

# Relational Databases and SQLite

Charles Severance



Python for Everybody  
[www.py4e.com/lectures3/](http://www.py4e.com/lectures3/)



# DB Browser for SQLite

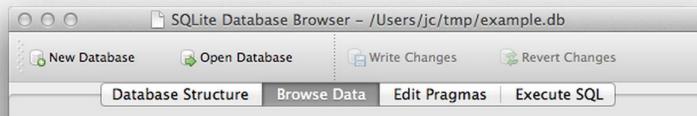
The Official home of the DB Browser for  
SQLite



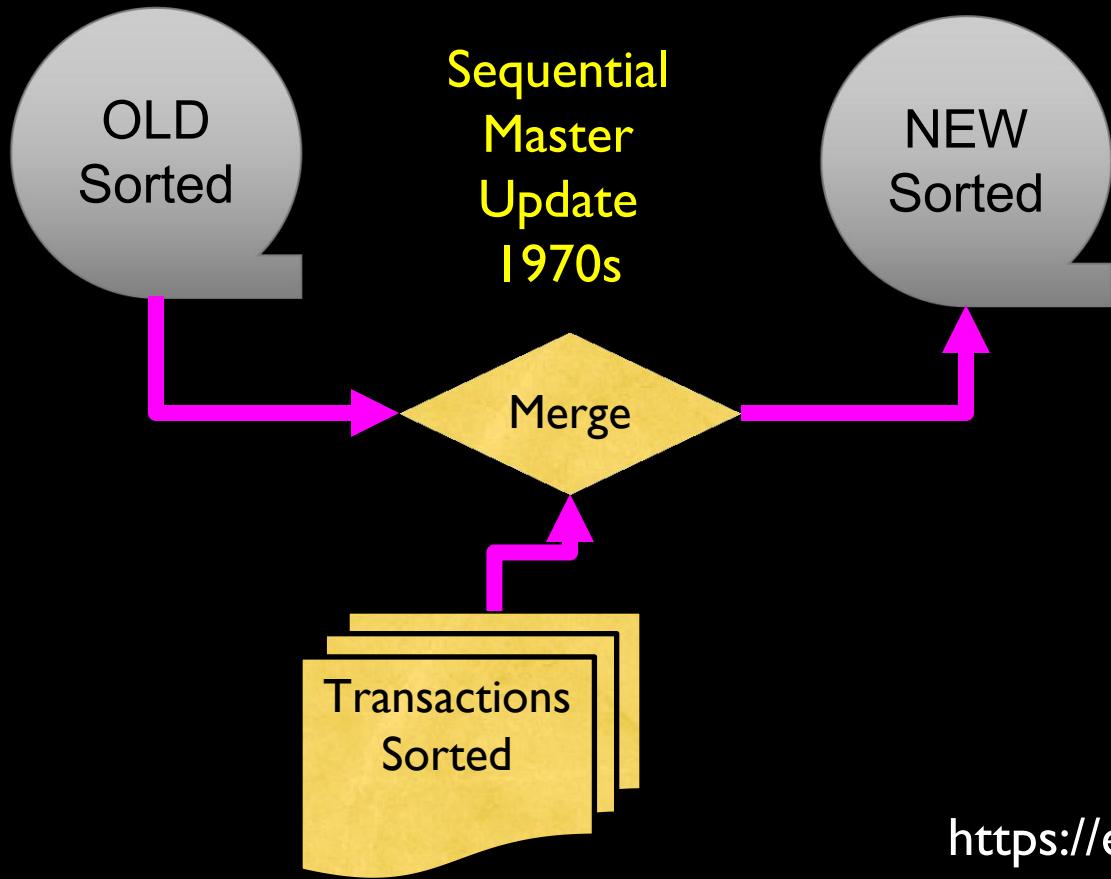
## // News

- 2015-07-07 - Added PortableApp version of 3.7.0. Thanks John. :)
- 2015-06-14 - Version 3.7.0 released. :)
- 2015-05-09 - Added PortableApp version of 3.6.0v3.

## // Screenshot



<http://sqlitebrowser.org/>



[https://en.wikipedia.org/wiki/IBM\\_729](https://en.wikipedia.org/wiki/IBM_729)

# Random Access

- When you can randomly access data...
- How can you layout data to be most efficient?
- Sorting might not be the best idea



[https://en.wikipedia.org/wiki/Hard\\_disk\\_drive\\_platter](https://en.wikipedia.org/wiki/Hard_disk_drive_platter)

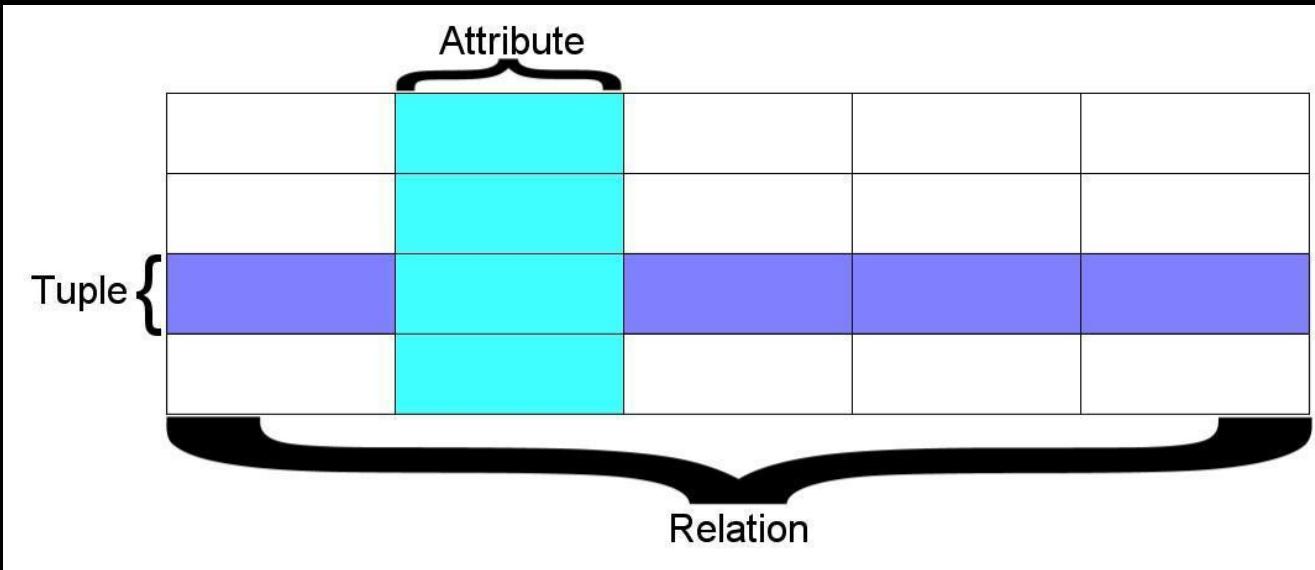
# Relational Databases

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](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

	A	B	C	D
1	TITLE	RATING	LEN	Rows / Tuples
2	About to Rock	3	354	
3	Who Made Who	4	252	
4				
5				
6				
7				
8				

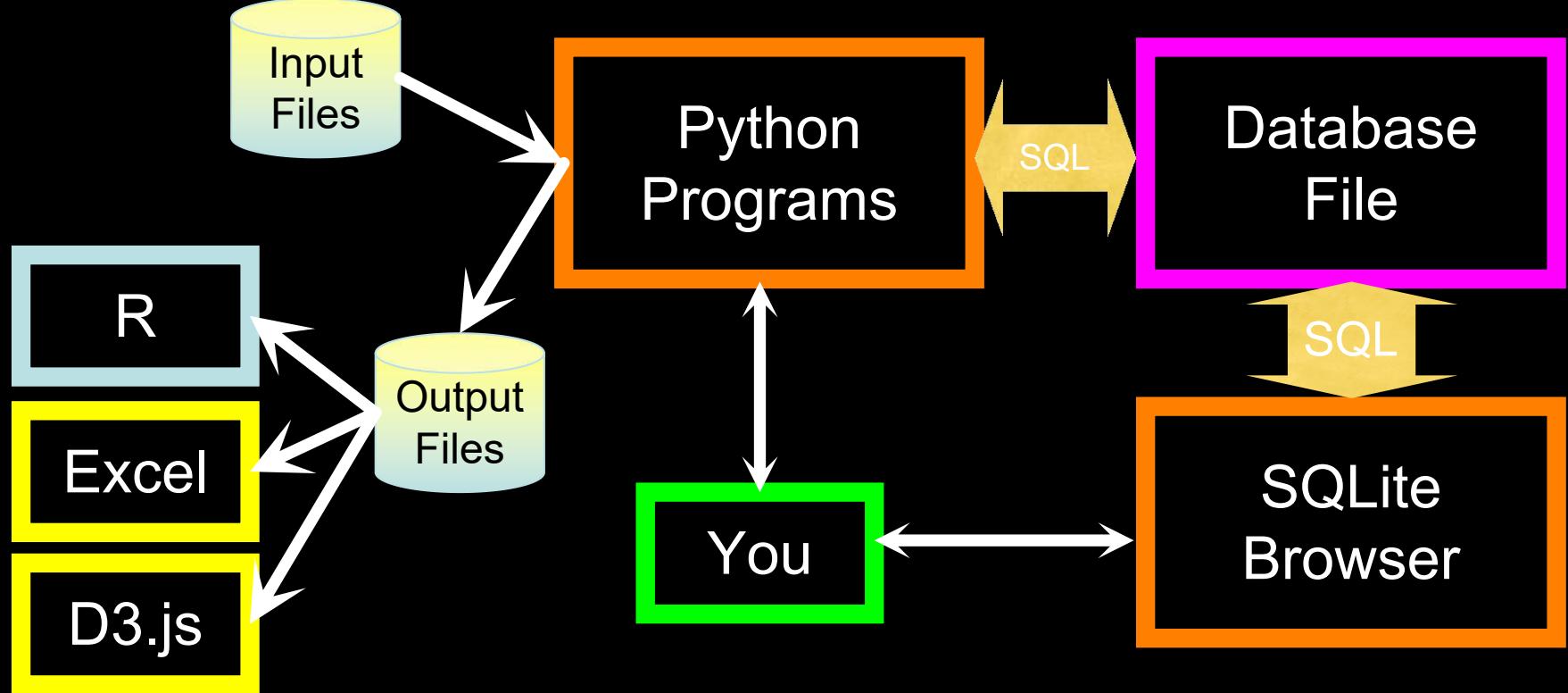
Tables / Relations

# SQL

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

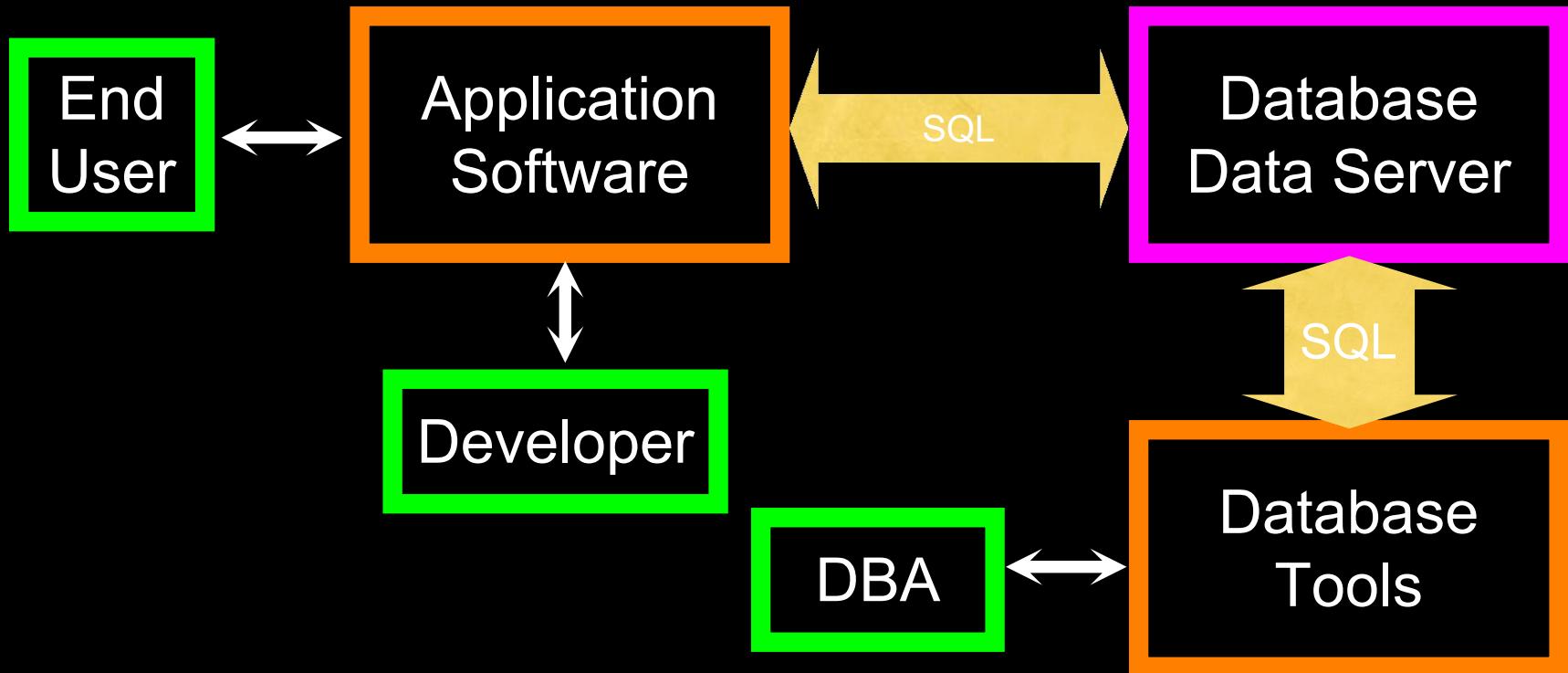
- Create data (a.k.a Insert)
- Retrieve data
- Update data
- Delete data

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



# 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”



# 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](http://en.wikipedia.org/wiki/Database_administrator)

# 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](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, Postgres, ...

# SQLite is in Lots of Software...

**symbian**

 **python™**



**skype**



**Microsoft®**

**McAfee®**



 **Adobe**



 **php**

**Google**

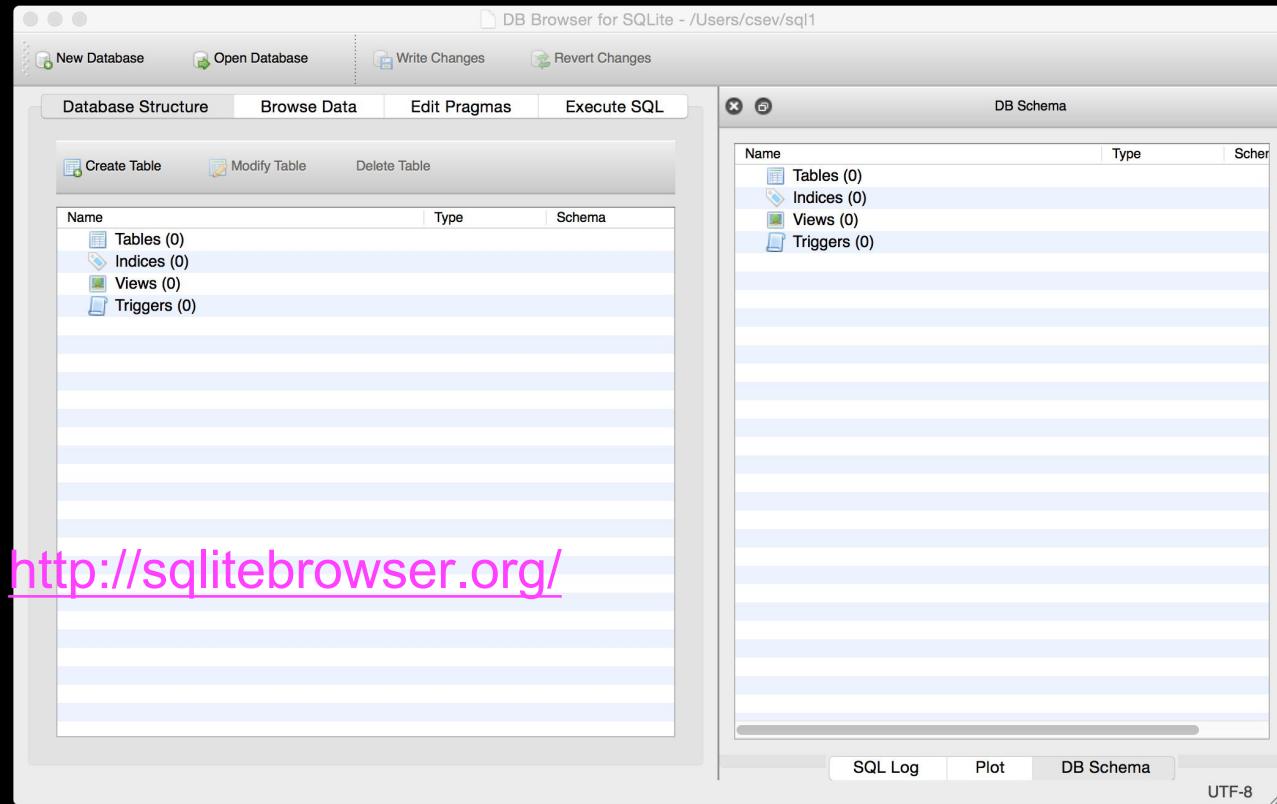
**TOSHIBA**

 **Sun  
microsystems**

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

# SQLite Browser

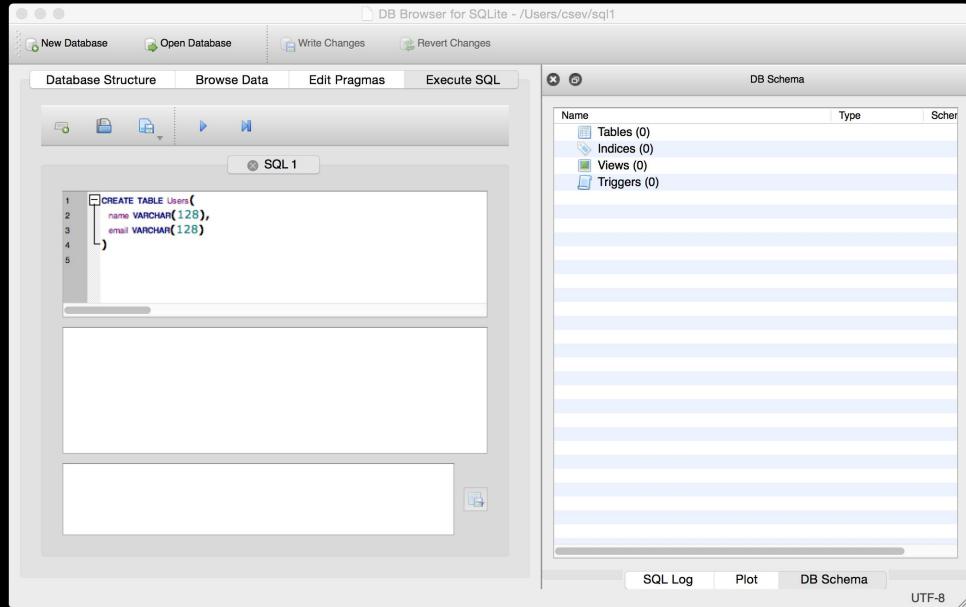
- SQLite is a very popular database - it is free and fast and small
- SQLite Browser allows us to directly manipulate SQLite files
  - <http://sqlitebrowser.org/>
- SQLite is embedded in Python and a number of other languages



# Lets Make a Database

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

# 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

The screenshot shows the DB Browser for SQLite interface. The main window displays the database structure on the left and the DB Schema on the right. In the database structure pane, there is one table named 'Users'. The schema for this table is shown as a CREATE TABLE statement: 'CREATE TABLE Users(name VARCHAR(128), email VARCHAR(128) )'. The DB Schema pane also lists the same table 'Users' with its schema. At the bottom, there are tabs for SQL Log, Plot, and DB Schema, with DB Schema being the active tab. The encoding is set to 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:  New Record Delete Record

name email  
Filter Filter

	name	email
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 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')
```

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	Table	CREATE TABLE Users( name VARCHAR(128), email VARCHAR(128) )
Indices (0)		
Views (0)		
Triggers (0)		

SQL Log Plot DB Schema UTF-8

The screenshot shows the DB Browser for SQLite interface. The main window has tabs for 'Database Structure', 'Browse Data', 'Edit Pragmas', and 'Execute SQL'. The 'Execute SQL' tab is active, displaying a query window with two lines of code: 'INSERT INTO Users (name, email) VALUES ('Kristin', 'kf@umich.edu')' and a blank line. Below the query window is a message: 'Query executed successfully: CREATE TABLE Users( name VARCHAR(128), email VARCHAR(128) ) (took 0ms)'. To the right is a 'DB Schema' panel titled 'DB Schema' which lists the database structure. It shows one table named 'Users' with the schema 'CREATE TABLE Users( name VARCHAR(128), email VARCHAR(128) )'. It also shows 0 indices, views, and triggers. At the bottom, there are tabs for 'SQL Log', 'Plot', 'DB Schema', and encoding 'UTF-8'.

DB Browser for SQLite - /Users/csev/sql1

New Database Open Database Write Changes Revert Changes

Database Structure Browse Data Edit Pragmas Execute SQL

SQL 1

```
1 INSERT INTO Users (name, email) VALUES ('Kristin', 'kf@umich.edu')
2
```

Query executed successfully: CREATE TABLE Users(
 name VARCHAR(128),
 email VARCHAR(128)
) (took 0ms)

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
1	Chuck	csev@umich...
2	Colleen	cvl@umich.edu
3	Ted	ted@umich....
4	Sally	a1@umich.edu
5	Kristin	kf@umich.edu

Go to: 1

DB Schema

Name	Type	Schema
Tables (1)		
Users		CREATE TABLE Users( name VARCHAR(128), email VARCHAR(128) )
Indices (0)		
Views (0)		
Triggers (0)		

SQL Log Plot DB Schema

UTF-8

# SQL: Delete

Deletes a row in a table based on selection criteria

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

DB Browser for SQLite - /Users/csev/sql1

New Database Open Database Write Changes Revert Changes

Database Structure Browse Data Edit Pragmas Execute SQL

SQL 1

```
1 DELETE FROM Users WHERE email='ted@umich.edu'
2
```

Query executed successfully: DELETE FROM Users WHERE email='ted@umich.edu'  
(took 0ms)

DB Schema

Name	Type	Schema
Tables (1)		
Users		CREATE TABLE Users( name VARCHAR(128), email VARCHAR(128) )
Indices (0)		
Views (0)		
Triggers (0)		

SQL Log Plot DB Schema UTF-8

This screenshot shows the DB Browser for SQLite application interface. The main window title is "DB Browser for SQLite - /Users/csev/sql1". The top menu bar includes "New Database", "Open Database", "Write Changes", and "Revert Changes". Below the menu is a tab bar with "Database Structure", "Browse Data", "Edit Pragmas", and "Execute SQL", where "Execute SQL" is currently selected. On the left, there's a toolbar with icons for file operations like New, Open, Save, and a dropdown arrow. A large central area contains a SQL editor with two lines of code: "DELETE FROM Users WHERE email='ted@umich.edu'" and a blank line. Below the editor is a message box stating "Query executed successfully: DELETE FROM Users WHERE email='ted@umich.edu' (took 0ms)". To the right, a "DB Schema" panel displays the database structure with one table named "Users" defined by the schema: "CREATE TABLE Users(name VARCHAR(128), email VARCHAR(128))". The panel also lists "Indices (0)", "Views (0)", and "Triggers (0)". At the bottom, there are tabs for "SQL Log", "Plot", and "DB Schema", with "DB Schema" being active. The status bar at the bottom right shows "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 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 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
1	Chuck	csev@umich...
2	Colleen	cvl@umich.edu
3	Sally	a1@umich.edu
4	Kristin	ki@umich.edu

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

The image displays two instances of the DB Browser for SQLite application running side-by-side. Both instances have the same title bar: "DB Browser for SQLite - /Users/csev/sql1".  
The left instance has a toolbar with "New Database", "Open Database", "Write Changes", and "Revert Changes". Below the toolbar are tabs: "Database Structure", "Browse Data", "Edit Pragmas", and "Execute". The "Execute" tab is active, showing a SQL command: "DELETE FROM Users WHERE email='ted@umich.edu'". The status bar at the bottom indicates the query was executed successfully in 0ms.  
The right instance also has a toolbar with the same options. Its tabs are: "Database Structure", "Browse Data", "Edit Pragmas", and "Execute SQL". The "Browse Data" tab is active, showing a table named "Users". The table has columns "name" and "email". It contains four rows of data:

	name	email
1	Chuck	csev@umich...
2	Colleen	cvl@umich.edu
3	Sally	a1@umich.edu
4	Kristin	ki@umich.edu

A large yellow arrow points from the right window back towards the left window, indicating a relationship or flow between the two screens.

# 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

The screenshot shows the DB Browser for SQLite interface. On the left, the 'Execute SQL' tab is active, displaying a single-line UPDATE query: 'UPDATE Users SET name='Charles' WHERE email='csev@umich.edu''. Below the query, a message indicates it was executed successfully: 'Query executed successfully: UPDATE Users SET name='Charles' WHERE email='csev@umich.edu' (took 0ms)'. On the right, the 'DB Schema' tab is active, showing the schema for the 'Users' table with columns 'name' and 'email'. The table has 1 row.

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

New Database Open Database Write Changes Revert Changes

Database Structure Browse Data Edit Pragmas Execute SQL

Table: Users New Record Delete Record

	name	email
1	Charles	csev@umich...
2	Colleen	cvl@umich.edu
3	Sally	a1@umich.edu
4	Kristin	kf@umich.edu

DB Schema

Name Type Schema

Tables (1)

- Users
- Indices (0)
- Views (0)
- Triggers (0)

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

SQL Log Plot DB Schema

UTF-8

The screenshot shows two instances of DB Browser for SQLite. The left instance has a SQL tab containing the query: 'UPDATE Users SET name='Charles' WHERE email='csev@umich.edu''. The status bar indicates the query was executed successfully. The right instance shows the 'Browse Data' tab with a table named 'Users' containing four rows: Charles, Colleen, Sally, and Kristin. A yellow arrow points from the right side of the table towards the schema viewer on the right. The schema viewer displays the table definition: 'CREATE TABLE Users(name VARCHAR(128), email VARCHAR(128))'. The bottom tabs for both instances are 'SQL Log', 'Plot', and 'DB Schema', with 'DB Schema' being the active tab.

# 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
```

	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

The screenshot shows the DB Browser for SQLite application interface. The main window has tabs for Database Structure, Browse Data, Edit Pragmas, and Execute SQL. The Execute SQL tab is active, displaying the query 'SELECT \* FROM Users' and its results. The results table has columns 'name' and 'email'. The data rows are: 1 Charles csev@umich.edu, 2 Colleen cvl@umich.edu, 3 Sally a1@umich.edu, and 4 Kristin kf@umich.edu. Below the table, it says '4 Rows returned from: SELECT \* FROM Users (took 0ms)'. To the right, the 'DB Schema' tab is selected, showing the table 'Users' with its CREATE statement: 'CREATE TABLE Users(name VARCHAR(128), email VARCHAR(128))'. Other schema elements like Indices, Views, and Triggers are listed as zero. At the bottom, there are tabs for SQL Log, Plot, and DB Schema, with DB Schema being the current tab.

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

The screenshot shows the DB Browser for SQLite application interface. The main window has tabs for Database Structure, Browse Data, Edit Pragmas, and Execute SQL. The Execute SQL tab is active, displaying the following SQL command:

```
1 SELECT * FROM Users WHERE email='csev@umich.edu'
2
```

Below the SQL input field is a table result set:

	name	email
1	Charles	csev@umich.edu

A status message at the bottom left indicates "1 Rows returned from: SELECT \* FROM Users WHERE email='csev@umich.edu' (took 0ms)".

The right side of the interface shows the "DB Schema" panel, which lists the database structure:

Name	Type	Schema
Tables (1)		
Users		CREATE TABLE Users( name VARCHAR(128), email VARCHAR(128) )
Indices (0)		
Views (0)		
Triggers (0)		

The "DB Schema" tab is highlighted at the bottom of the interface.

# Sorting with ORDER BY

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

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

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

DB Browser for SQLite - /Users/csev/sql1

New Database Open Database Write Changes Revert Changes

Database Structure Browse Data Edit Pragmas Execute SQL

SQL 1

```
1 SELECT * FROM Users ORDER BY email
2
```

	name	email
1	Sally	a1@umich.edu
2	Charles	csev@umich.edu
3	Colleen	cvl@umich.edu
4	Kristin	kf@umich.edu

4 Rows returned from: SELECT \* FROM Users ORDER BY email (took 0ms)

DB Schema

Name	Type	Schema
Tables (1)		
Users	Table	CREATE TABLE Users( name VARCHAR(128), email VARCHAR(128) )
Indices (0)		
Views (0)		
Triggers (0)		

SQL Log Plot DB Schema

UTF-8

The screenshot shows the DB Browser for SQLite application interface. The main window has tabs for 'Database Structure', 'Browse Data', 'Edit Pragmas', and 'Execute SQL'. The 'Execute SQL' tab is active, displaying the results of a query: 'SELECT \* FROM Users ORDER BY email'. The results are presented in a table with columns 'name' and 'email'. The table contains four rows with data: Sally (a1@umich.edu), Charles (csev@umich.edu), Colleen (cvl@umich.edu), and Kristin (kf@umich.edu). Below the table, a message indicates 4 rows were returned from the query. To the right of the main window is a separate 'DB Schema' window, which lists the database's structure. It shows one table named 'Users' with the following schema: CREATE TABLE Users(name VARCHAR(128), email VARCHAR(128)). It also lists 'Indices (0)', 'Views (0)', and 'Triggers (0)'. The bottom of the screen shows the application's status bar with tabs for 'SQL Log', 'Plot', and 'DB Schema', and the encoding '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)

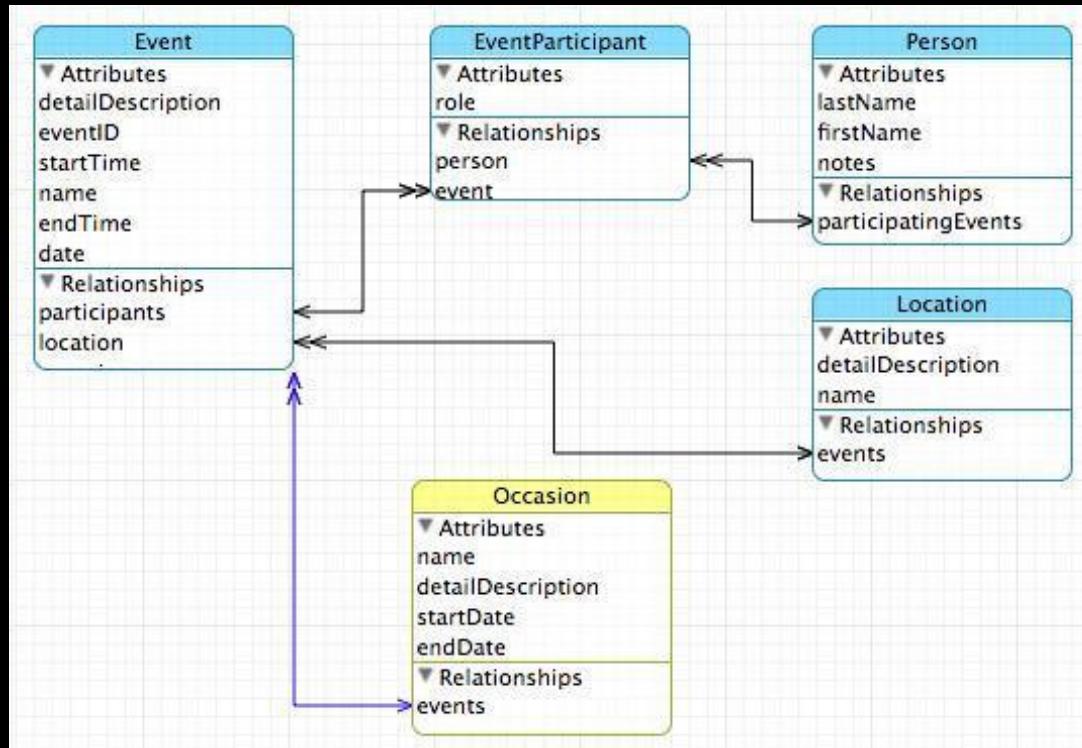
- Tables pretty much look like big fast programmable spreadsheets with rows, columns, and commands
- The power comes when we have more than one table and we can exploit the relationships between the tables

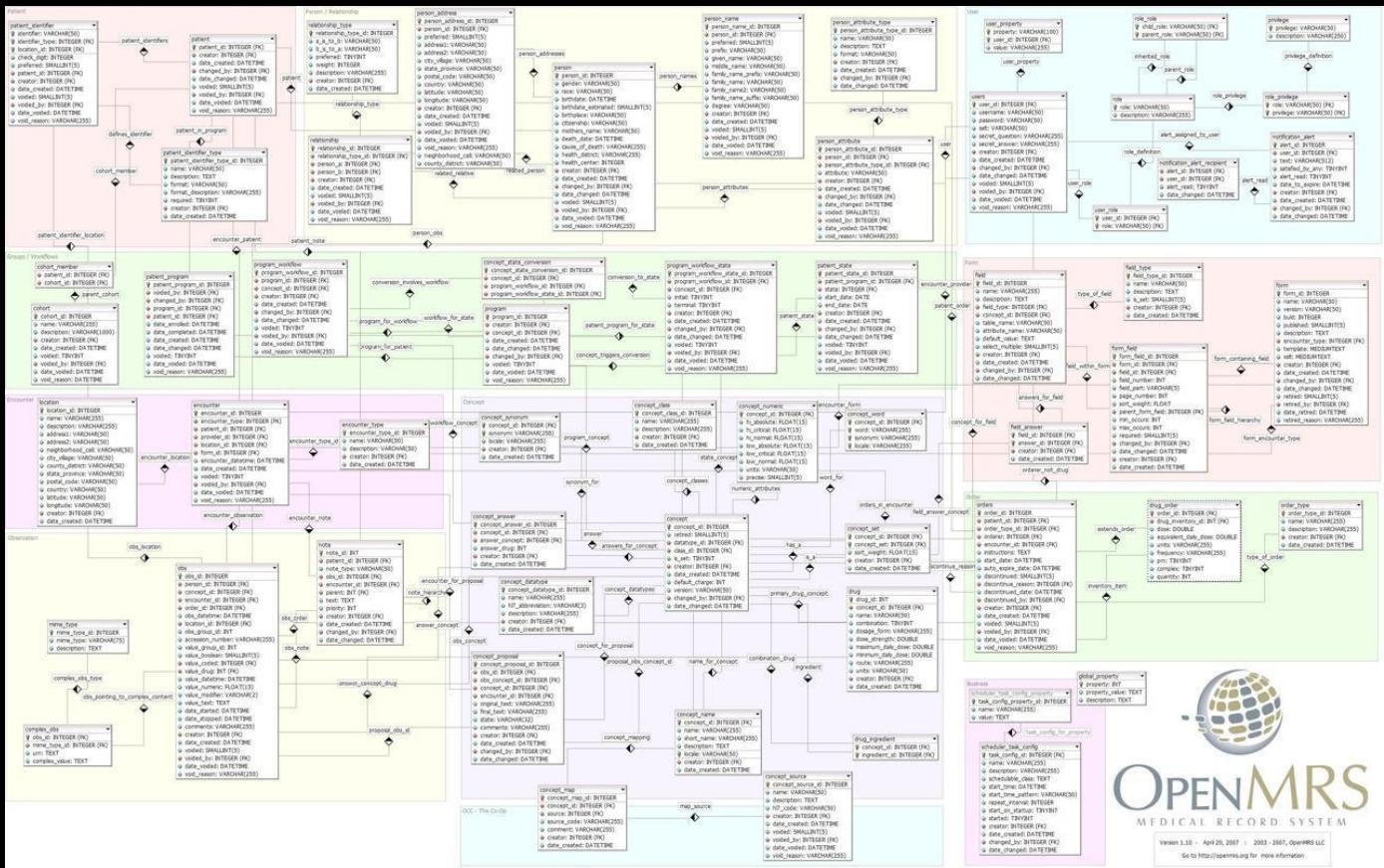
# Complex Data Models and Relationships

[http://en.wikipedia.org/wiki/Relational\\_model](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...





OPENMRS  
MEDICAL RECORD SYSTEM

Version 1.10 - April 29, 2007 | 2003-2007, OpenNMS.org

# 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
-------	-----	--------	-------	-------	--------	-------

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

# For each “piece of info”...

- Is the column an object or an attribute of another object?  
Len                      Album  
Genre
- Once we define objects, we need to define the relationships between objects  
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
☒ Tie Me	2:20	America	Greatest Hits	Easy Listen...	★★★★★	22

Track

Album

Artist

Genre

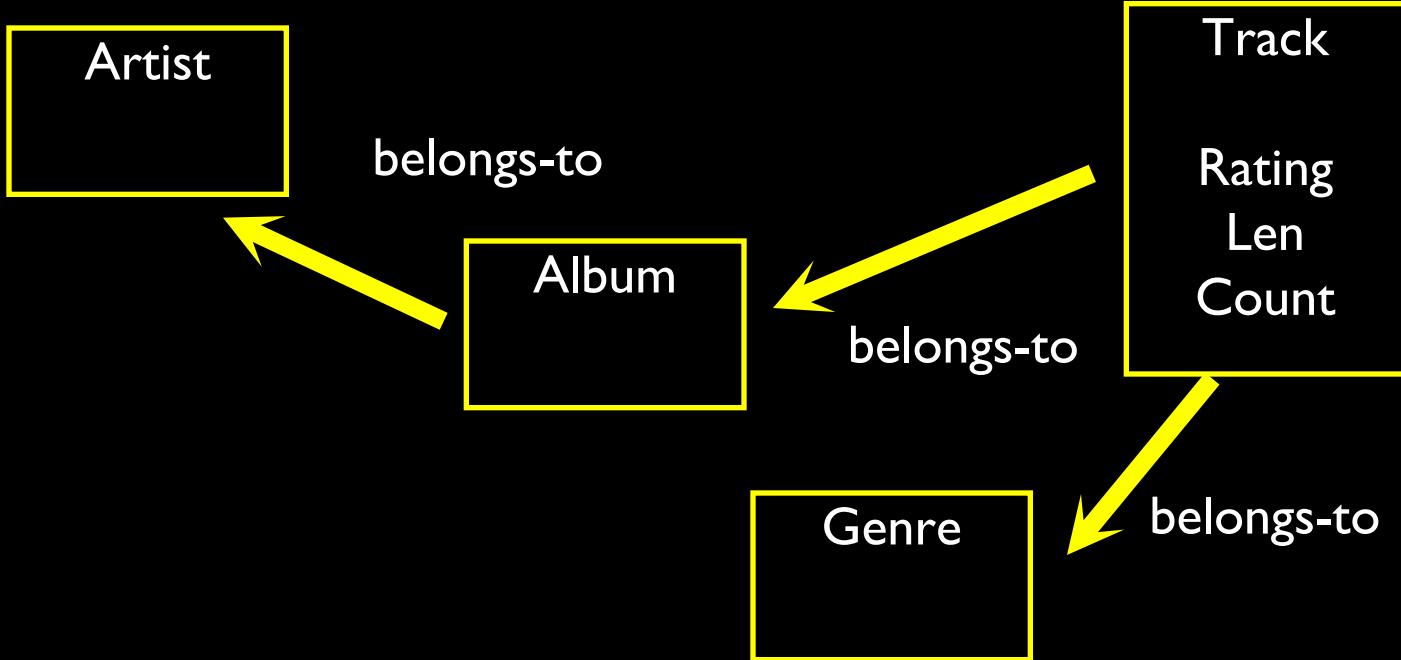
Rating

Len

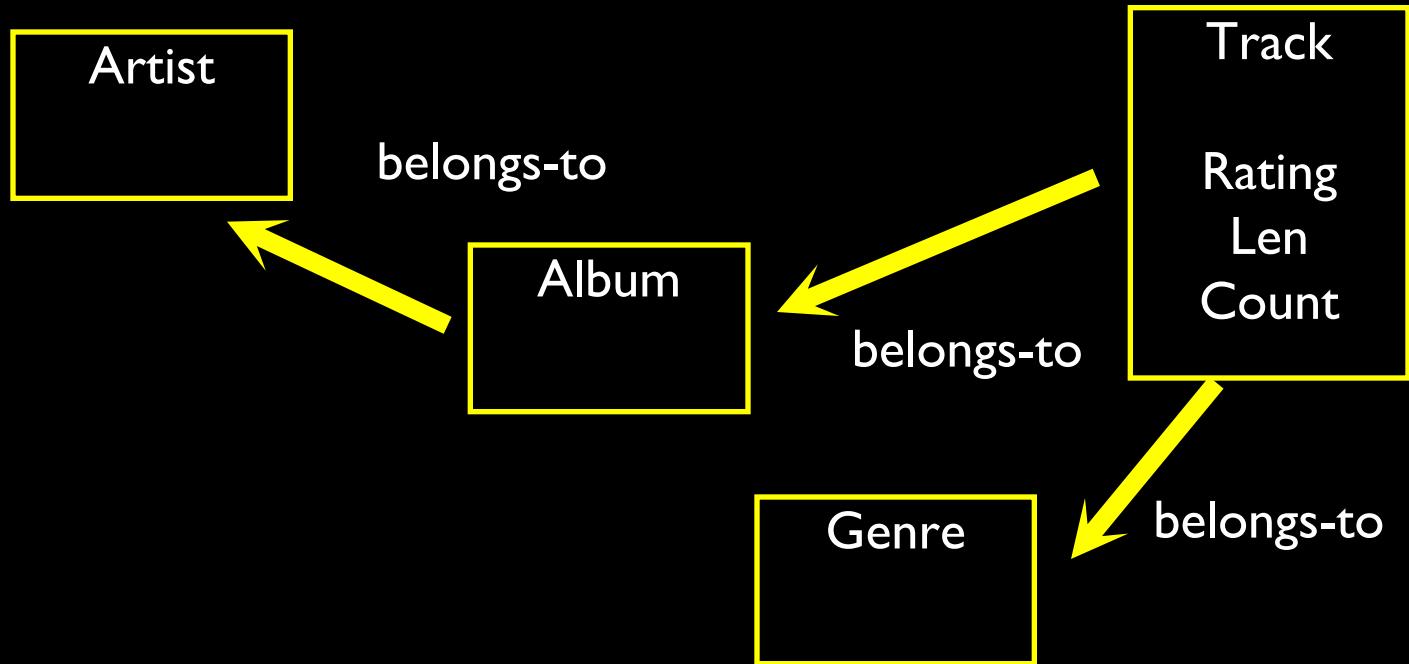
Count

<input checked="" type="checkbox"/> Hells Bells	5:13	AC/DC	Who Made Who	Rock	★★★★★	61
<input checked="" type="checkbox"/> Shake Your Foundations	3:54	AC/DC	Who Made Who	Rock	★★★★★	70
<input checked="" type="checkbox"/> Chase the Ace	3:01	AC/DC	Who Made Who	Rock		56
<input checked="" type="checkbox"/> For Those About To Rock (We ...	5:54	AC/DC	Who Made Who	Rock	★★★★★	61
<input checked="" type="checkbox"/> Dúlamán	3:43	Altan	Natural Wonders M...	New Age		31
<input checked="" type="checkbox"/> Rode Across the Desert	4:10	America	Greatest Hits	Easy Listen...	★★★★★	23
<input checked="" type="checkbox"/> Now You Are Gone	3:08	America	Greatest Hits	Easy Listen...	★★★★★	18
<input checked="" type="checkbox"/> Tin Man	2:20	America	Greatest Hits	Easy Listen...	★★★★★	22

Track  
Album  
Artist  
Genre  
Rating  
Len  
Count



<input checked="" type="checkbox"/> Hells Bells	5:13	AC/DC	Who Made Who	Rock	★★★★★	61
<input checked="" type="checkbox"/> Shake Your Foundations	3:54	AC/DC	Who Made Who	Rock	★★★★★	70
<input checked="" type="checkbox"/> Chase the Ace	3:01	AC/DC	Who Made Who	Rock		56
<input checked="" type="checkbox"/> For Those About To Rock (We ...	5:54	AC/DC	Who Made Who	Rock	★★★★★	61
<input checked="" type="checkbox"/> Dúlamán	3:43	Altan	Natural Wonders M...	New Age		31
<input checked="" type="checkbox"/> Rode Across the Desert	4:10	America	Greatest Hits	Easy Listen...	★★★★★	23
<input checked="" type="checkbox"/> Now You Are Gone	3:08	America	Greatest Hits	Easy Listen...	★★★★★	18
in Tin Man	2:20	America	Greatest Hits	Easy Listen...	★★★★★	22



<input checked="" type="checkbox"/> Hells Bells	5:13	AC/DC	Who Made Who	Rock	★★★★★	61
<input checked="" type="checkbox"/> Shake Your Foundations	3:54	AC/DC	Who Made Who	Rock	★★★★★	70
<input checked="" type="checkbox"/> Chase the Ace	3:01	AC/DC	Who Made Who	Rock		56
<input checked="" type="checkbox"/> For Those About To Rock (We ...	5:54	AC/DC	Who Made Who	Rock	★★★★★	61
<input checked="" type="checkbox"/> Dúlamán	3:43	Altan	Natural Wonders M...	New Age		31
<input checked="" type="checkbox"/> Rode Across the Desert	4:10	America	Greatest Hits	Easy Listen...	★★★★★	23
<input checked="" type="checkbox"/> Now You Are Gone	3:08	America	Greatest Hits	Easy Listen...	★★★★★	18
in Tin Man	2:20	America	Greatest Hits	Easy Listen...	★★★★★	22

# 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](http://en.wikipedia.org/wiki/Database_normalization)

<input checked="" type="checkbox"/> Hells Bells	5:13	AC/DC	Who Made Who	Rock		61
<input checked="" type="checkbox"/> Shake Your Foundations	3:54	AC/DC	Who Made Who	Rock		70
<input checked="" type="checkbox"/> Chase the Ace	3:01	AC/DC	Who Made Who	Rock		56
<input checked="" type="checkbox"/> For Those About To Rock (We ...	5:54	AC/DC	Who Made Who	Rock		61
<input checked="" type="checkbox"/> Dúlamán	3:43	Altan	Natural Wonders M...	New Age		31
<input checked="" type="checkbox"/> Rode Across the Desert	4:10	America	Greatest Hits	Easy Listen...		23
<input checked="" type="checkbox"/> Now You Are Gone	3:08	America	Greatest Hits	Easy Listen...		18
<input type="checkbox"/> Tie Me	3:20	America	Greatest Hits	Easy Listen...		22

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

# 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

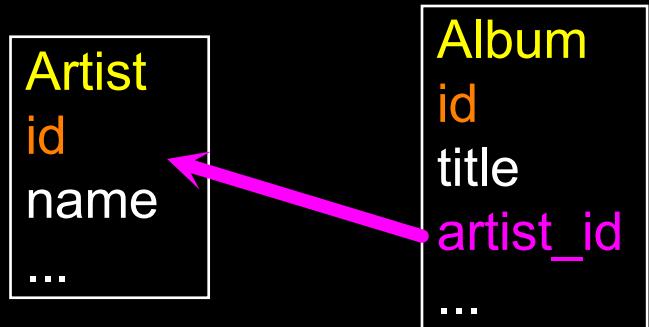
## Best practices

- 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



# Relationship Building (in tables)

Artist

Album

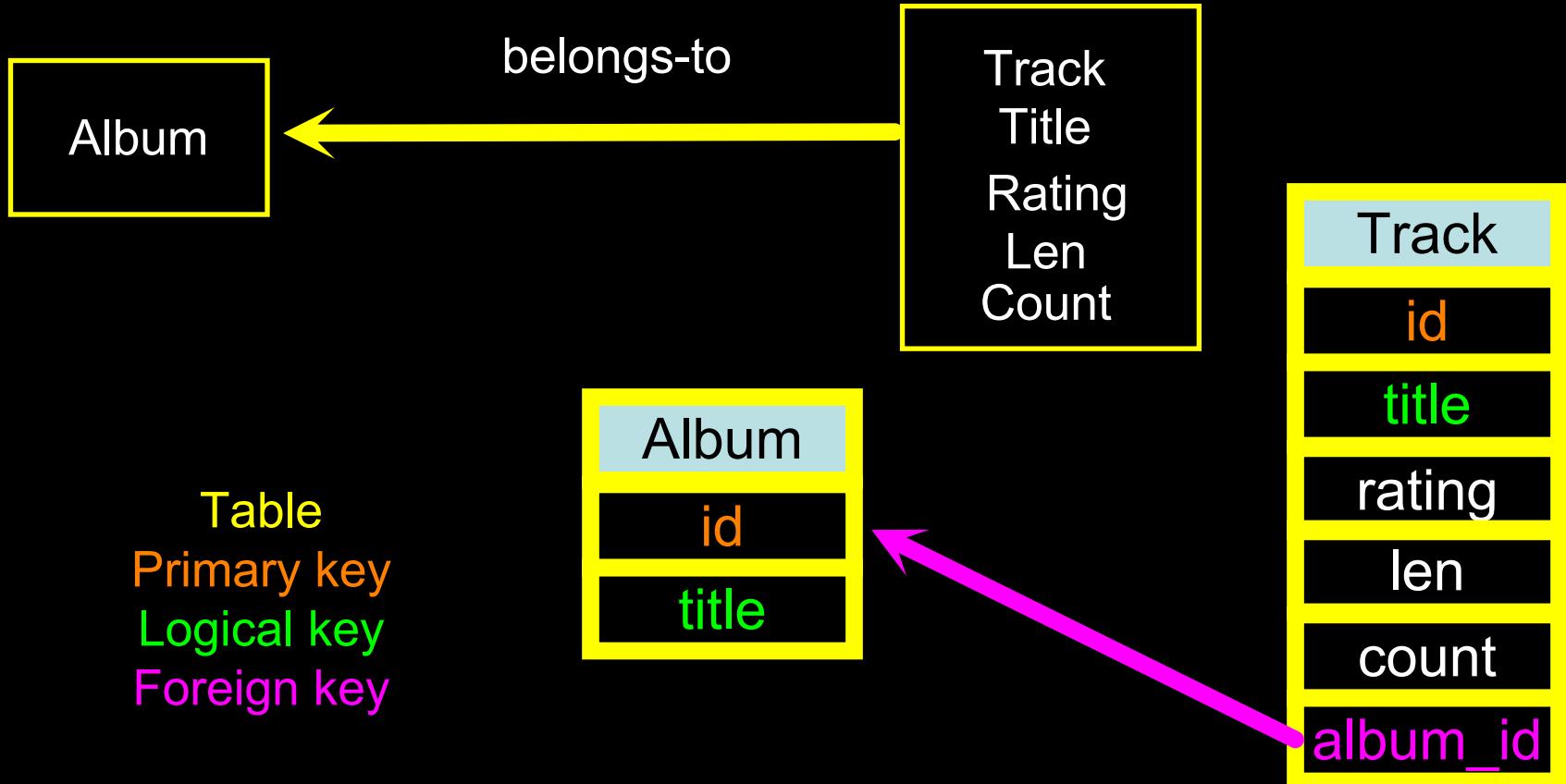
Track  
Rating  
Len  
Count

belongs-to

belongs-to

belongs-to

✓ 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
✓ Tie Man	2:20	America	Greatest Hits	Easy Listen...	★★★★★	22



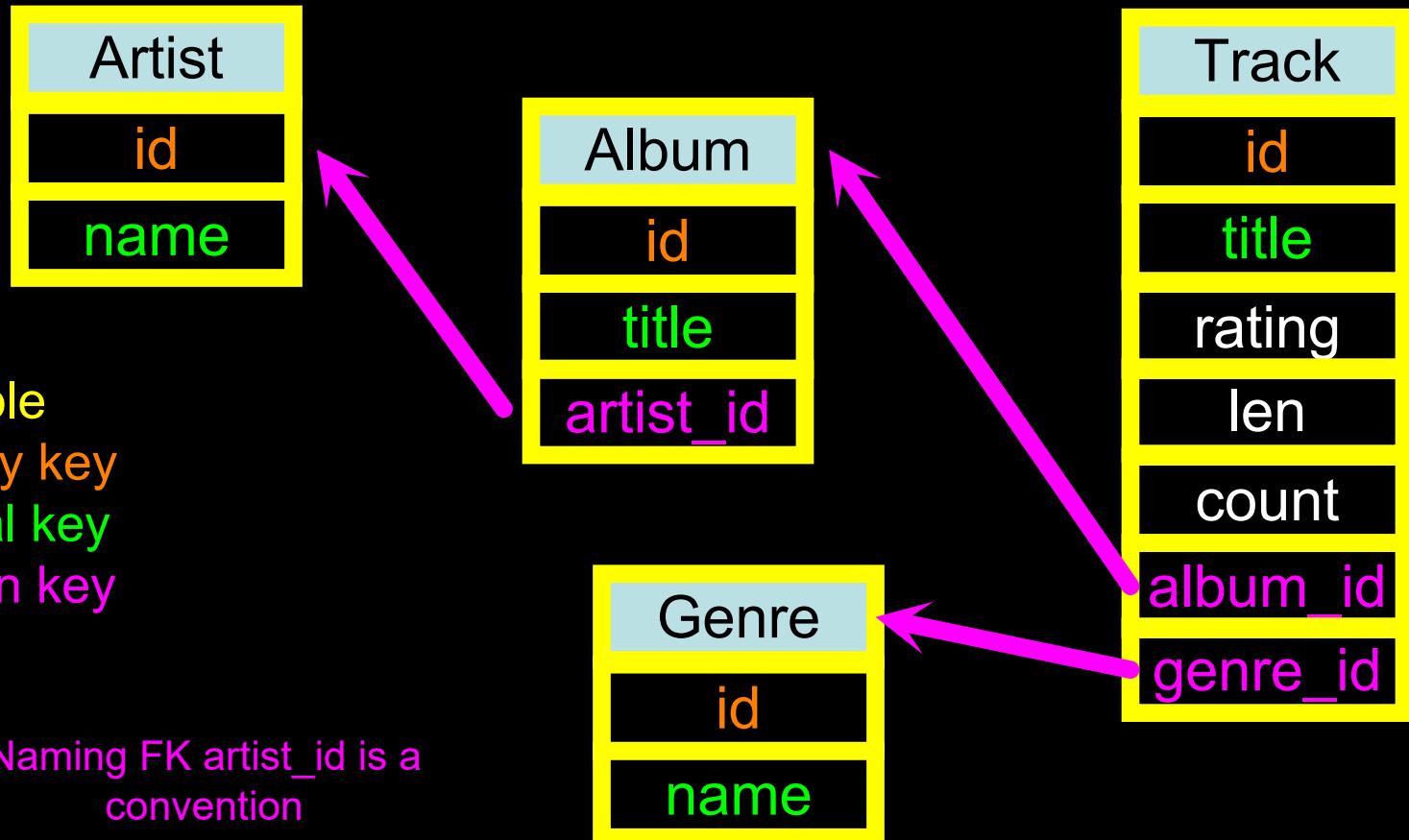
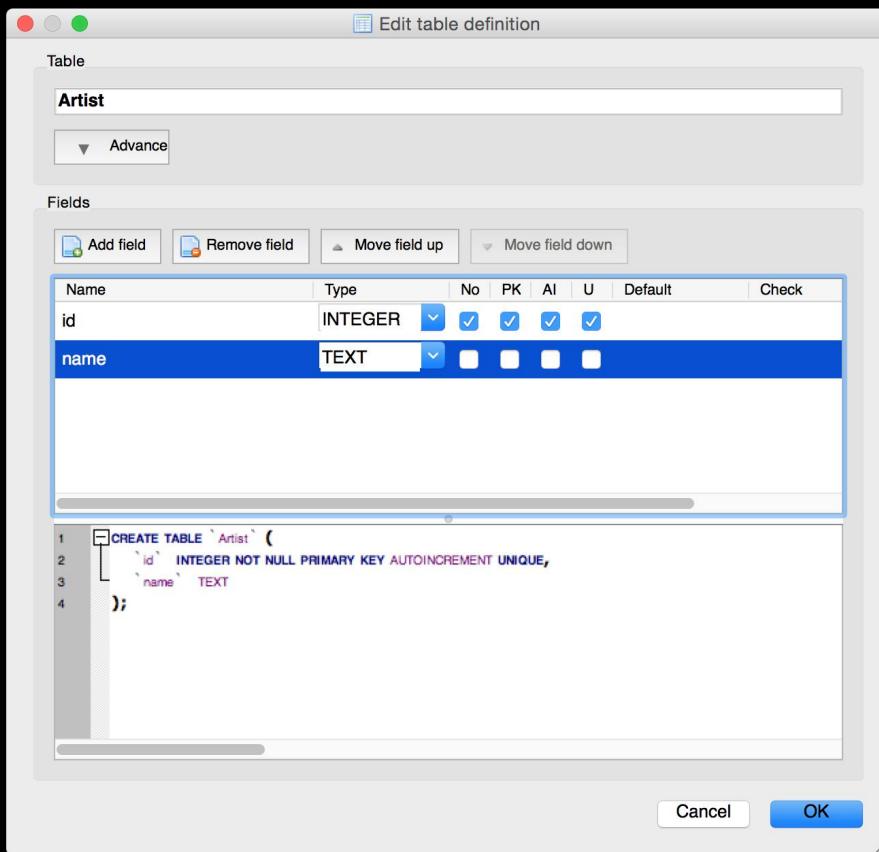
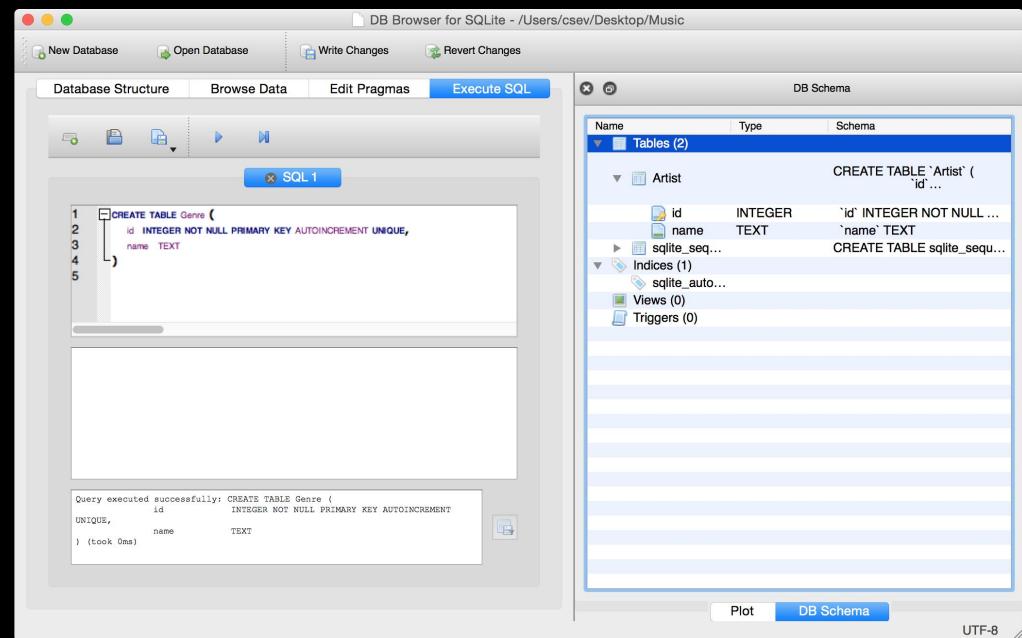


Table  
Primary key  
Logical key  
Foreign key

Naming FK `artist_id` is a convention





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

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

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

DB Browser for SQLite - /Users/csev/Desktop/Music

Database Structure    Browse Data    Edit Pragmas    Execute SQL

Create Table    Modify Table    Delete Table

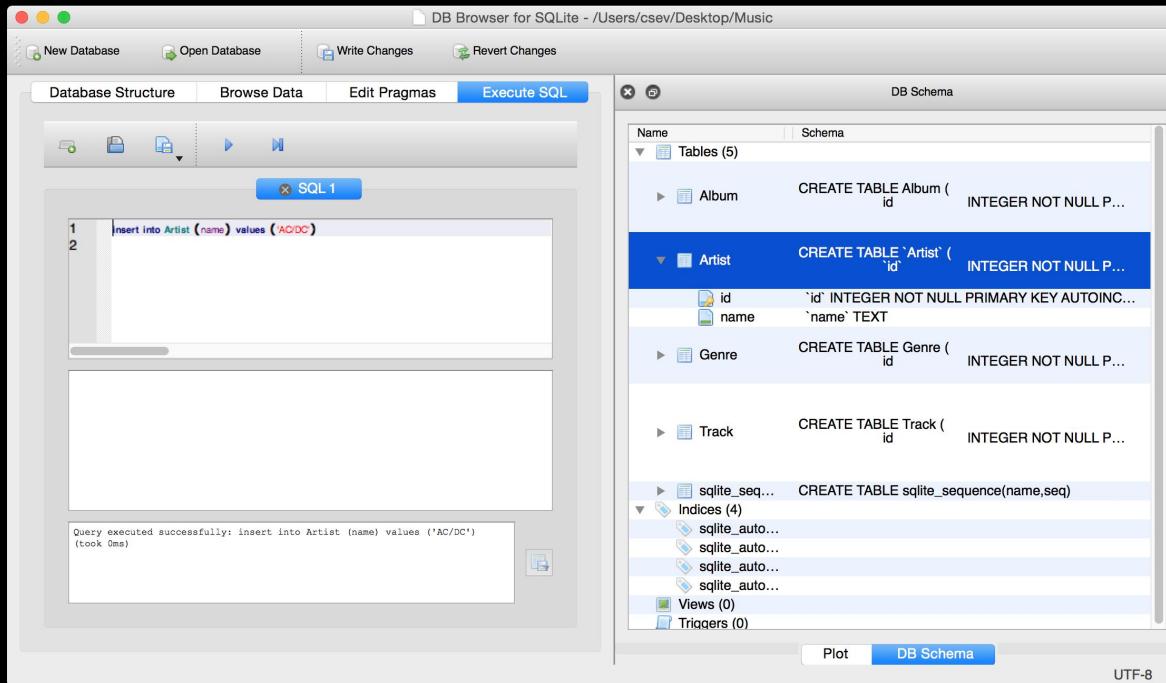
Name	Type	Schema
Tables (5)		
Album		<pre>CREATE TABLE "Album" (     `id` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,     `artist_id` INTEGER,     `title` TEXT )</pre>
Artist		<pre>CREATE TABLE "Artist" (     `id` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,     `name` TEXT )</pre>
Genre		<pre>CREATE TABLE Genre (     id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,     name TEXT )</pre>
Track		<pre>CREATE TABLE Track (     id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,     title TEXT,     album_id INTEGER,     genre_id INTEGER,     len INTEGER,     rating INTEGER,     count INTEGER )</pre>
		<pre>    `id` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,     `title` TEXT )</pre>
		<pre>    `album_id` INTEGER )</pre>
		<pre>    `genre_id` INTEGER )</pre>
		<pre>    `len` INTEGER )</pre>
		<pre>    `rating` INTEGER )</pre>
		<pre>    `count` INTEGER )</pre>

DB Schema

Name	Schema
Tables (5)	
Album	<pre>CREATE TABLE "Album" (     `id` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,     `artist_id` INTEGER,     `title` TEXT )</pre>
Artist	<pre>CREATE TABLE "Artist" (     `id` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,     `name` TEXT )</pre>
Genre	<pre>CREATE TABLE Genre (     id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,     name TEXT )</pre>
Track	<pre>CREATE TABLE Track (     id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,     title TEXT,     album_id INTEGER,     genre_id INTEGER,     len INTEGER,     rating INTEGER,     count INTEGER )</pre>
sqlite_sequence	<pre>CREATE TABLE sqlite_sequence(name,seq)</pre>
Indices (4)	
sqlite_autoindex_Album_1	
sqlite_autoindex_Track_1	
sqlite_autoindex_Track_2	

Plot    DB Schema

UTF-8



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

The image shows three windows of DB Browser for SQLite:

- SQL Editor (Left):** Shows the SQL query `insert into Artist (name) values ('AC/DC')` in the SQL tab.
- Database Structure (Middle):** Shows the `Artist` table with two rows: `1 Led Zeppelin` and `2 AC/DC`. A yellow arrow points from this window to the `DB Schema` window.
- DB Schema (Right):** Shows the database schema with the following CREATE TABLE statements:
  - `CREATE TABLE Album (id INTEGER NOT NULL PRIMARY KEY)`
  - `CREATE TABLE Artist (id INTEGER NOT NULL PRIMARY KEY)`
  - `CREATE TABLE Genre (id INTEGER NOT NULL PRIMARY KEY)`
  - `CREATE TABLE Track (id INTEGER NOT NULL PRIMARY KEY)`
  - `CREATE TABLE sqlite_sequence(name,seq)`

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

The screenshot shows the DB Browser for SQLite interface. On the left, the 'Browse Data' tab is selected, displaying the 'Genre' table with two rows: '1 Rock' and '2 Metal'. On the right, the 'DB Schema' tab is selected, showing the CREATE statements for all tables in the database:

- Album: CREATE TABLE Album (id INTEGER NOT NULL PRIMARY KEY)
- Artist: CREATE TABLE `Artist` (id INTEGER NOT NULL PRIMARY KEY)
- Genre: CREATE TABLE Genre (id INTEGER NOT NULL PRIMARY KEY)
- Track: CREATE TABLE Track (id INTEGER NOT NULL PRIMARY KEY)
- sqlite\_sequence: CREATE TABLE sqlite\_sequence(name,seq)

Below the schema, there are sections for Indices (4), Views (0), and Triggers (0).

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

DB Browser for SQLite - /Users/csev/Desktop/Music

New Database Open Database Write Changes Revert Changes

Database Structure Browse Data Edit Pragmas Execute SQL

Table: Album New Record Delete Record

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

< < 1 - 2 of 2 > > Go to: 1

UTF-8

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

Track

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

Album

id	artist_id	title
1	2	Who Made Who
2	1	IV

Artist

id	name
1	Led Zeppelin
2	AC/DC

id	name
1	Rock
2	Metal

Genre

# Using Join Across Tables

[http://en.wikipedia.org/wiki/Join\\_\(SQL\)](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**

<b>id</b>	<b>artist_id</b>	<b>title</b>
Filter	Filter	Filter
1	2	Who Made Who
2	1	IV

Album

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

Artist

<b>id</b>	<b>name</b>
Filter	Filter
1	Led Zepplin
2	AC/DC

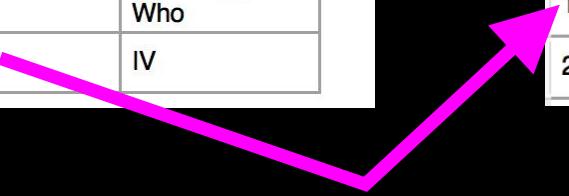
select Album.title, Artist.name from Album join Artist on Album.artist\_id = Artist.id

What we want  
to see

The tables that  
hold the data

How the tables  
are linked

<b>id</b>	<b>artist_id</b>	<b>title</b>	<b>id</b>	<b>name</b>
Filter	Filter	Filter	Filter	Filter
1	2	Who Made Who	1	Led Zepplin
2	1	IV	2	AC/DC



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

select Album.title, Album.artist\_id, Artist.id, Artist.name  
from Album join Artist on Album.artist\_id = Artist.id

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

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

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

	title	name
id		
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

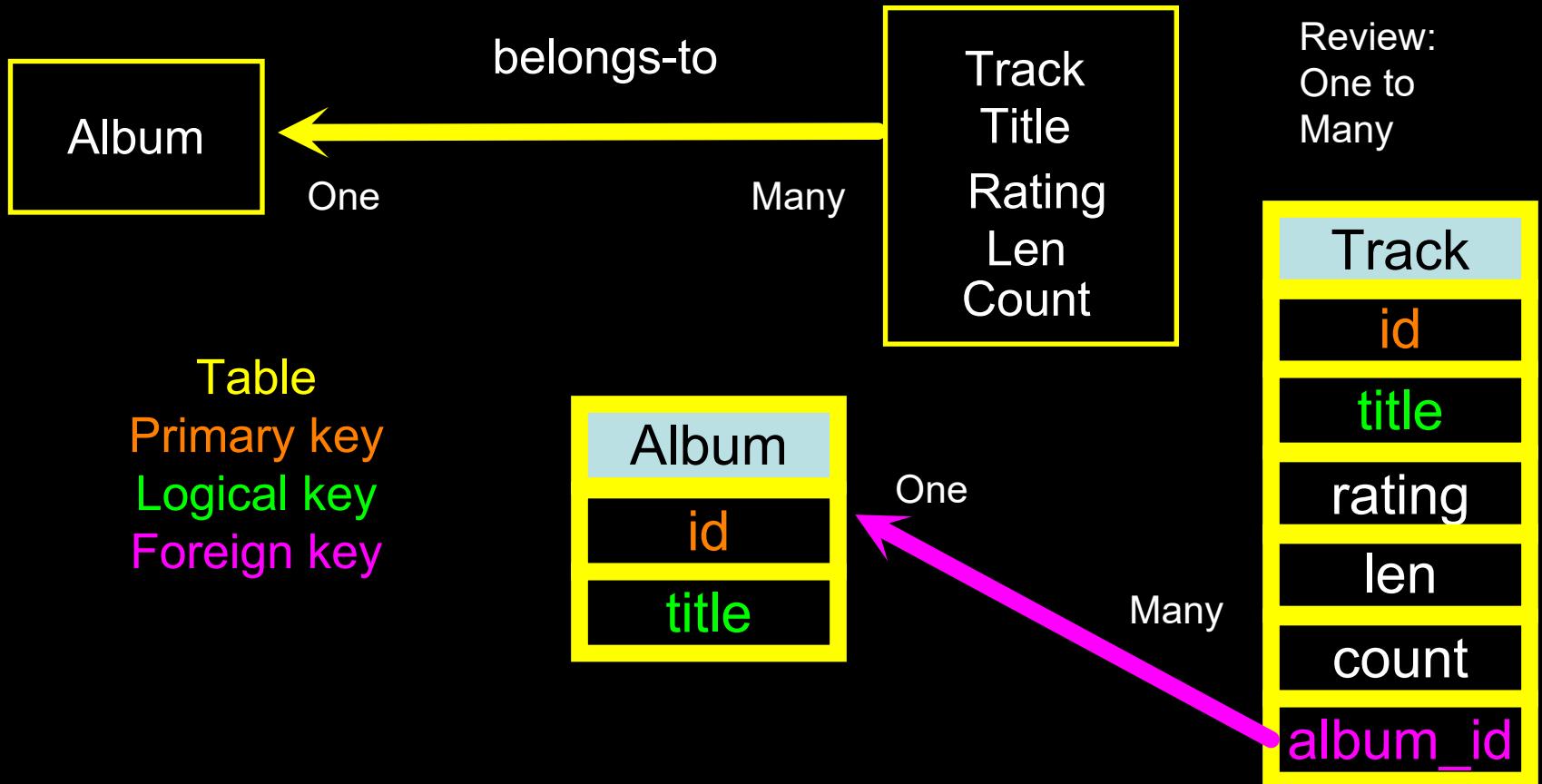
How the tables are  
linked

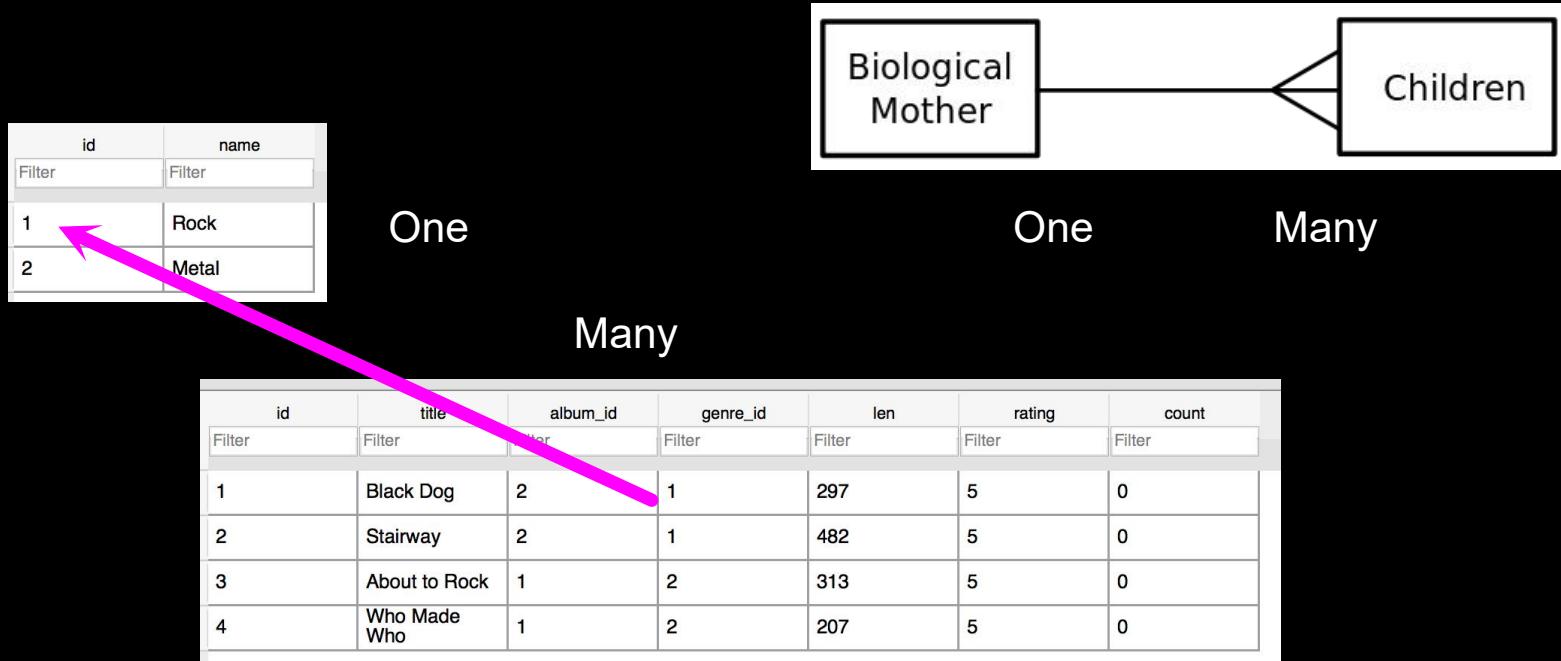
<input checked="" type="checkbox"/> Hells Bells	5:13	AC/DC	Who Made Who	Rock	★★★★★	61
<input checked="" type="checkbox"/> Shake Your Foundations	3:54	AC/DC	Who Made Who	Rock	★★★★★	70
<input checked="" type="checkbox"/> Chase the Ace	3:01	AC/DC	Who Made Who	Rock		56
<input checked="" type="checkbox"/>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

		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

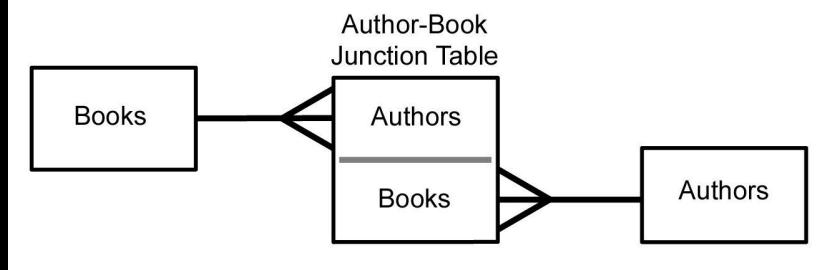
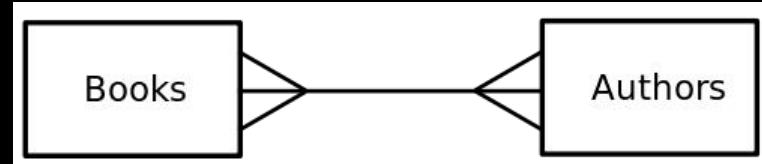
[https://en.wikipedia.org/wiki/Many-to-many\\_\(data\\_model\)](https://en.wikipedia.org/wiki/Many-to-many_(data_model))



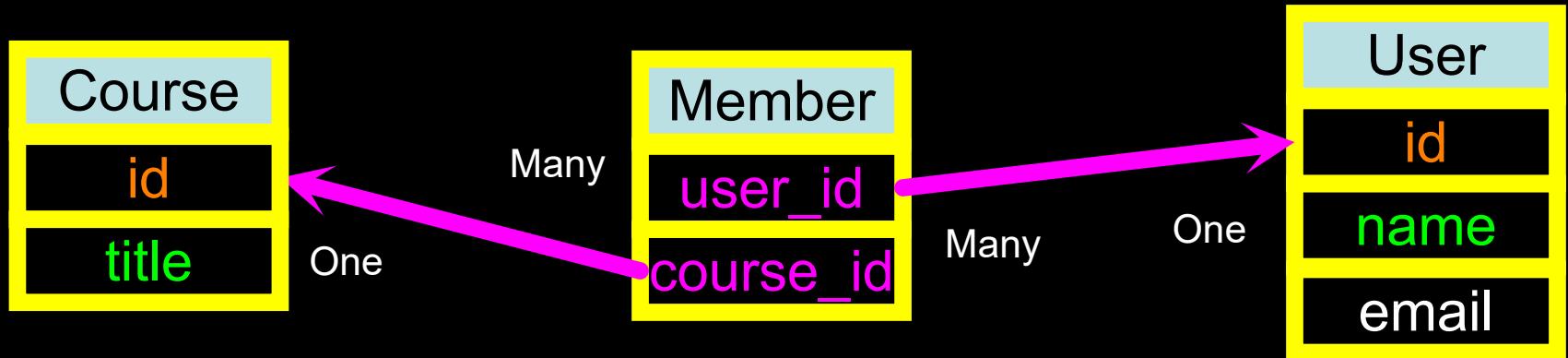


# Many to Many

- 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)
)
```

Start with a Fresh  
Database

DB Browser for SQLite - /Users/csev/Desktop/si502\_database

New Database Open Database Write Changes Revert Changes

Database Structure Browse Data Edit Pragmas Execute SQL

Create Table Modify Table Delete Table

Name	Type	Schema
Tables (4)		
Course		CREATE TABLE Course ( id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE, title TEXT )
Member		CREATE TABLE Member ( user_id INTEGER, course_id INTEGER, PRIMARY KEY (user_id, course_id) )
User		CREATE TABLE User ( id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE, name TEXT, email TEXT )
sqlite_sequence		CREATE TABLE sqlite_sequence(name,seq)
Indices (3)		
sqlite_autoindex_Course_1		
sqlite_autoindex_Member_1		
sqlite_autoindex_User_1		
Views (0)		
Triggers (0)		

UTF-8

# Insert Users and Courses

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

DB Browser for SQLite - /Users/csev/Desktop/si502\_database

New Database Open Database Write Changes Revert Changes

Database Structure Browse Data Edit Pragmas

Table: Course

	id	title
	1	Python
	2	SQL
	3	PHP

Filter Filter

1 < < 1 - 3 of 3 > > Go to:

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

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

Filter Filter Filter

New Record Delete Record

1 < < 1 - 3 of 3 > > Go to: 1 Go to: UTF-8

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

id	title
Filter	Filter
1	Python
2	SQL
3	PHP

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

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

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

DB Browser for SQLite - /Users/csev/Desktop/si502\_database

New Database Open Database Write Changes Revert Changes

Database Structure Browse Data Edit Pragmas Execute SQL

Table: Member New Record Delete Record

	user_id	course_id	role
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

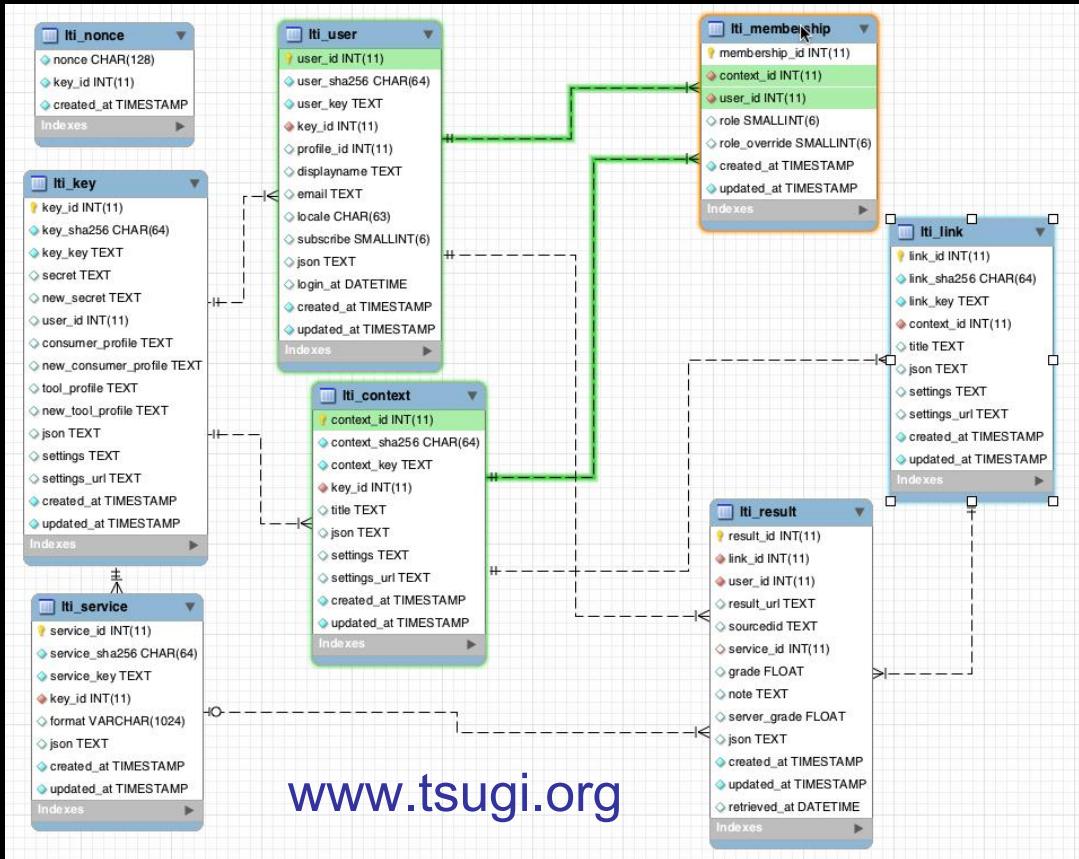
	id	name	email
	Filter	Filter	Filter
1	Jane	jane@tsugi.org	
2	Ed	ed@tsugi.org	
3	Sue	sue@tsugi.org	

	user_id	course_id	role
	Filter	Filter	Filter
1	1	1	1
2	1	1	0
3	1	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
	Filter	Filter	Filter
2	Sue	0	PHP
3	Jane	1	Python
4	Ed	0	Python
5	Sue	0	Python
6	Ed	1	SQL



[www.tsugi.org](http://www.tsugi.org)

# Complexity Enables Speed

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

# Additional SQL Topics

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

# 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



These slides are Copyright 2010- Charles R. Severance  
([www.dr-chuck.com](http://www.dr-chuck.com)) of the University of Michigan School of  
Information and [open.umich.edu](http://open.umich.edu) and made available under a  
Creative Commons Attribution 4.0 License. Please maintain this  
last slide in all copies of the document to comply with the  
attribution requirements of the license. If you make a change, feel  
free to add your name and organization to the list of contributors  
on this page as you republish the materials.

...

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

... Insert new Contributors here