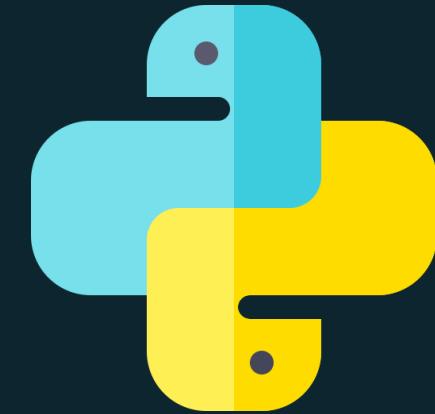


Brief Python

Python Course for Programmers



Learn Python Language for Data Science

CHAPTER 12B: DATABASE (SQL)

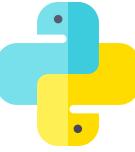
DR. ERIC CHOU

IEEE SENIOR MEMBER

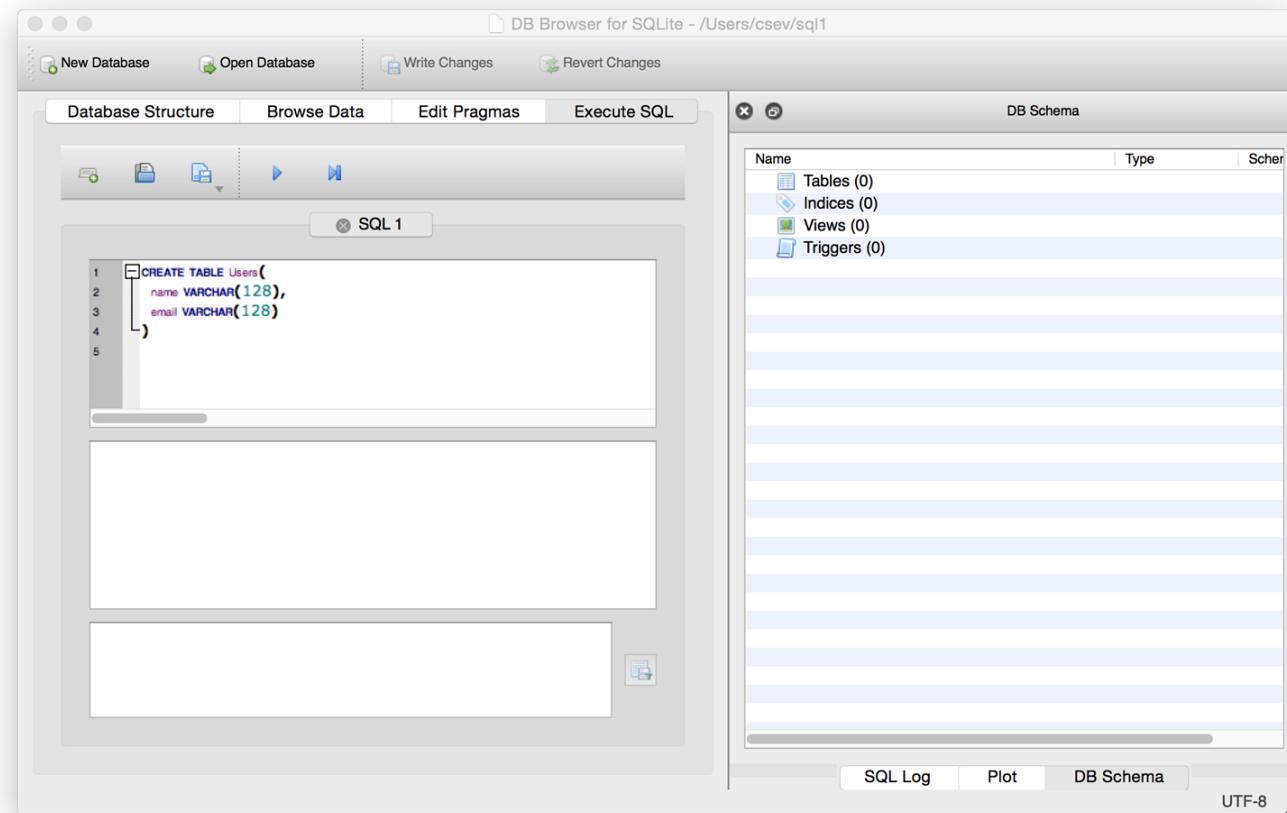


SQL Table Design

LECTURE 3



Start Simple - A Single Table



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

DB Browser for SQLite - /Users/csev/sql1

New Database Open Database Write Changes Revert Changes

Database Structure Browse Data Edit Pragmas Execute SQL

Create Table Modify Table Delete Table

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

DB Schema

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

SQL Log Plot DB Schema UTF-8

The screenshot shows the DB Browser for SQLite interface. The left panel, titled 'Database Structure', displays a table structure with one entry: 'Tables (1)' containing a single table named 'Users'. The schema for 'Users' is shown as a CREATE TABLE statement: 'CREATE TABLE Users(name VARCHAR(128), email VARCHAR(128))'. Below the table are sections for 'Indices (0)', 'Views (0)', and 'Triggers (0)'. The right panel, titled 'DB Schema', shows the same information in a more detailed tree view. 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: 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

Filter Filter

< < 0 - 0 of 0 > > Go to: 1

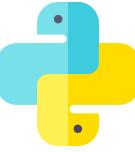
DB Schema

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

SQL Log Plot DB Schema UTF-8

Our table with four rows

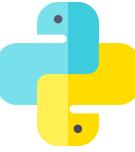
The screenshot shows the DB Browser for SQLite application interface. The main window displays the 'Browse Data' tab for a table named 'Users'. The table has two columns: 'name' and 'email'. There are four rows of data: 1. Chuck, csev@umich...; 2. Colleen, cvl@umich.edu; 3. Ted, ted@umich....; 4. Sally, a1@umich.edu. Below the table are navigation buttons for sorting and filtering. To the right, the 'DB Schema' tab is open, showing the CREATE TABLE statement for the 'Users' table. The schema table lists 'Tables (1)', 'Users', with its definition: 'CREATE TABLE Users(name VARCHAR(128), email VARCHAR(128))'. It also shows 'Indices (0)', 'Views (0)', and 'Triggers (0)'. At the bottom, there are tabs for 'SQL Log', 'Plot', 'DB Schema', and encoding 'UTF-8'.



SQL

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

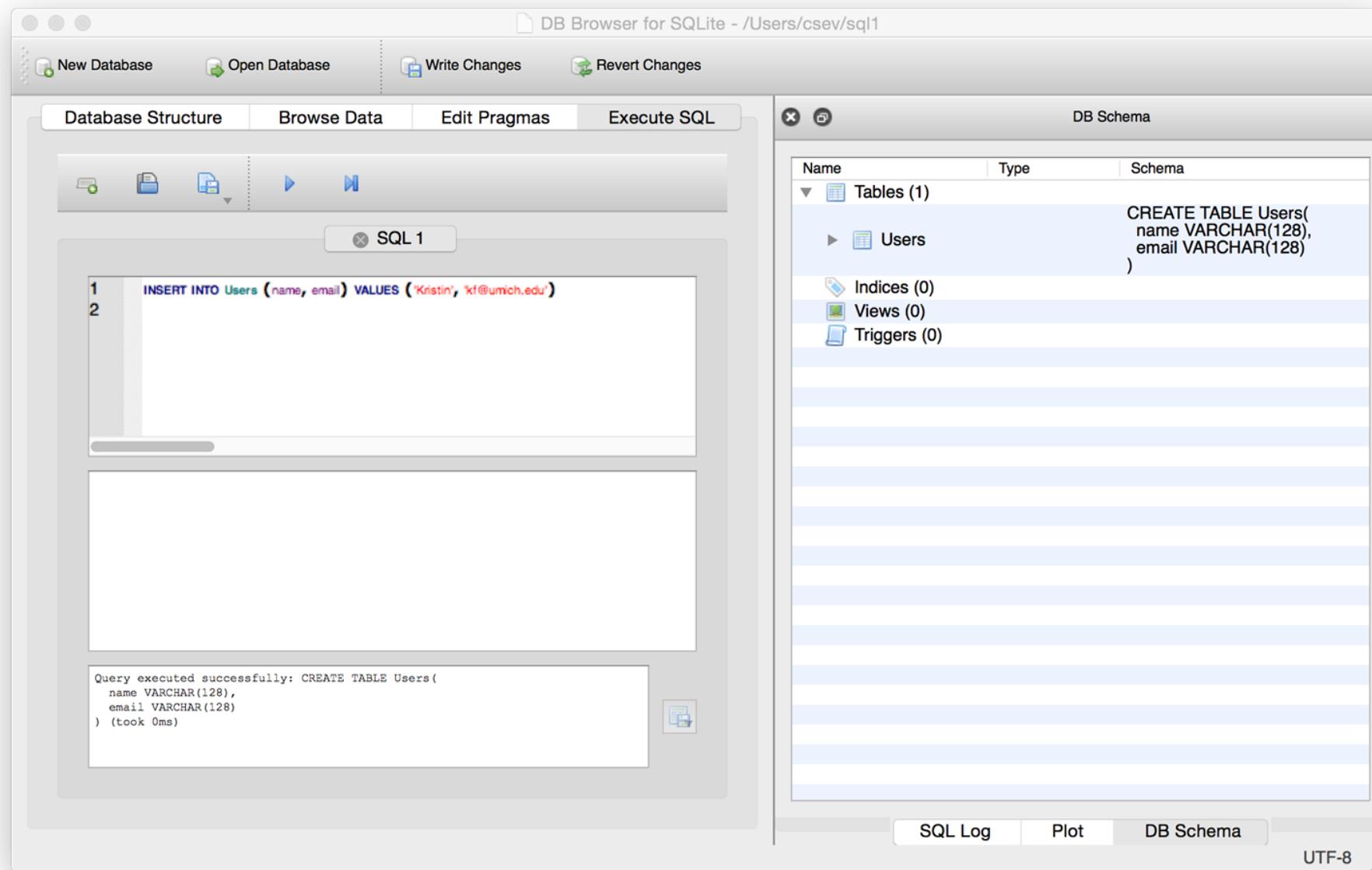
- Create data (a.k.a Insert)
- Retrieve data
- Update data
- Delete data



SQL: Insert

The Insert statement inserts a row into a table

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



The screenshot shows three windows of the DB Browser for SQLite application.

Left Window: Shows the SQL tab with the following content:

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

Below the SQL tab, the status bar displays:

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

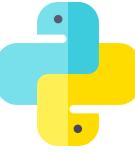
Middle Window: Shows the Browse Data tab for the 'Users' table. The table has columns 'name' and 'email'. The data is:

	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

Right Window: Shows the DB Schema tab. It lists the 'Tables (1)' section with one entry: 'Users'. The schema for the 'Users' table is:

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

A yellow arrow points from the 'DB Schema' tab towards the 'Browse Data' tab.



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

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, with Execute SQL selected. In the SQL pane, there is a single query: "DELETE FROM Users WHERE email='ted@umich.edu'". A message below the query states "Query executed successfully: DELETE FROM Users WHERE email='ted@umich.edu' (took 0ms)". To the right, the DB Schema pane displays the database structure with one table named "Users" defined by the schema: "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 selected. 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

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)



DB Browser for SQLite - /Users/csev/sql1

New Database Open Database Write Changes Revert Changes

Database Structure Browse Data Edit Pragmas Execute SQL

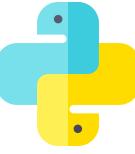
Table: Users New Record Delete Record

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

< < 1 - 4 of 4 > >| Go to: 1

SQL Log Plot DB Schema

UTF-8



SQL: Update

Allows the updating of a field with a where clause

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

DB Browser for SQLite - /Users/csev/sql1

New Database Open Database Write Changes Revert Changes

Database Structure Browse Data Edit Pragmas Execute SQL

SQL 1

1 UPDATE Users SET name='Charles' WHERE email='csev@umich.edu'
2 |

Query executed successfully: UPDATE Users SET name='Charles' WHERE email='csev@umich.edu' (took 0ms)

DB Schema

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

SQL Log Plot DB Schema

UTF-8

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 an UPDATE query: 'UPDATE Users SET name='Charles' WHERE email='csev@umich.edu''. Below the query is a message: 'Query executed successfully: UPDATE Users SET name='Charles' WHERE email='csev@umich.edu' (took 0ms)'. To the right, the 'DB Schema' tab is selected, showing a table named 'Users' with the schema: 'CREATE TABLE Users(name VARCHAR(128), email VARCHAR(128))'. Other sections like Indices, Views, and Triggers are listed as empty.

The screenshot shows three windows of the DB Browser for SQLite application.

Left Window (SQL Editor):

- Toolbar: New Database, Open Database, Write Changes, Revert Changes.
- Tab Bar: Database Structure, Browse Data, Edit Pragmas, Execute SQL (selected).
- Toolbar: Create Table, Create Index, Create View, Create Trigger, SQL 1.
- Text Area:

```
1 UPDATE Users SET name='Charles' WHERE email='csev@umich.edu'
2
```
- Status Bar: Query executed successfully: UPDATE Users SET name='Charles' WHERE email='csev@umich.edu' (took 0ms)

Middle Window (Data Browser):

- Toolbar: New Database, Open Database, Write Changes, Revert Changes.
- Tab Bar: Database Structure, Browse Data, Edit Pragmas, Execute SQL.
- Table View:

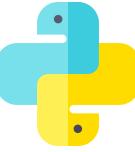
	name	email
1	Charles	csev@umich...
2	Colleen	cvl@umich.edu
3	Sally	a1@umich.edu
4	Kristin	kf@umich.edu
- Buttons: New Record, Delete Record.
- Page Navigation: < < 1 - 4 of 4 > > Go to: 1

Right Window (Schema Browser):

- Toolbar: New Database, Open Database, Write Changes, Revert Changes.
- Tab Bar: Database Structure, Browse Data, Edit Pragmas, Execute SQL.
- Table of Contents:

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

A yellow arrow points from the "DB Schema" tab in the middle window to the "DB Schema" tab in the right window.



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

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', containing four rows of data: Charles (csev@umich.edu), Colleen (cvl@umich.edu), Sally (a1@umich.edu), and Kristin (kf@umich.edu). Below the table, a message indicates 4 Rows returned from the query. To the right, the DB Schema panel shows the table 'Users' with its CREATE TABLE definition: 'CREATE TABLE Users(name VARCHAR(128), email VARCHAR(128))'. Other schema elements like Indices, Views, and Triggers are listed as zero. The bottom navigation bar includes SQL Log, Plot, and DB Schema tabs, with DB Schema being the active one.



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)

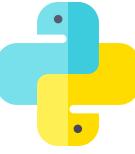
DB Schema

Name	Type	Schema
Tables (1)		
Users	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 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' (which is selected). On the left, there's a toolbar with icons for file operations like New, Open, Save, and Print. A central panel contains an SQL editor with the query 'SELECT * FROM Users WHERE email='csev@umich.edu' and its result set, which shows one row: Charles with email csev@umich.edu. Below the result set is a message indicating 1 row was returned. To the right is a 'DB Schema' panel showing the table 'Users' with its schema: 'CREATE TABLE Users(name VARCHAR(128), email VARCHAR(128))'. The bottom of the window has tabs for 'SQL Log', 'Plot', and 'DB Schema' (selected), and a status bar showing 'UTF-8'.



Sorting with ORDER BY

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

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

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

DB Browser for SQLite - /Users/csev/sql1

New Database Open Database Write Changes Revert Changes

Database Structure Browse Data Edit Pragmas Execute SQL

SQL 1

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

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

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

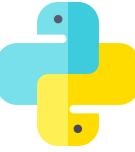
DB Schema

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

SQL Log Plot DB Schema

UTF-8

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 ORDER BY email' and its results. The results table has columns 'name' and 'email', containing 4 rows of data. Below the table, a message indicates 4 rows were returned. To the right, the DB Schema tab is active, showing the table 'Users' with its schema: 'CREATE TABLE Users(name VARCHAR(128), email VARCHAR(128))'. The schema also lists Indices, Views, and Triggers, all of which are currently 0.



SQL Summary

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

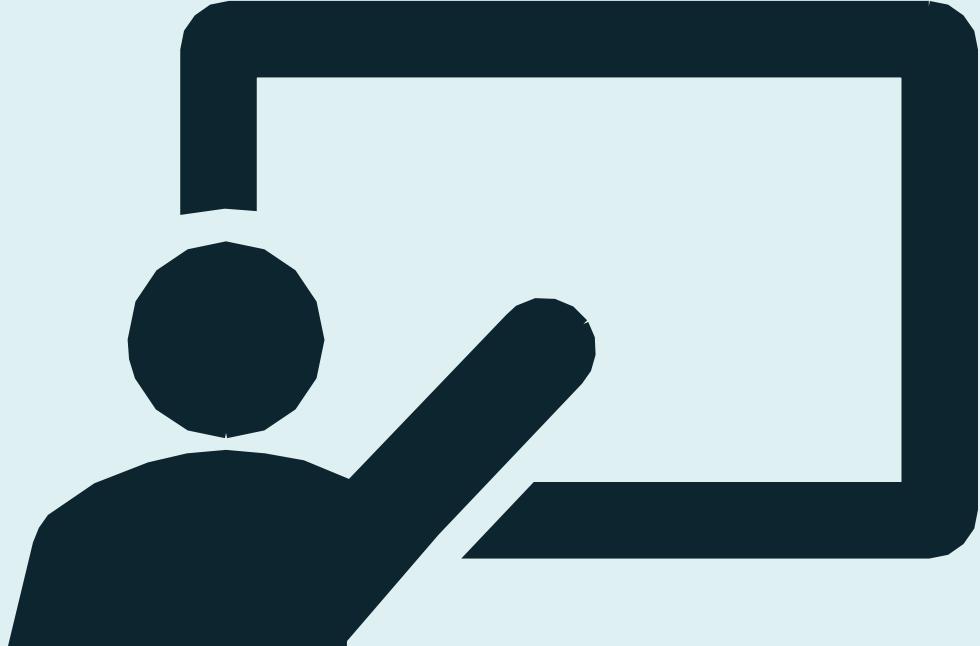
```
DELETE FROM Users WHERE email='ted@umich.edu'
```

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

```
SELECT * FROM Users
```

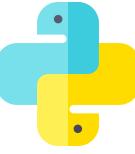
```
SELECT * FROM Users WHERE email='csev@umich.edu'
```

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



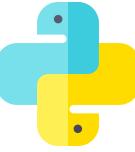
Data Models

LECTURE 4



From the Use of Tools to Programming

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



Complex Data Models and Relationships

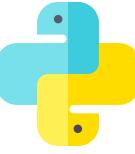
Relational Model

Activity Code	Activity Name
23	Patching
24	Overlay
25	Crack Sealing

Key = 24

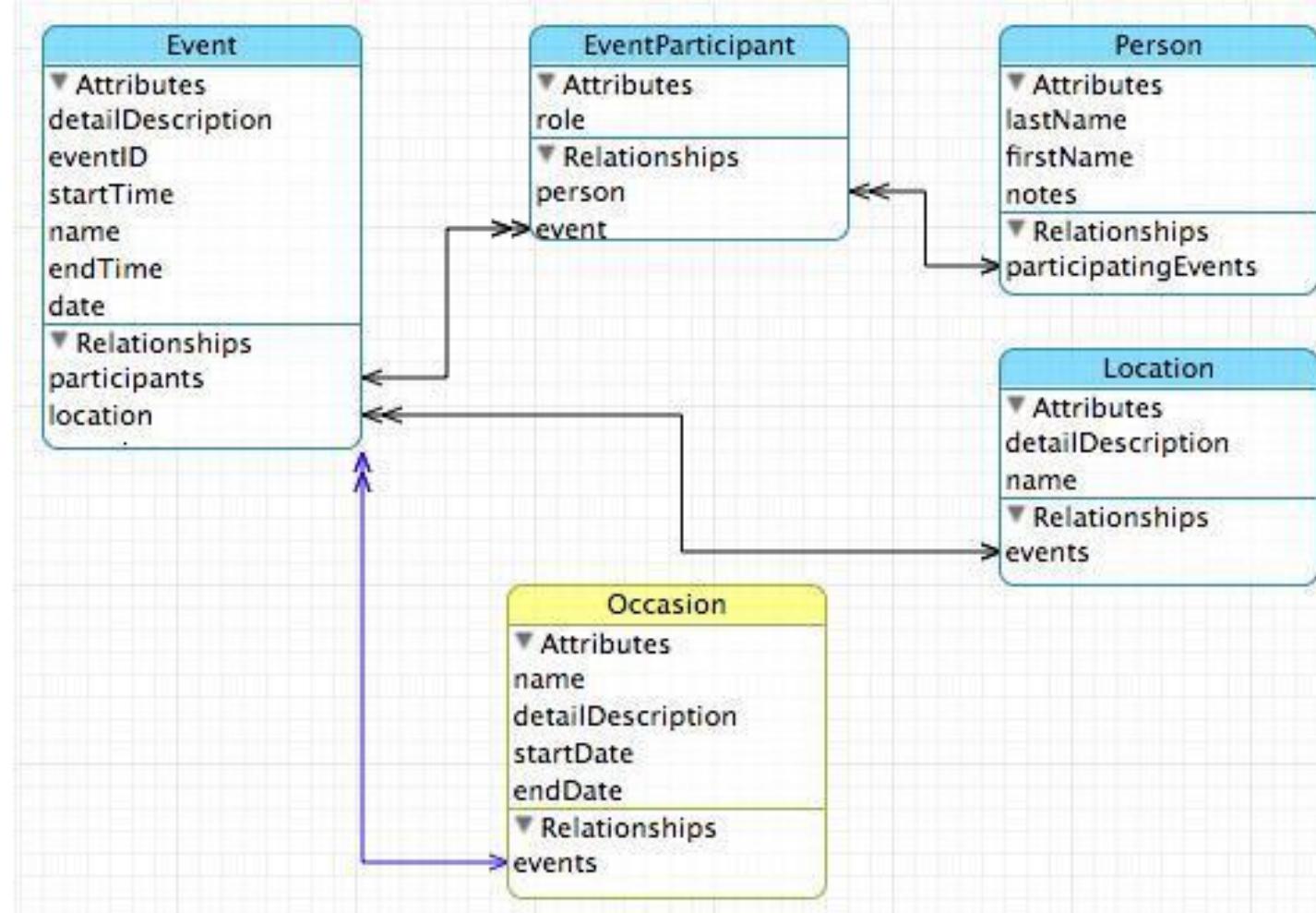
Activity Code	Date	Route No.
24	01/12/01	I-95
24	02/08/01	I-66

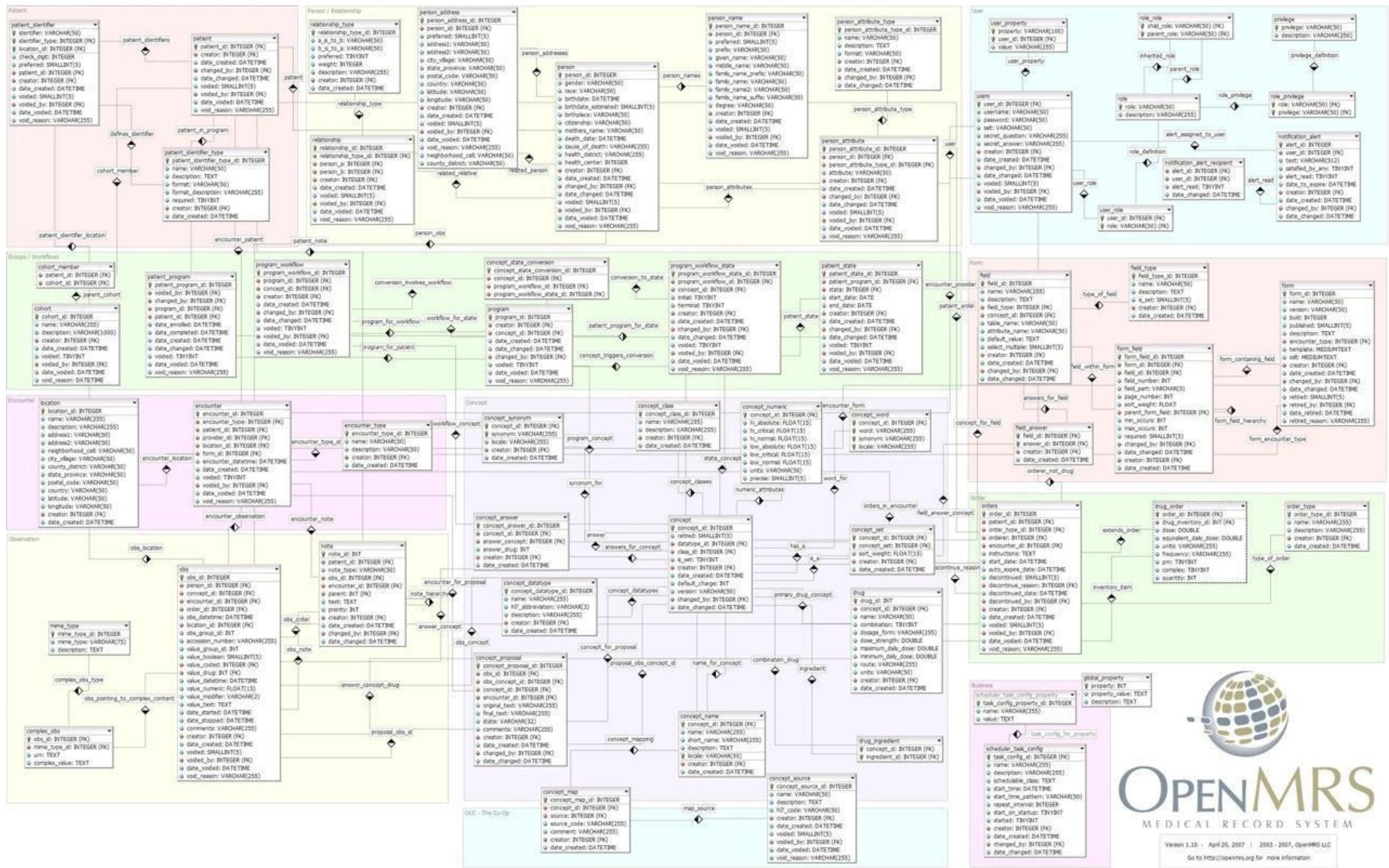
Date	Activity Code	Route No.
01/12/01	24	I-95
01/15/01	23	I-495
02/08/01	24	I-66



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...



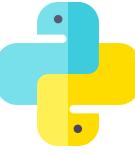


OPENMRS
MEDICAL RECORD SYSTEM

Version 1.10 - April 20, 2007 | 2003 - 2007, OpenHRS LLC
Go to <http://openhrs.org> for more information.



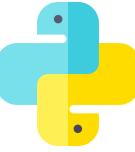
Learning Channel



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

<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?

- Once we define objects, we need to define the relationships between objects

Len	Album	
Genre	Artist	Rating
Track	Count	

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

Track

Album

Artist

Genre

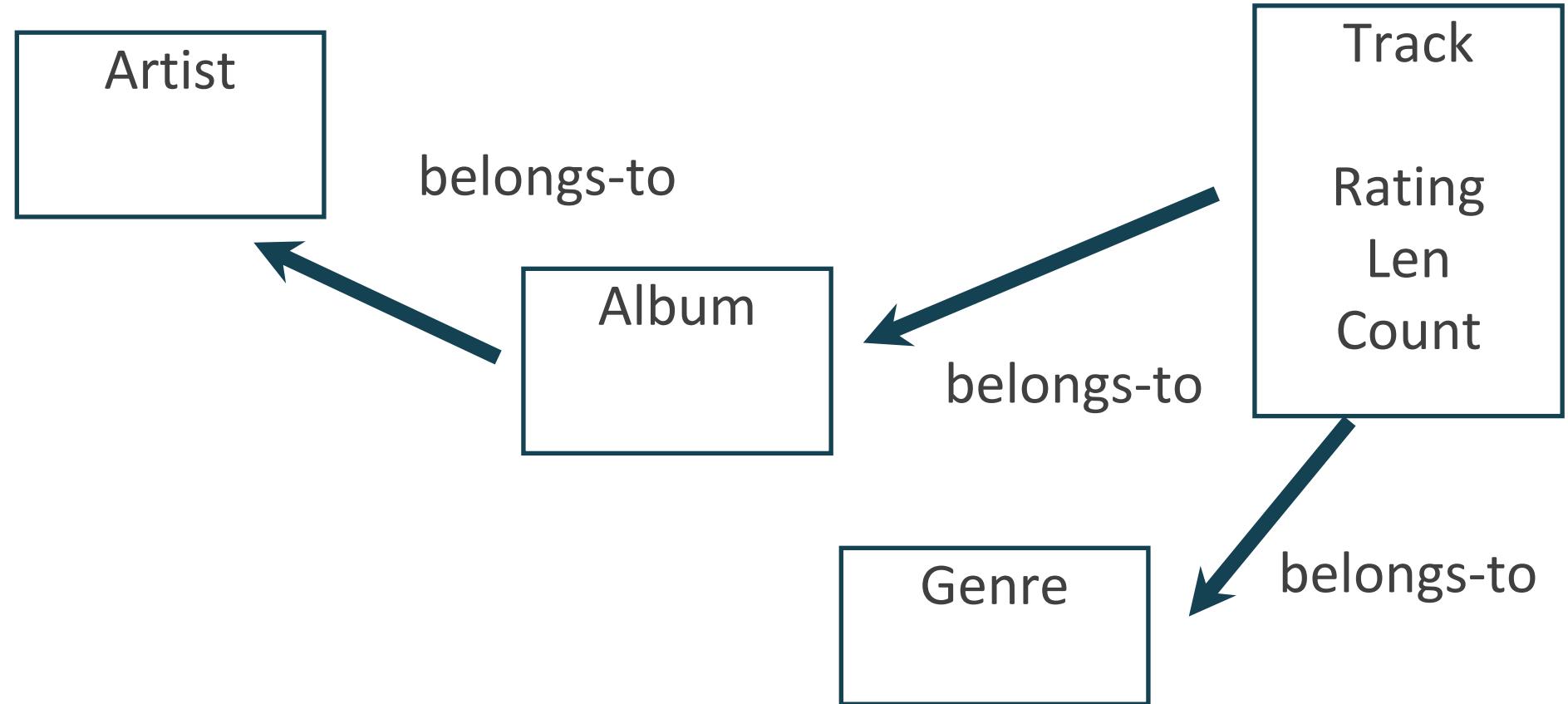
Rating

Len

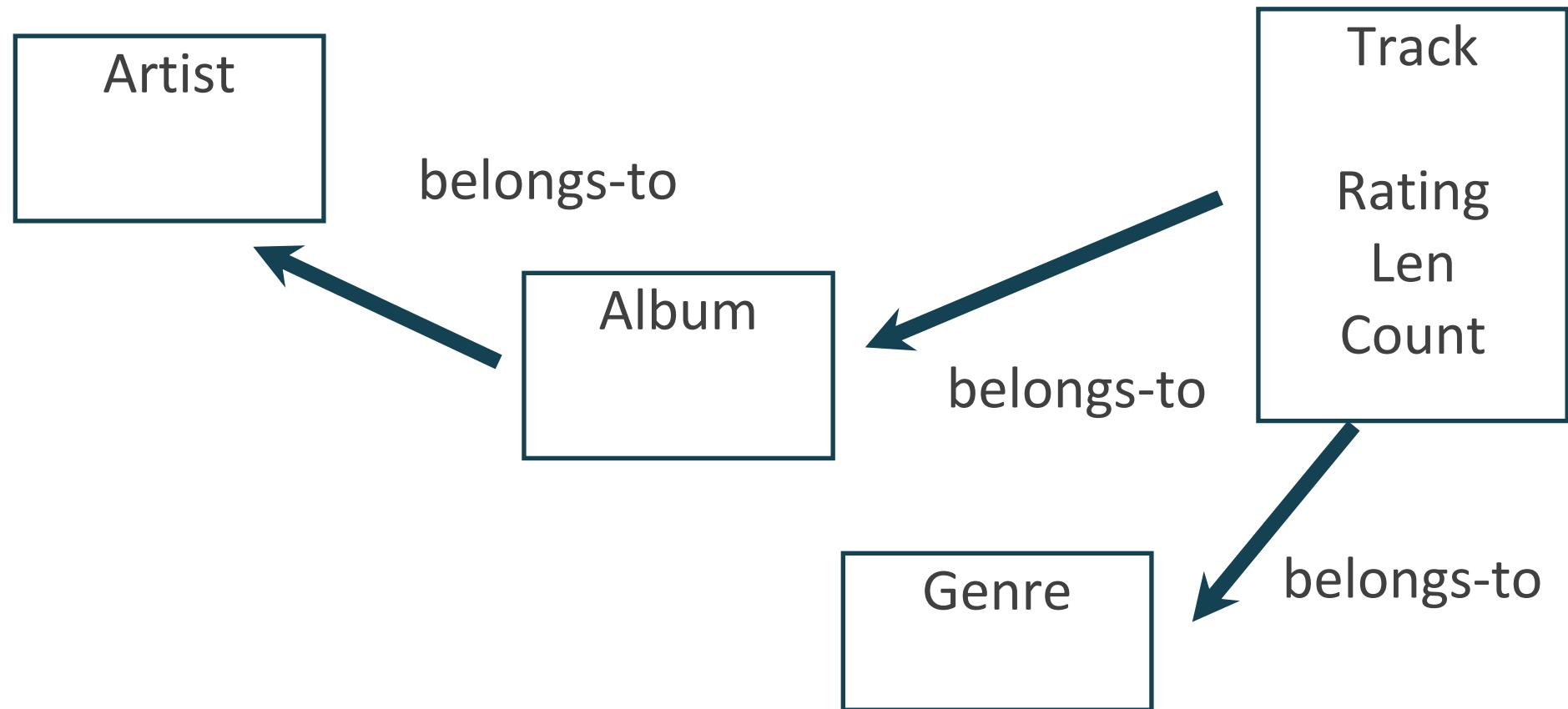
Count

<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

Track
Album
Artist
Genre
Rating
Len
Count



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



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



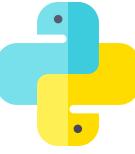
Relational Database

LECTURE 5



Representing Relationships in a Database

ACTIVITY

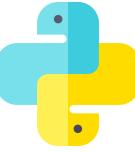


Database Normalization (3NF)

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

<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

- 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?

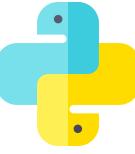


Integer Reference Pattern

We use integers to reference rows in another table

id	name
Filter	Filter
1	Led Zeppelin
2	AC/DC

id	artist_id	title
1	2	Who Made Who
2	1	IV



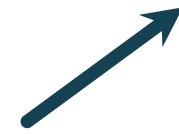
Three Kinds of Keys

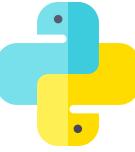
Primary key - generally an integer auto-increment field

Logical key - What the outside world uses for lookup

Foreign key - generally an integer key pointing to a row in another table

*Album
id
title
artist_id
...*



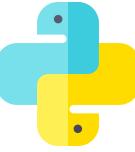


Key Rules

Best practices

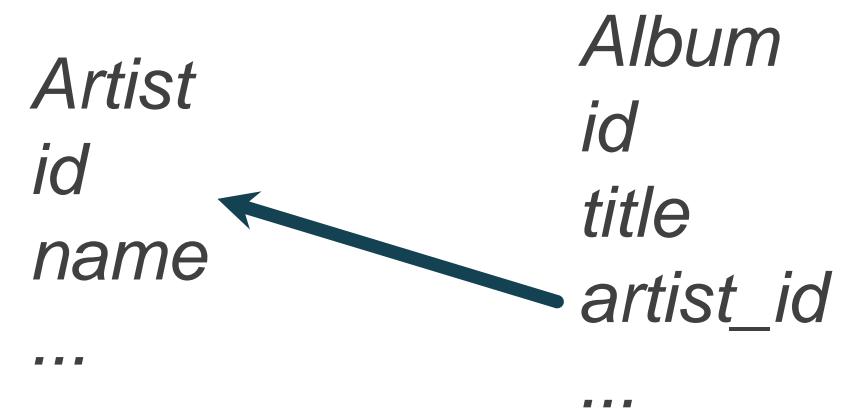
- Never use your logical key as the primary key
- Logical keys can and do change, albeit slowly
- Relationships that are based on matching string fields are less efficient than integers

*User
id
login
password
name
email
created_at
modified_at
login_at*



Foreign Keys

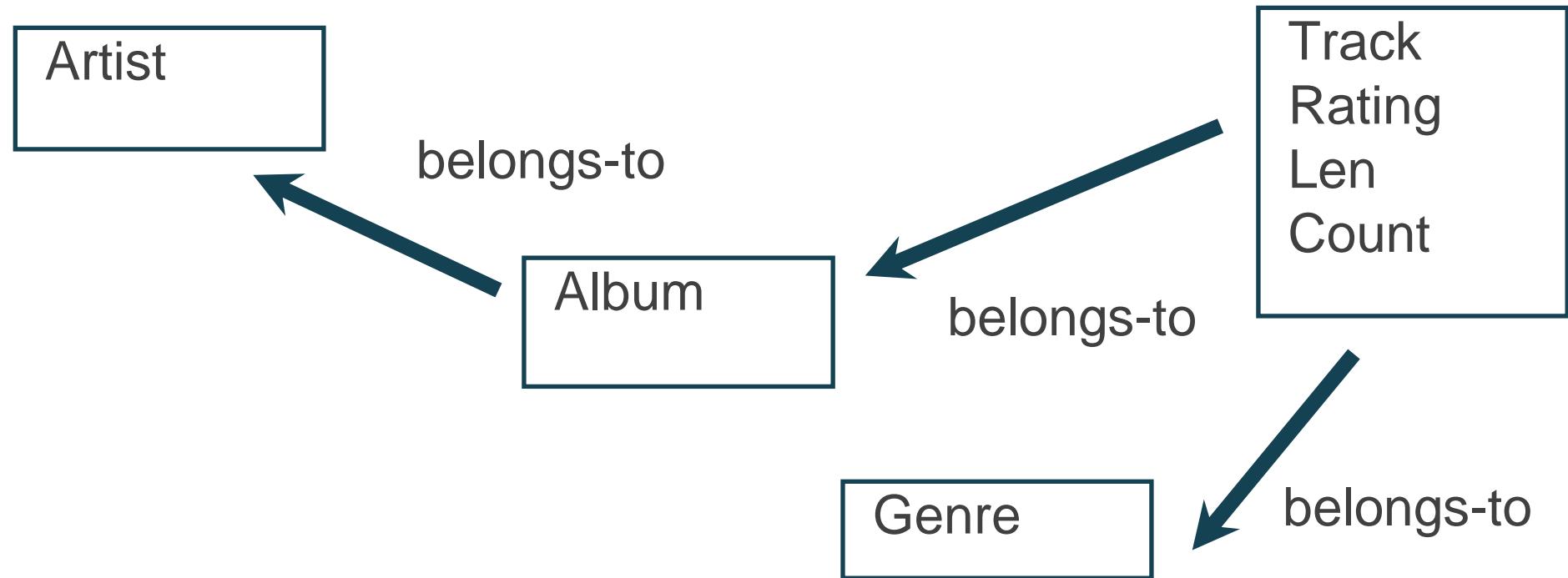
- A foreign key is when a table has a column that contains a key which points to the primary key of another table.
- When all primary keys are integers, then all foreign keys are integers - this is good - very good



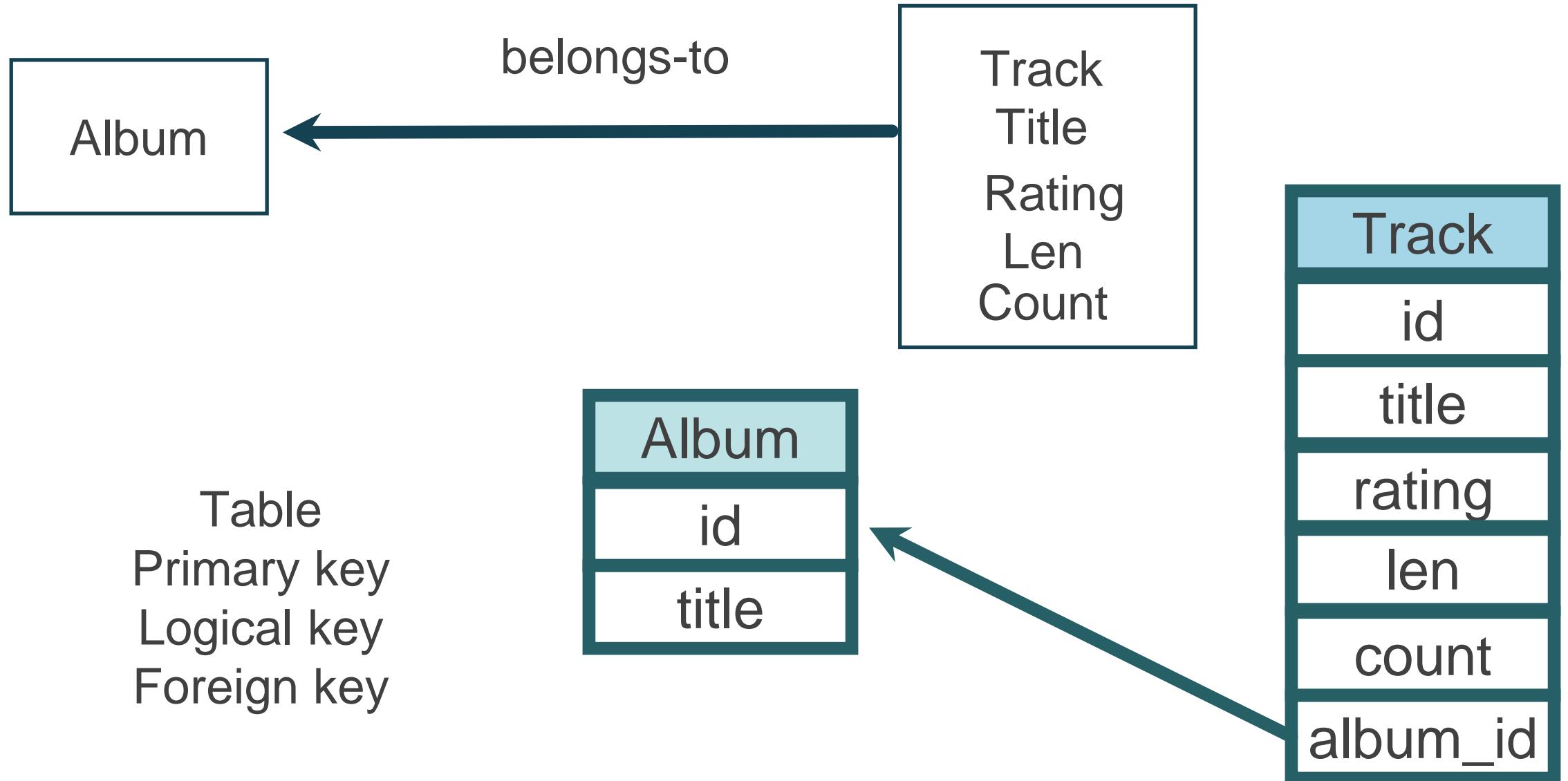


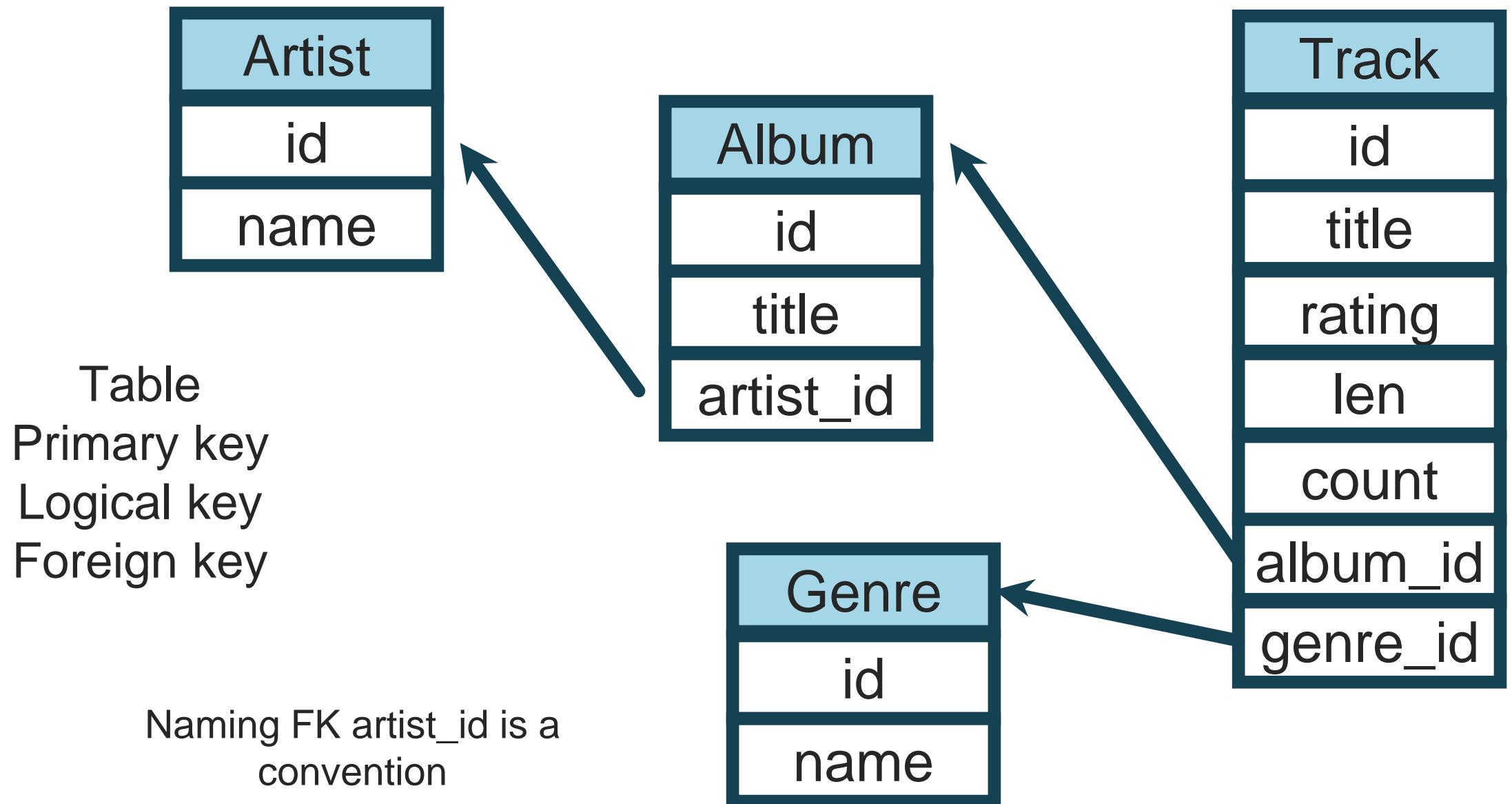
Relationship Building (in tables)

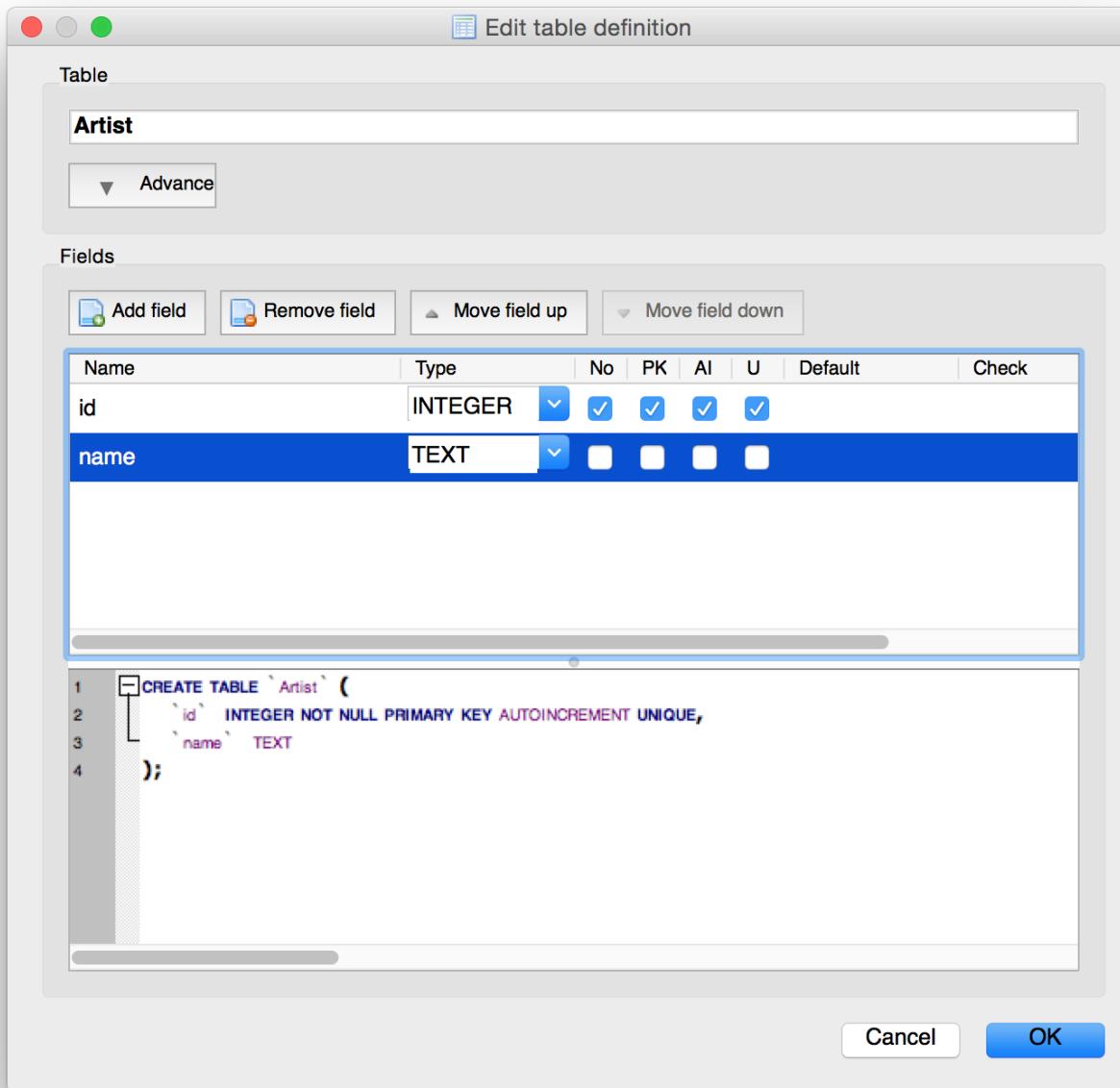
ACTIVITY

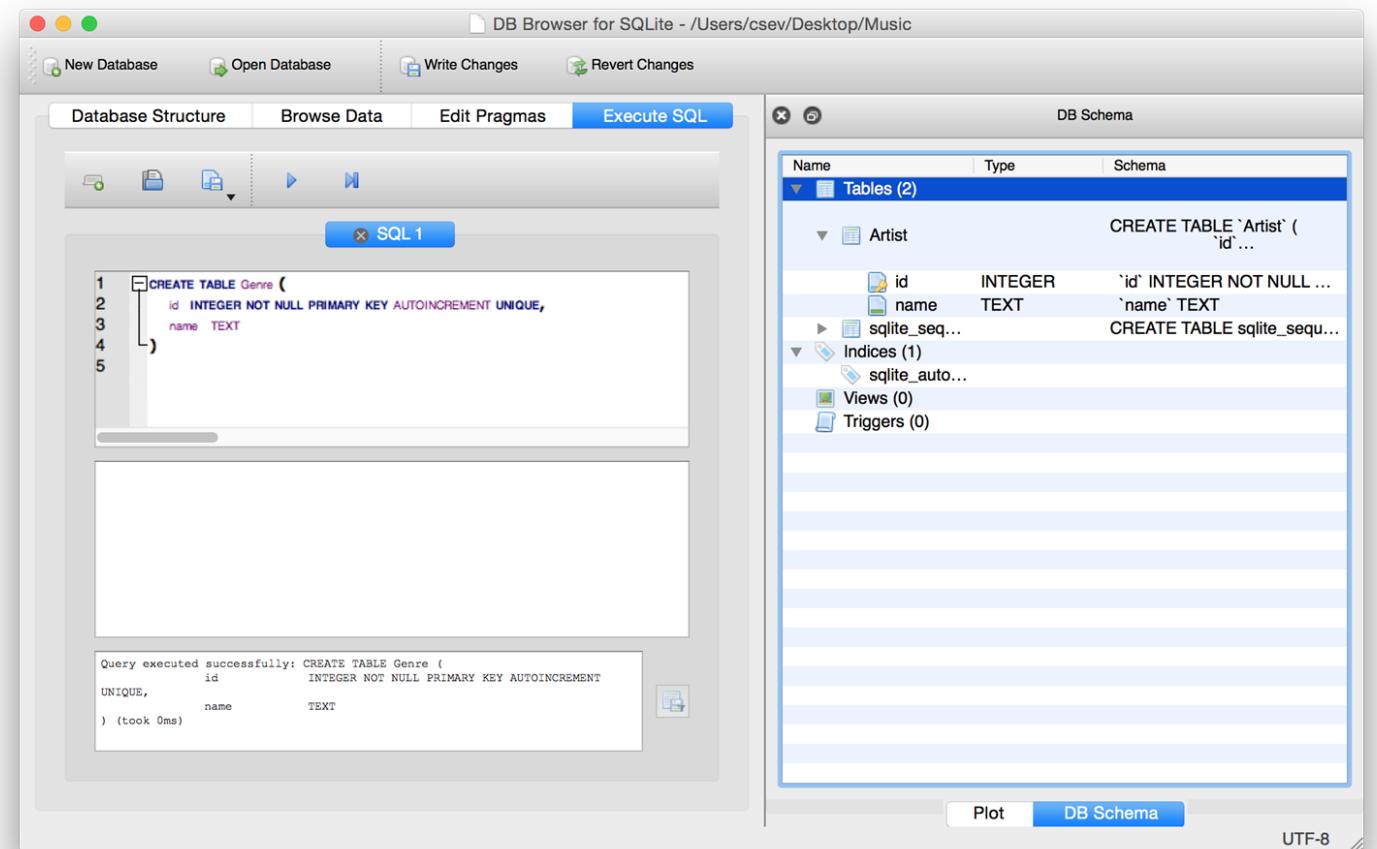


✓ Hells Bells	5:13	AC/DC	Who Made Who	Rock	★★★★★	61
✓ Shake Your Foundations	3:54	AC/DC	Who Made Who	Rock	★★★★★	70
✓ Chase the Ace	3:01	AC/DC	Who Made Who	Rock		56
✓ For Those About To Rock (We ...	5:54	AC/DC	Who Made Who	Rock	★★★★★	61
✓ Dúlamán	3:43	Altan	Natural Wonders M...	New Age		31
✓ Rode Across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









```
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

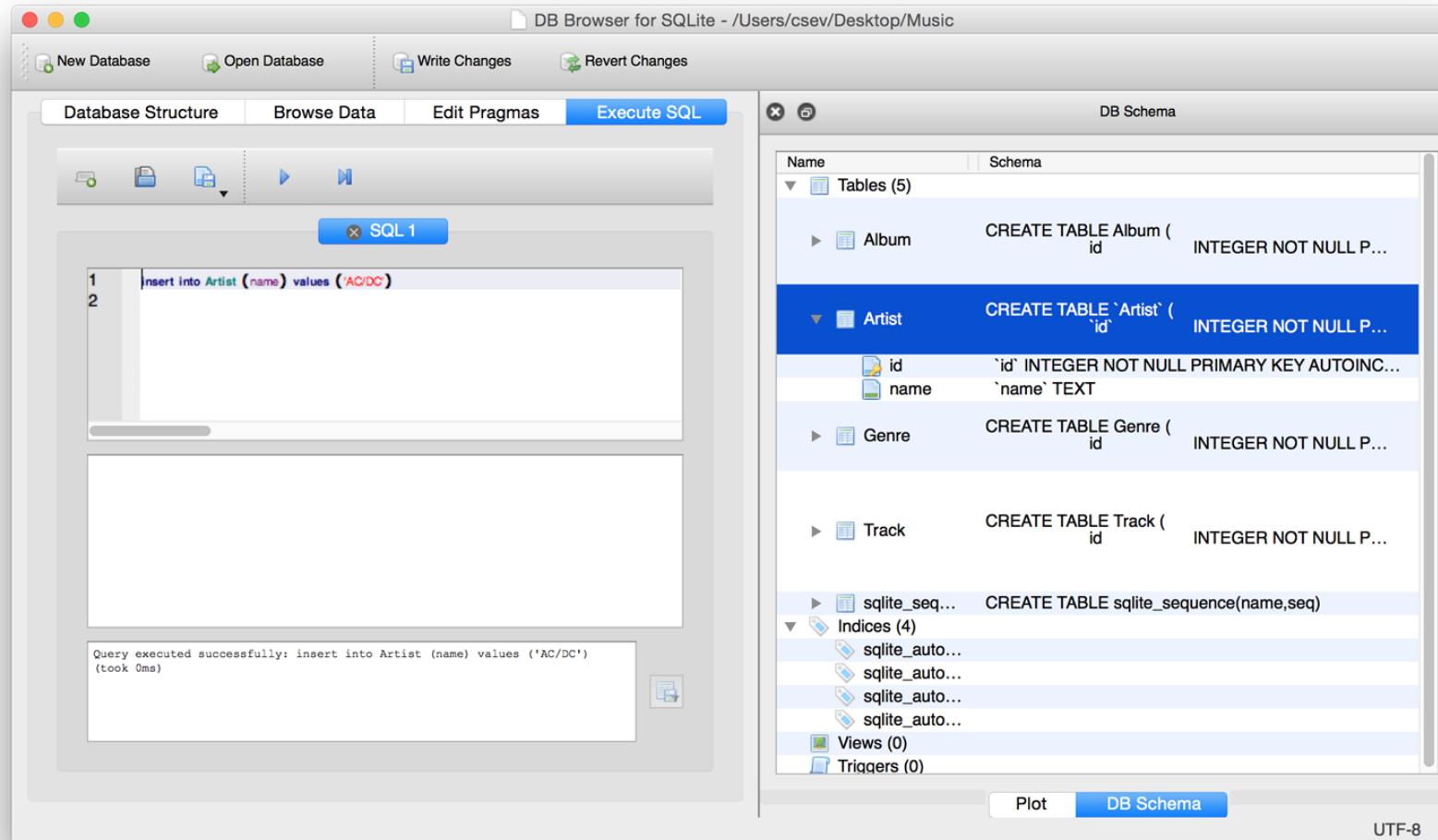
Name	Type	Schema
Tables (5)		
Album		<pre>CREATE TABLE "Album" (`id` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE, `artist_id` INTEGER, `title` TEXT)</pre>
Artist		<pre>CREATE TABLE `Artist` (`id` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE, `name` TEXT)</pre>
Genre		<pre>CREATE TABLE Genre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

DB Schema

Name	Schema
Tables (5)	
Album	<pre>CREATE TABLE "Album" (`id` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,</pre>
Artist	<pre>CREATE TABLE `Artist` (`id` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,</pre>
Genre	<pre>CREATE TABLE Genre (`id` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,</pre>
Track	<pre>CREATE TABLE Track (id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,</pre>
sqlite_sequence	<pre>CREATE TABLE sqlite_sequence(name,seq)</pre>
Indices (4)	
sqlite_autoindex_Album_1	
sqlite_autoindex_Track_1	

Plot DB Schema

UTF-8



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

The screenshot shows three windows of the DB Browser for SQLite application:

- SQL Editor (Left Window):** Shows the SQL query `insert into Artist (name) values ('AC/DC')`. A message below states: "Query executed successfully: insert into Artist (name) values ('AC/DC') (took 0ms)".
- Database Structure (Middle Window):** Displays the "Artist" table with two records:

id	name
1	Led Zeppelin
2	AC/DC
- DB Schema (Right Window):** Shows the database schema with the following CREATE TABLE statements:
 - 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)`
 - Indices (4):
 - sqlite_autoindex_Album_1
 - sqlite_autoindex_Album_2
 - sqlite_autoindex_Artist_1
 - sqlite_autoindex_Track_1

A yellow arrow points from the "Artist" table in the Database Structure window to the "Artist" table in the DB Schema window.

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

The screenshot shows the DB Browser for SQLite application interface. The left pane displays the 'Genre' table with two rows: Rock and Metal. The right pane shows the database schema with five tables: Album, Artist, Genre, Track, and sqlite_sequence. The 'Genre' table's schema is displayed.

Table: Genre

	id	name
1	1	Rock
2	2	Metal

DB Schema

Name	Schema
Tables (5)	
Album	CREATE TABLE Album (id INTEGER NOT NULL PRI...)
Artist	CREATE TABLE `Artist` (`id` INTEGER NOT NULL PRI...)
Genre	CREATE TABLE Genre (id INTEGER NOT NULL PRI...)
Track	CREATE TABLE Track (id INTEGER NOT NULL PRI...)
sqlite_sequence	CREATE TABLE sqlite_sequence(name,seq)
Indices (4)	
sqlite_autoindex_Album_1	
sqlite_autoindex_Artist_1	
sqlite_autoindex_Genre_1	
sqlite_autoindex_Track_1	
Views (0)	
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	Filter
1	1	Black Dog	2	1	297	5	0
2	2	Stairway	2	1	482	5	0
3	3	About to Rock	1	2	313	5	0
4	4	Who Made Who	1	2	207	5	0

Track

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

Album

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

Artist

id	name
Filter	Filter
1	Led Zeppelin
2	AC/DC

Genre

id	name
Filter	Filter
1	Rock
2	Metal

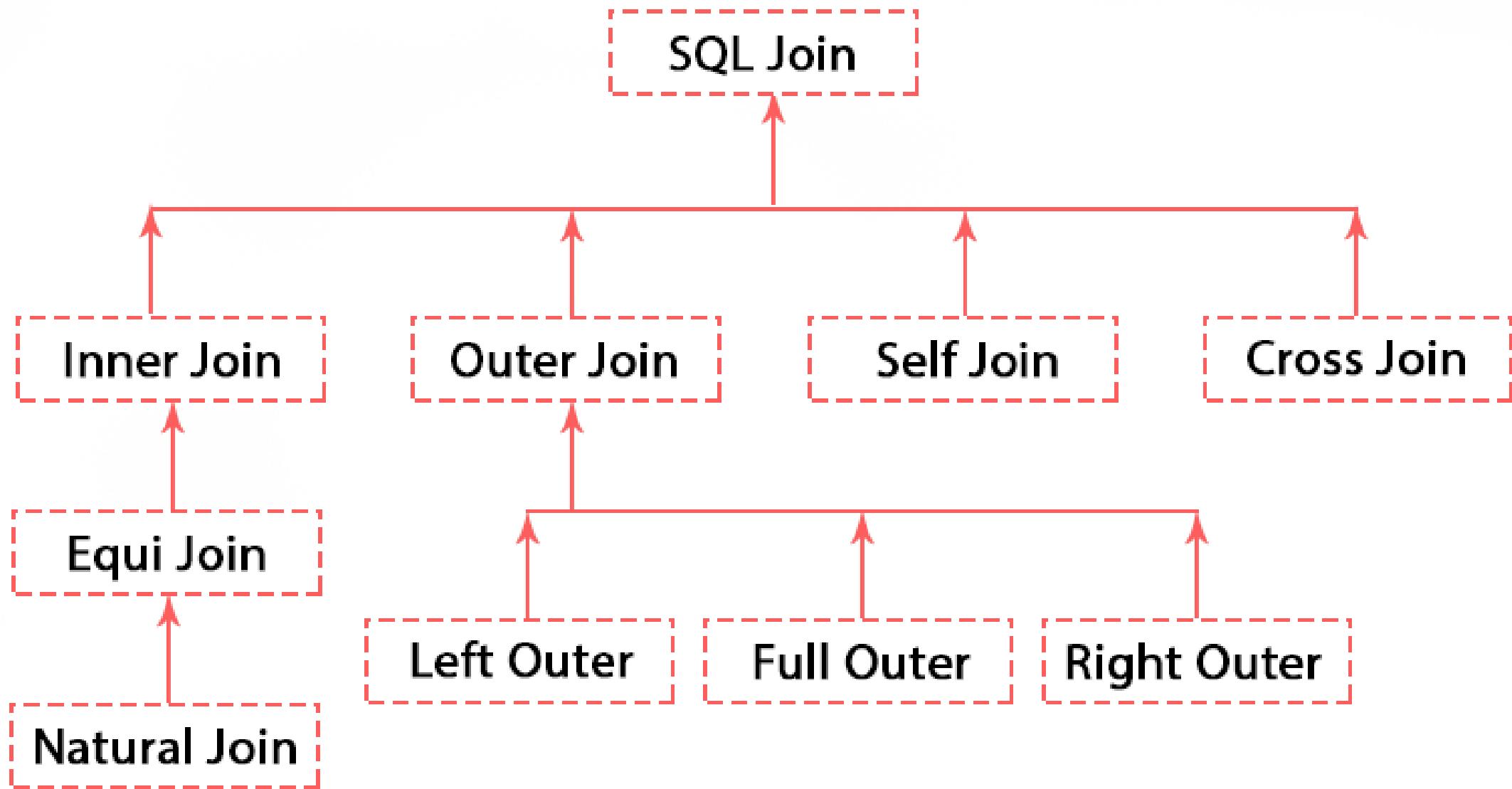


Learning Channel



Using Join Across Tables

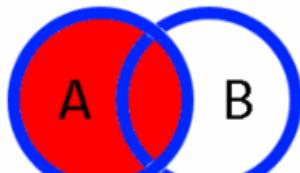
ACTIVITY



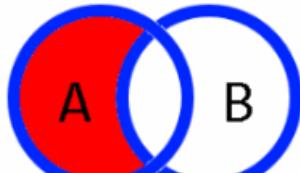
(SQL JOIN TYPES)

SQL JOINS

LEFT OUTER JOIN



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



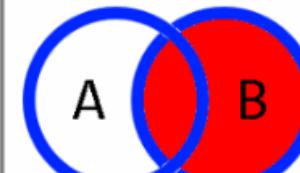
```
SELECT *  
FROM TableA a  
LEFT JOIN TableB b  
ON a.KEY = b.KEY  
WHERE b.KEY IS NULL
```

INNER JOIN

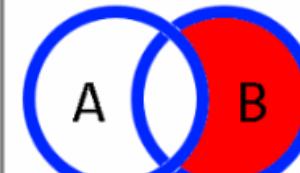


```
SELECT *  
FROM TableA a  
INNER JOIN TableB b  
ON a.KEY = b.KEY
```

RIGHT OUTER JOIN

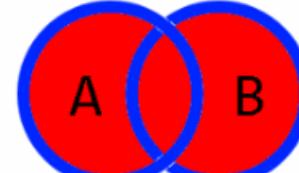


```
SELECT *  
FROM TableA a  
RIGHT JOIN TableB b  
ON a.KEY = b.KEY
```

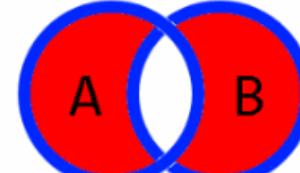


```
SELECT *  
FROM TableA a  
RIGHT JOIN TableB b  
ON a.KEY = b.KEY  
WHERE a.KEY IS NULL  
OR b.KEY IS NULL
```

FULL OUTER JOIN

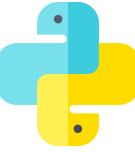


```
SELECT *  
FROM TableA a  
FULL OUTER JOIN TableB b  
ON a.KEY = b.KEY
```



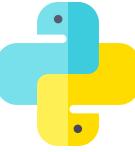
```
SELECT *  
FROM TableA a  
FULL OUTER JOIN TableB b  
ON a.KEY = b.KEY  
WHERE a.KEY IS NULL  
OR b.KEY IS NULL
```





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

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

Album

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

Artist

id	name
Filter	Filter
1	Led Zepplin
2	AC/DC

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

What we want
to see

The tables that
hold the data

How the tables
are linked



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

id	name
Filter	Filter
1	Led Zepplin
2	AC/DC

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

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

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

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

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

	title	name
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

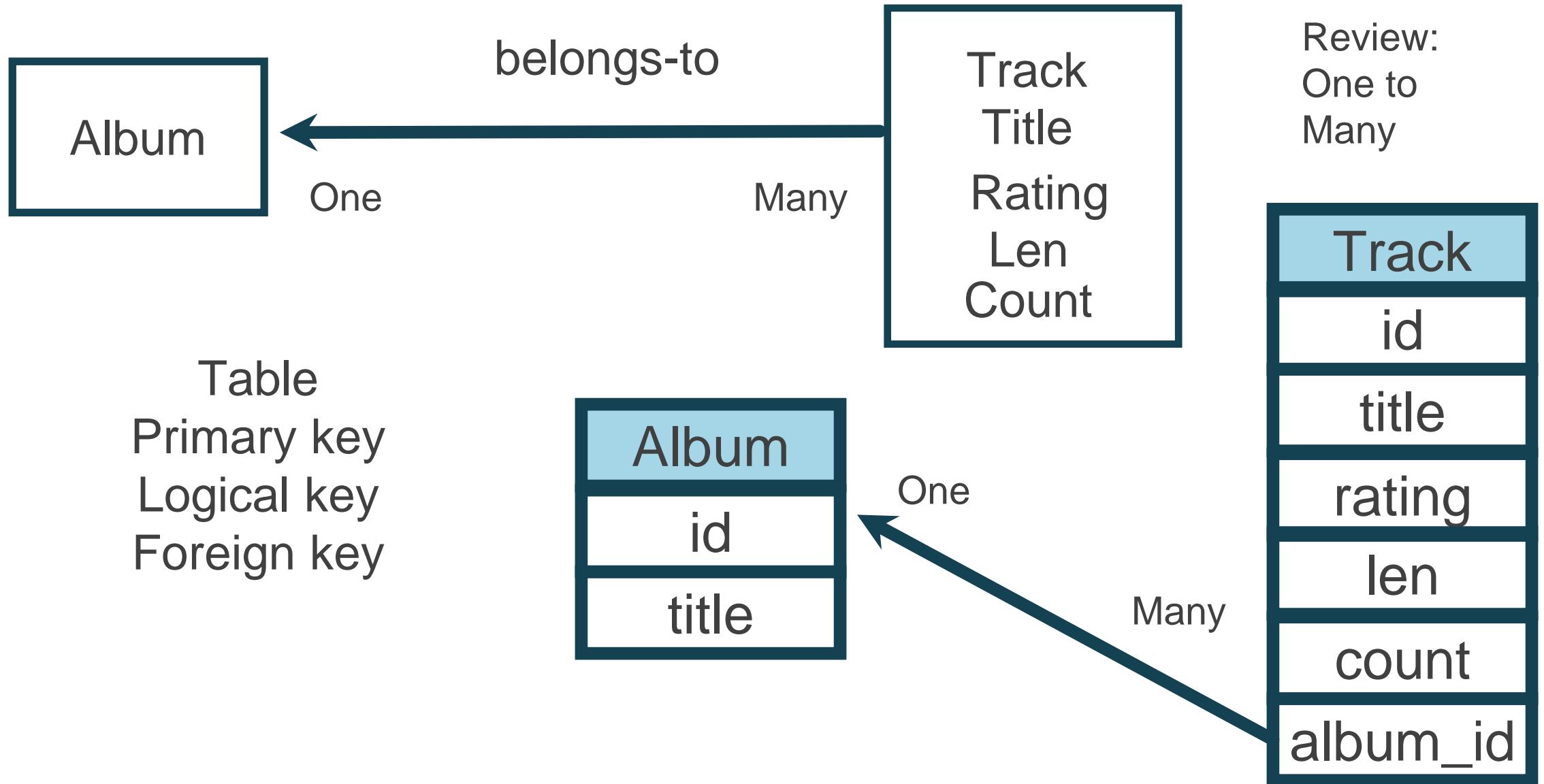
- What we want to see
- The tables which hold the data
- How the tables are linked

<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	Danger Zone	Blues/R&B	★★★★★	10
<input checked="" type="checkbox"/> Track 03	3:26	Billy Price	Danger Zone	Blues/R&B	★★★★★	10
<input checked="" type="checkbox"/> Track 04	4:17	Billy Price	Danger Zone	Blues/R&B	★★★★★	10
<input checked="" type="checkbox"/> Track 05	3:50	Billy Price	Danger Zone	Blues/R&B	★★★★★	10
<input checked="" type="checkbox"/> War Pigs/Luke's Wall	7:58	Black Sabbath	Paranoid	Led Zepplin	IV	Rock
<input checked="" type="checkbox"/> Paranoid	2:53	Black Sabbath	Paranoid	Led Zepplin	IV	Rock
<input checked="" type="checkbox"/> Planet Caravan	4:35	Black Sabbath	Paranoid	Led Zepplin	IV	Rock
<input checked="" type="checkbox"/> Iron Man	5:59	Black Sabbath	Paranoid	Led Zepplin	IV	Rock
<input checked="" type="checkbox"/> Electric Funeral	4:53	Black Sabbath	Paranoid	Led Zepplin	IV	Rock
<input checked="" type="checkbox"/> Hand of Doom	7:10	Black Sabbath	Paranoid	Led Zepplin	IV	Rock
<input checked="" type="checkbox"/> Rat Salad	2:30	Black Sabbath	Paranoid	Led Zepplin	IV	Rock
<input checked="" type="checkbox"/> Jack the Stripper/Fairies Wear ...	6:14	Black Sabbath	Paranoid	AC/DC	Who Made Who	Metal
<input checked="" type="checkbox"/> Bomb Squad (TECH)	3:28	Brent	Brent's Album	AC/DC	Who Made Who	Metal
<input checked="" type="checkbox"/> clay techno	4:36	Brent	Brent's Album	AC/DC	Who Made Who	Metal
<input checked="" type="checkbox"/> Heavy	3:08	Brent	Brent's Album	AC/DC	Who Made Who	Metal
<input checked="" type="checkbox"/> Hi metal man	4:20	Brent	Brent's Album	AC/DC	Who Made Who	Metal
<input checked="" type="checkbox"/> Mistro	2:58	Brent	Brent's Album	AC/DC	Who Made Who	Metal



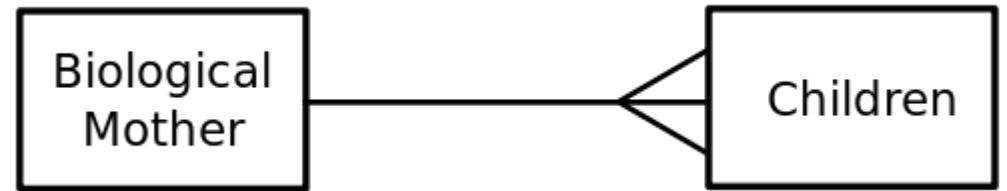
Many-To-Many Relationships

ACTIVITY



id		name	
Filter	Filter	Filter	Filter
1		Rock	
2		Metal	

One

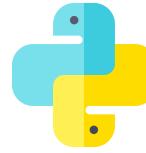


One

Many

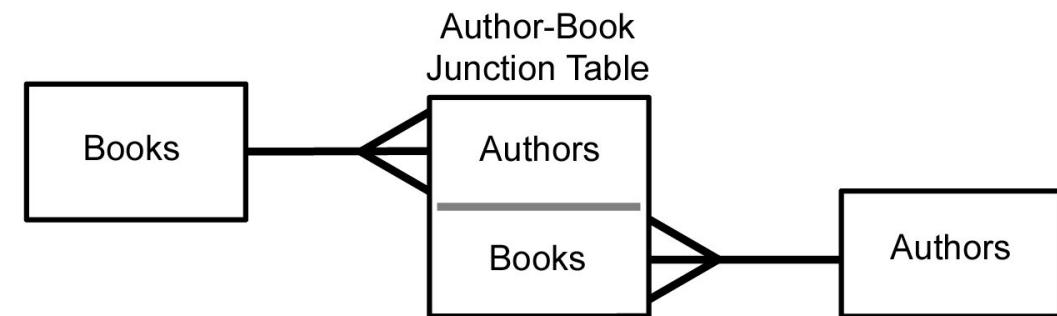
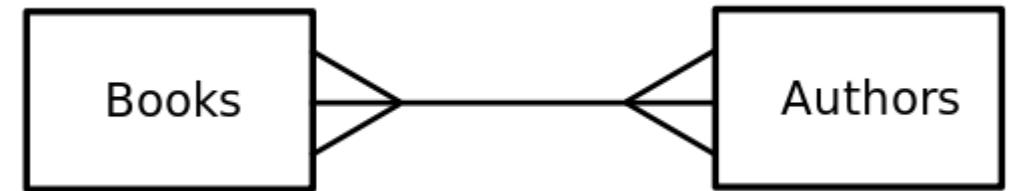
Many

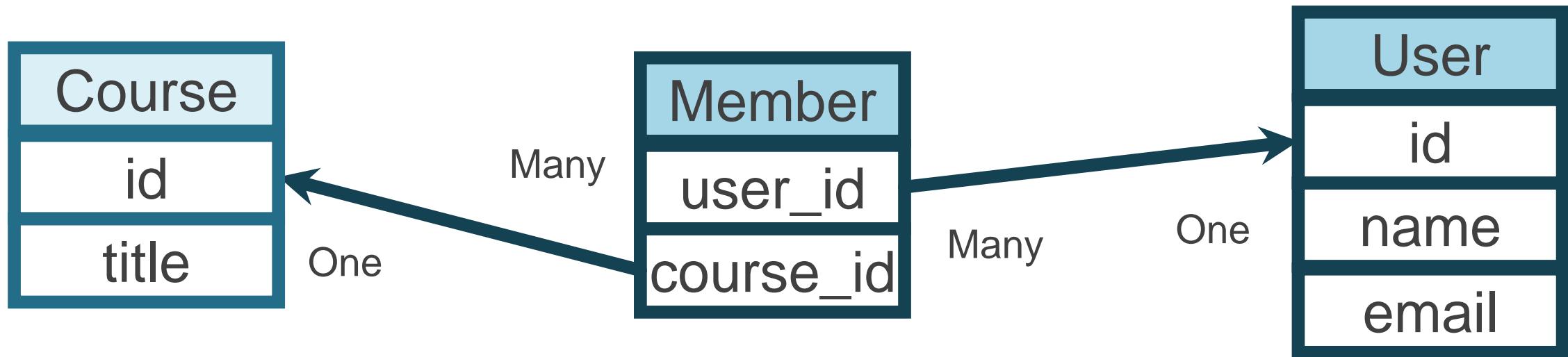
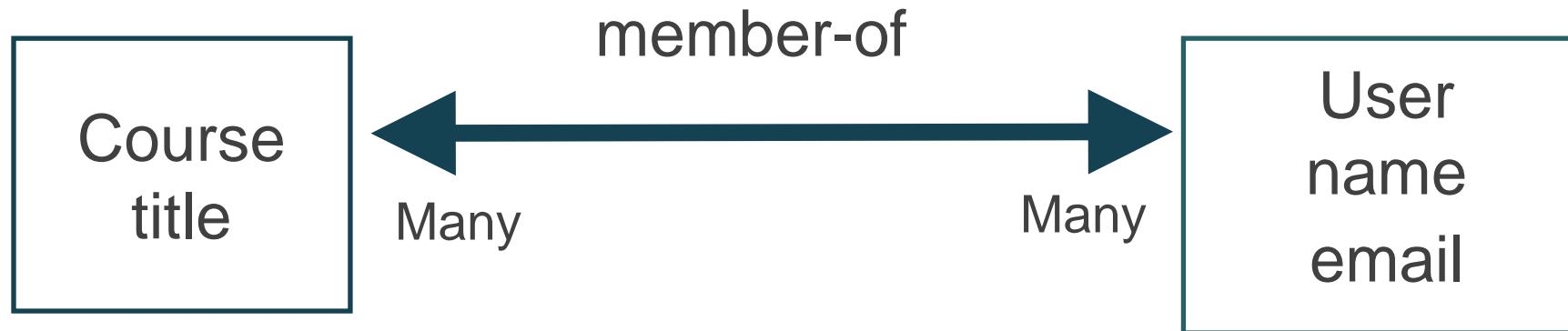
id		title		album_id	genre_id	len	rating	count
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1		Black Dog		2	1	297	5	0
2		Stairway		2	1	482	5	0
3		About to Rock		1	2	313	5	0
4		Who Made Who		1	2	207	5	0



Many to Many

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





```
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

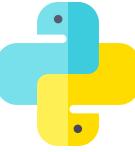
New Database Open Database Write Changes Revert Changes

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	1	Python
2	2	SQL
3	3	PHP

Filter Filter

< < 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	1	Jane	jane@tsugi.org
2	2	Ed	ed@tsugi.org
3	3	Sue	sue@tsugi.org

Filter Filter Filter

< < 1 - 3 of 3 > >>

Go to: 1

UTF-8

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

id	title
Filter	Filter
1	Python
2	SQL
3	PHP

```

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

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

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

```

DB Browser for SQLite - /Users/csev/Desktop/si502_database

New Database Open Database Write Changes Revert Changes

Database Structure Browse Data Edit Pragmas Execute SQL

Table: Member

New Record Delete Record

user_id	course_id	role
1	1	1
2	2	0
3	3	0
4	1	0
5	2	1
6	2	1
7	3	0

< < 1 - 7 of 7 > >

Go to: 1

UTF-8

The screenshot shows the DB Browser for SQLite application interface. The title bar indicates the database is located at /Users/csev/Desktop/si502_database. The main window has tabs for Database Structure, Browse Data, Edit Pragmas, and Execute SQL, with Browse Data selected. A sub-header shows the current table is Member. Below this are buttons for New Record and Delete Record. The main area displays a grid of data with three columns: user_id, course_id, and role. The data consists of 7 rows, each with values: (1, 1, 1), (2, 2, 0), (3, 3, 0), (4, 1, 0), (5, 2, 1), (6, 2, 1), and (7, 3, 0). At the bottom, there are navigation buttons (<, <<, >, >>) and a Go to: input field set to 1. The status bar at the bottom right shows UTF-8 encoding.

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



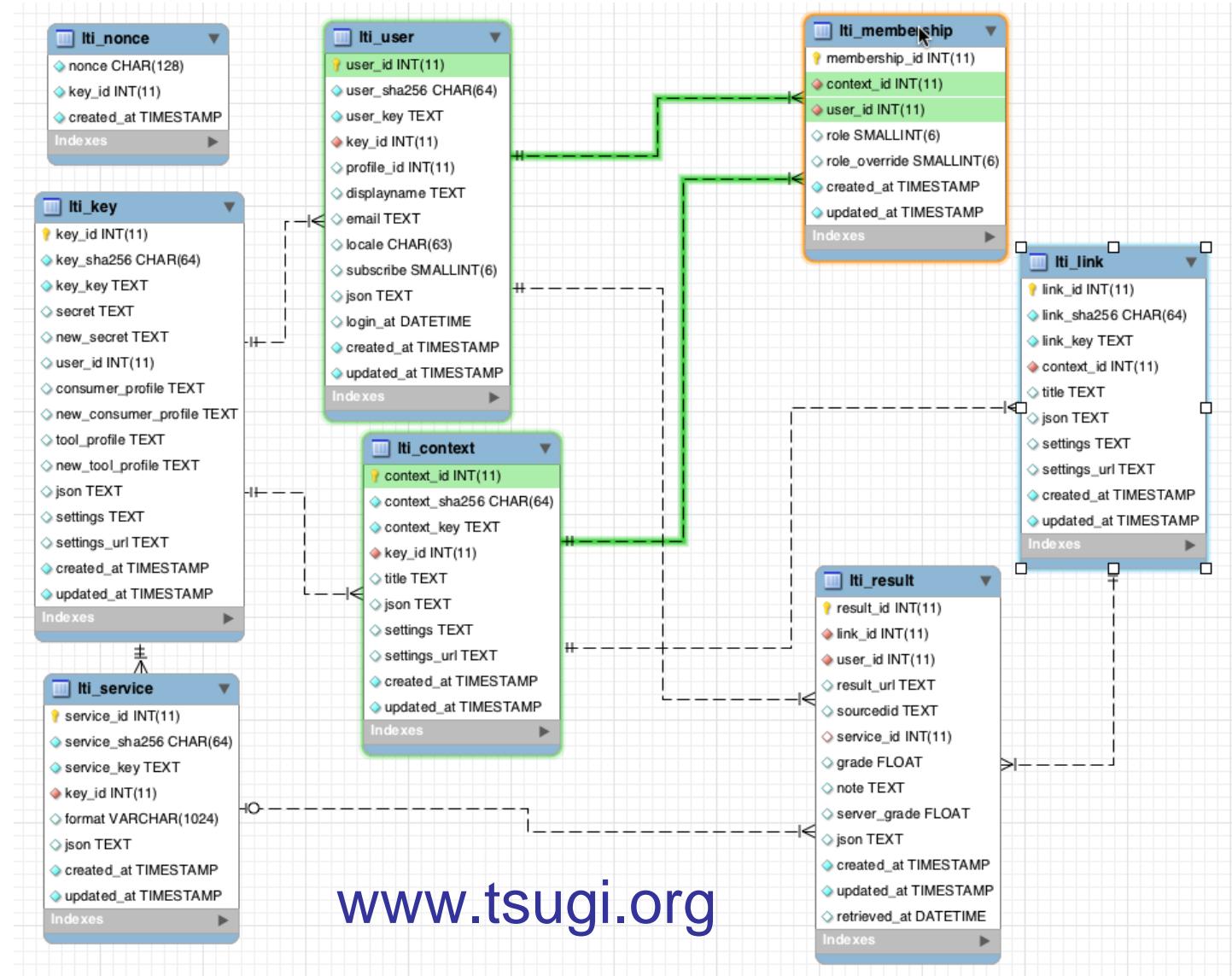
user_id	course_id	role
Filter	Filter	Filter
1	1	1
2	1	0
3	1	0
1	2	0
2	2	1
2	3	1
3	3	0



id	title
Filter	Filter
1	Python
2	SQL
3	PHP

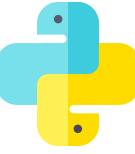
```
SELECT User.name, Member.role, Course.title
FROM User JOIN Member JOIN Course
ON Member.user_id = User.id AND
Member.course_id = Course.id
ORDER BY Course.title, Member.role DESC, User.name
```

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



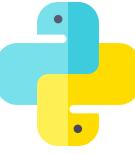
www.tsugi.org





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