Relational Databases

Charles Severance

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 relatinships between those tables involved in the query.

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

SQLite Database Browser

- SQLite is a very popular browser - it is free and fast and small
- We have a program to manipulate SQLite databases
- http://sqlitebrowser.sourceforge.net/
- SQLite is embedded in Python and a number of other languages

SQLite is in lots of software...

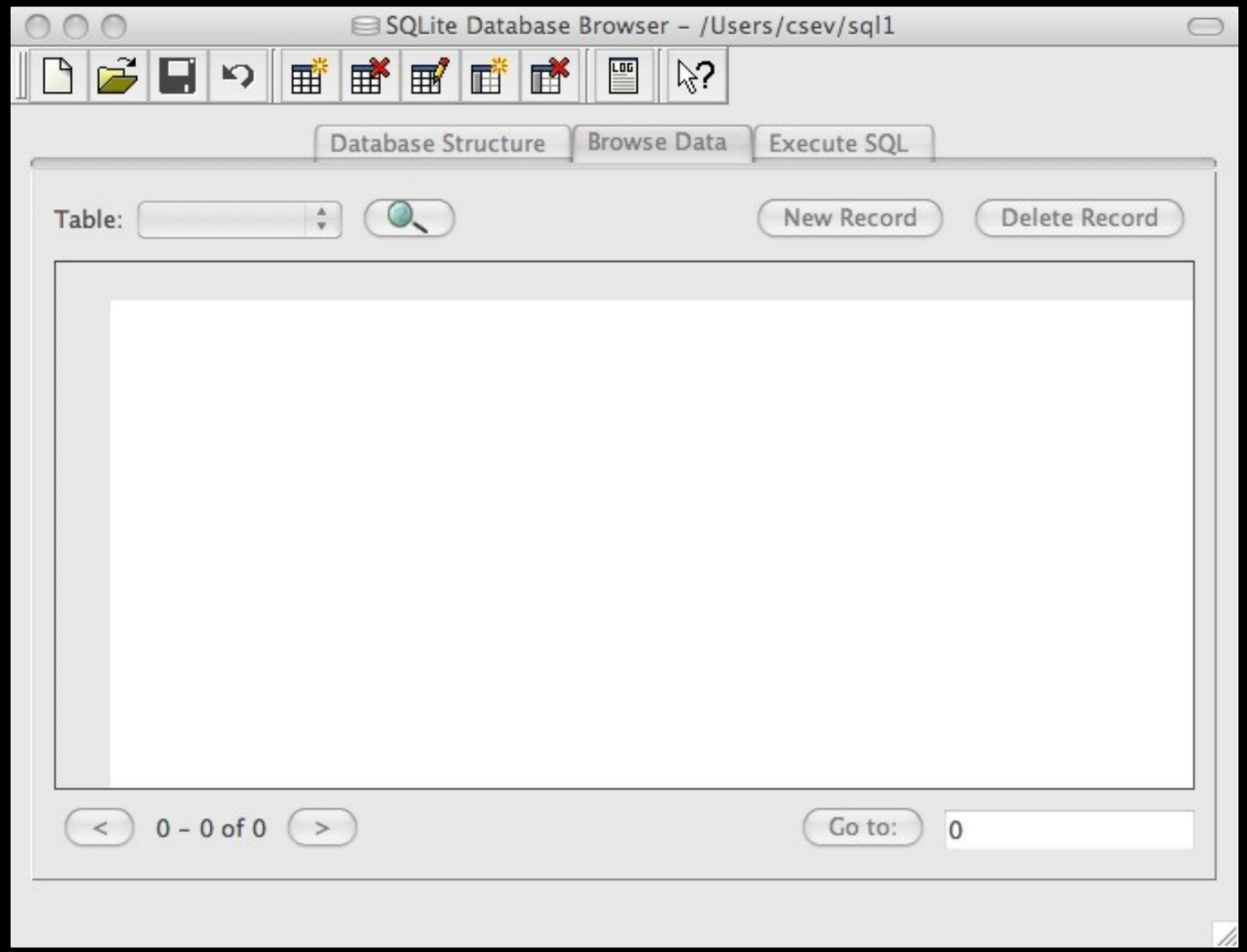
Symbian Python Philips Skype

GE Microsoft McAfee Apple

Adobe Firefox PHP Toshiba Sun Microsystems

Google

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



- http://sqlitebrowser.sourceforge.net/

Source: SQLite Terminal

Start Simple - A Single Table

- Lets make a table of People - with a Name and an E-Mail

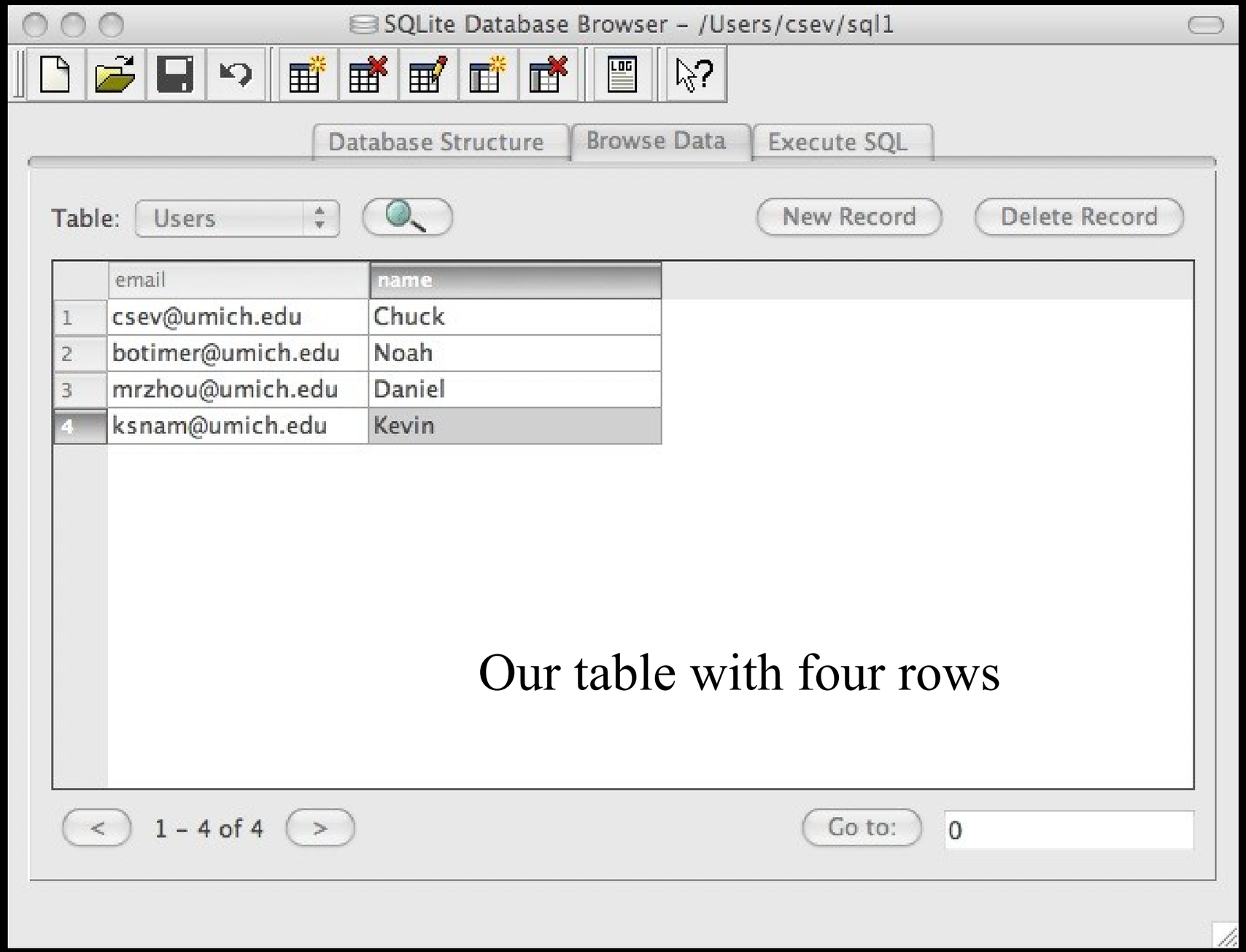
SQLite Database Browser – /Users/csev/sql1

Database Structure | Browse Data | Execute SQL

Name	Object	Type	Schema
Users	table		CREATE TABLE Users (email TEXT, name TEXT)
email	field	TEXT	
name	field	TEXT	

Our first table with two columns

Source: SQLite Terminal



Source: SQLite Terminal

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 ('Ted', 'ted@umich.edu')
```

SQLite Database Browser – /Users/csev/sql1

Database Structure | Browse Data | Execute SQL

SQL string:

```
insert into Users (name, email) values ('Ted', 'ted@umich.edu')
```

Execute query

Error message from database engine:

No error

Data returned:

SQLite Database Browser – /Users/csev/sql1

Database Structure | Browse Data | Execute SQL

Table: Users

New Record | Delete Record

	email	name
1	csev@umich.edu	Chuck
2	botimer@umich.edu	Noah
3	mrzhou@umich.edu	Daniel
4	ksnam@umich.edu	Kevin
5	ted@umich.edu	Ted

< 1 – 5 of 5 >

Go to: 0

Sources: SQLite Terminal

SQL Delete

- Deletes a row in a table based on a selection criteria

```
delete from Users where email='ted@umich.edu'
```

SQLite Database Browser – /Users/csev/sql1

Database Structure | Browse Data | Execute SQL

SQL string:

```
delete from Users where email='ted@umich.edu'
```

Execute query

Error message from database engine:

No error

Data returned:

SQLite Database Browser – /Users/csev/sql1

Database Structure | Browse Data | Execute SQL

Table: Users

New Record | Delete Record

	email	name
1	csev@umich.edu	Chuck
2	botimer@umich.edu	Noah
3	mrzhou@umich.edu	Daniel
4	ksnam@umich.edu	Kevin

< 1 – 4 of 4 >

Go to: 0

Sources: SQLite Terminal

SQL: Update

- Allows the updating of a field with a where clause

```
update Users set name="Charles" where email='csev@umich.edu'
```

SQLite Database Browser – /Users/csev/sql1

Database Structure | Browse Data | Execute SQL

SQL string:

```
update Users set name="Charles" where email='csev@umich.edu'
```

Execute query

Error message from database engine:

No error

Data returned:

SQLite Database Browser – /Users/csev/sql1

Database Structure | Browse Data | Execute SQL

Table: Users

New Record | Delete Record

	email	name
1	csev@umich.edu	Charles
2	botimer@umich.edu	Noah
3	mrzhou@umich.edu	Daniel
4	ksnam@umich.edu	Kevin

< 1 – 4 of 4 >

Go to: 0

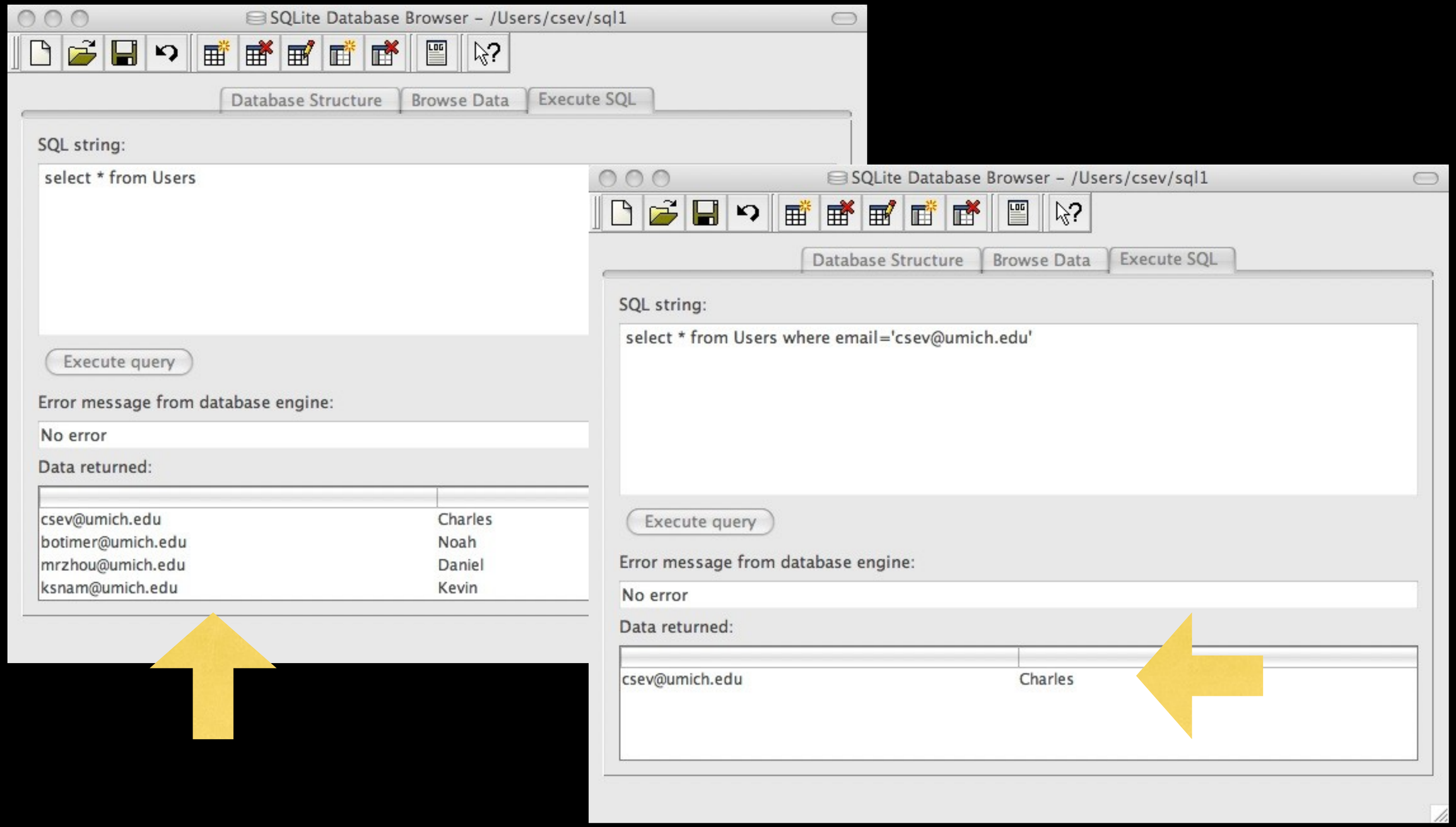
Sources: SQLite Terminal

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'
```



Sources: SQLite Terminal

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
```

SQLite Database Browser – /Users/csev/sql1

Database Structure | Browse Data | Execute SQL

SQL string:

```
select * from Users order by name
```

Execute query

Error message from database engine:

No error

Data returned:

csev@umich.edu	Charles
mrzhou@umich.edu	Daniel
ksnam@umich.edu	Kevin
botimer@umich.edu	Noah

SQLite Database Browser – /Users/csev/sql1

Database Structure | Browse Data | Execute SQL

SQL string:

```
select * from Users order by email
```

Execute query

Error message from database engine:

No error

Data returned:

botimer@umich.edu	Noah
csev@umich.edu	Charles
ksnam@umich.edu	Kevin
mrzhou@umich.edu	Daniel

Sources: SQLite Terminal

SQL Summary

insert into Users (name, email) values (‘Ted’, ‘ted@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 spreadsheet with rows, columns, and commands

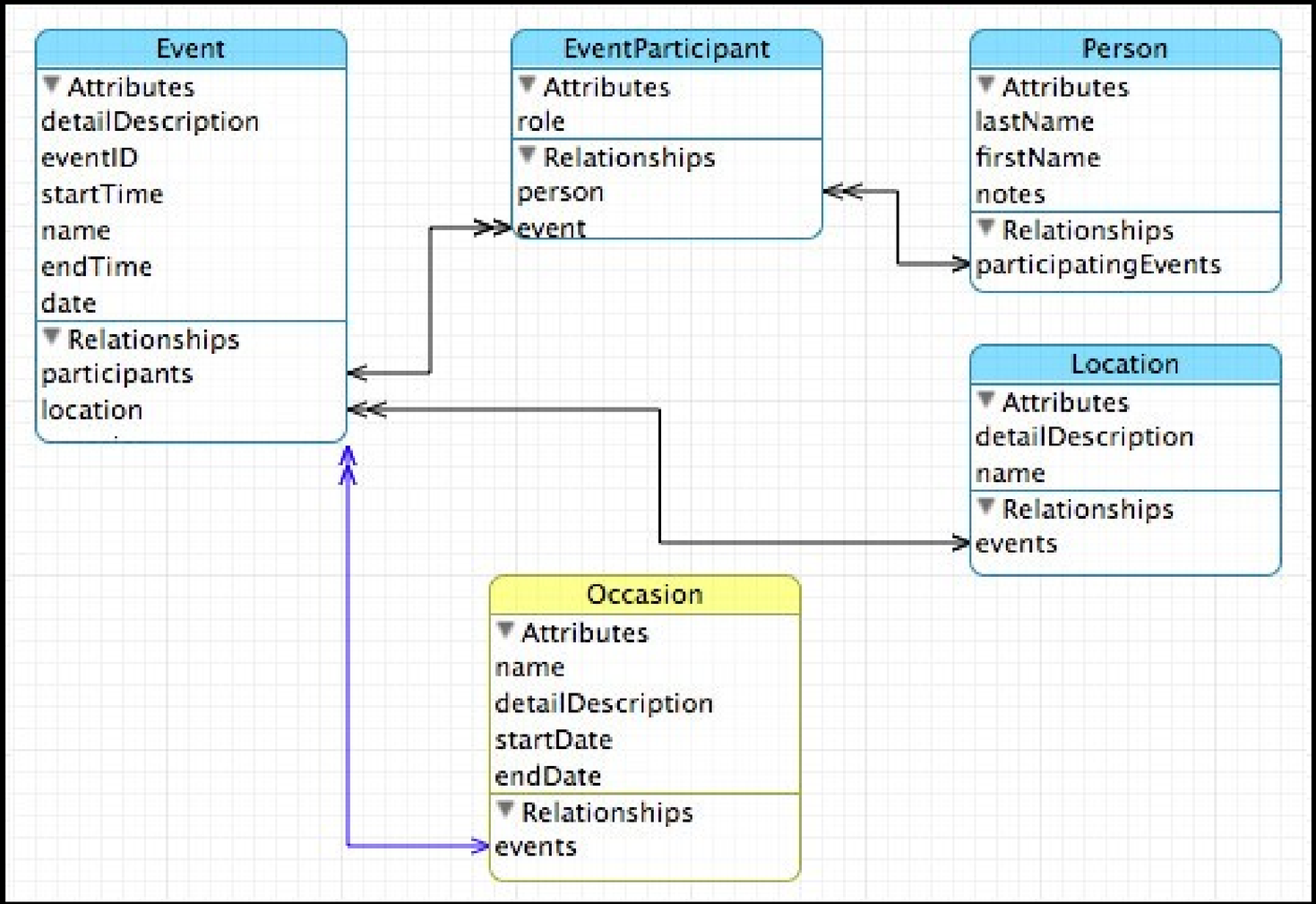
- The power comes when we have more than one table and we can exploit the relationships between the tables

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

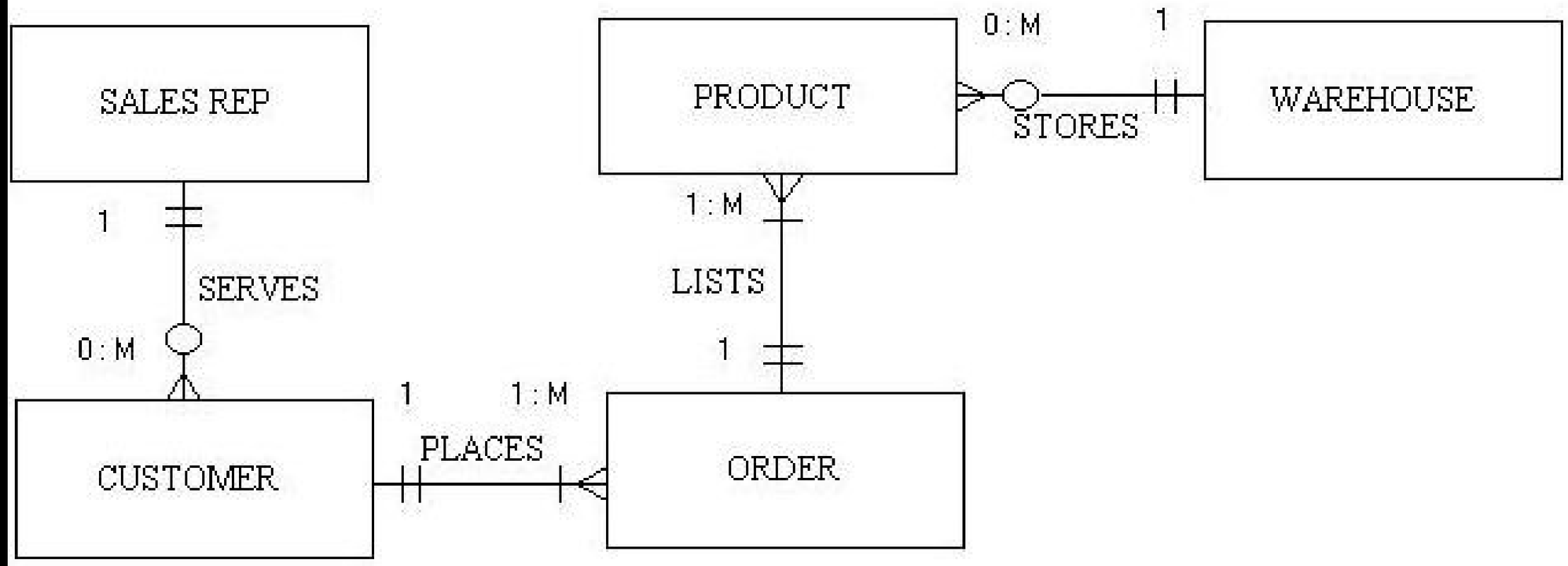
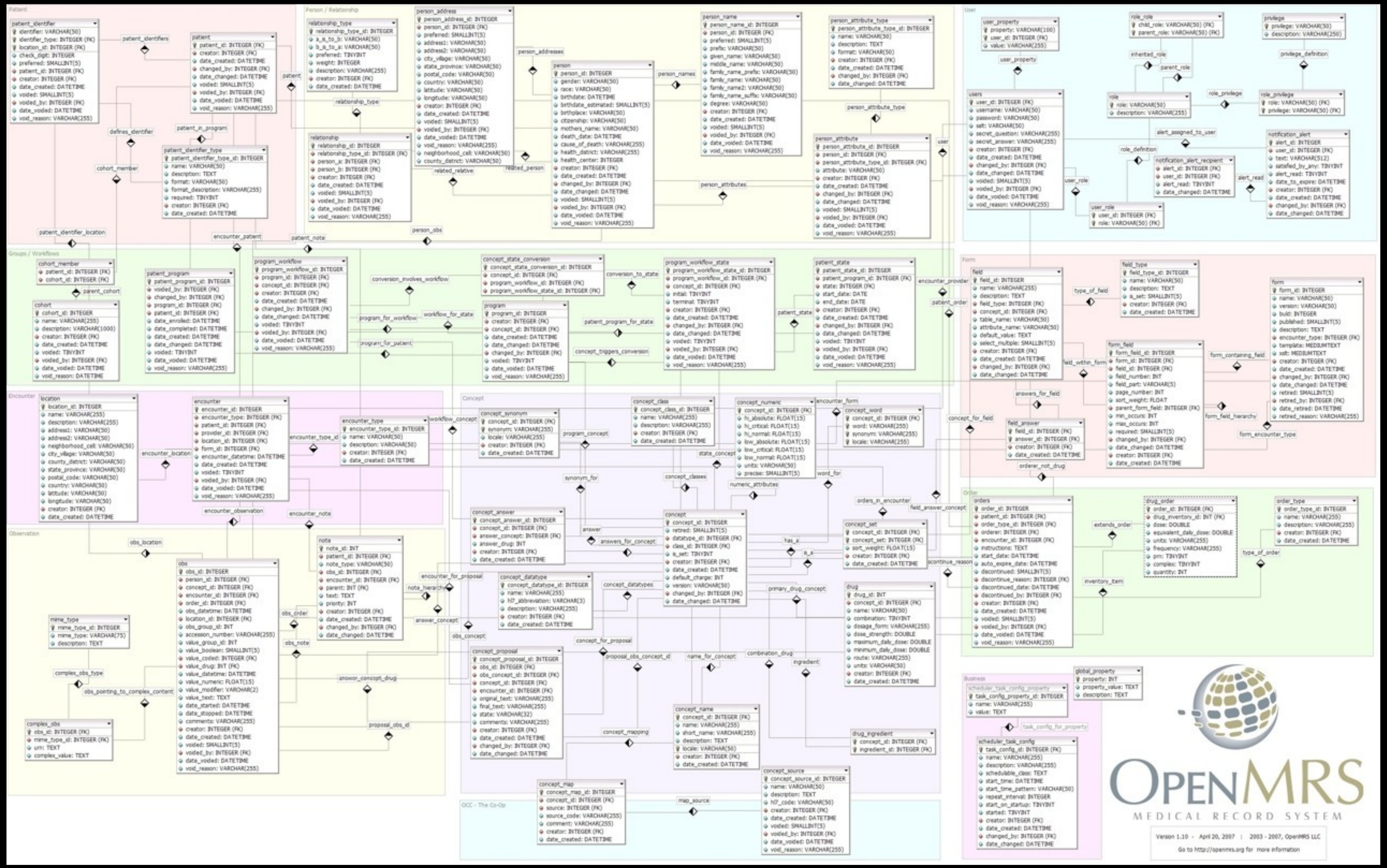


Figure 1. Entity-Relationship Diagram

* 1 INSTANCE OF A SALES REP SERVES 1 TO MANY CUSTOMERS
* 1 INSTANCE OF A CUSTOMER PLACES 1 TO MANY ORDERS
* 1 INSTANCE OF AN ORDER LISTS 1 TO MANY PRODUCTS
* 1 INSTANCE OF A WAREHOUSE STORES 0 TO MANY PRODUCTS


Patient
Person / Relationship
User
Groups / Workflows
Form
Encounter
Concept
Observation
Order
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

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

Source: Apple iTunes Terminal

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

Source: Apple iTunes Terminal

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

Track

Artist

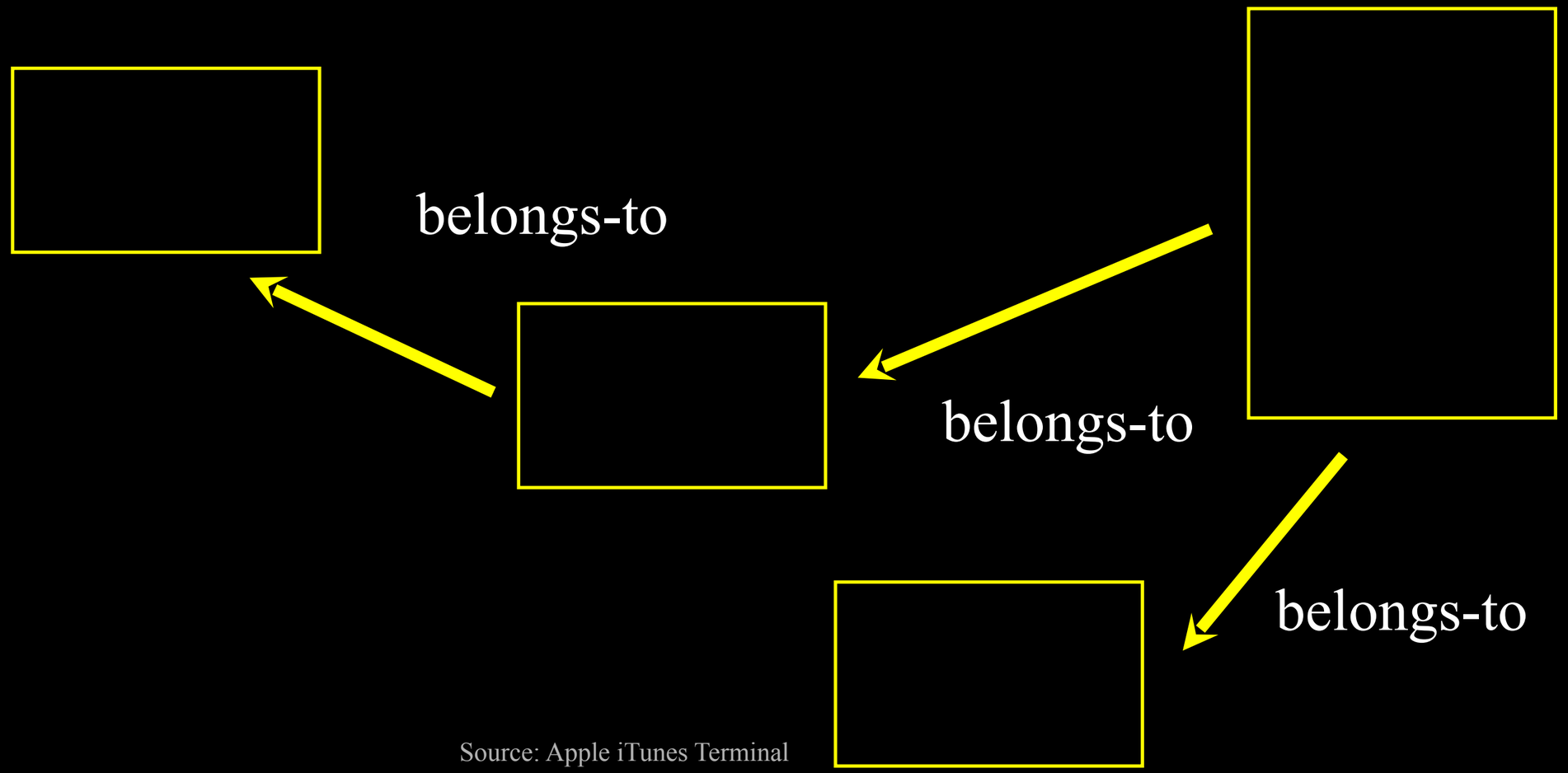
Album

Genre

Rating

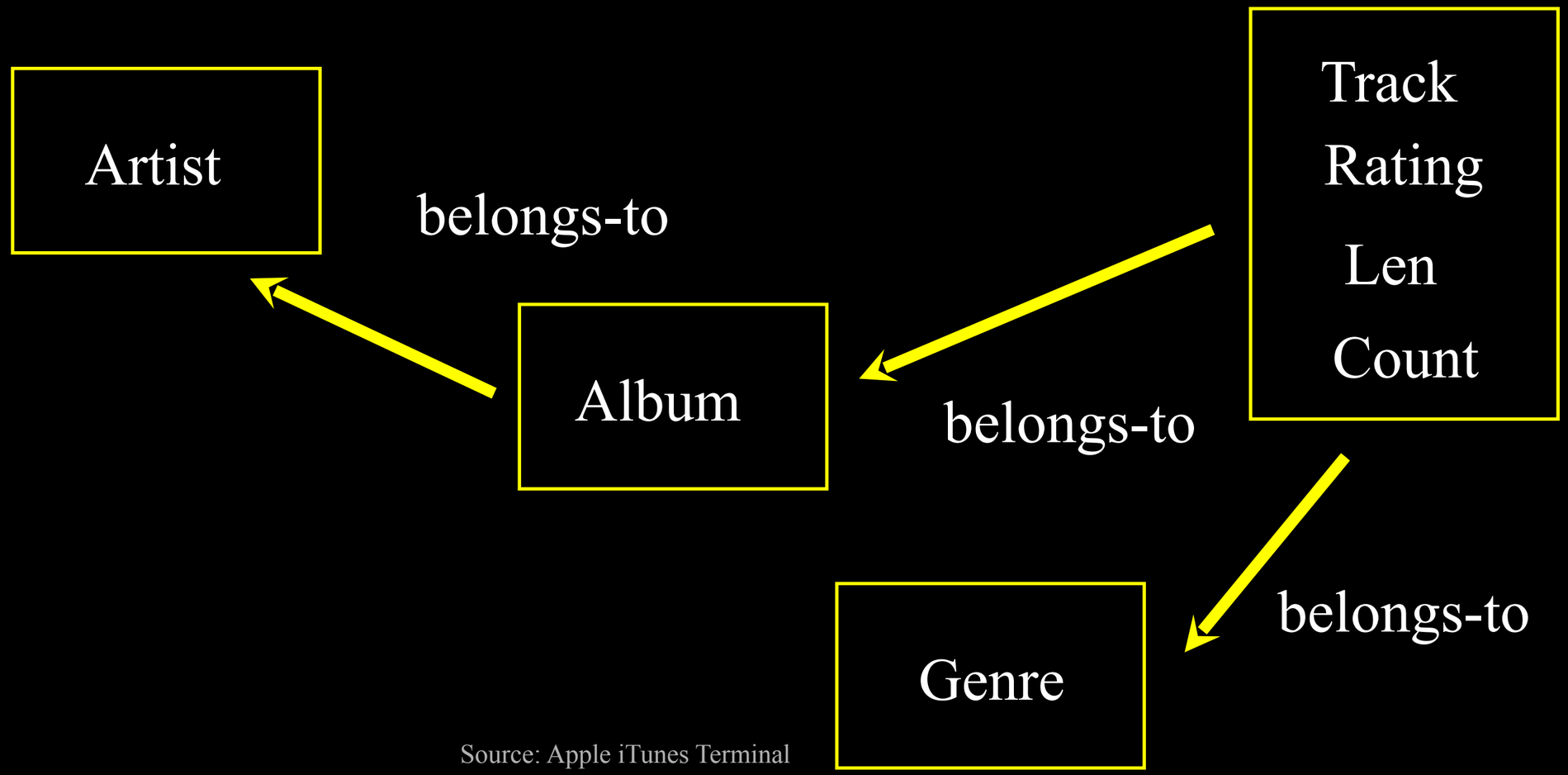
Len

Count



Source: Apple iTunes Terminal

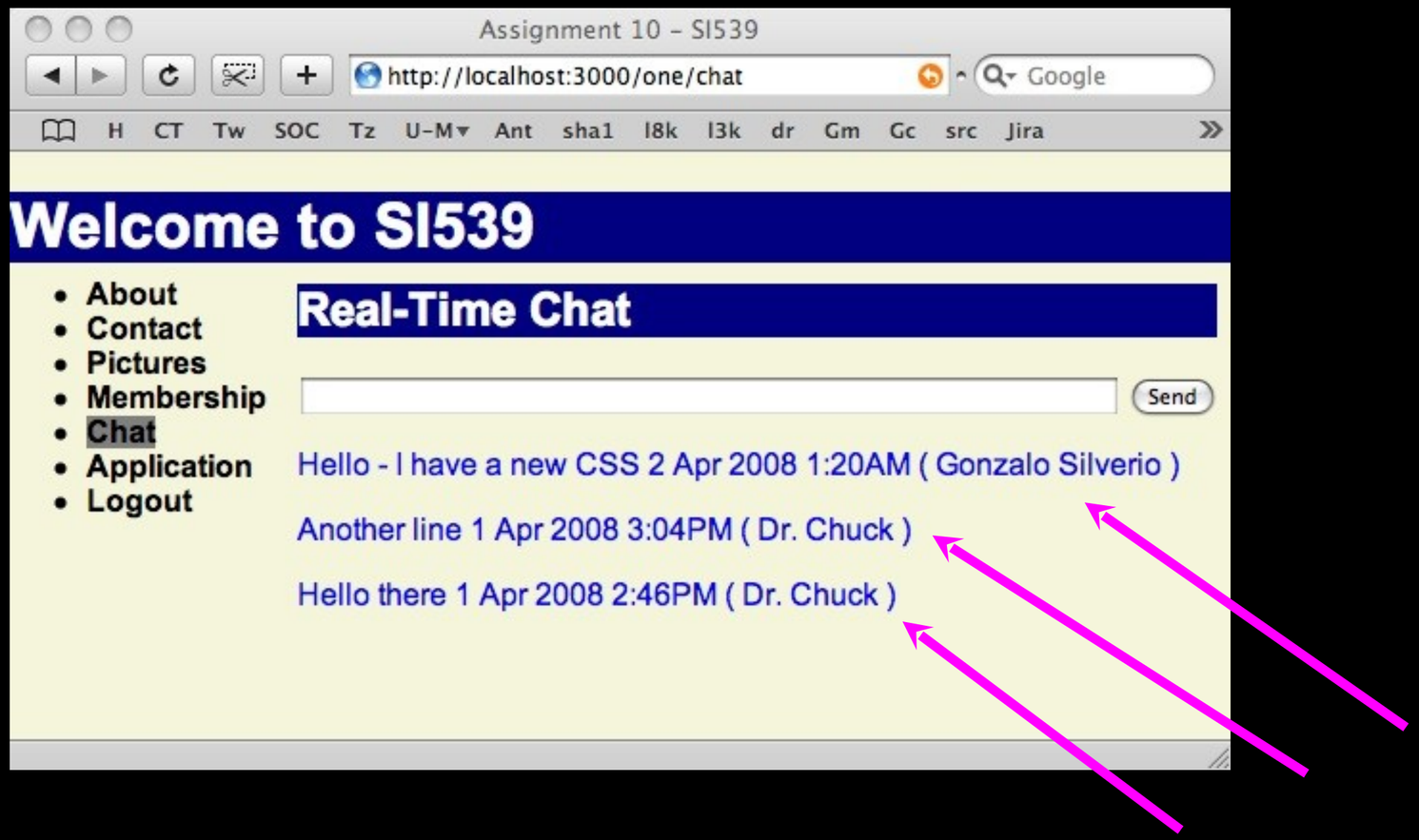
Hells Bells	5:13	AC/DC	Who Made Who	Rock	★★★★★	61
Shake Your Foundations	3:54	AC/DC	Who Made Who	Rock	★★★★★	70
Chase the Ace	3:01	AC/DC	Who Made Who	Rock		56
For Those About To Rock (We …	5:54	AC/DC	Who Made Who	Rock	★★★★★	61
Dúlamán	3:43	Altan	Natural Wonders M…	New Age		31
Rode Across the Desert	4:10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



Source: Apple iTunes Terminal

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	3:20	America	Greatest Hits	Easy Listen…	★★★★★	23

Representing Relationships in a Database



We want to keep track of who is the "owner" of each chat message...
Who does this chat message "belong to"???

Source: CTools http://ctools.umich.edu

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” to each table which we will reference - by convention many programmers call this “id”

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

Better Reference Pattern

We use integers to reference rows in another table.

Table: members

New Record

	id	name	email
1	1	Dr. Chuck	csev@umich.edu
2	2	Gonzalo Silverio	gsilver@umich.edu

Table: chats

New Record

	id	chatmsg	member_id	created_at
1	1	Hello there	1	2008-04-01 14
2	2	Another line	1	2008-04-01 15
3	3	Hello - I have a	2	2008-04-02 01

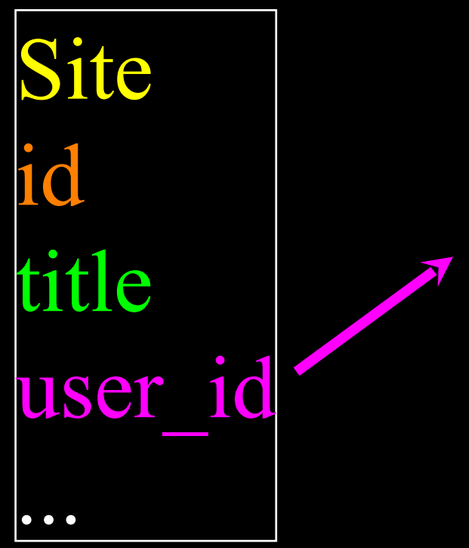
Sources: SQLite Terminal

Keys

Finding our way around....

Three Kinds of Keys

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



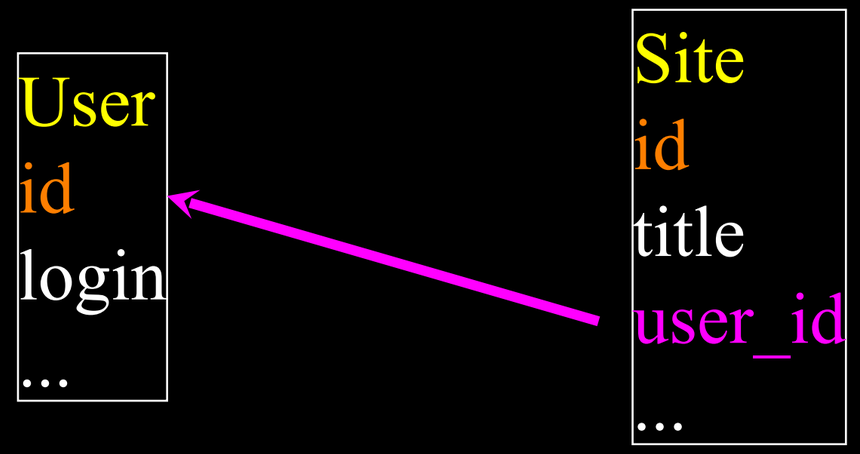
Primary Key Rules

- Rails enourages you to follow 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 far less efficient than integers performance-wise

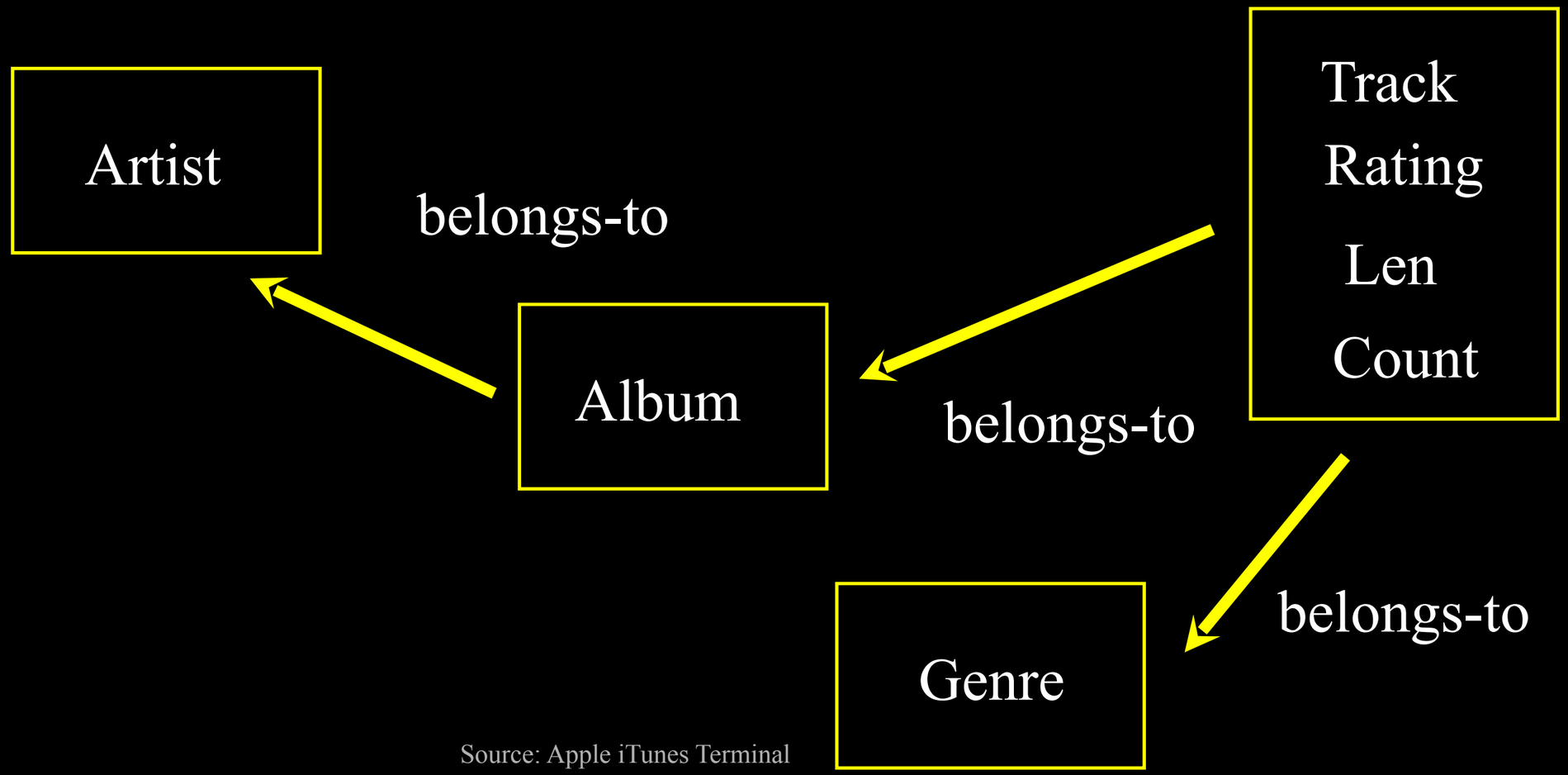
User
id
login
password
name
email
created_at
modified_at
login_at

Foreign Keys

- A foreign key is when a table has a column that contains a key which points the primary key of another table.
- When all primary keys are integers, then all foreign keys are integers - this is good - very good
- If you use strings as foreign keys - you show yourself to be an uncultured swine

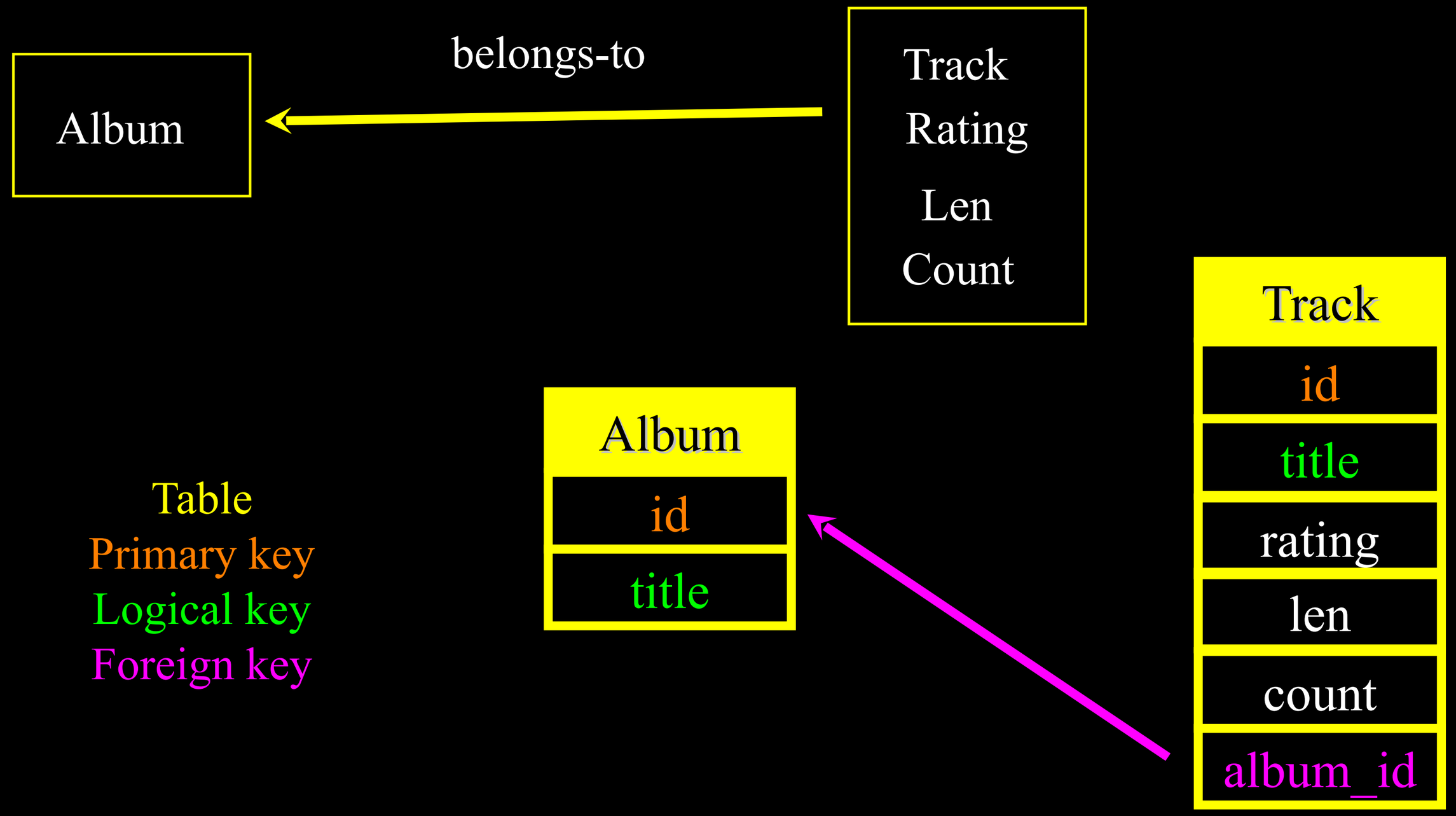


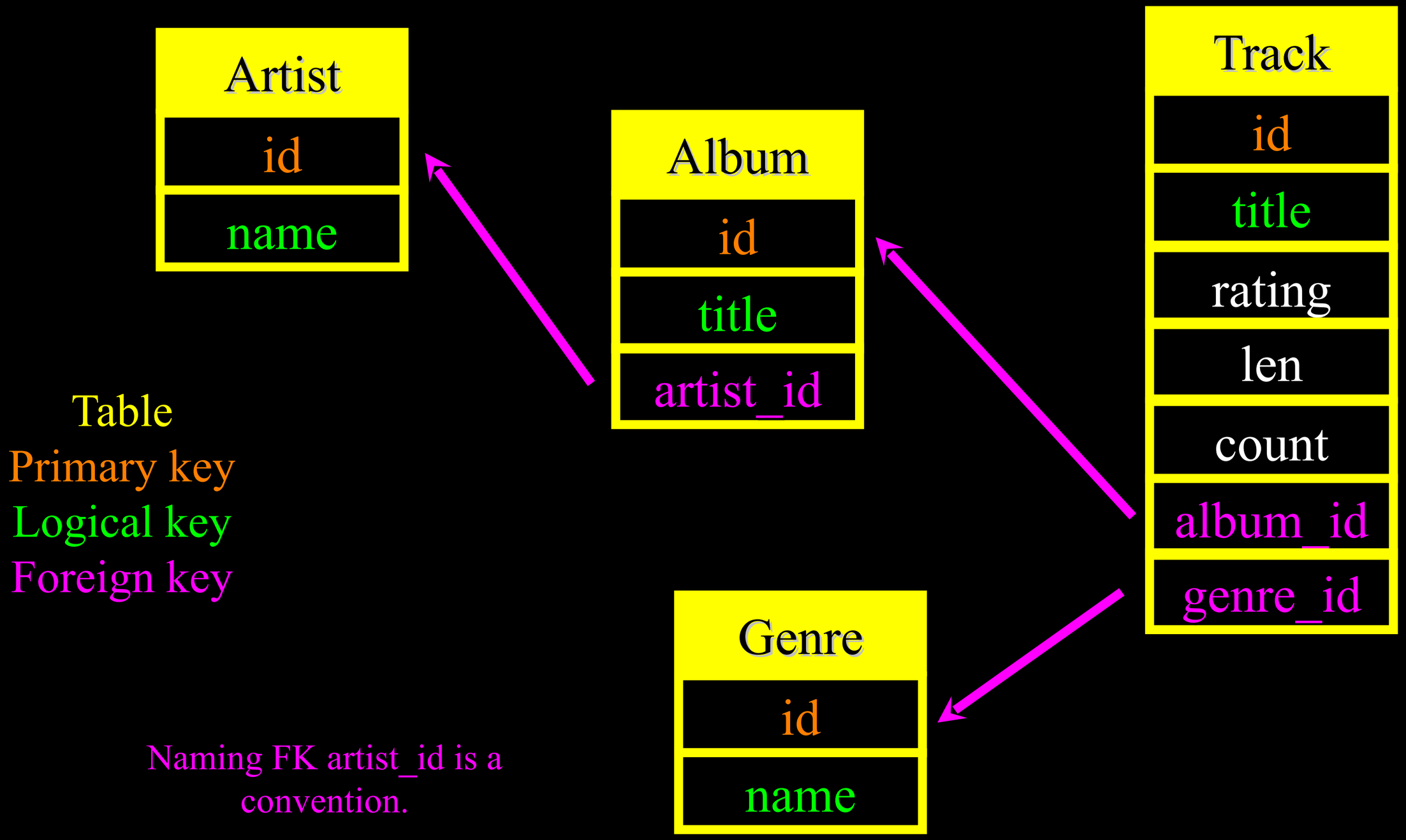
Relationship Building (in tables)

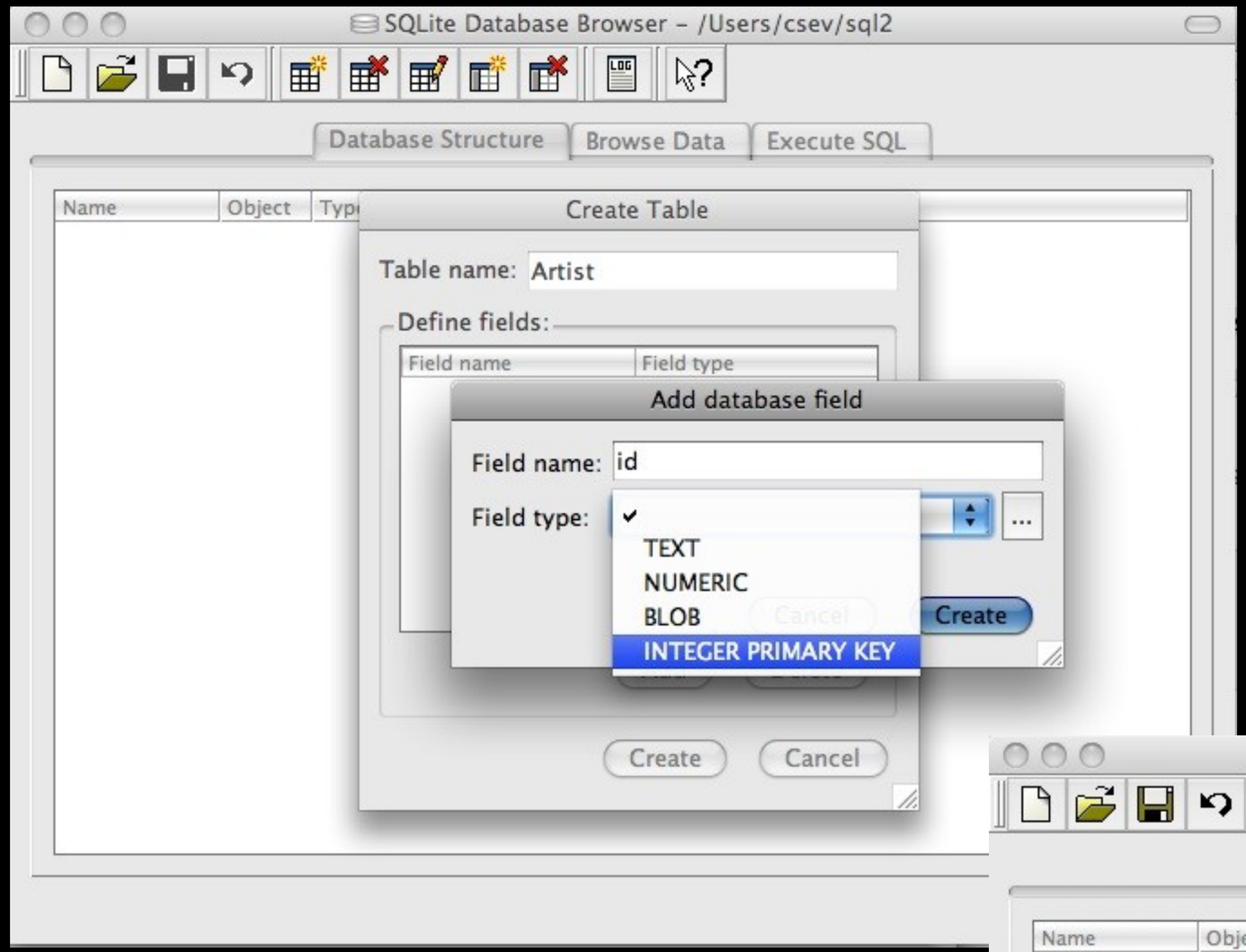


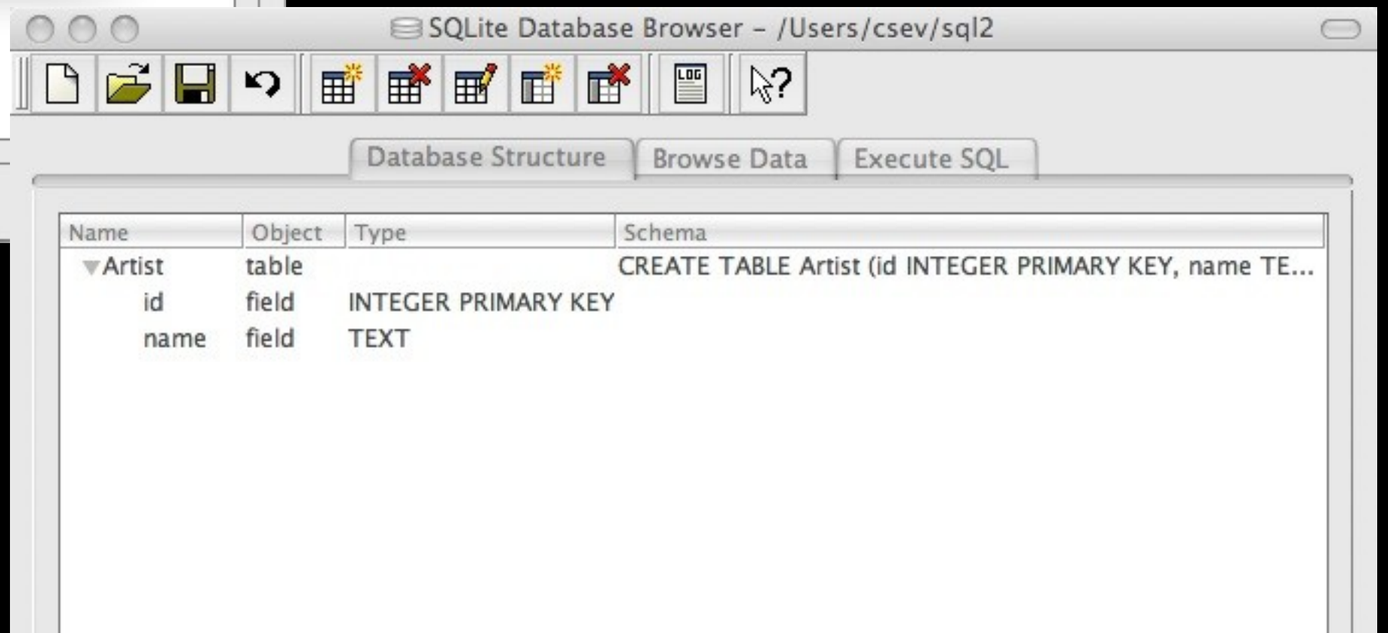
Source: Apple iTunes Terminal

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	3:20	America	Greatest Hits	Easy Listen…	★★★★★	23


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


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





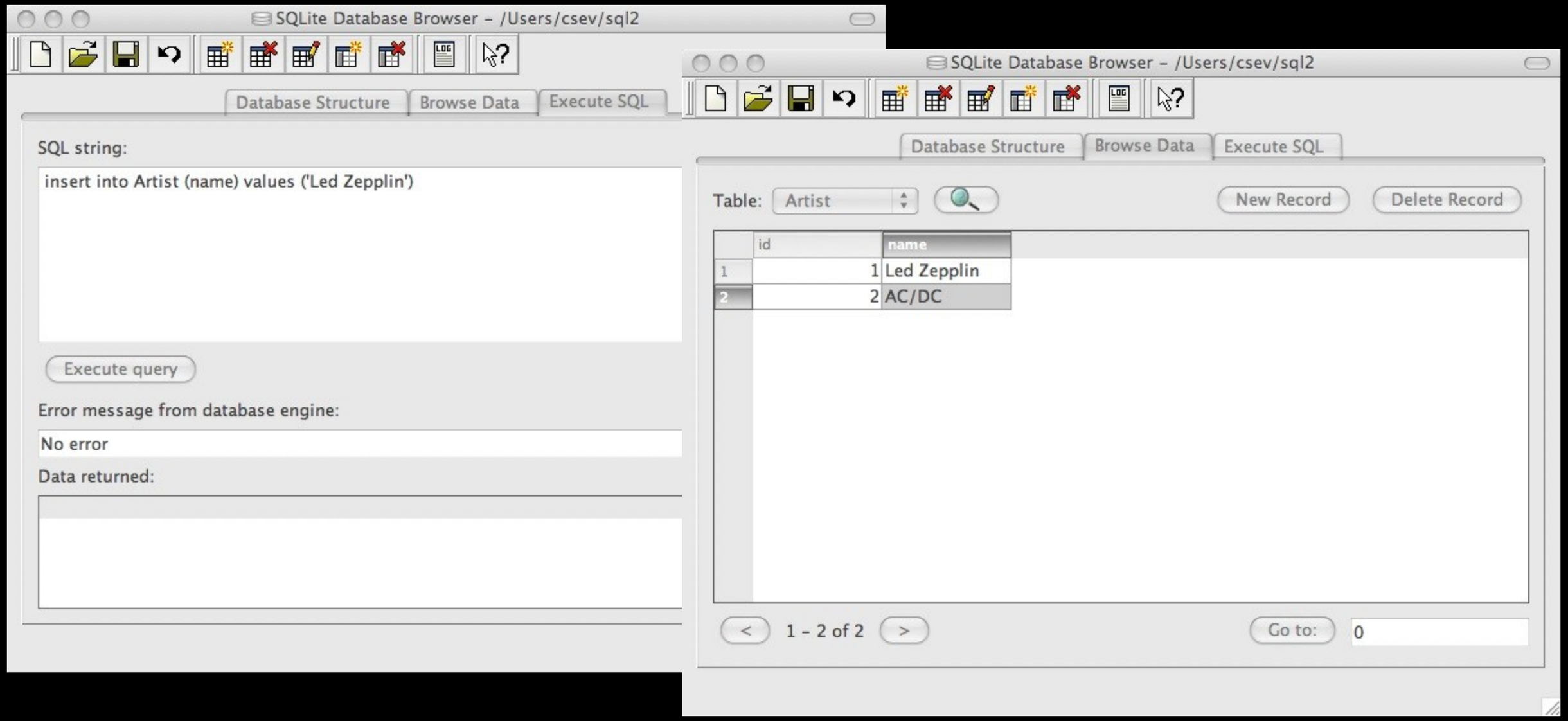
Sources: SQLite Terminal

SQLite Database Browser – /Users/csev/sql2

Database Structure | Browse Data | Execute SQL

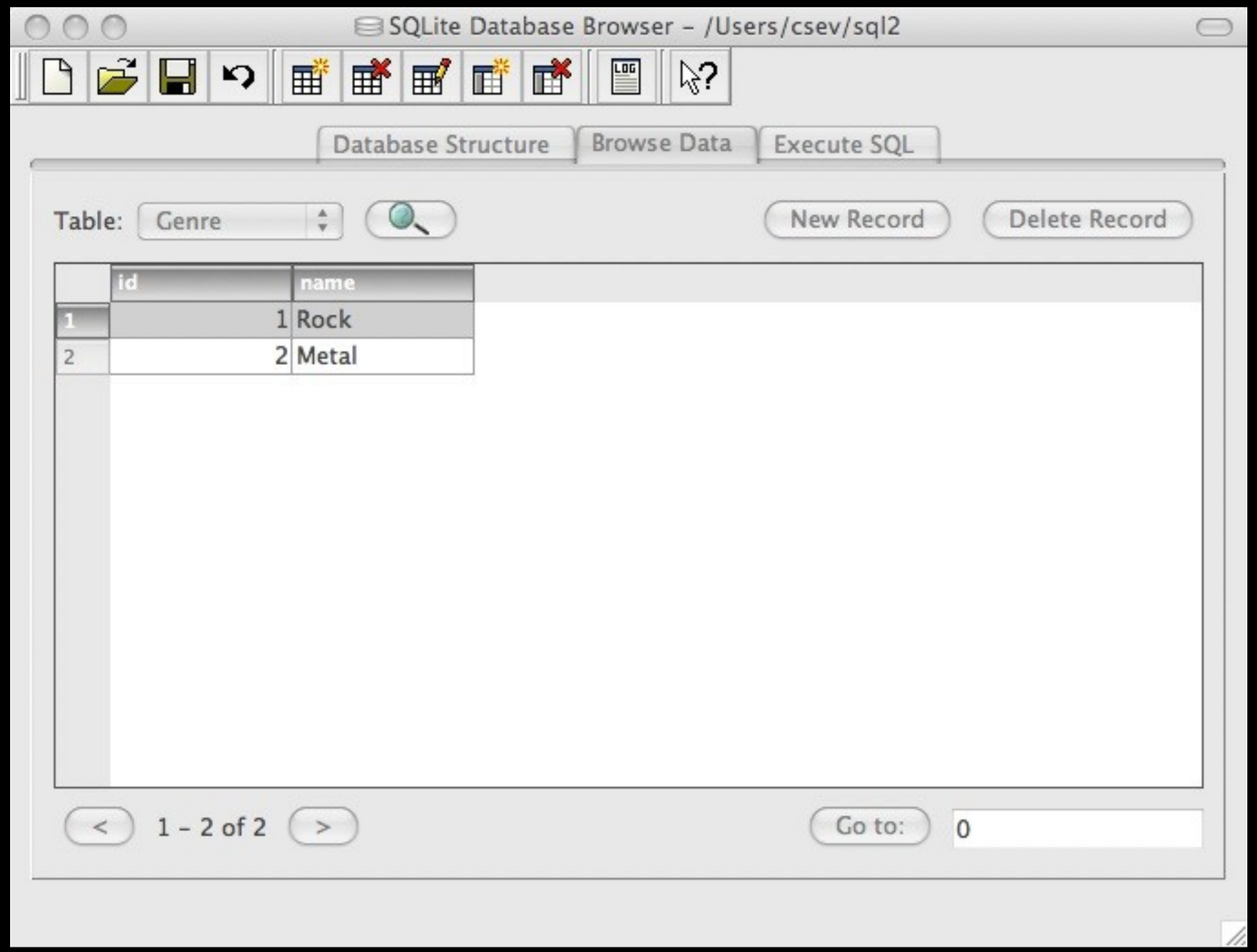
Name	Object	Type	Schema
Album	table		CREATE TABLE Album (artist_id NUMERIC, id INTEGER PRI...
artist_id	field	NUMERIC	
id	field	INTEGER PRIMARY KEY	
title	field	TEXT	
Artist	table		CREATE TABLE Artist (id INTEGER PRIMARY KEY, name TE...
id	field	INTEGER PRIMARY KEY	
name	field	TEXT	
Genre	table		CREATE TABLE Genre (id INTEGER PRIMARY KEY, name T...
id	field	INTEGER PRIMARY KEY	
name	field	TEXT	
Track	table		CREATE TABLE Track (album_id NUMERIC, count NUMERI...
album_id	field	NUMERIC	
count	field	NUMERIC	
genre_id	field	NUMERIC	
id	field	INTEGER PRIMARY KEY	
len	field	NUMERIC	
rating	field	NUMERIC	
title	field	TEXT	

Sources: SQLite Terminal



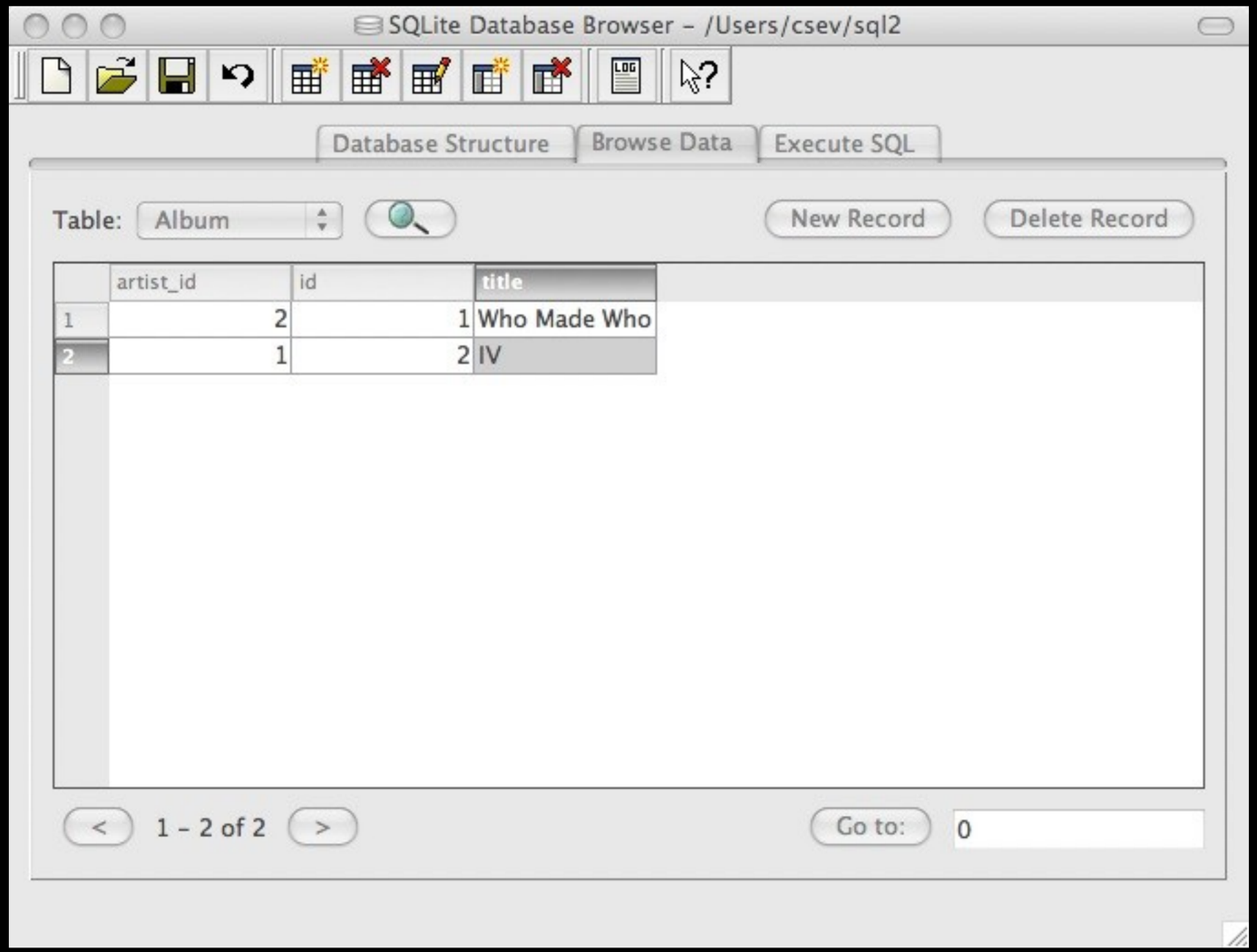
insert into Artist (name) values ('Led Zepplin')
insert into Artist (name) values ('AC/DC')

Sources: SQLite Terminal



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

Source: SQLite Terminal



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

Source: SQLite Terminal

```
insert into Track (title, rating, len, count, album_id, genre_id)
    values ('Black Dog', 5, 297, 0, 1, 1)
insert into Track (title, rating, len, count, album_id, genre_id)

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

Database Structure | Browse Data | Execute SQL

Table: Track

New Record | Delete Record

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

Source: SQLite Terminal

We have relationships!

Table: Track

New Record | Delete Record

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

Table: Album

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

Table: Artist

	id	name
1	1	Led Zepplin
2	2	AC/DC

Table: Genre

	id	name
1	1	Rock
2	2	Metal

Sources: SQLite Terminal

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 the keys which make the connection between the tables using an ON clause

Table: Track

New Record | Delete Record

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

Table: Genre

	id	name
1	1	Rock
2	2	Metal

Data returned:

Black Dog	Rock
Stairway	Rock
About to Rock	Metal
Who Made Who	Metal

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

What we want to see

The tables which hold the data

How the tables are linked

Sources: SQLite Terminal

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

Data returned:

Black Dog	AC/DC	Who Made Who	Rock
Stairway	AC/DC	Who Made Who	Rock
About to Rock	Led Zepplin	IV	Metal
Who Made Who	Led Zepplin	IV	Metal

What we want to see

The tables which hold the data

How the tables are linked

Sources: SQLite Terminal

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						25
Paranoid						22
Planet Caravan						25
Iron Man						26
Electric Funeral						22
Hand of Doom						23
Rat Salad						31
Jack the Stripper/Fairies						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

Data returned:

Black Dog	AC/DC	Who Made Who	Rock
Stairway	AC/DC	Who Made Who	Rock
About to Rock	Led Zepplin	IV	Metal
Who Made Who	Led Zepplin	IV	Metal

Sources: SQLite Terminal

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

Python and SQLite3

http://www.python.org/doc/2.5.2/lib/module-sqlite3.html

SQLite3 is built into Python

- Since SQLite is simple and small and designed to be “embedded” - Python decided to embed SQLite into Python
- You simply “import sqlite3” and open a connection to the database and start doing SQL commands

http://www.python.org/doc/2.5.2/lib/module-sqlite3.html

SQLite3 is built into Python

```
import sqlite3

# Open up the database file and get a cursor
conn = sqlite3.connect('music.db')
c = conn.cursor()

print "Genre Rows"
c.execute('select * from Genre')
for row in c :
   print row
```

SQLite stores all tables and data in a single file.

```
$ python sql1.py
Genre Rows
(1, u'Rock')
(2, u'Metal')
$ ls
music.db     sql1.py   sql2.py
```

```
import sqlite3

# Open up the database file and get a cursor
conn = sqlite3.connect('music.db')
c = conn.cursor()

print "Inserting Country"
c.execute('insert into Genre (name) values ( ? )', ( 'Country', ) )

print "Genre Rows"
c.execute('select * from Genre')
for row in c :
   print row

print "Deleting Country"
c.execute("delete from Genre where name='Country'")

print "Genre Rows"
c.execute('select * from Genre')
for row in c :
   print row
```

```
$ python sql2.py
Inserting Country
Genre Rows
(1, u'Rock')
(2, u'Metal')
(3, u'Country')
Deleting Country
Genre Rows
(1, u'Rock')
(2, u'Metal')
```

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
- See SI572 - Database Design

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