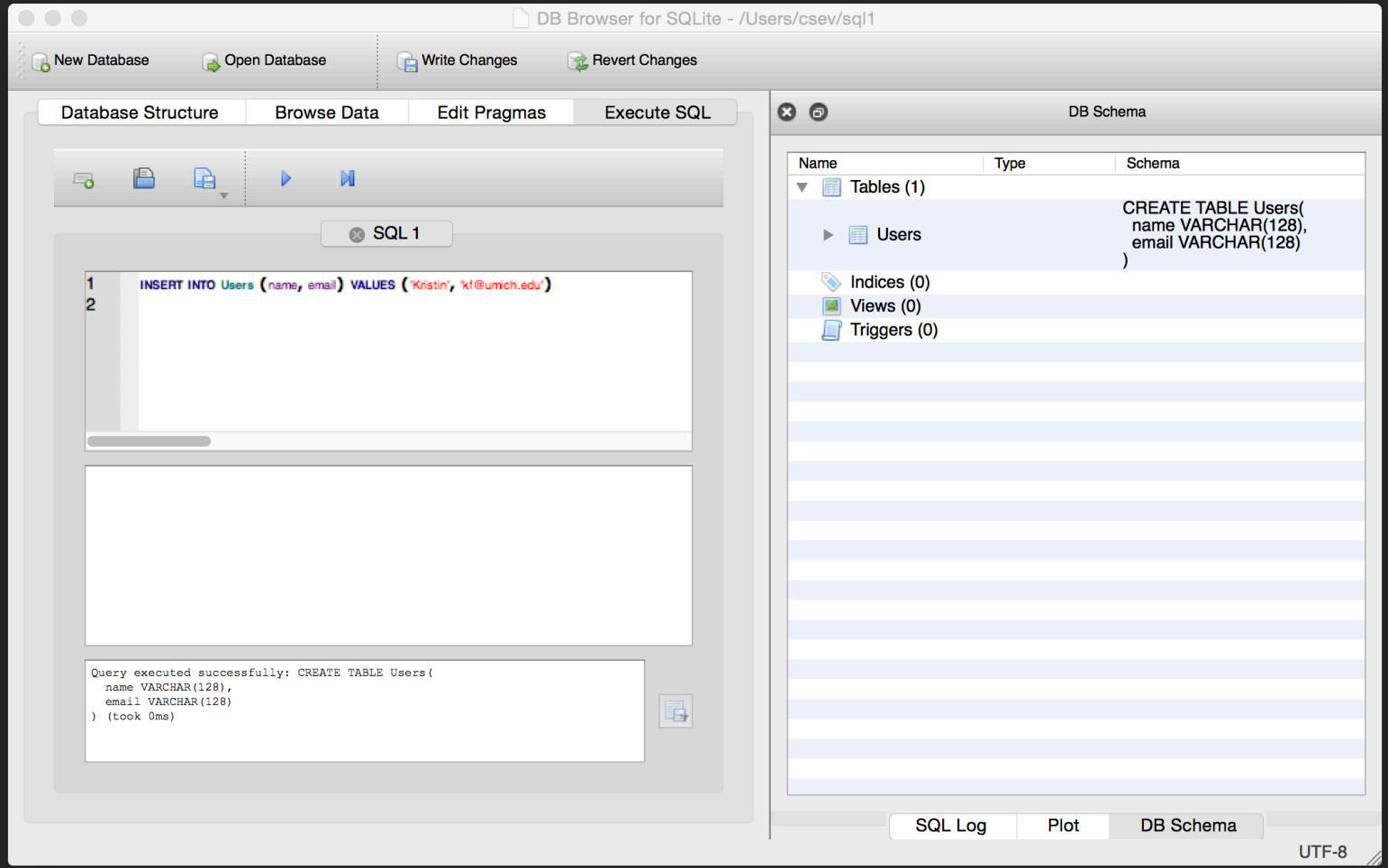
- **Structured Query Language** is the language we use to issue commands to the database
  - Create a table
  - Retieve some data
  - Insert data
  - Delete data

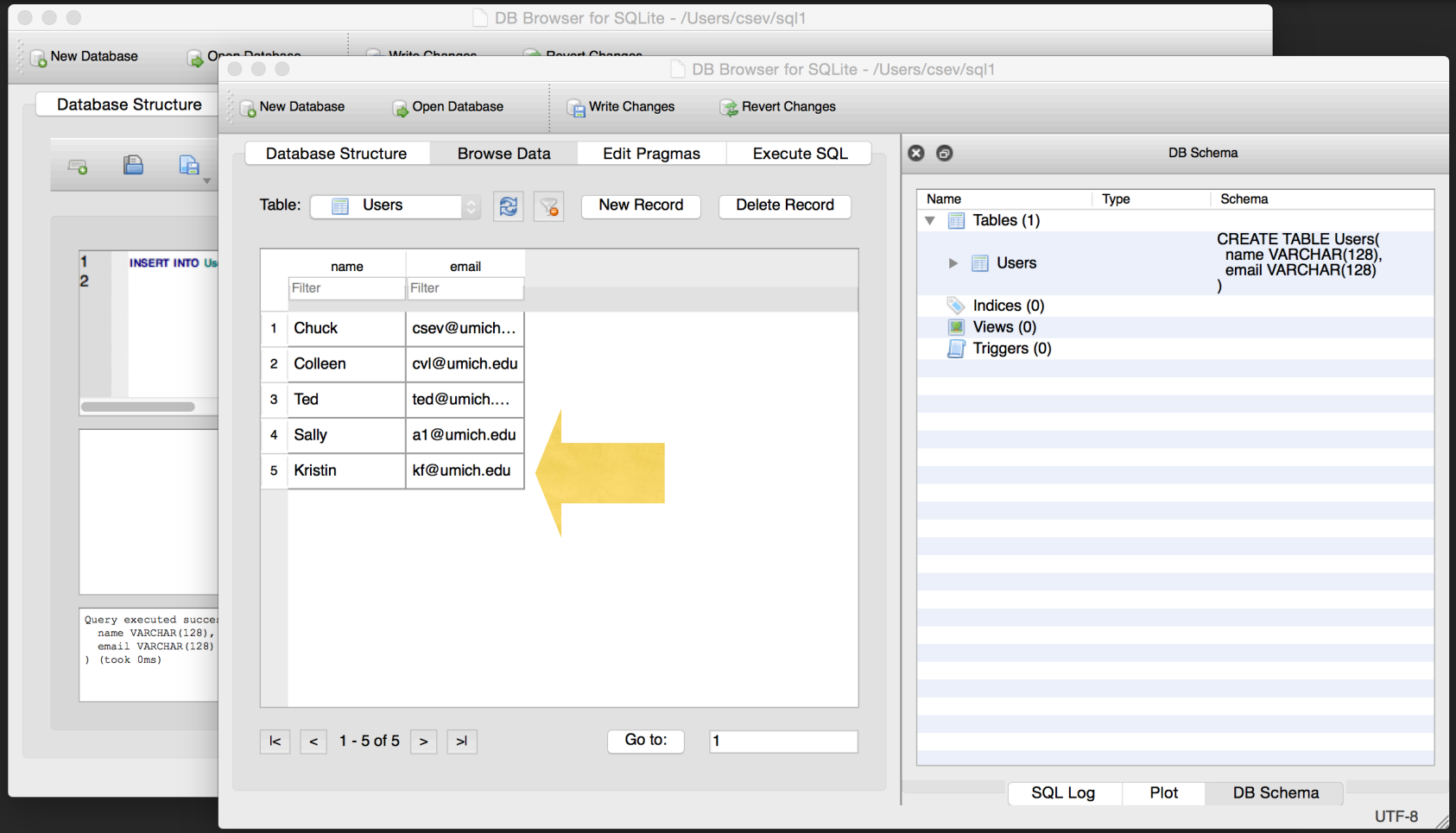
http://en.wikipedia.org/wiki/SQL

SQL Insert

- The Insert statement inserts a row into a table

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

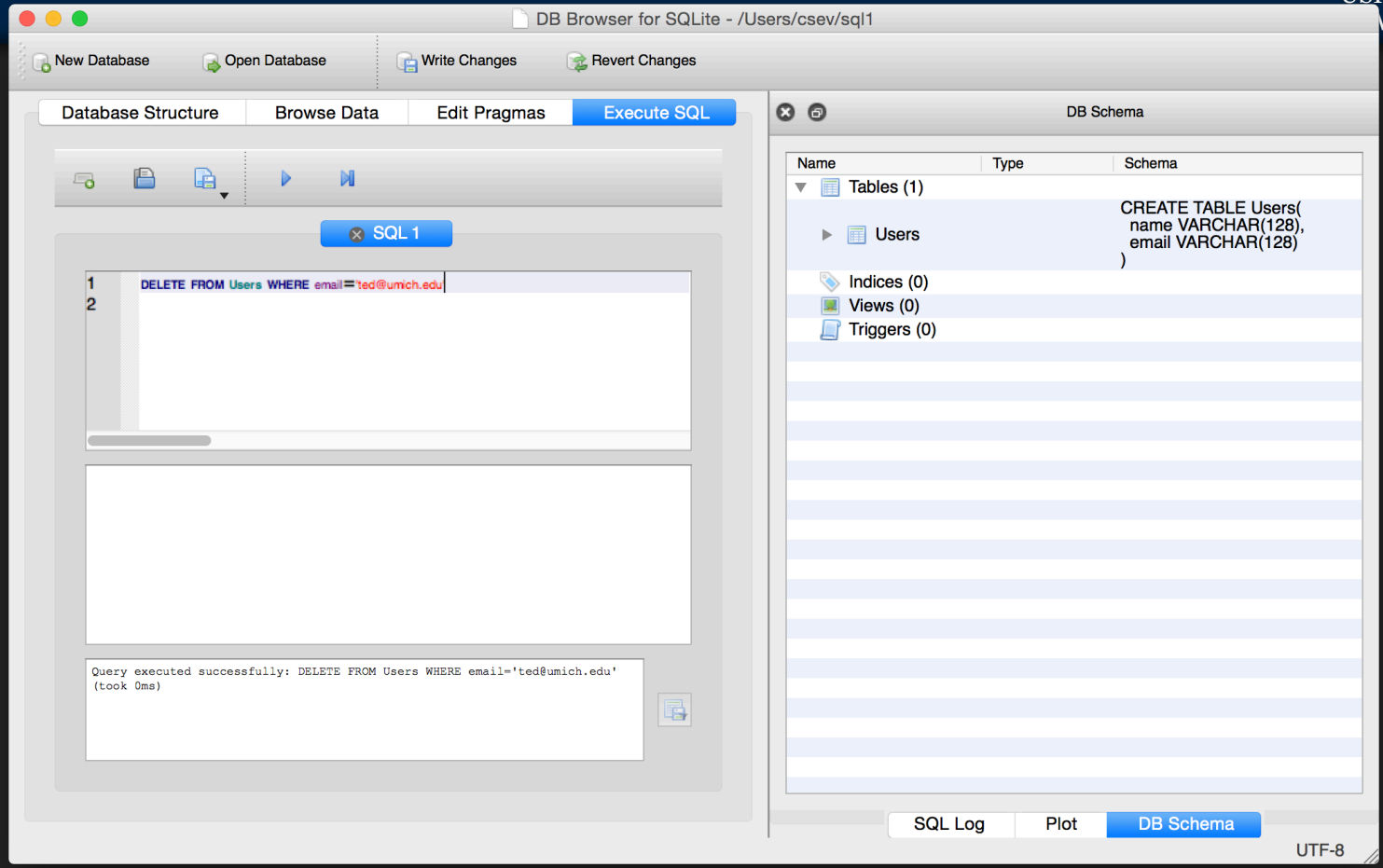

DB Browser for SQLite - /Users/csev/sql1
New Database
Open Database
Write Changes
Revert Changes
Database Structure
Browse Data
Edit Pragmas
Execute SQL
SQL 1
1 INSERT INTO Users (name, email) VALUES ('Kristin', 'kf@umich.edu')
2
Query executed successfully: CREATE TABLE Users(
name VARCHAR(128),
email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

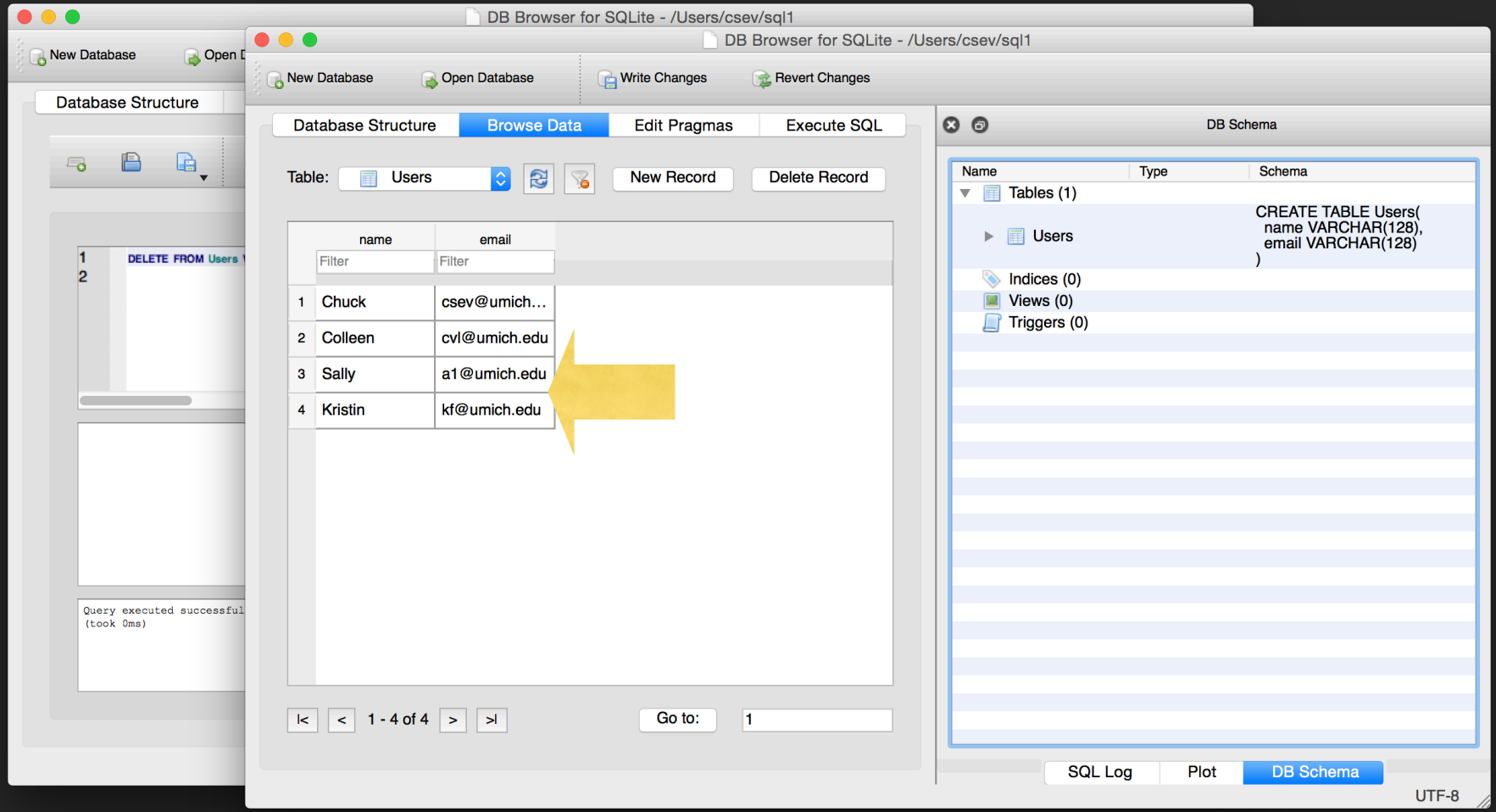

DB Browser for SQLite - /Users/csev/sql1
New Database
Open Database
Write Changes
Revert Changes
Database Structure
Browse Data
Edit Pragmas
Execute SQL
Table: Users
New Record
Delete Record
name
email
Filter
1 Chuck csev@umich...
2 Colleen cvl@umich.edu
3 Ted ted@umich....
4 Sally a1@umich.edu
5 Kristin 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 a selection criteria

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

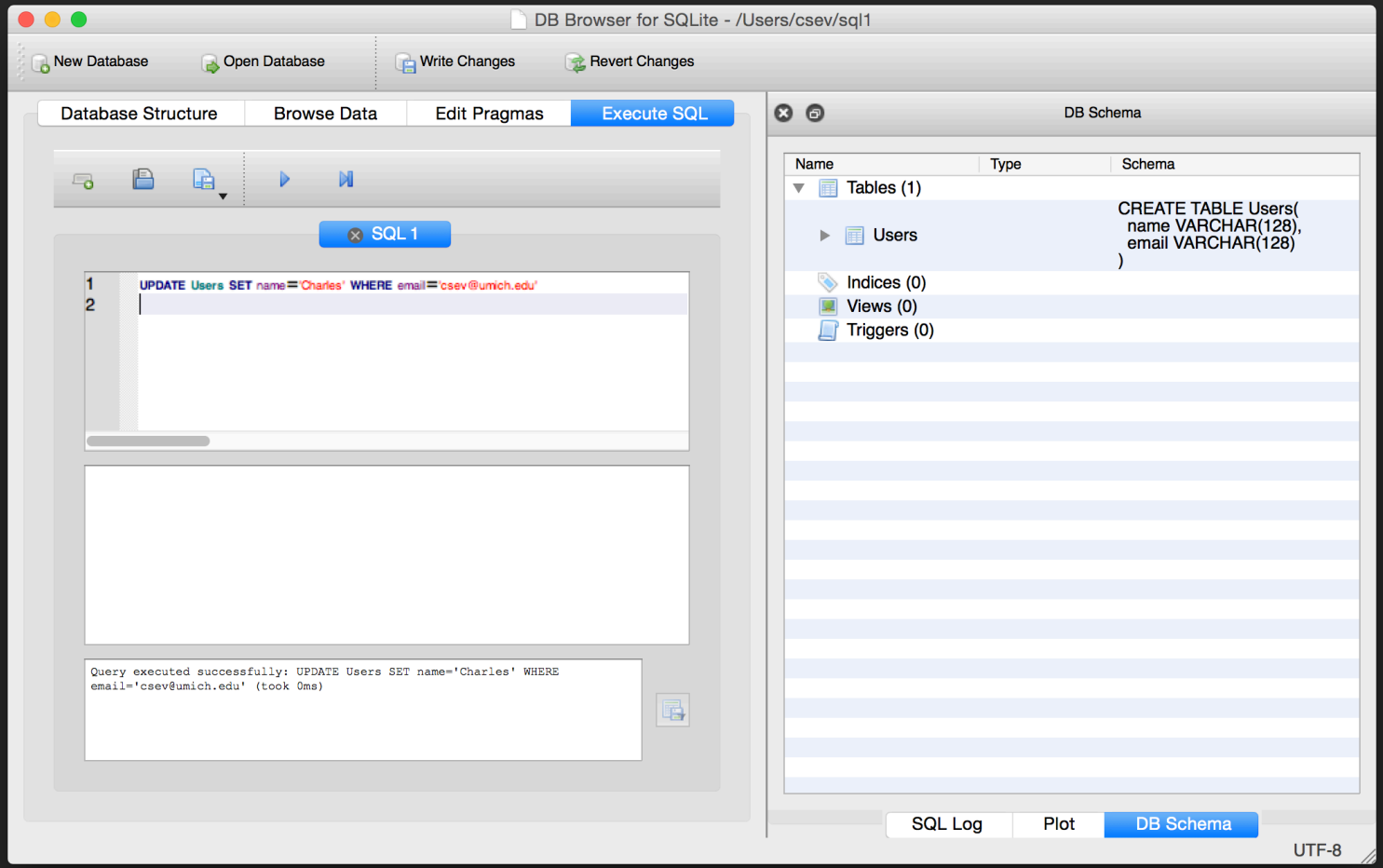

DB Browser for SQLite - /Users/csev/sql1
New Database
Open Database
Write Changes
Revert Changes
Database Structure
Browse Data
Edit Pragmas
Execute SQL
SQL 1
1 DELETE FROM Users WHERE email='ted@umich.edu
2
Query executed successfully: DELETE FROM Users WHERE email='ted@umich.edu'
(took 0ms)
DB Schema
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

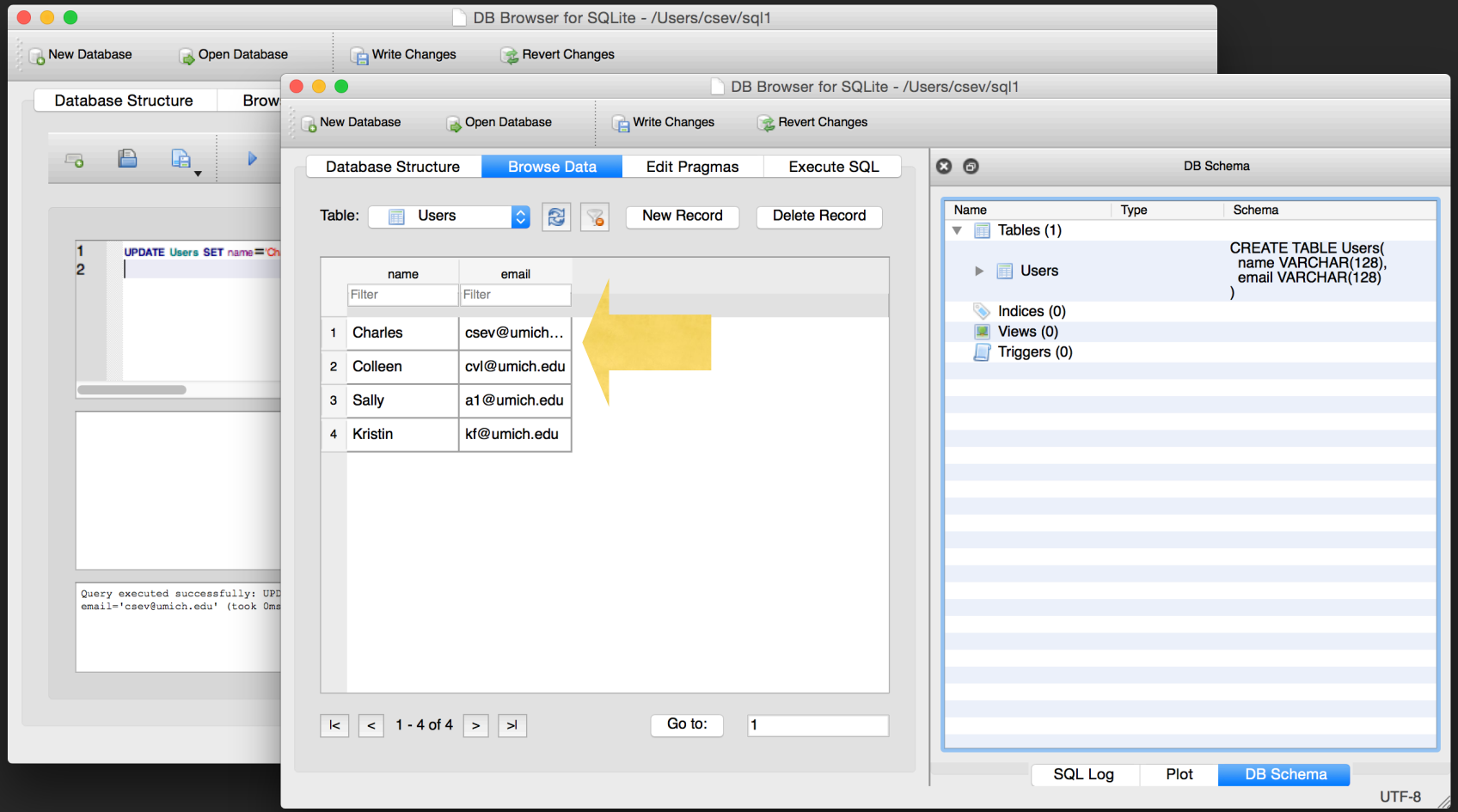

DB Browser for SQLite - /Users/csev/sql1
New Database
Open Database
Write Changes
Revert Changes
Database Structure
Browse Data
Edit Pragmas
Execute SQL
Table: Users
New Record
Delete Record
name
email
Filter
Filter
1 Chuck csev@umich...
2 Colleen cvl@umich.edu
3 Sally a1@umich.edu
4 Kristin kf@umich.edu
|< < 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
DELETE FROM Users
Query executed successful
(took 0ms)

SQL: Update

- Allows the updating of a field with a where clause

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

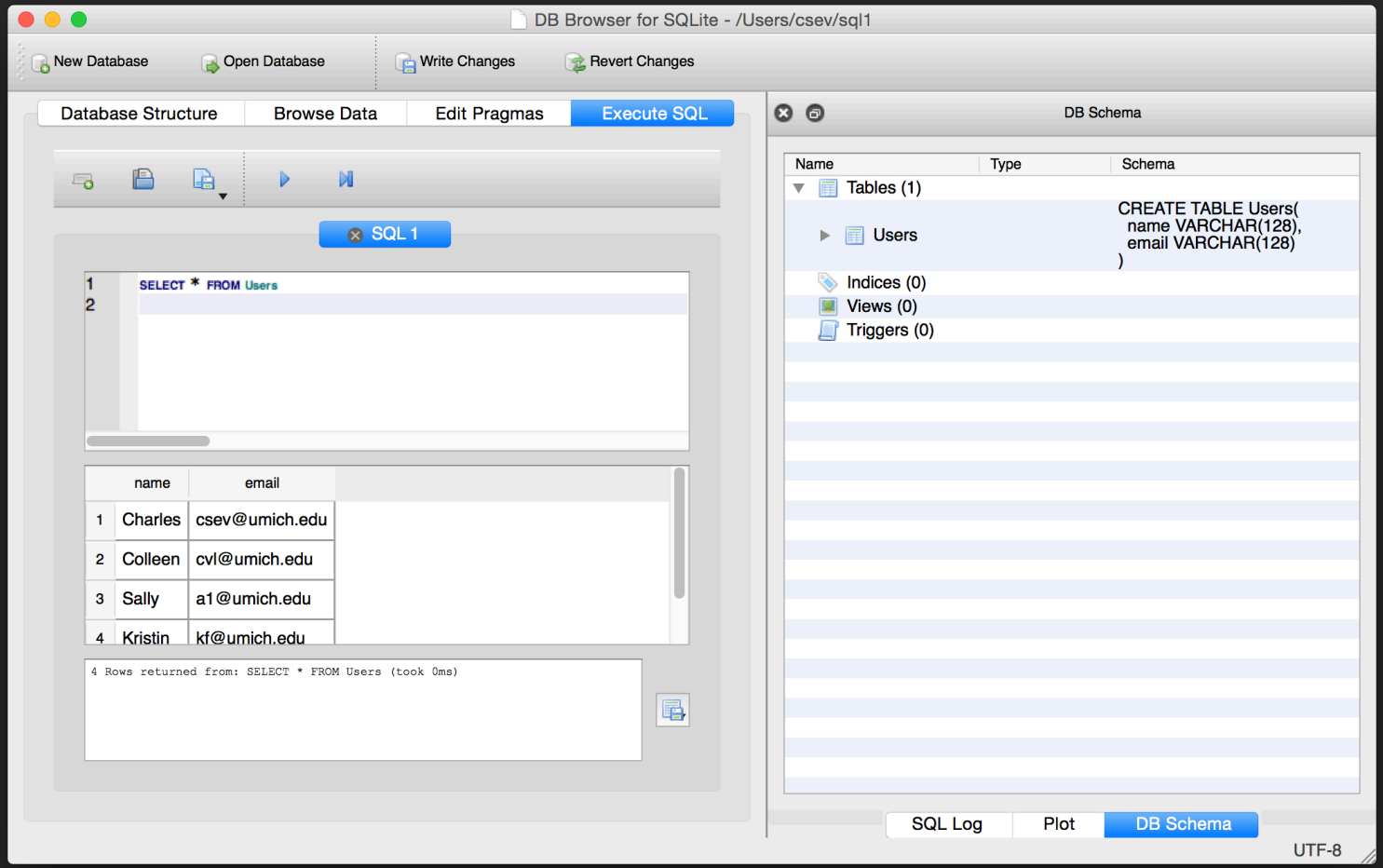

DB Browser for SQLite - /Users/csev/sql1
New Database
Open Database
Write Changes
Revert Changes
Database Structure
Browse Data
Edit Pragmas
Execute SQL
SQL 1
1 UPDATE Users SET name='Charles' WHERE email='csev@umich.edu'
2
Query executed successfully: UPDATE Users SET name='Charles' WHERE email='csev@umich.edu' (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

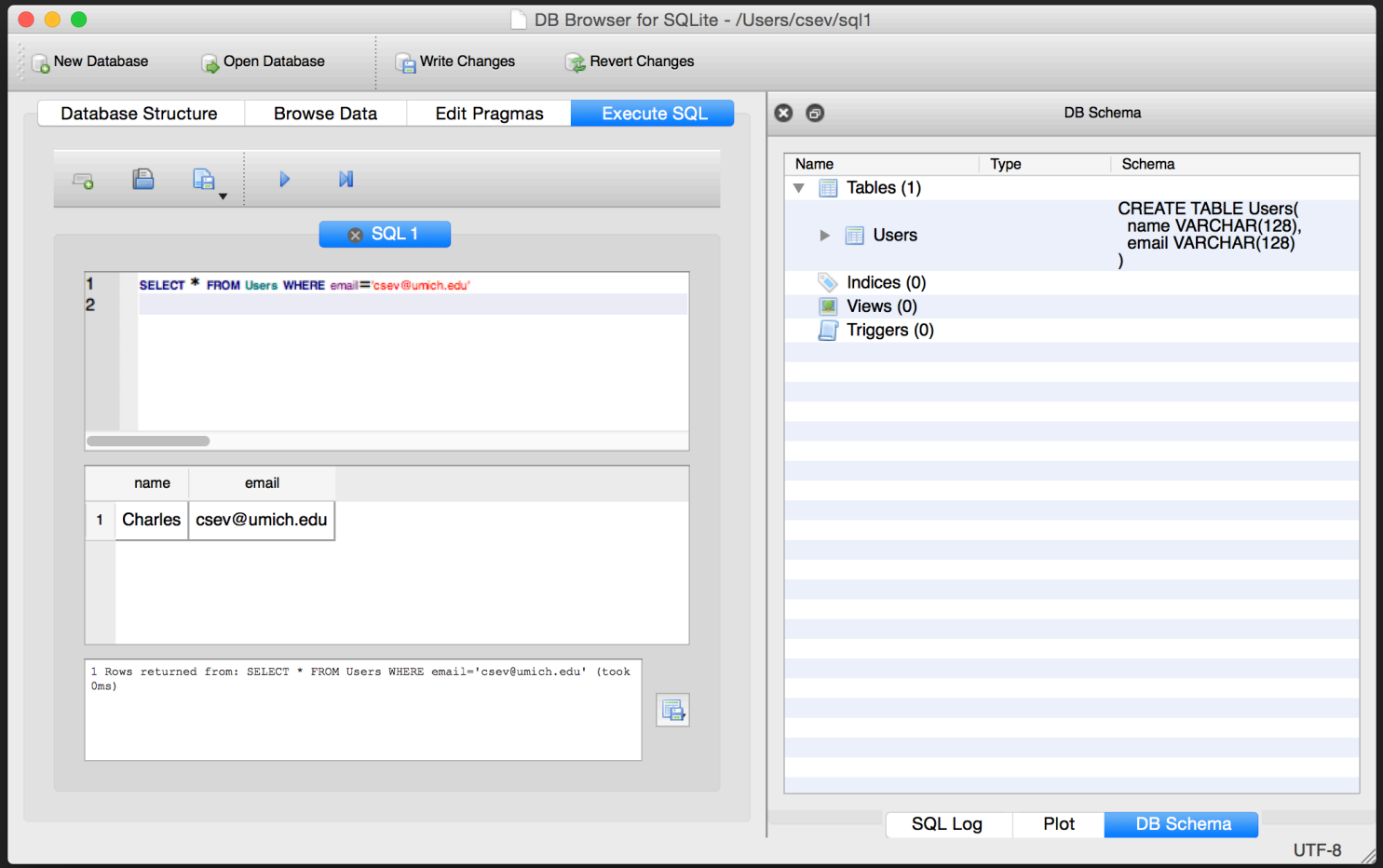

DB Browser for SQLite - /Users/csev/sql1
New Database
Open Database
Write Changes
Revert Changes
Database Structure
Browse Data
Edit Pragmas
Execute SQL
Table:
Users
New Record
Delete Record
name
email
Filter
1 Charles csev@umich...
2 Colleen cvl@umich.edu
3 Sally a1@umich.edu
4 Kristin kf@umich.edu
1 - 4 of 4
Go to:
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
UPDATE Users SET name
Query executed successfully: UPD
email='csev@umich.edu' (took 0ms

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'
```

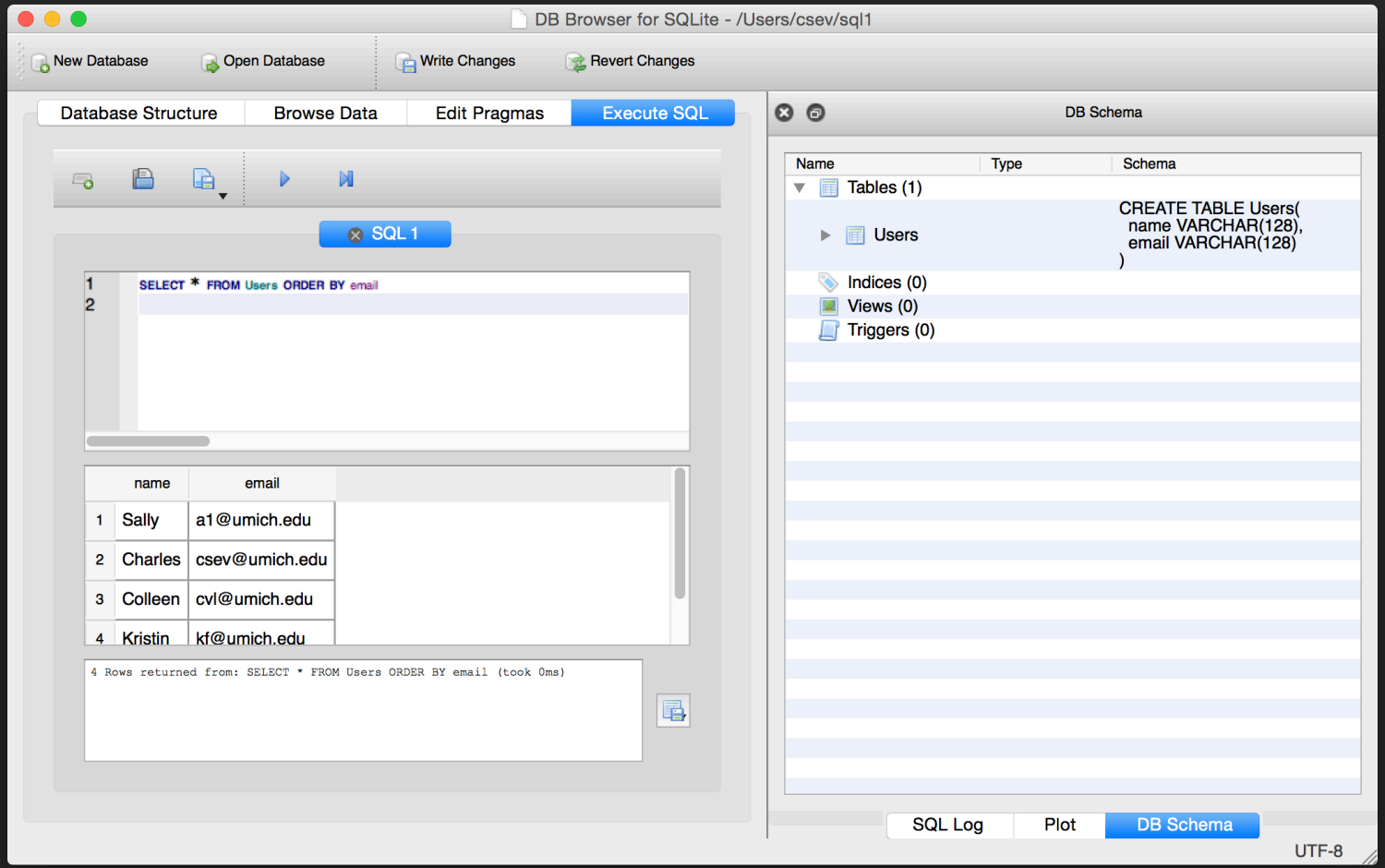

DB Browser for SQLite - /Users/csev/sql1
New Database
Open Database
Write Changes
Revert Changes
Database Structure
Browse Data
Edit Pragmas
Execute SQL
SQL 1
1 SELECT * FROM Users
2
name email
1 Charles csev@umich.edu
2 Colleen cvl@umich.edu
3 Sally a1@umich.edu
4 Kristin kf@umich.edu
4 Rows returned 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
```


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
```

This is not too exciting (so far)

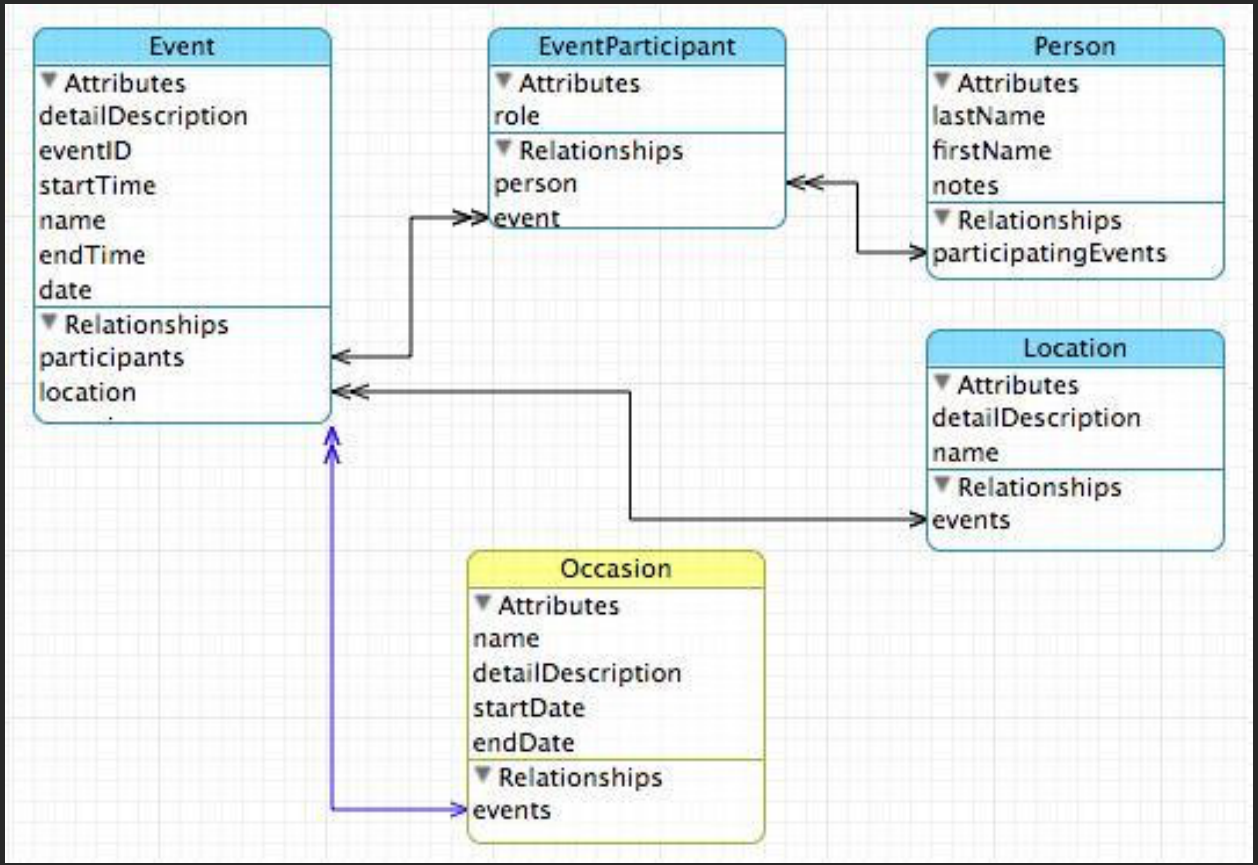
- Tables pretty much look like big fast programmable spreadsheets with rows, columns, and commands
- The power comes when we have more than one table and we can exploit the relationships between the tables

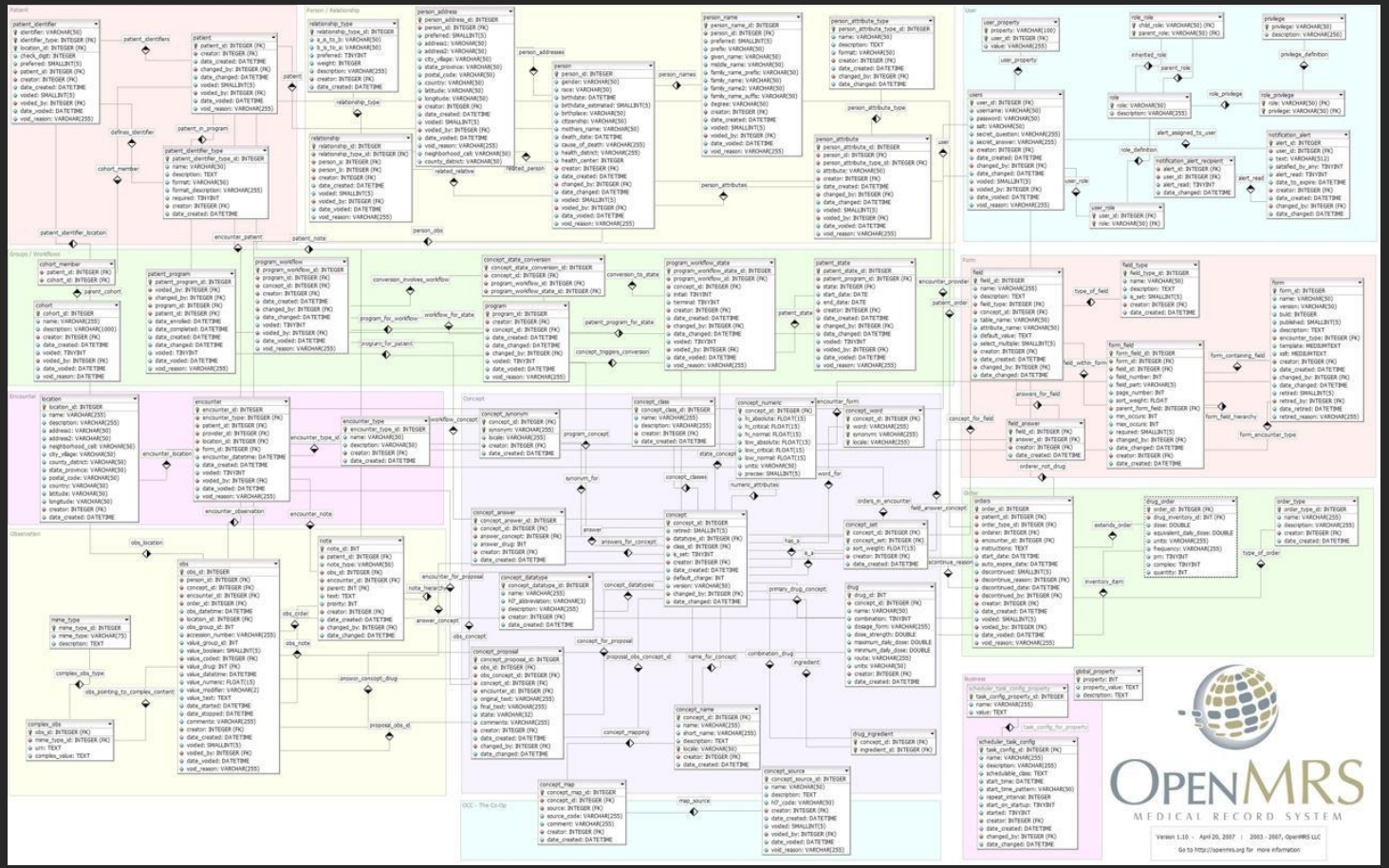
Complex Data Models and Relationships

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...


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


OpenMRS
MEDICAL RECORD SYSTEM

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

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.

Len

Album

Genre

Artist

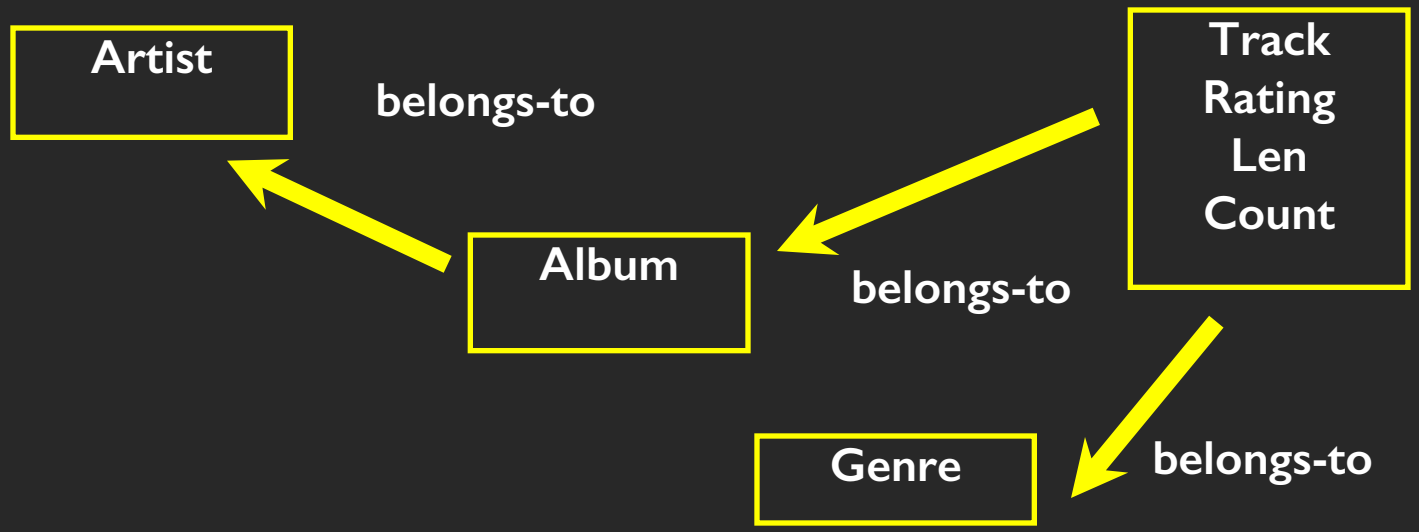
Rating

Track

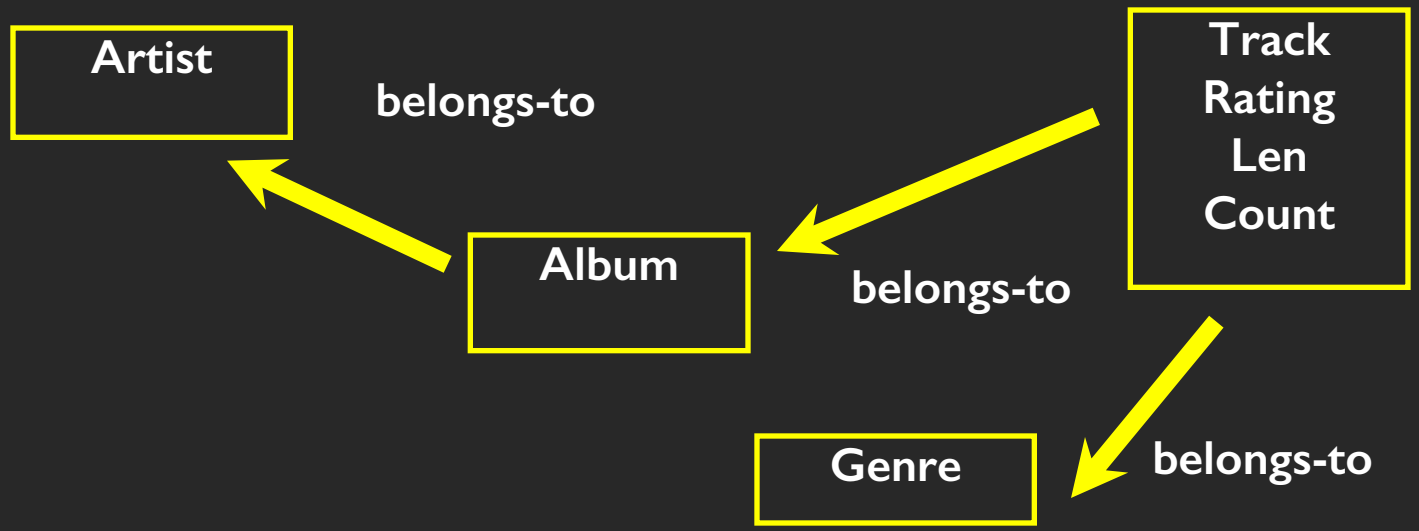
Count

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
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



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

Representing Relationships in a Database

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"

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

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	[illegible]	★★★★★	61
Dúlamán	3:43	Altan	Natural Wonders M...	New [illegible]ge		31
Rode Across the Desert	[illegible]10	America	Greatest Hits	Easy Liste...	★★★★★	23
Now You Are Gone	3:08	America	Greatest Hits	Easy Listen...	★★★★★	18
Tin Man	3:30	America	Greatest Hits	Easy Listen...	★★★★★	23

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

id	name
Filter	Filter
1	Led Zepplin
2	AC/DC

Artist

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

Album

Key Terminology

Finding our way around....

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

Primary Key Rules

Best practices

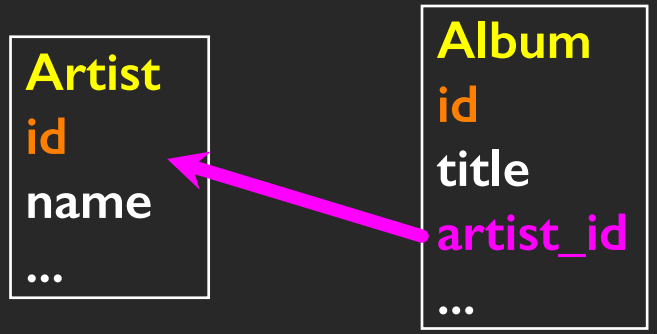
- Never use your logical key as the primary key
- Logical keys can and do change, albeit slowly
- Relationships that are based on matching string fields are less efficient than integers

User
id
login
password
name
email
created_at
modified_at
login_at

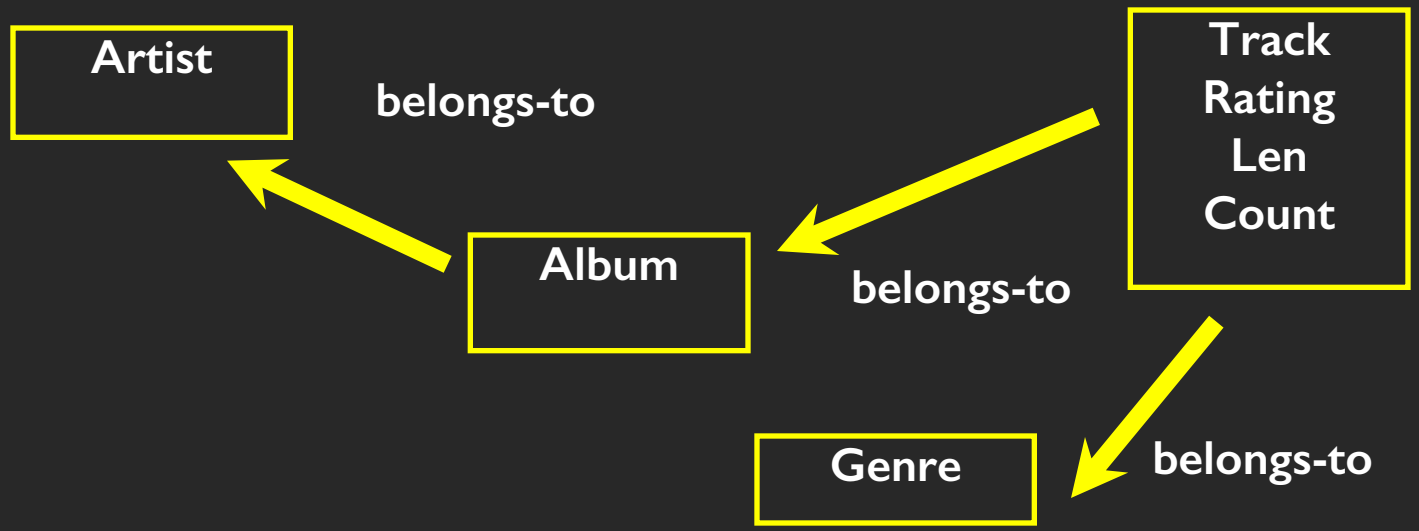
Foreign Keys

- A foreign key is when a table has a column that contains a key which points to the primary key of another table.

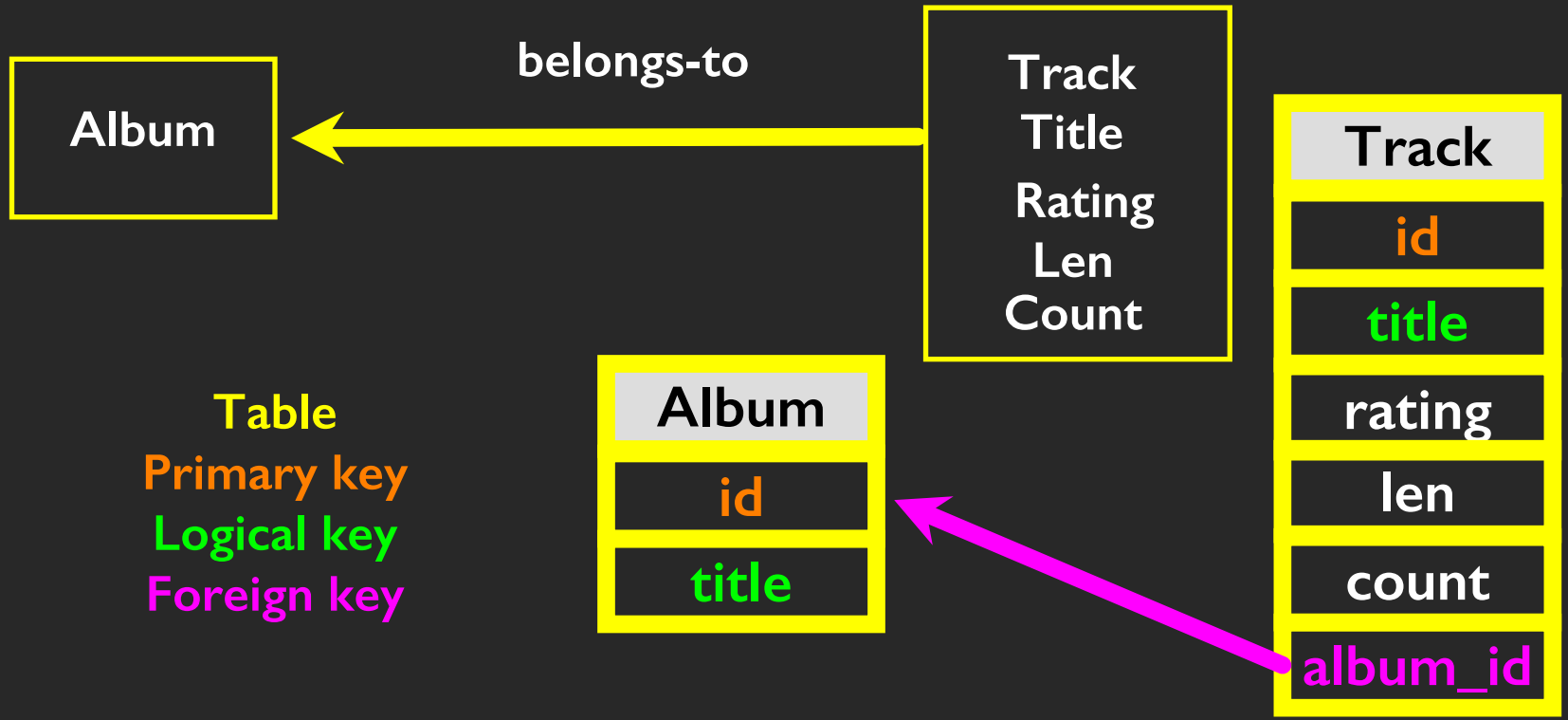
- When all primary keys are integers, then all foreign keys are integers - this is good - very good

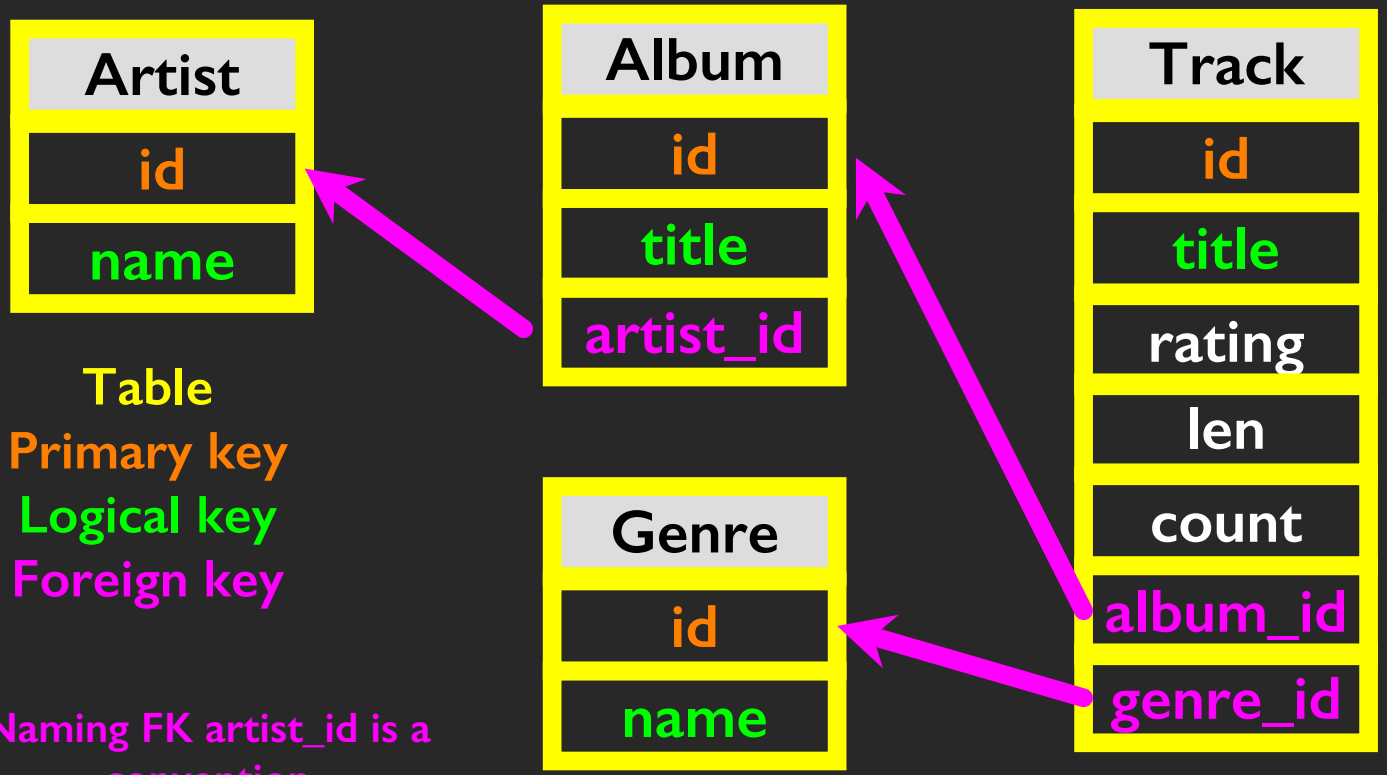


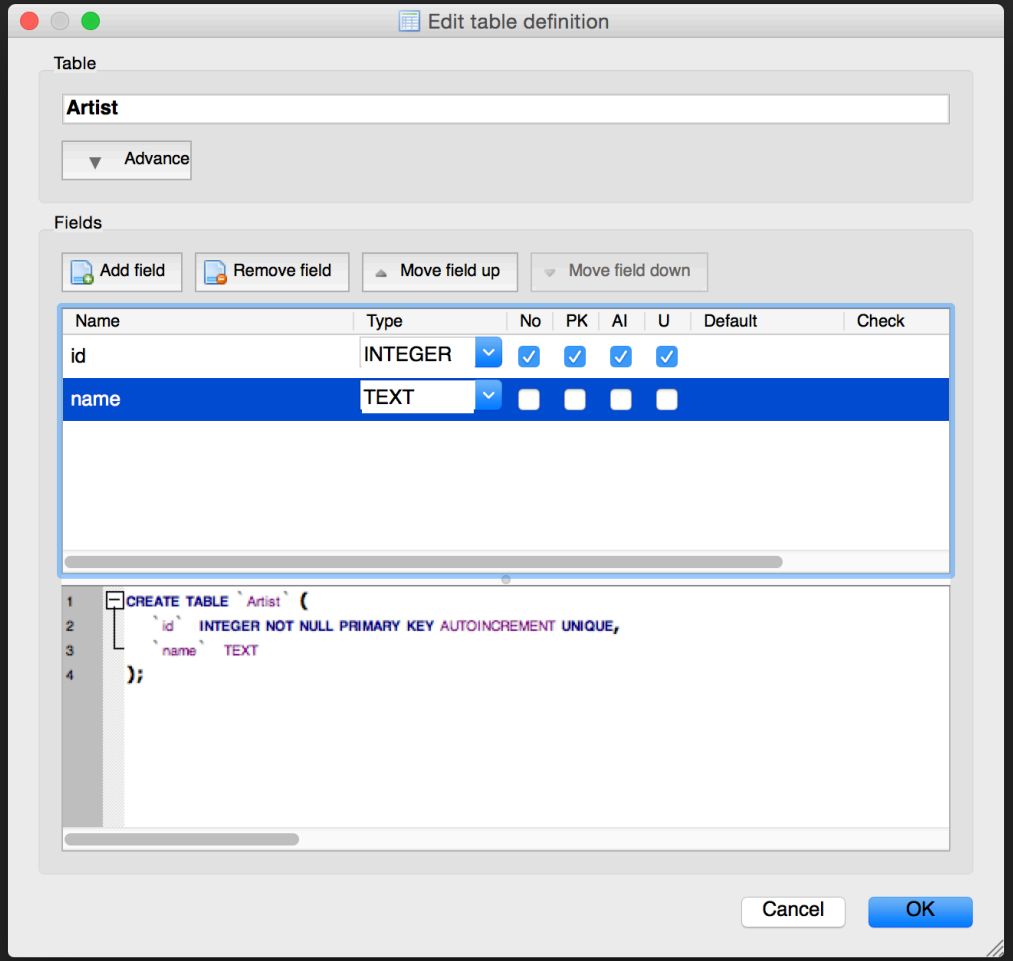
Relationship Building (in tables)

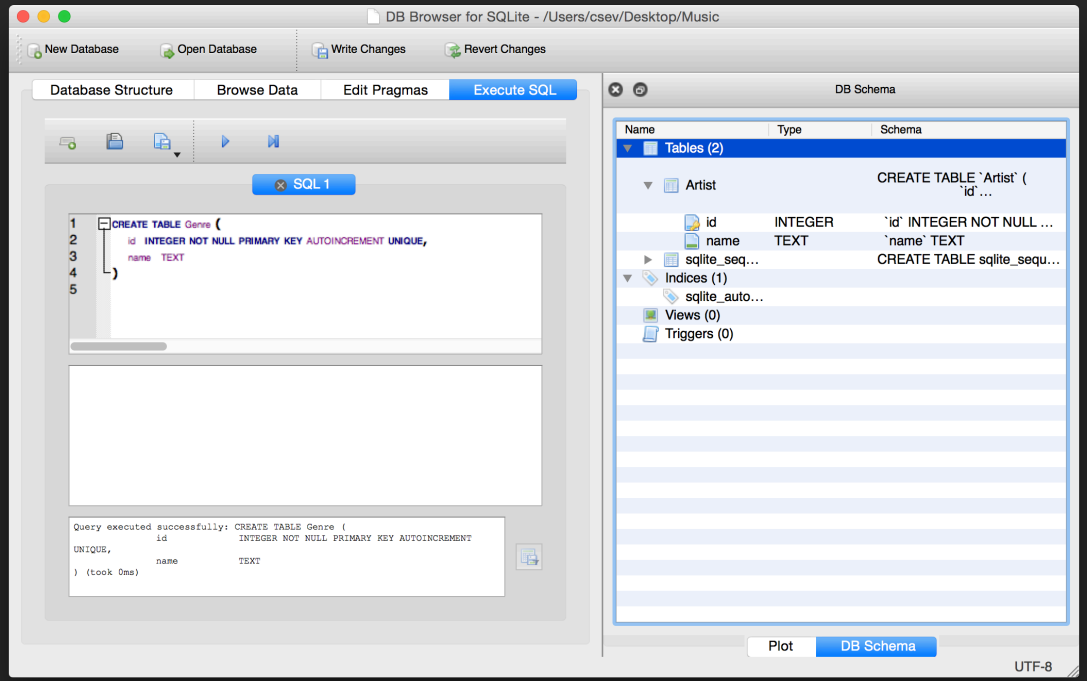


Hells Bells	5:13	AC/DC	Who Made Who	Rock	★★★★★	61
Shake Your Foundations	3:54	AC/DC	Who Made Who	Rock	★★★★★	70
Chase the Ace	3:01	AC/DC	Who Made Who	Rock		56
For Those About To Rock (We ...	5:54	AC/DC	Who Made Who	Rock	★★★★★	61
Dúlamán	3:43	Altan	Natural Wonders M...	New Age		31
Rode Across the Desert	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
Table
Primary key
Logical key
Foreign key
Album
id
title

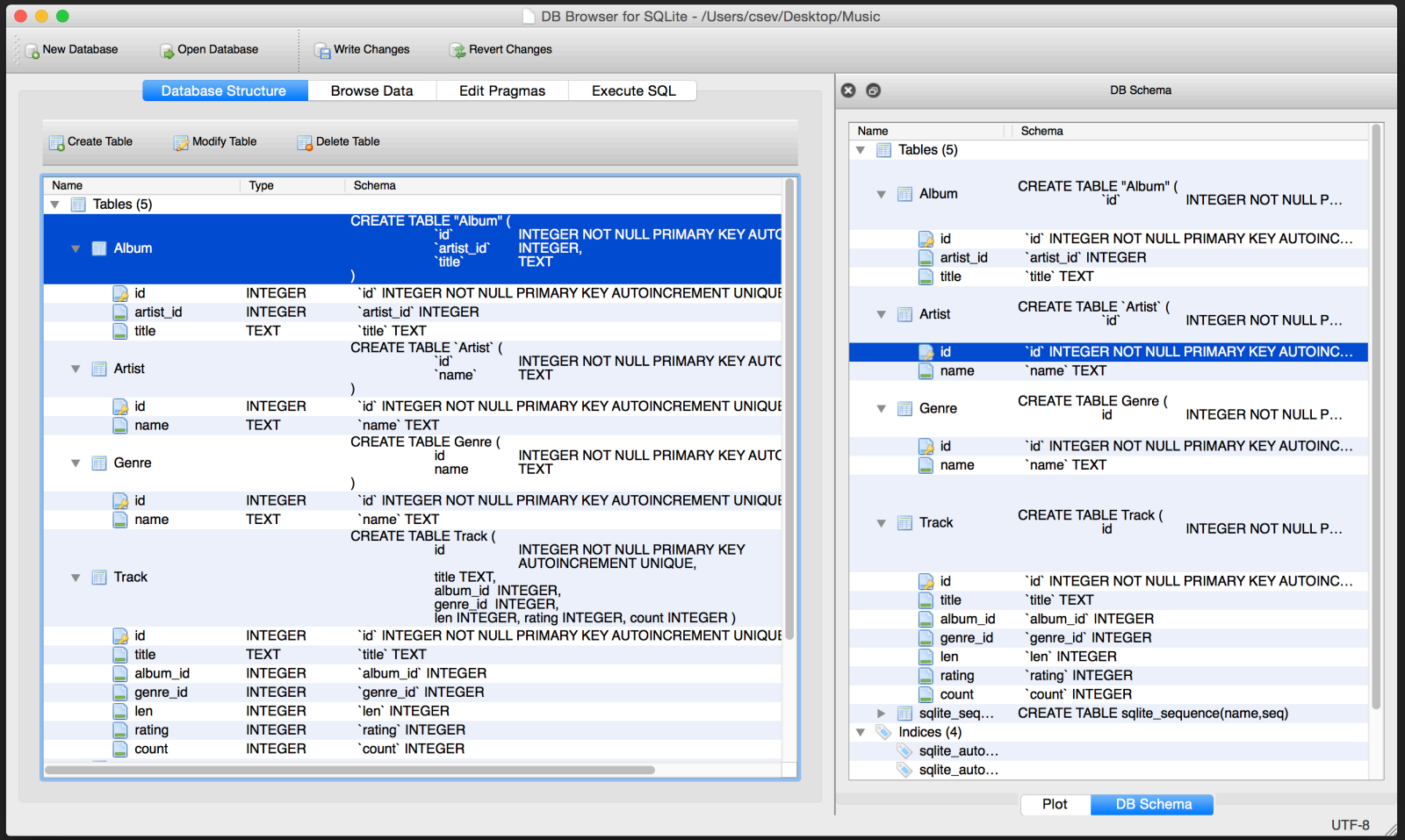

Artist
id
name
Album
id
title
artist_id
Track
id
title
rating
len
count
album_id
genre_id
Genre
id
name
Table
Primary key
Logical key
Foreign key
Naming FK artist_id is a convention

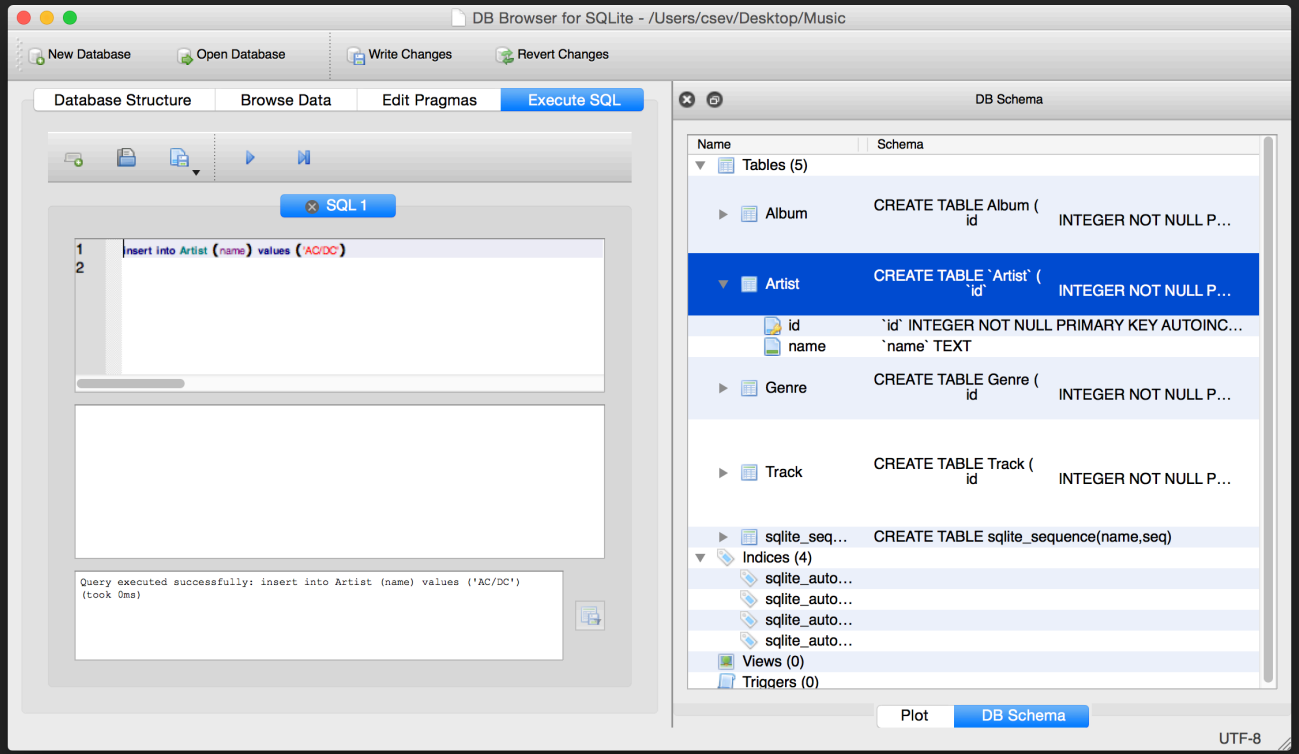

Edit table definition
Table
Artist
Advance
Fields
Add field
Remove field
Move field up
Move field down
Name
Type
No
PK
AI
U
Default
Check
id
INTEGER
name
TEXT
1 CREATE TABLE `Artist` (
2 `id` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
3 `name` TEXT
4);
Cancel
OK



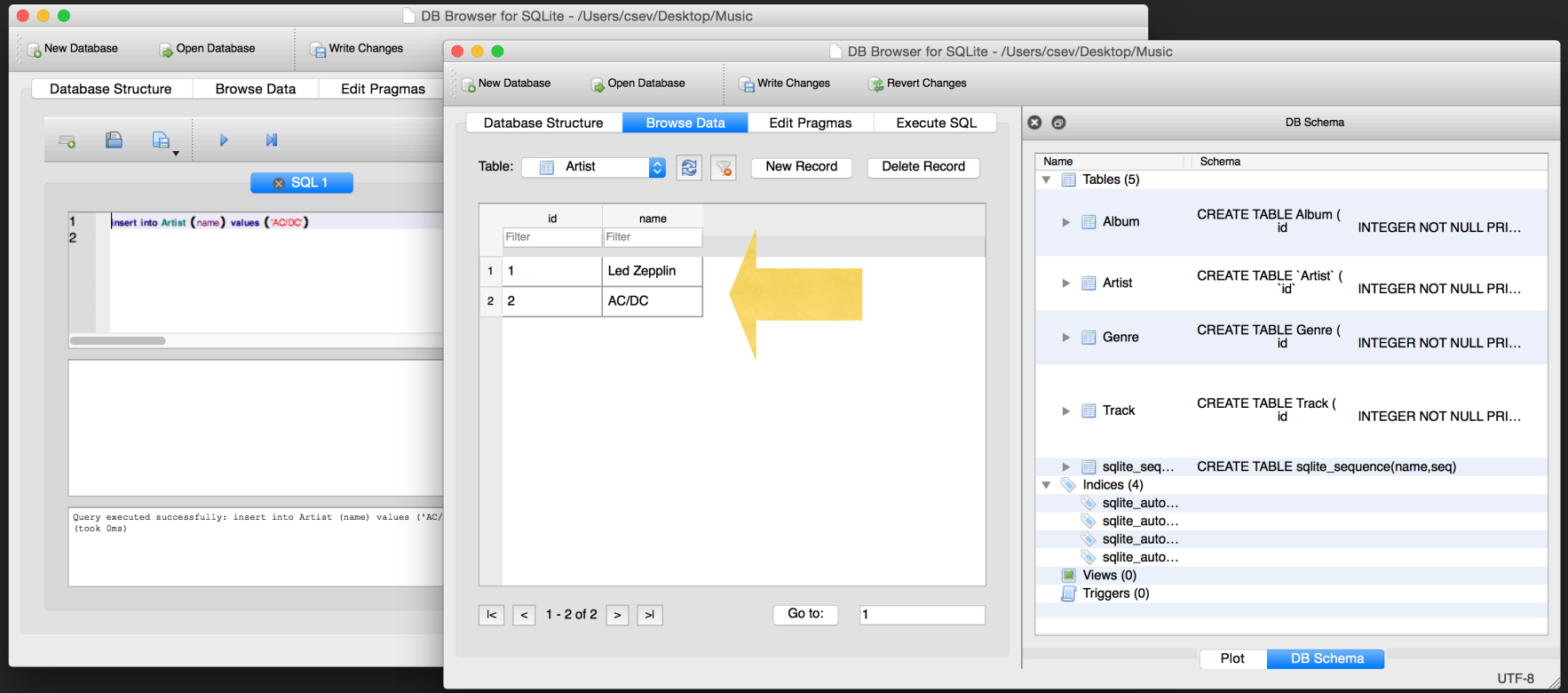

```
CREATE TABLE Genre (
        id          INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
        name        TEXT
)
```

```
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
)
```

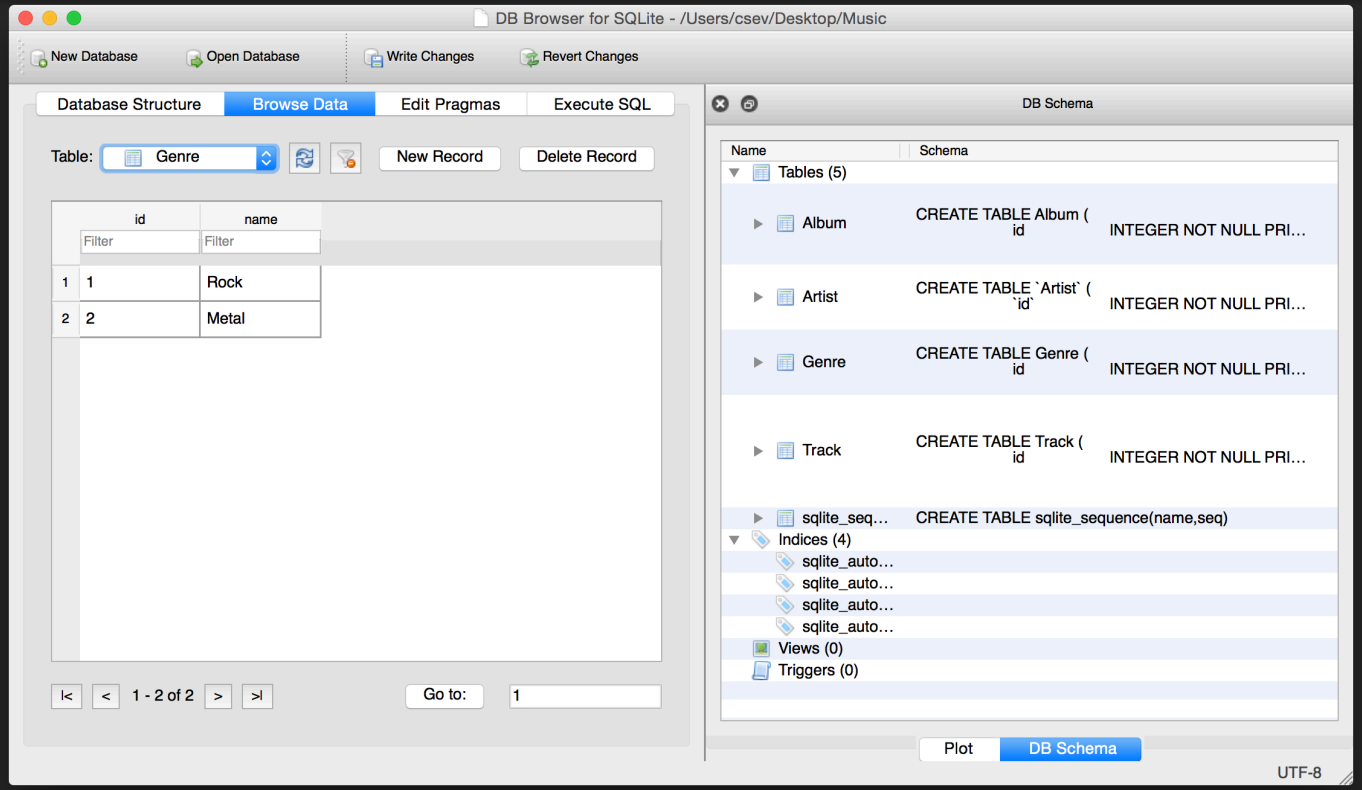

DB Browser for SQLite - /Users/csev/Desktop/Music
New Database
Open Database
Write Changes
Revert Changes
Database Structure
Browse Data
Edit Pragmas
Execute SQL
Create Table
Modify Table
Delete Table
Name
Type
Schema
Tables (5)
Album
CREATE TABLE "Album" (`id` INTEGER NOT NULL PRIMARY KEY AUTO `artist_id` 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



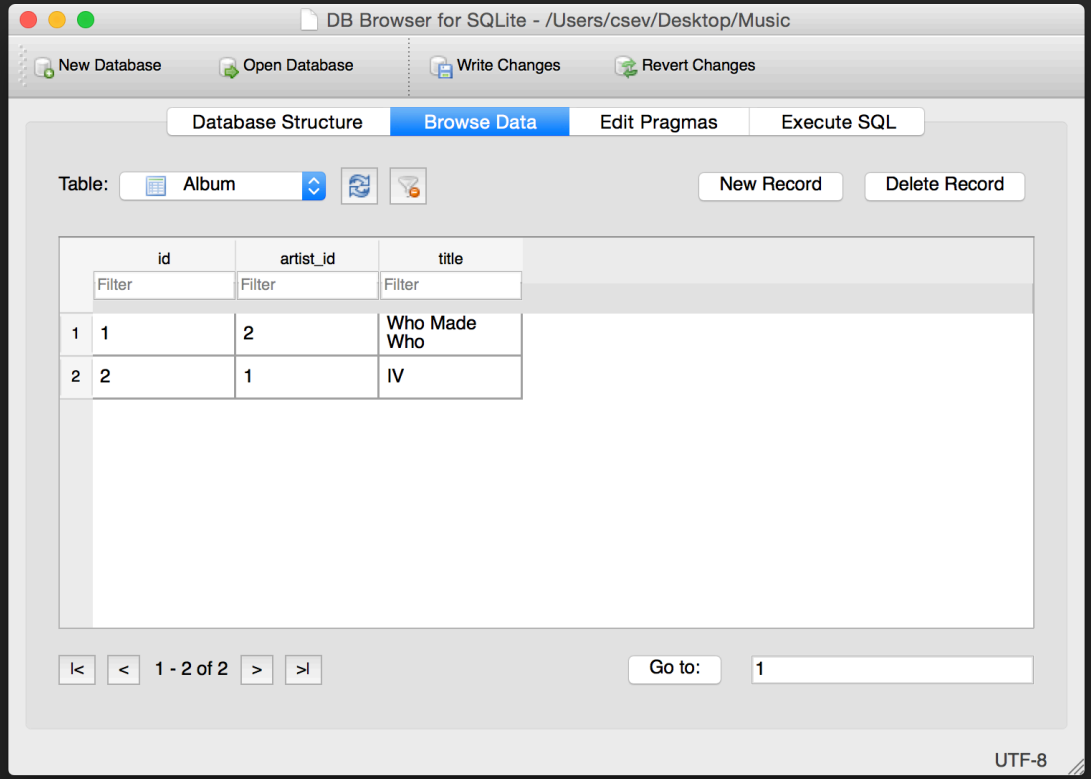
insert into Artist (name) values ('Led Zepplin')
insert into Artist (name) values ('AC/DC')



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

We have relationships!

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

Using Join Across Tables

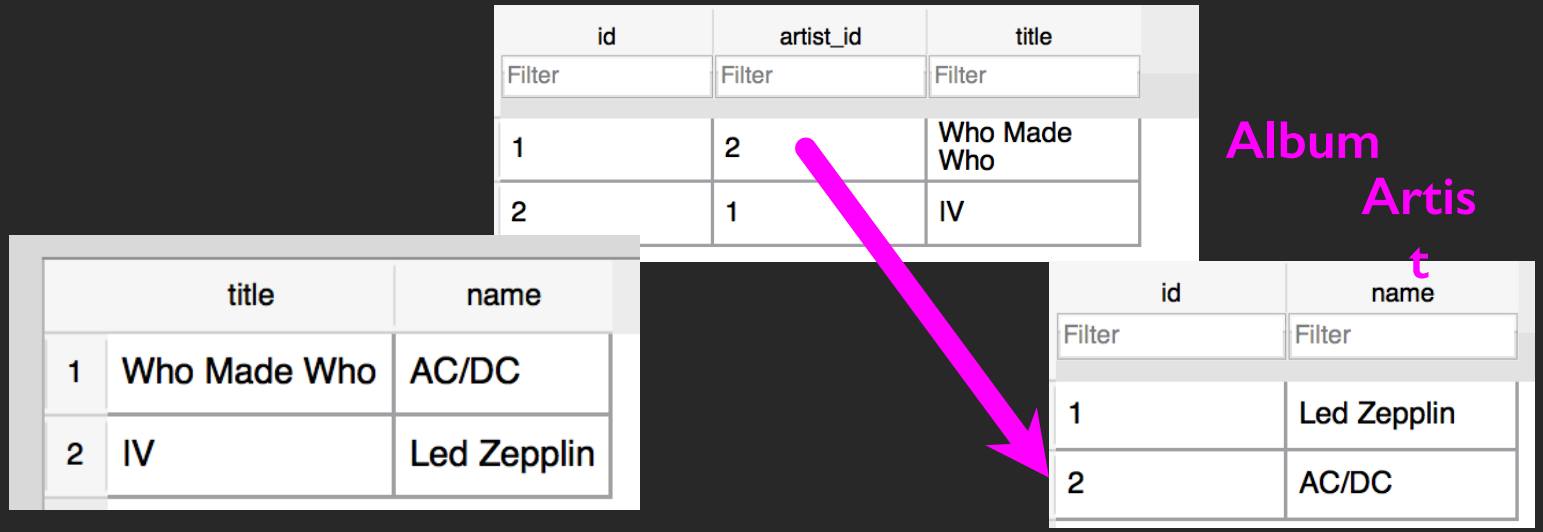
http://en.wikipedia.org/wiki/Join_(SQL)

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 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

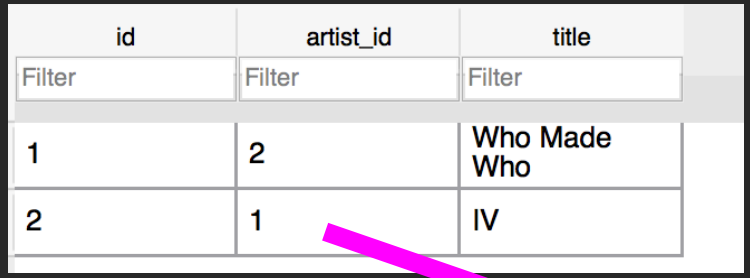



```
select Album.title, Artist.name from Album join Artist on Album.artist_id = Artist.id
```

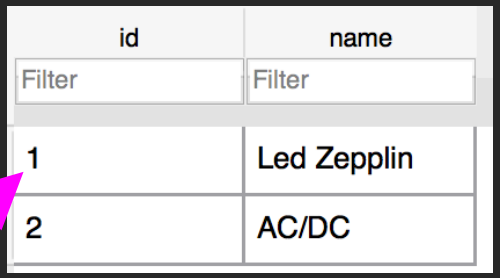
What we want to see

The tables that hold the data

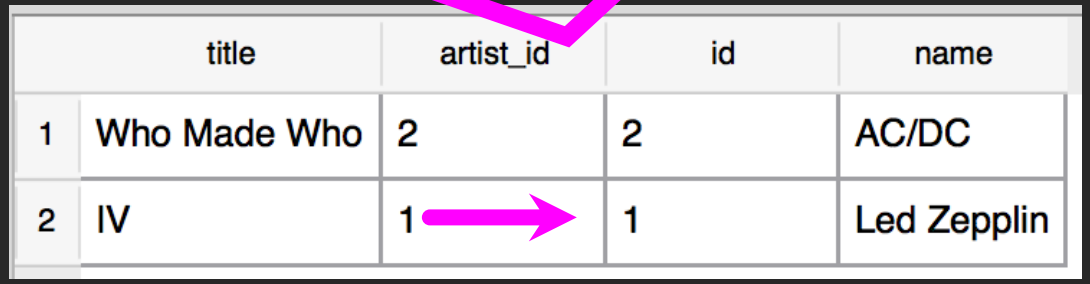
How the tables are linked



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	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

	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.

It can get complex...

```
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

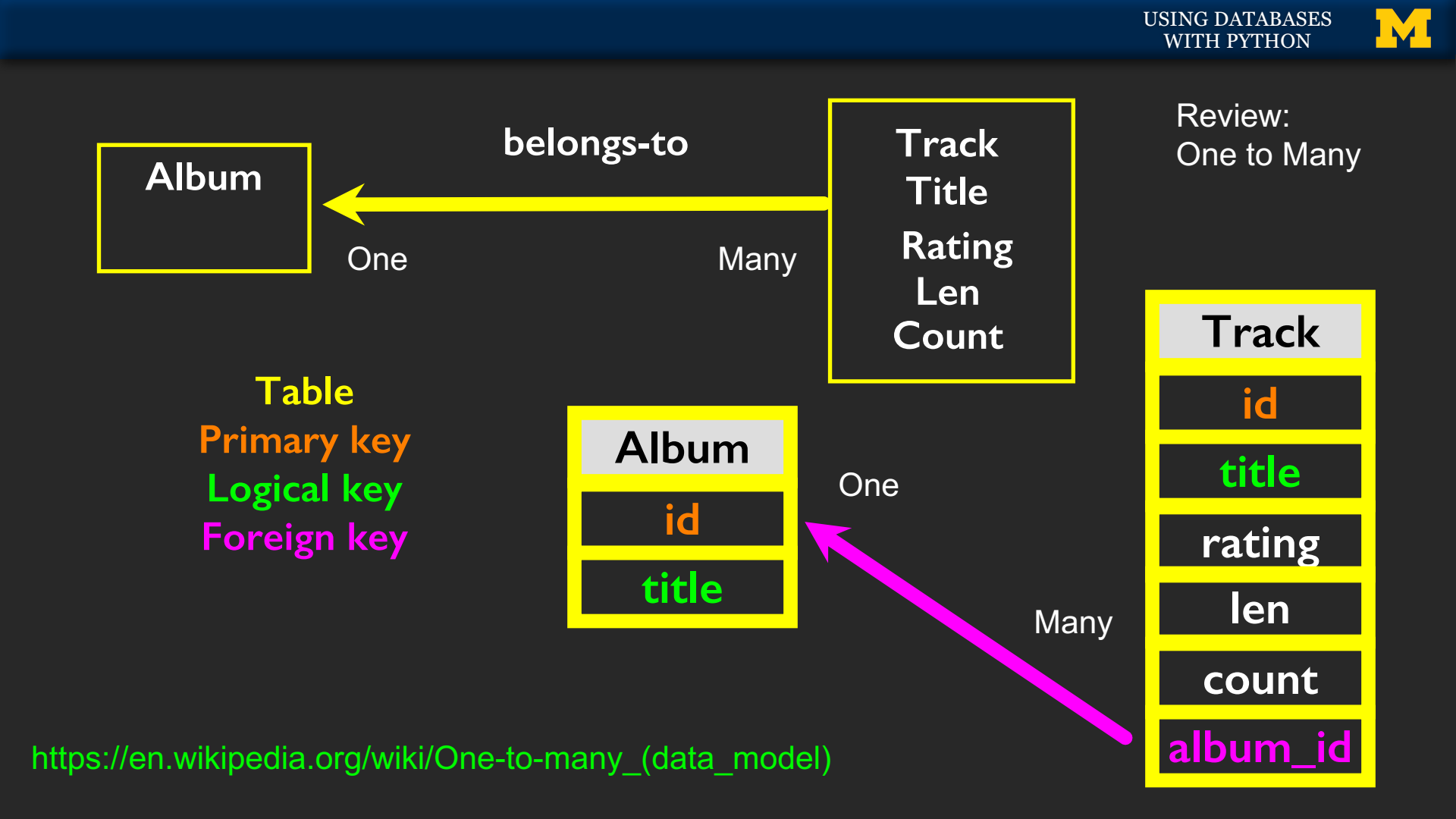
How the tables are linked

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	Danger Zone	Blues/R&B	★★★★★	18
Track 03	3:26	Billy Price	Danger Zone	Blues/R&B	★★★★★	22
Track 04						18
Track 05						21
War Pigs/Luke's Wall						25
Paranoid						22
Planet Caravan						25
Iron Man						26
Electric Funeral						22
Hand of Doom						23
Rat Salad						31
Jack the Stripper/Fairies Wear ...						24
Bomb Squad (TECH)						1
clay techno						2
Heavy						1
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

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)

One Many

id	name
Filter	Filter
1	Rock
2	Metal

One

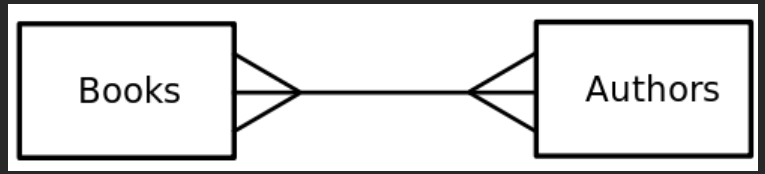
Many

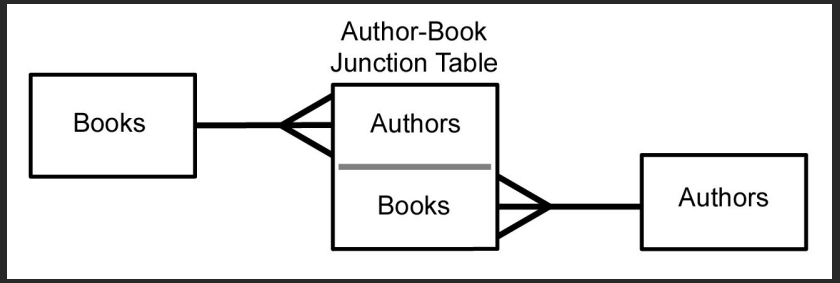
id	title	album_id	genre_id	len	rating	count
Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	Black Dog	2	1	297	5	0
2	Stairway	2	1	482	5	0
3	About to Rock	1	2	313	5	0
4	Who Made Who	1	2	207	5	0

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

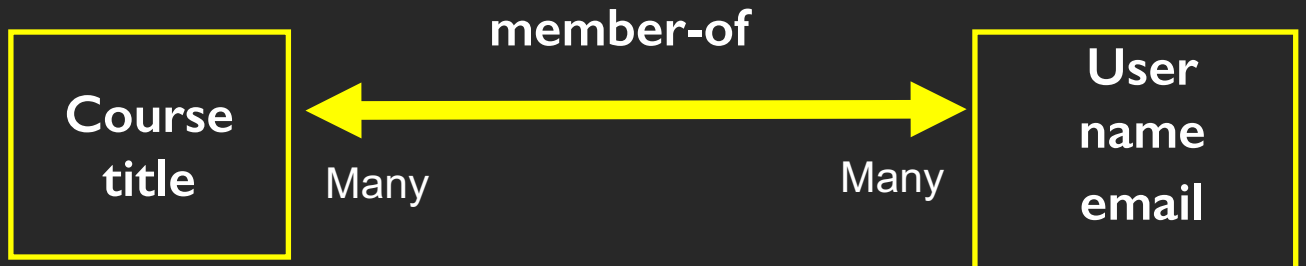
Many to Many

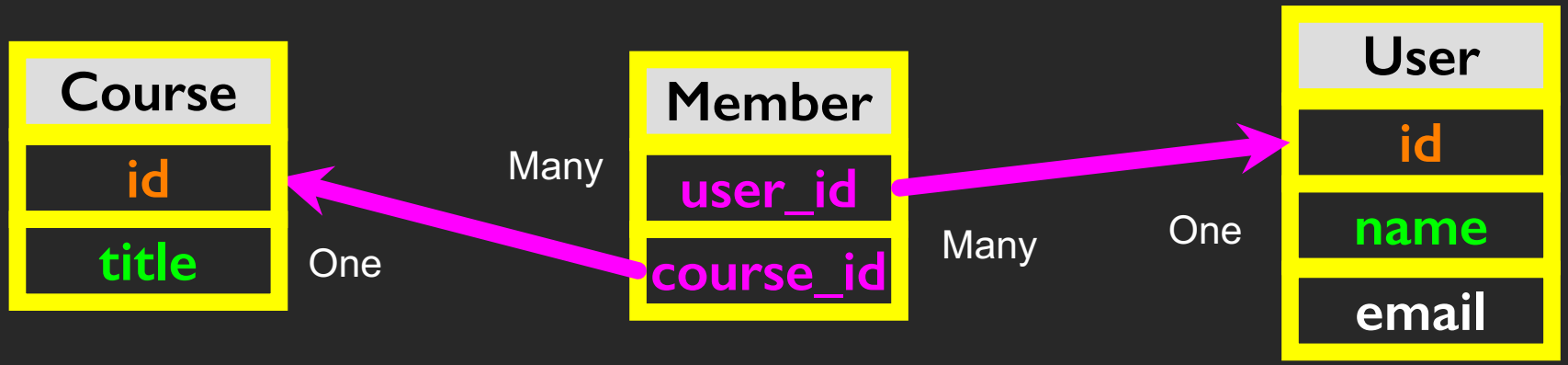
- Sometimes we need to model a relationship that is many-to-many
- We need to add a "connection" table with two foreign keys
- There is usually no separate primary key





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

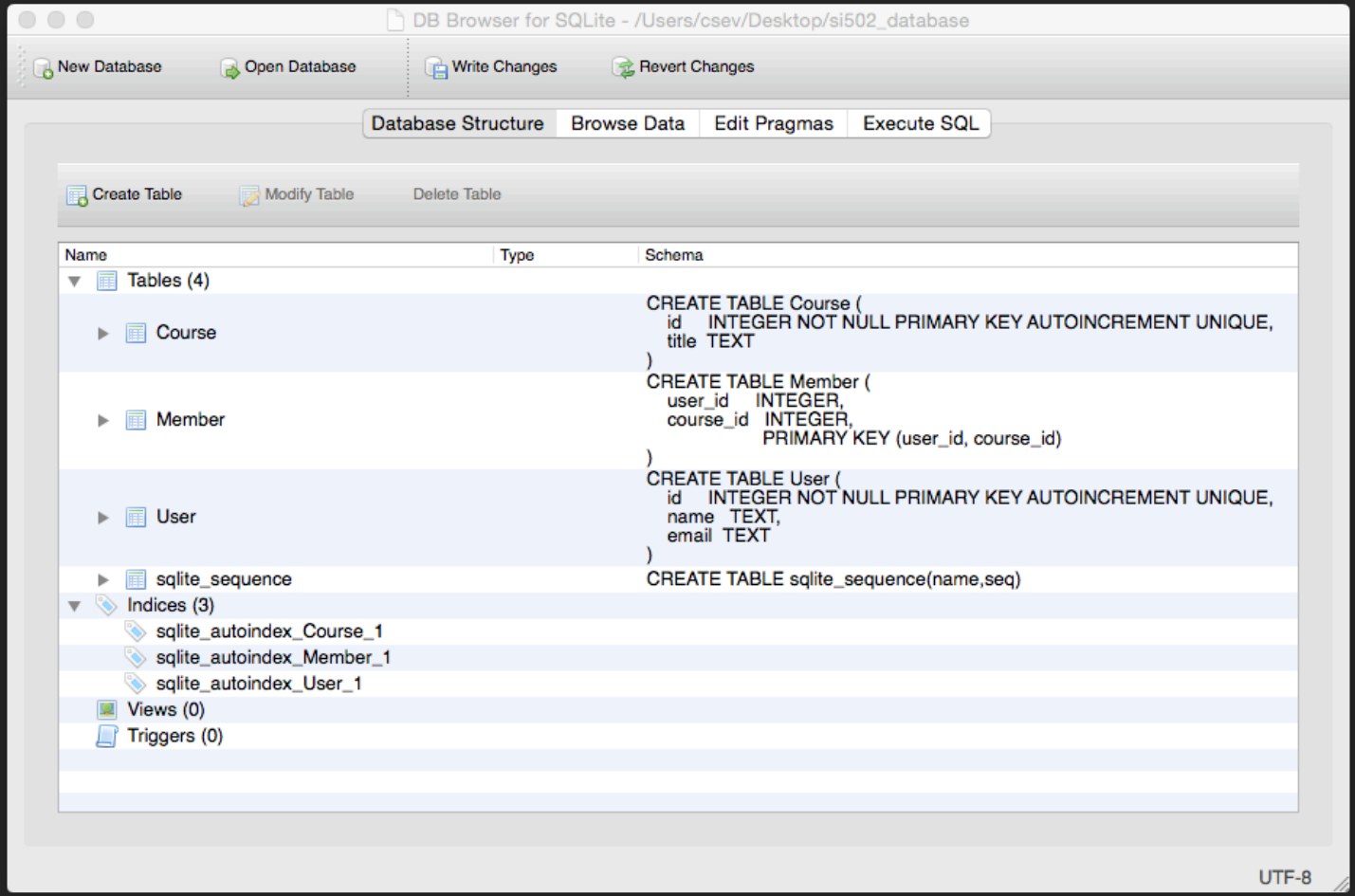




https://en.wikipedia.org/wiki/One-to-many_(data_model)

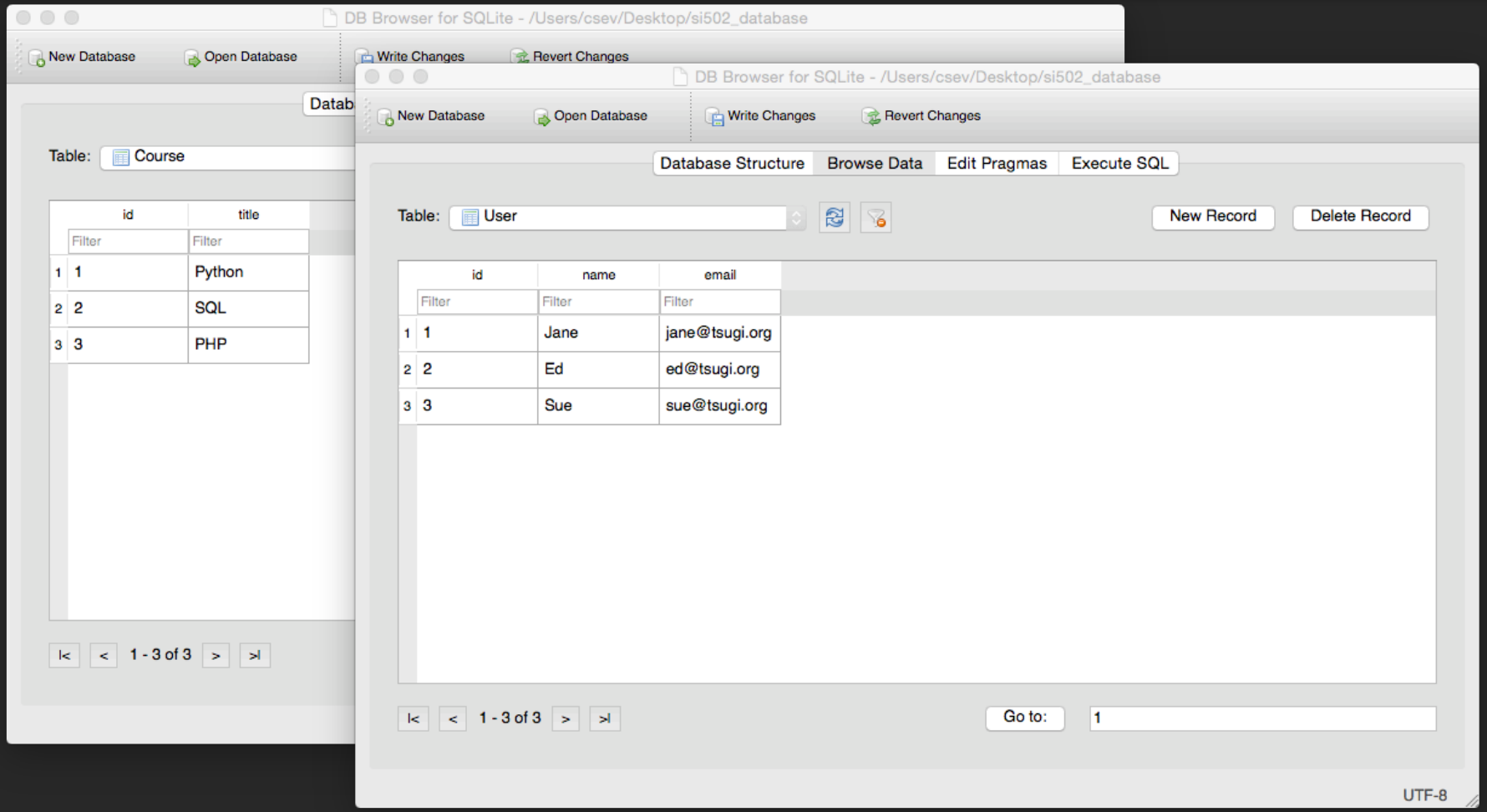
Start with a Fresh Database

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

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

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


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');
```

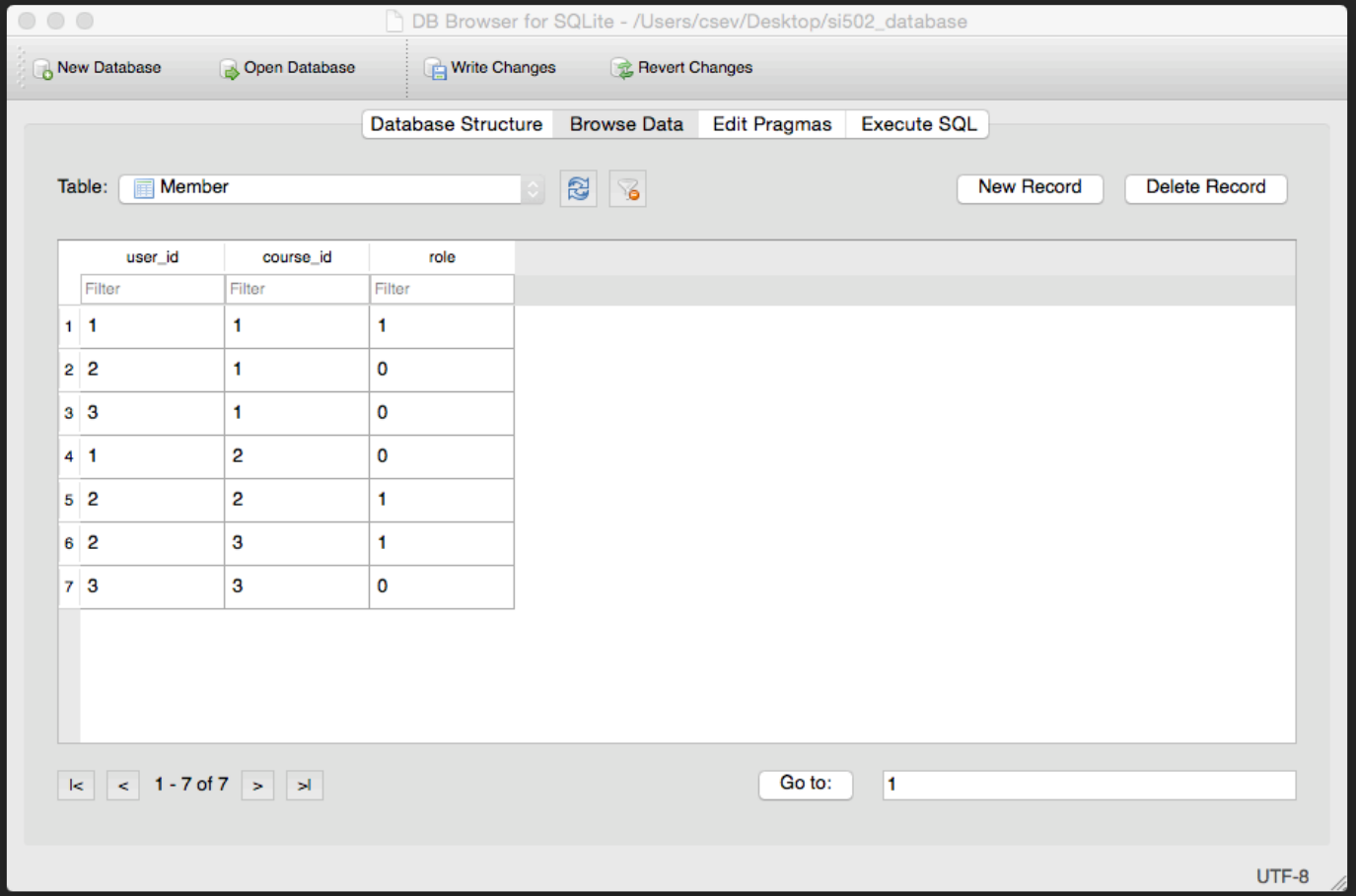

DB Browser for SQLite - /Users/csev/Desktop/si502_database
New Database
Open Database
Write Changes
Revert Changes
Table: Course
id
title
Filter
1 1 Python
2 2 SQL
3 3 PHP
1 - 3 of 3
DB Browser for SQLite - /Users/csev/Desktop/si502_database
New Database
Open Database
Write Changes
Revert Changes
Database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

Insert Memberships

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

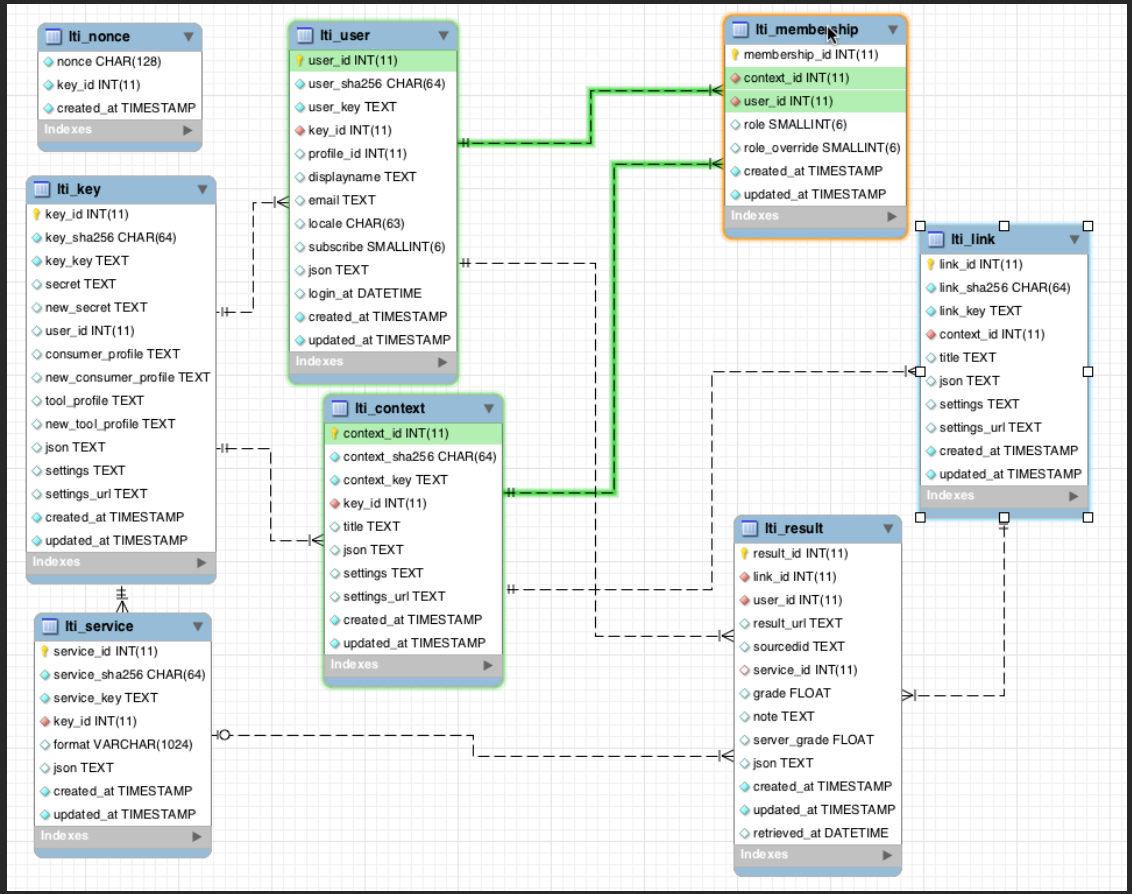
id	name	email
Filter	Filter	Filter
1	Jane	jane@tsugi.org
2	Ed	ed@tsugi.org
3	Sue	sue@tsugi.org

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

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

```
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
```


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_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
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

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

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

Acknowledgements / Contributions

Initial Development: Charles Severance, University of Michigan School of Information

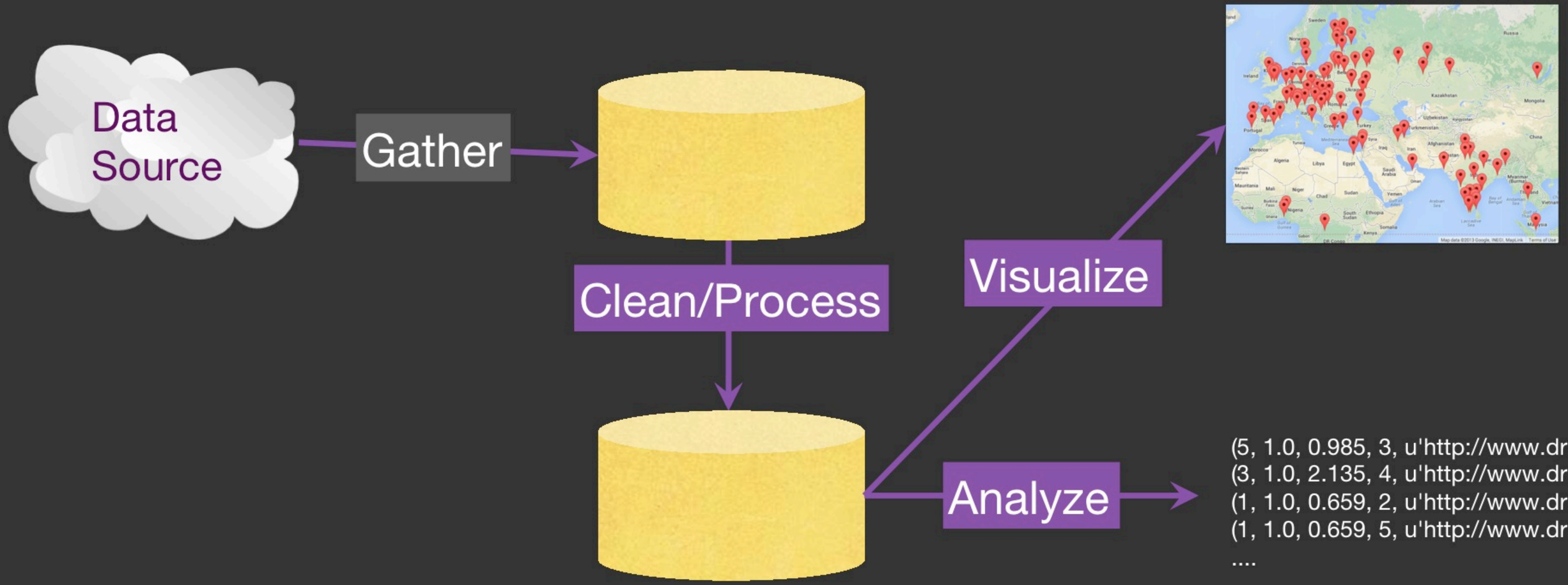
… Insert new Contributors here

Retrieving and Visualizing Data

Charles Severance

Python for Everybody
www.py4e.com

Multi-Step Data Analysis



Many Data Mining Technologies

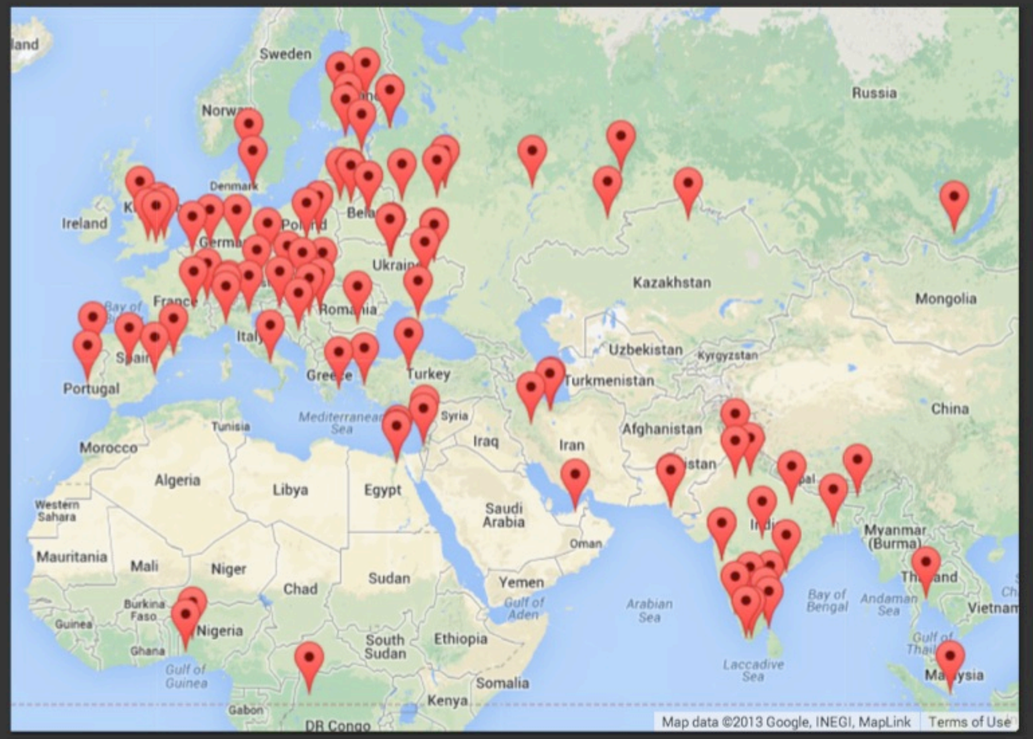
- https://hadoop.apache.org/
- http://spark.apache.org/
- https://aws.amazon.com/redshift/
- http://community.pentaho.com/
- ....

"Personal Data Mining"

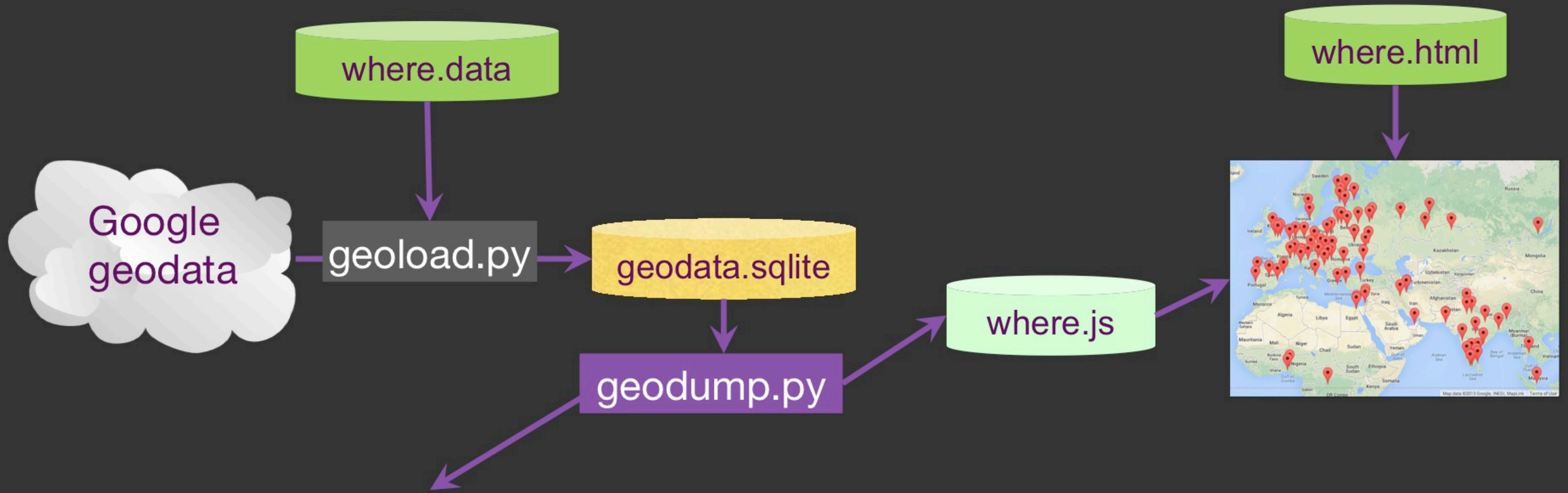
- Our goal is to make you better programmers – not to make you data mining experts

GeoData

- Makes a Google Map from user entered data
- Uses the Google Geodata API
- Caches data in a database to avoid rate limiting and allow restarting
- Visualized in a browser using the Google Maps API



http://www.py4e.com/code3/geodata.zip




```
Northeastern University, ... Boston, MA 02115, USA 42.3396998 -71.08975
Bradley University, 1501 ... Peoria, IL 61625, USA 40.6963857 -89.6160811
...
Technion, Viazman 87, Kesalsaba, 32000, Israel 32.7775 35.0216667
Monash University Clayton ... VIC 3800, Australia -37.9152113 145.134682
Kokshetau, Kazakhstan 53.2833333 69.3833333
...
12 records written to where.js
Open where.html to view the data in a browser
```

http://www.py4e.com/code3/geodata.zip

Search Engine and PageRank

Acknowledgements / Contributions

Initial Development: Charles Severance, University of Michigan School of Information

… Insert new Contributors here

...