Spring 2024 CSE 380

1-23-2024

Office Hours Update

- Abigail's office hours
  - Mondays: 10:30-12:00pm (3203 EB)
  - Wednesdays: 10:30-12:00pm (3203 EB)
  - Fridays: 10:30-11:30am (always Zoom)
  - Abigail's Zoom link is on Piazza
- Professor Mariani's office hours (always 3145 EB)
  - Mondays: 1:30pm-3:00pm
  - Wednesdays: 2:00pm-3:30pm

Quiz Reminder

- We have our first quiz on Thursday (January 25rd)
- Please remember, no phones, cheatsheets, computers, etc. of any kind
- Make sure you're studying and practicing
- Quiz 2 might be moved, will keep you updated

Project Updates

- Project 1 Released on Feb 16, Due March 6
- Project 2 Released March 8, Due March 20
- Project 3 Released March 22, Due April 3
- Project 4 Released April 5, Due April 17
- All projects will be done in Flask (Python)
  - I will do a small Flask review, and a review of SQLite in Python before Project 1
  - If you want to get a head start, can look up some tutorials on Flask

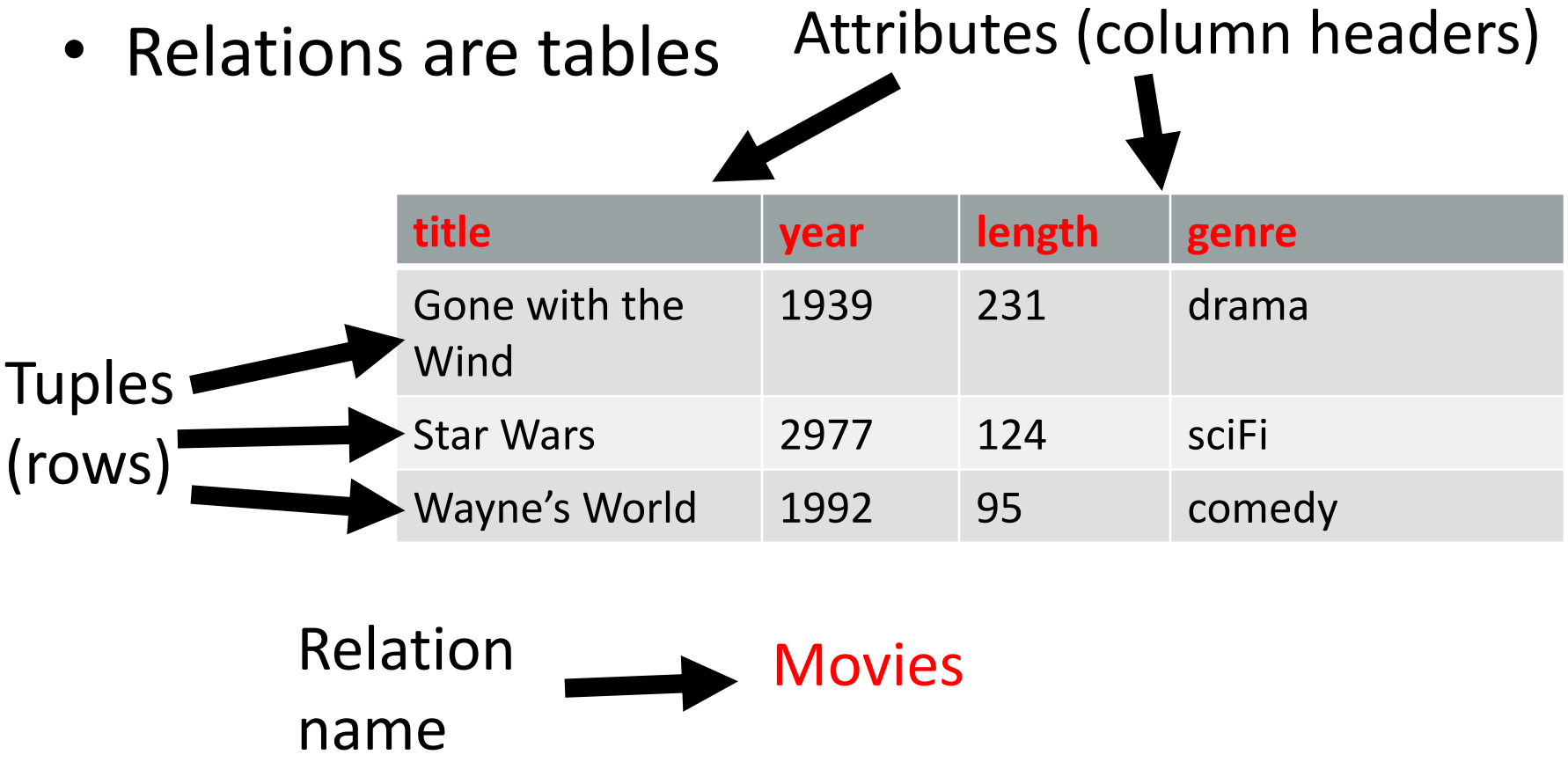
Review

Introduction to SQL

- Data Models – Components
  - Mathematical representation of the data
    - Relational Models (CSV) = tables
    - Semi-structured Models (XML, JSON) = trees/graphs
  - Operations on data (what is allowed, i.e. comparisons, equality, math)
  - Constraints (what types of data are allowed, and where)

Introduction to SQL

- Relations are tables



title	year	length	genre
Gone with the Wind	1939	231	drama
Star Wars	2977	124	sciFi
Wayne's World	1992	95	comedy

Introduction to SQL

- Schemas (sound familiar?)
  - Relation schema = relation name and attribute list
    - Optionally: types of attributes
    - Example: Movies (title, year, length genre) or Movies (title TEXT, year INTEGER, length INTEGER, genre TEXT)
  - Database = collection of relations
  - Database schema = set of all relation schemas in the database

Introduction to SQL

- Why Relations (tables)?
  - Very simple model
  - Often (but not always) matches how we think about data
  - Abstract model that underlies SQL, the most important database language today

Basic SQL Querying

- Example: Create Table

```
CREATE TABLE Movies (
        title TEXT,
        year INTEGER,
        length INTEGER,
        genre TEXT
);
```

Basic SQL Querying

- SQL Values
  - Integers and real values (floats) are represented as you would expect
  - Strings are as well, except they require single quotes
    - Two single quotes = one real quote
      - 'Joe''s Bar' = Joe's Bar
  - Any value can be NULL
    - For now, we will learn later how this can be prevented

Basic SQL Querying

- Insertion
  - To insert a single record:

    INSERT INTO <relation>

    VALUES (<list of values>);
  - Example: add Star Wars to our Movies relation

    INSERT INTO Movies

    VALUES ('Star Wars', 1977, 124, 'sciFi');

Basic SQL Querying

- Select
  - Using our Movies relation, what movies have been released?

```
SELECT title
FROM Movies;
```

Basic SQL Querying

- Select and Order By
  - Using our Movies relation, what movies came out, ordered from oldest to newest

```
SELECT title
FROM Movies
ORDER BY year;
```

Column we want to select

Table we want to select from

Column to order results by

Basic SQL Querying

- Select and Order By
  - Using our Movies relation, what movies were released, ordered by release date

```
SELECT title
FROM Movies
ORDER BY year, length;
```

If same year, break ties by 'length' column

Basic SQL Querying

- Select Multiple Attributes
  - Using our Movies relation, what are the names of the movies and the year they were released

```
SELECT title, year
FROM Movies
ORDER BY year;
```

Basic SQL Querying

- Select *
  - What are all of the attributions of the relation 'players'?

```
SELECT *
FROM Movies
ORDER BY year;
```

SQLite Demo

- Download sqlite
- Type:
  - sqlite3 test.db

If you're starting a new DB, you can name this whatever you want

```
host-262820:~ cse498$ sqlite3 test.db
SQLite version 3.30.0 2019-10-04 15:03:17
Enter ".help" for usage hints.
sqlite>
```

SQLite Demo

- Create the table

```
host-262820:~ cse498$ sqlite3 test.db
SQLite version 3.30.0 2019-10-04 15:03:17
Enter ".help" for usage hints.
sqlite> CREATE TABLE Movies(title TEXT, year INTEGER, length INTEGER, genre TEXT);
sqlite>
```

- Insert values into the table

```
host-262820:~ cse498$ sqlite3 test.db
SQLite version 3.30.0 2019-10-04 15:03:17
Enter ".help" for usage hints.
sqlite> CREATE TABLE Movies(title TEXT, year INTEGER, length INTEGER, genre TEXT);
sqlite> INSERT INTO Movies VALUES ('Gone With the Wind', 1939, 231, 'drama');
sqlite> INSERT INTO Movies VALUES ('Star Wars', 1977, 123, 'sciFi');
sqlite> INSERT INTO Movies VALUES ('Wayne''s World', 1992, 95, 'comedy');
sqlite>
```

SQLite Demo

- Let's see what's in the DB

```
sqlite> SELECT * FROM Movies;
Gone With the Wind|1939|231|drama
Star Wars|1977|123|sciFi
Wayne's World|1992|95|comedy
sqlite>
```

- That's Ugly

```
sqlite> .headers on
sqlite> .mode columns
```

SQLite Demo

- One more SELECT

```
sqlite> .headers on
sqlite> .mode columns
sqlite> SELECT * FROM Movies;
title                year        length      genre
-------------------  ----------  ----------  ----------
Gone With the Wind   1939        231         drama
Star Wars            1977        123         sciFi
Wayne's World        1992        95          comedy
sqlite>
```

SQLite Demo NEW

- Saving the SQL

```
sqlite> .dump
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
CREATE TABLE Movies(title TEXT, year INTEGER, length INTEGER, genre TEXT);
INSERT INTO Movies VALUES('Gone With the Wind',1939,231,'drama');
INSERT INTO Movies VALUES('Star Wars',1977,123,'sciFi');
INSERT INTO Movies VALUES('Wayne''s World',1992,95,'comedy');
COMMIT;
sqlite> .output test.sql
```

New Material

WHERE Clauses

- SELECT ... FROM ... WHERE ...;
- The where clause stipulates a test (predicate) that each row must pass to be returned in the select statement.

SELECT title, genre FROM Movies WHERE length < 120;

WHERE Clauses

SELECT title FROM Movies WHERE length < 120;

```
sqlite> SELECT title FROM Movies WHERE length < 120;
title
-------------
Wayne's World
Back to the F
```

WHERE Clauses

SELECT title, genre FROM Movies WHERE length < 120;

```
sqlite> SELECT title, genre FROM Movies WHERE length < 120;
title          genre
-------------  ----------
Wayne's World  comedy
Back to the F  adventure
```

SQL Operators

- Arithmetic Operators:
  - + - * / %
- Comparison Operators:
  - = != <> > < >= <=
- Logical Operators
  - ALL AND ANY BETWEEN EXISTS IN LIKE NOT OR IS

Between

SELECT title, genre FROM Movies WHERE length BETWEEN 90 AND 130;

note: between is inclusive of the bounds

```
sqlite> SELECT title, genre FROM Movies WHERE length BETWEEN 90 AND 130;
title       genre
----------  ----------
Star Wars   sciFi
Wayne's Wo  comedy
Back to th  adventure
```

Between

SELECT title, genre FROM Movies WHERE length NOT BETWEEN 90 AND 130;

```
sqlite> SELECT title, genre FROM Movies WHERE length NOT BETWEEN 90 AND 130;
title               genre
------------------  ----------
Gone With the Wind  drama
The Shawshank Rede  drama
The Godfather       drama
The Dark Knight     action
The Matrix          sciFi
```

Between

- Why can't we do?

SELECT title, genre FROM Movies WHERE length >= 90 AND length <= 130;

IS and IS NOT

SELECT title, genre FROM Movies WHERE length IS NULL;

'IS' operator is only to be used to test for NULL values

```
sqlite> SELECT title, genre FROM Movies WHERE length is NULL;
title              genre
-----------------  ----------
Good Will Hunting  drama
```

AND and OR

SELECT title, genre FROM Movies WHERE length BETWEEN 90 AND 130 AND genre = ‘comedy’;

```
sqlite> SELECT title, genre FROM Movies WHERE length BETWEEn 90 AND 130 AND genre = 'comedy';
title          genre
-------------  ----------
Wayne's World  comedy
```

An Aside: COUNT()

SELECT COUNT(*) FROM Movies WHERE length BETWEEN 90 AND 130 AND genre = ‘comedy’;

```
sqlite> SELECT COUNT(*) FROM Movies WHERE length BETWEEn 90 AND 130 AND genre = 'comedy';
COUNT(*)
----------
1
```

SELECT COUNT(*) FROM Movies WHERE genre = ‘drama’;

```
sqlite> SELECT COUNT(*) FROM Movies WHERE genre = 'drama';
COUNT(*)
----------
4
```

LIKE

SELECT title FROM Movies WHERE title LIKE 'The%';

```
sqlite> SELECT title FROM Movies WHERE title LIKE 'The%';
title
----------------------
The Shawshank Redemption
The Godfather
The Dark Knight
The Matrix
```

LIKE

SELECT title FROM Movies WHERE title LIKE 'The%';

- % is a wildcard that can match 0 or more characters
  - Probably should have been '*', but already being used
- _ (underscore) is a wildcard representing a single character
  - Should have been '.' but again, already taken

LIKE

SELECT title FROM Movies WHERE title LIKE 'The%' AND genre = 'drama';

```
sqlite> SELECT title FROM Movies WHERE title LIKE 'The%' AND genre = 'drama';
title
------------------------
The Shawshank Redemption
The Godfather
```

IN

SELECT title, genre FROM Movies WHERE length IN(90,130);

- IN tests for inclusion in a static, parenthesized list

```
sqlite> SELECT title, genre FROM Movies WHERE length IN (90, 130);
sqlite>
```

Question

- What should happen for a query where nothing matches the WHERE clause, but one of the rows has a NULL in the WHERE column
  - It should return 0 rows
  - It should raise a syntax error
  - It should return 1 row (the row with the NULL)
  - It should raise a "no rows" error

Comparing NULLS to Values

- Comparing any value (including NULL itself) with NULL yields UNKNOWN.
- A tuple is in a query answer if and only if the WHERE clause is TRUE (not FALSE or UNKNOWN).

SQL Column Constraints

- Unique:
  - Used to specify a column whose values will always be unique for every row.
  - CREATE TABLE students (msu_id TEXT UNIQUE,
    pid INTEGER UNIQUE,
    first_name TEXT);

  The DBMS will raise an error if the constraint is violated.

Should NULLS be Allowed in UNIQUE columns?

- No
- Only one
- As many as desired
- Depends on the data type

NOT NULL

- Used to designate columns that may not be NULL

  CREATE TABLE students (msu_id TEXT UNIQUE NOT NULL, section INTEGER NOT NULL);

- Raises error if a NULL value is inserted.

Primary key

- Each table can have at most one PRIMARY KEY.
  - It is implicitly NOT NULL and UNIQUE
  - It is used to specify a specific column in the table

```
CREATE TABLE students (msu_id TEXT PRIMARY KEY,
                       grade REAL);
```

Composite PRIMATE KEY

- You can create a primary key out of multiple columns if the combination is unique

```
CREATE TABLE students (first_name TEXT,
                       last_name TEXT,
                       PRIMARY KEY (first_name, last_name));
```

Why Use PRIMARY KEYS?

- They indicate values that can be used to easily connect two relations together (joins).
- They indicate to the database good attributes to organize the rows by.
  - This allows for more optimized queries.
- Best practice is for every table to have a primary key
- Will cover very in depth, later in the course

Which Column(s) Would Make the Best Primary Key for a Students Table?

- Social Security Number
- First and Last Name
- Student Number
- MSU Net ID

Qualified Table Names

- The qualified name of a table is <name of database>.<name of relation> (e.g. main.students)
- The qualified name of a column is <qualified relation>.<name of column> (e.g. main.students.msu_id)
- Sometimes it is simpler to preface column names with their relation (we'll see why shortly)

SELECT name FROM students;

- Is exactly the same as:

SELECT students.name FROM students;

Qualified Table Names

- You can also use the pattern anywhere you use a column name:

```
SELECT * FROM students WHERE students.name = 'James';
```

SQL Alias (AS)

- Another helpful piece of syntax is aliases if you want to refer to a column by a different name.

```
SELECT msu_net_id AS id FROM students
                    WHERE id = 'mariani4';
```

- You can also use it on tables:

```
SELECT cse380.msu_net_id FROM students AS cse380;
```

SELECT cse380.msu_net_id FROM students cse380;

  - Implicit Alias, not recommended

Joins

- Often, queries need to combine (join) rows from multiple tables.
- Joins specify which rows get merged with rows from a second table.
- As there are many possible methods to match rows together, there are many types of joins.
- You will need to know (memorize) the different joins.
- Annoying, but very useful knowledge.

Joins

professors

id	name
esfahanian	Abdol
mariani	James
dyksen	Wayne
NULL	demo-prof

```
CREATE TABLE professors (id TEXT, name TEXT);
INSERT INTO professors VALUES('esfahanian','Abdol');
INSERT INTO professors VALUES('mariani','James');
INSERT INTO professors VALUES('dyksen','Wayne');
INSERT INTO professors VALUES(NULL, 'demo-student');
```

```
CREATE TABLE associates (responder_id TEXT, associate_id TEXT);
INSERT INTO associates VALUES('mariani','dyksen');
INSERT INTO associates VALUES('mariani','esfahanian');
INSERT INTO associates VALUES('dyksen','mariani');
INSERT INTO associates VALUES('esfahanian','dyksen');
```

associates

responder_id	associate_id
mariani	dyksen
mariani	esfahanian
dyksen	mariani
esfahanian	dyksen

Cross Join

- Combine every row in one table with every row in the other

```
SELECT * FROM professors CROSS JOIN associates;
```

- Implicit Cross Join (equivalent)

```
SELECT * FROM students, associates;
```

- Also called the Cartesian Product (as it results in every possible combination)
- Not often used, because you rarely need all possible combinations.

Inner Join

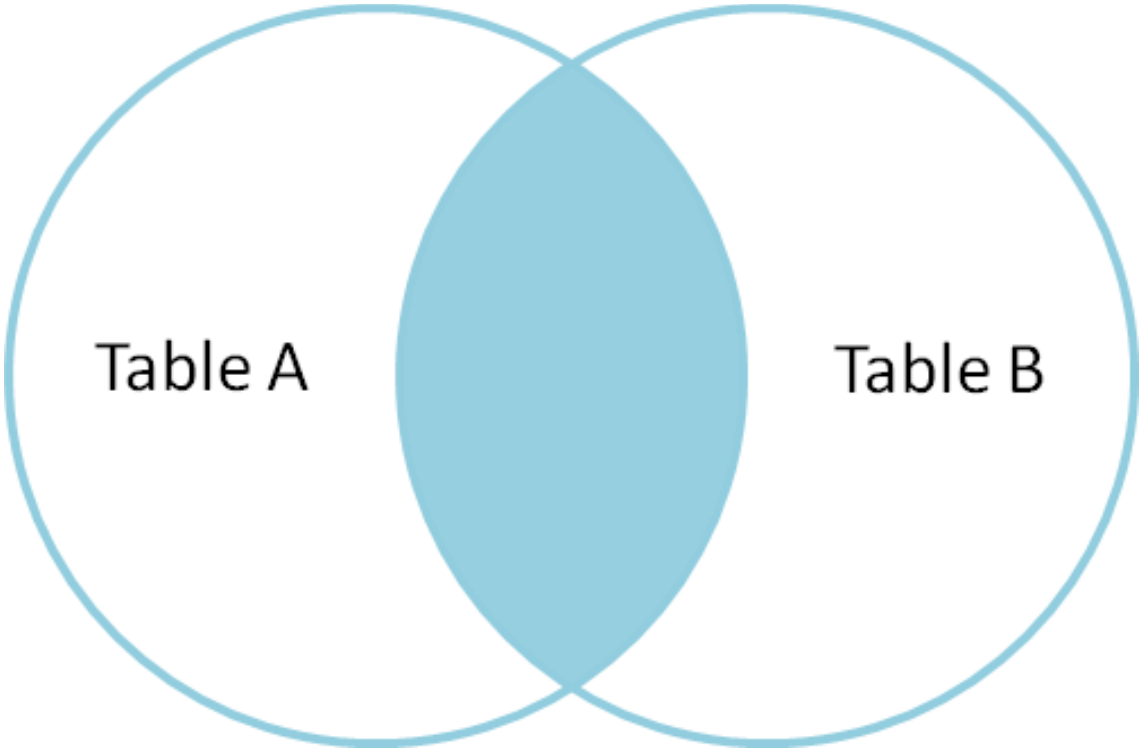
- INNER JOIN is like the CROSS JOIN, except only rows that match a predicate are allowed. Frequently that predicate includes primary keys to match associated data

```
SELECT * FROM professors INNER JOIN
associates
    ON id = responder_id;
```

- You can equivalently do an implicit inner join and use WHERE to filter rows:

```
SELECT * FROM students, associates
    WHERE id = responder_id;
```

Inner Join



Retuned rows are those that match both tables (shaded).

Inner Join

- SELECT * FROM professors INNER JOIN associates ON id = responder_id;

```
sqlite> SELECT * FROM professors INNER JOIN associates ON id = responder_id;
id          name        responder_id  associate_id
----------  ----------  ------------  ------------
esfahanian  Abdol       esfahanian    dyksen
mariani     James       mariani       dyksen
mariani     James       mariani       esfahanian
dyksen      Wayne       dyksen        mariani
```

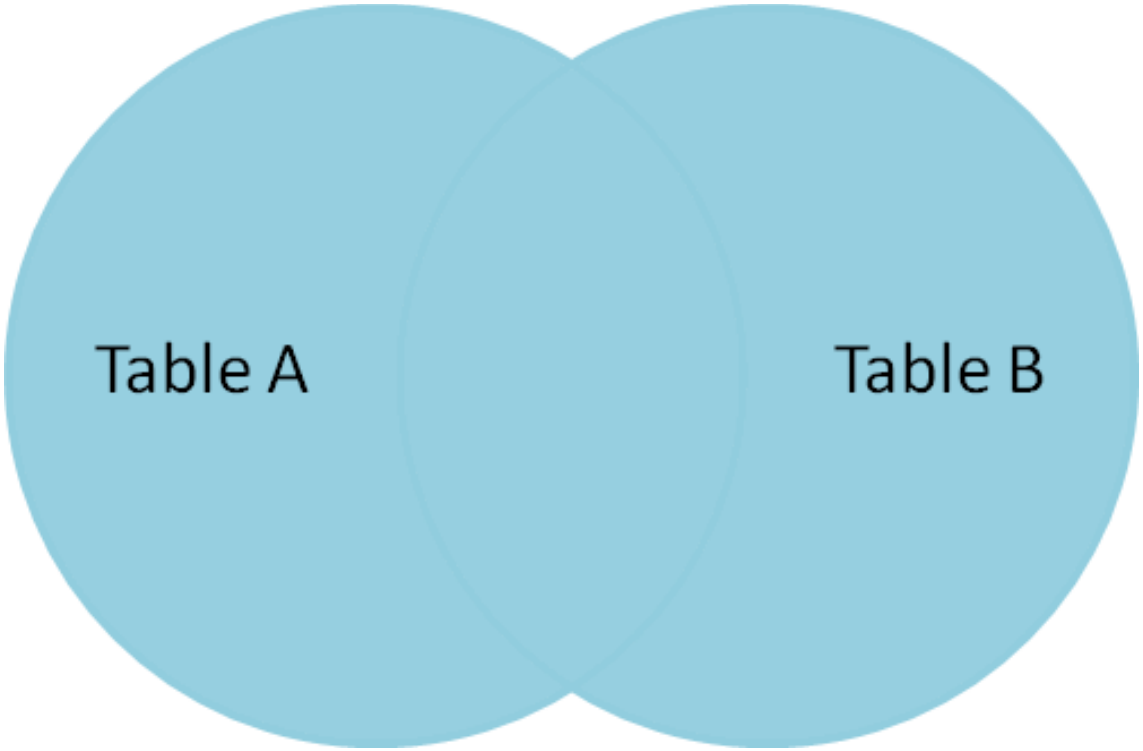
Full Outer Join

- Like an INNER JOIN, the FULL OUTER JOIN includes all rows that match a predicate. In addition however, rows that don't have a match are included with NULL values where their corresponding data would be

```
SELECT * FROM professors FULL OUTER JOIN associates
                    ON id = responder_id;
```

- SQLite doesn't implement FULL OUTER JOIN,): . But I'll show you how to do it anyways in a later lecture.

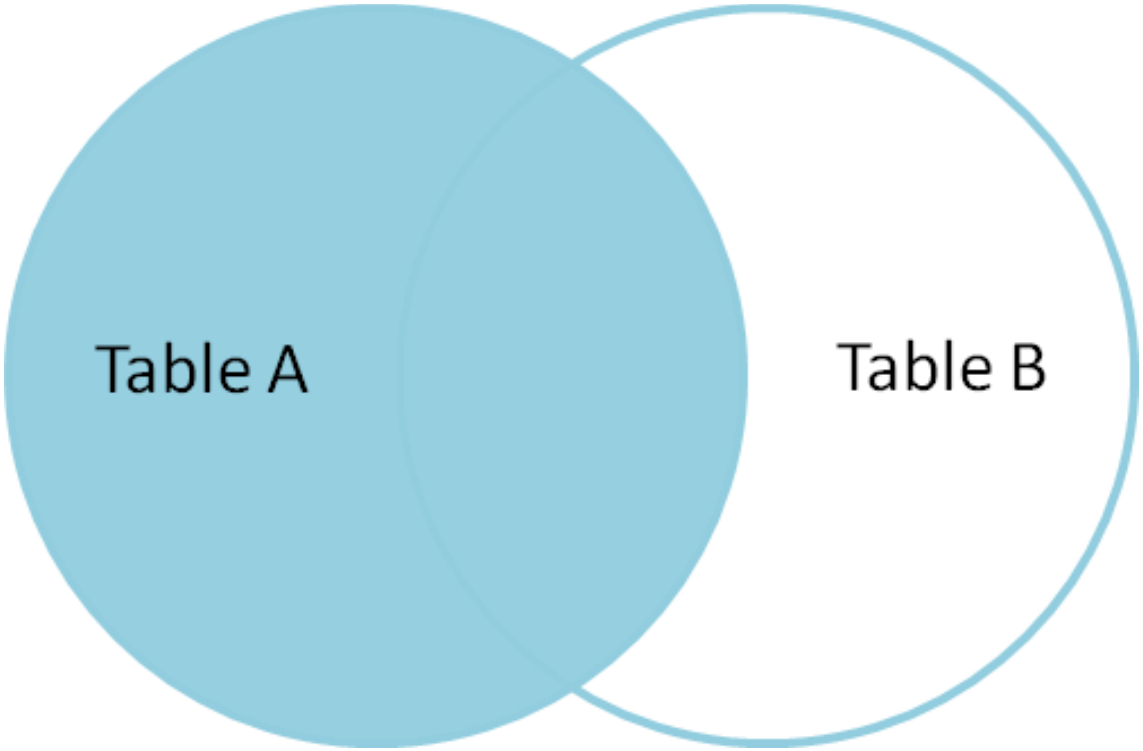
Full Outer Join



Left Outer Join

- LEFT OUTER JOIN includes all the rows to the left of the JOIN keyword, where there is a match, it add the matching column data to rows. If there isn't a match NULL values are used instead.
- SELECT * FROM professors LEFT OUTER JOIN associates
  ON id = responder_id;
- Note: order matters!!!
- SELECT * FROM associates LEFT OUTER JOIN professors
  ON id = responder_id;

Left Outer Join



Left Outer Join

SELECT * FROM professors LEFT OUTER JOIN associates ON id = responder_id;

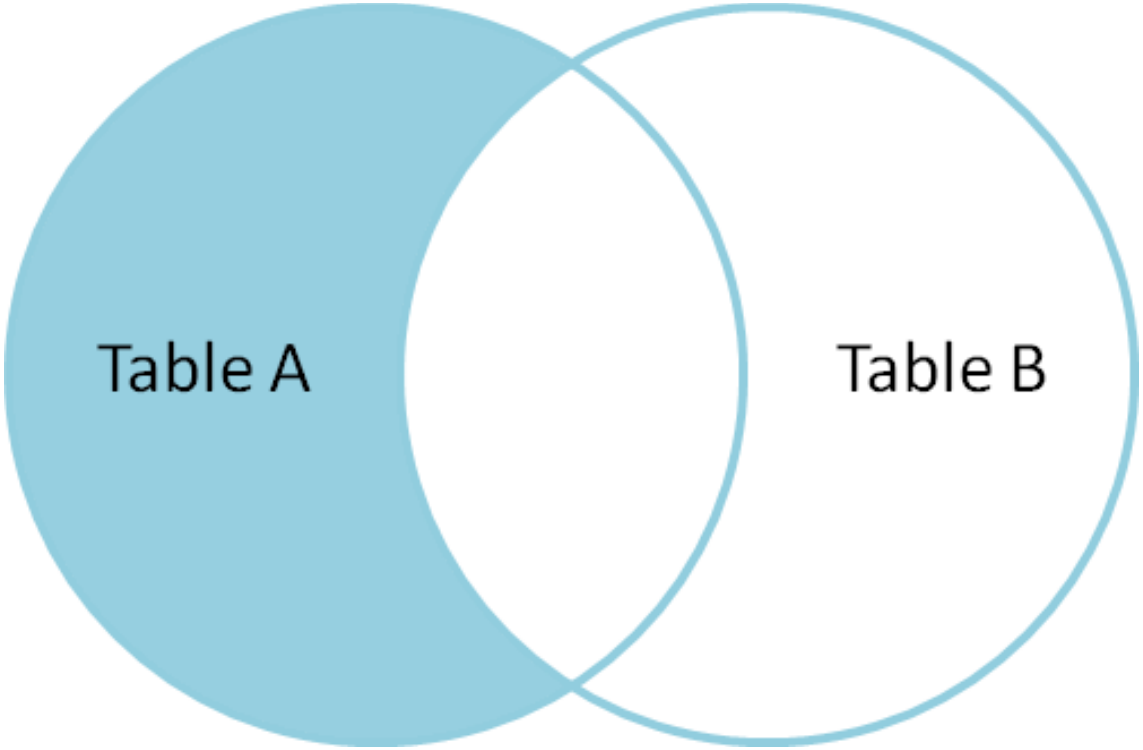
```
sqlite> SELECT * FROM professors LEFT OUTER JOIN associates ON id=responder_id;
id          name        responder_id  associate_id
----------  ----------  ------------  ------------
esfahanian  Abdol       esfahanian    dyksen
mariani     James       mariani       dyksen
mariani     James       mariani       esfahanian
dyksen      Wayne       dyksen        mariani
            demo-stude
```

Left Outer Join Without Matches

- If we want to find the rows in the left table that don't have matches, we can do a LEFT OUTER JOIN and filter out the rows with matches.

```
SELECT * FROM professors LEFT OUTER JOIN associates
        ON id = responder_id WHERE responder_id IS NULL;
```

Left Outer Join Without Matches

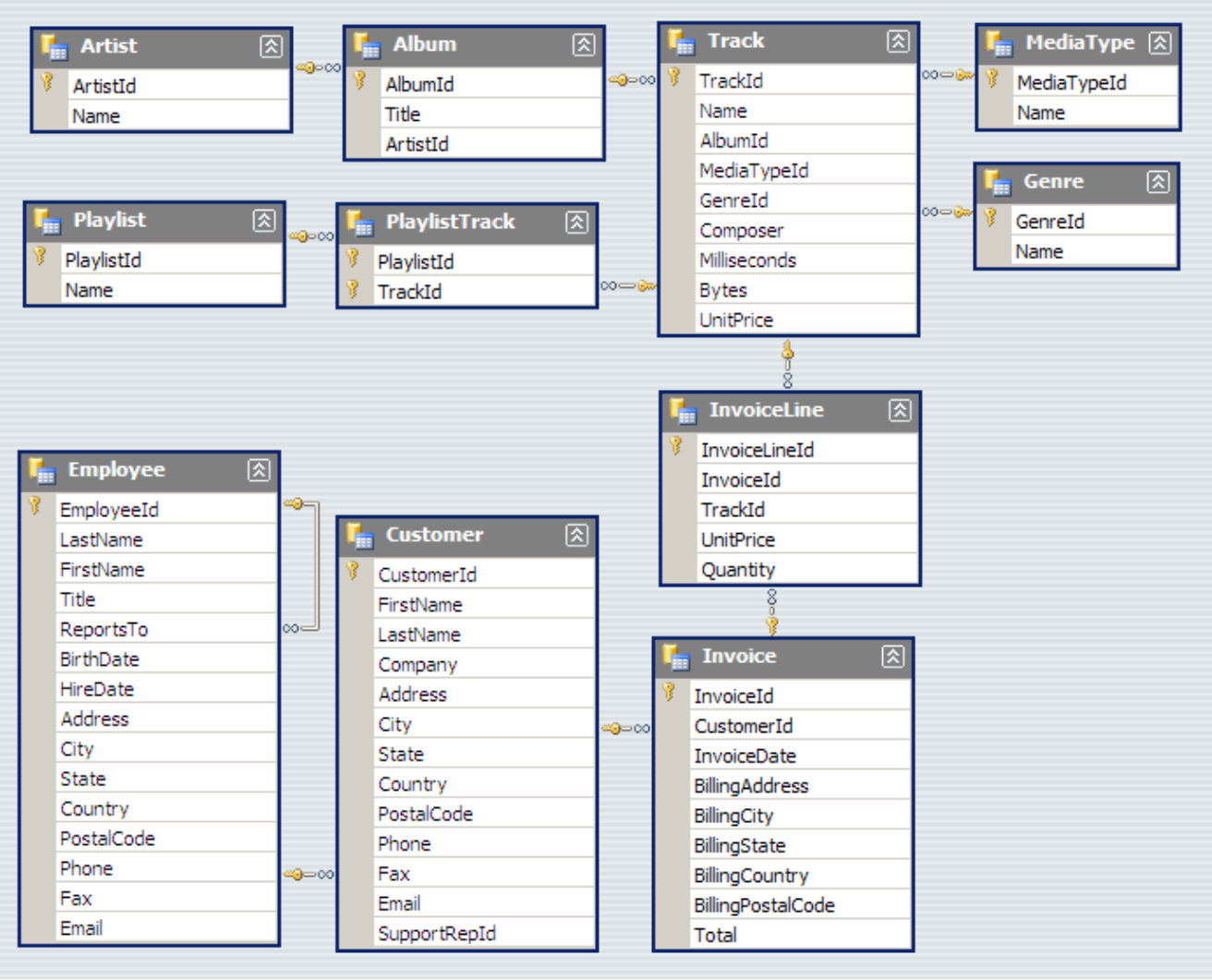


What rows should a left inner join return?

- Rows that match the predicate
- Row that match plus the unmatched rows on the left
- Rows that are unmatched and the matches on the left
- ???

Practice

Chinook Dataset

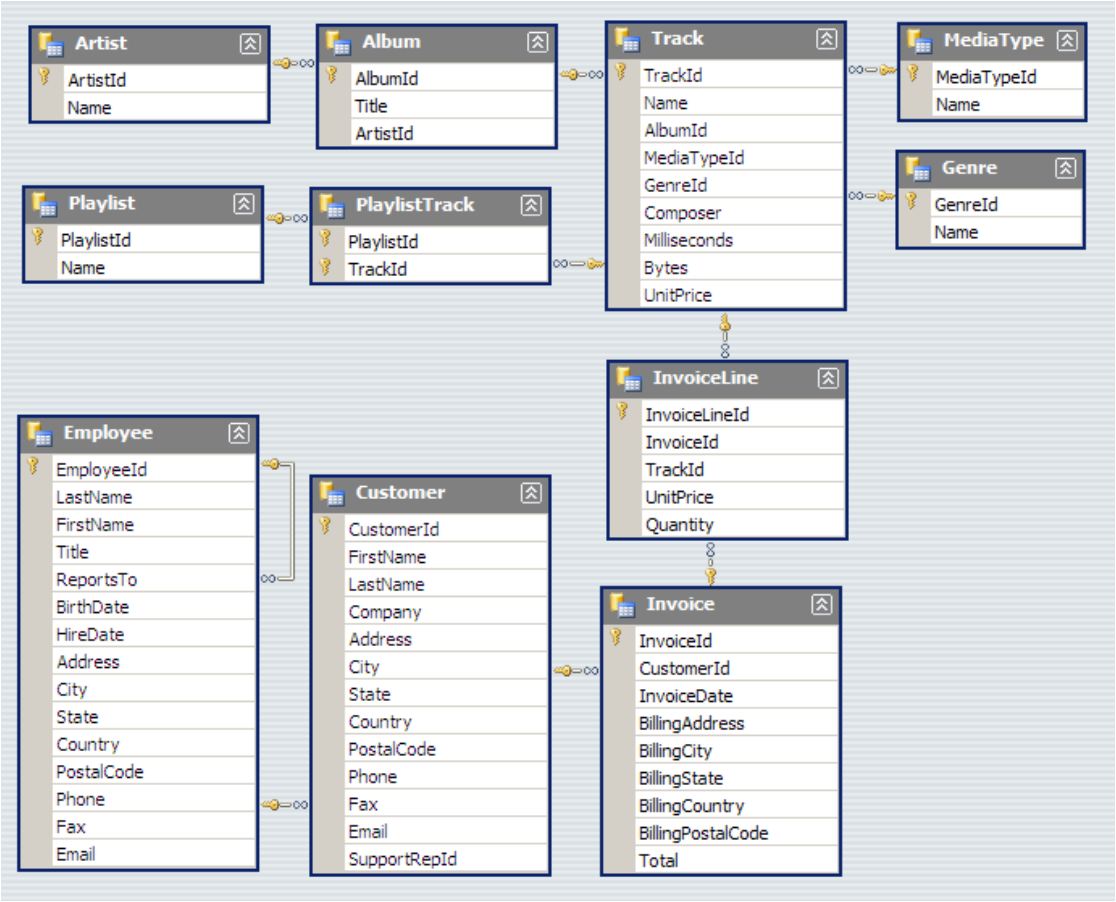


Test Databases

- Chinook Music Database
  - Used for examples
- Create chinook DB instance - .sql file on D2L

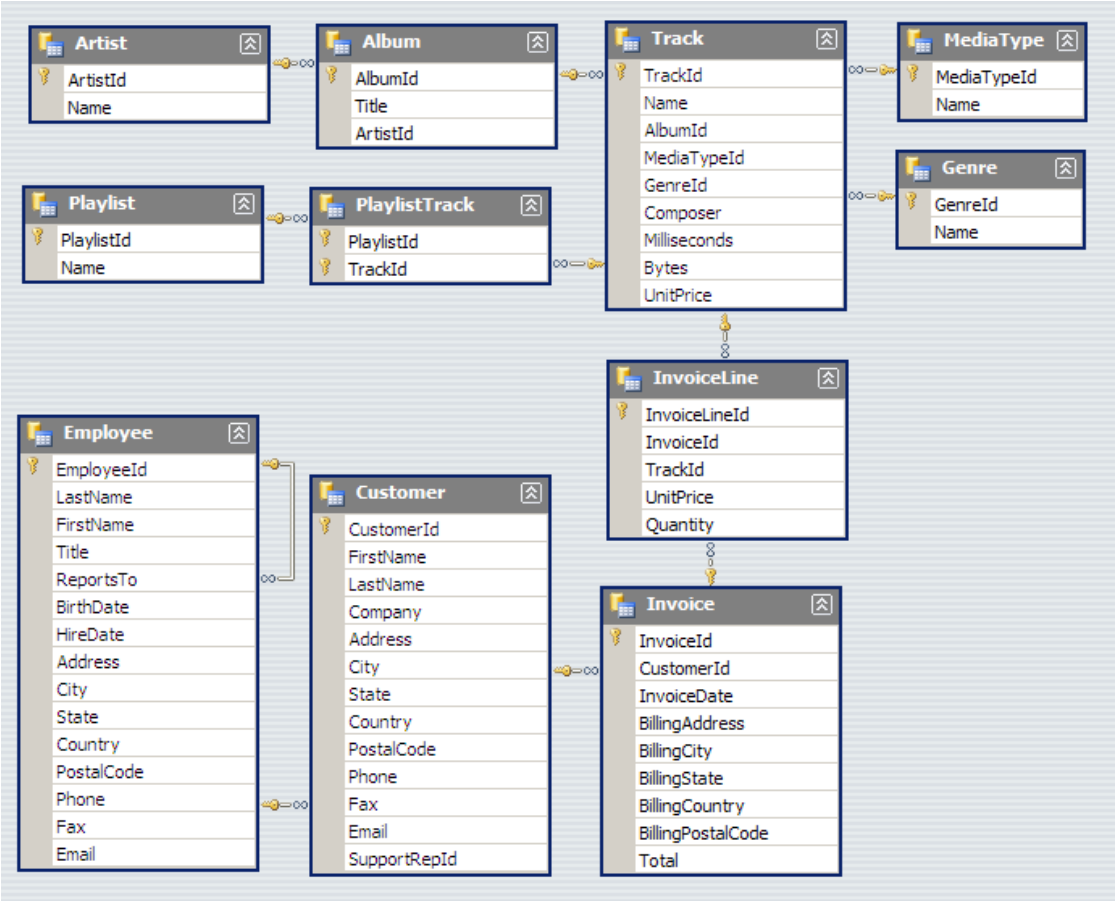
sqlite3 chinook.db < Chinook_Sqlite.sql

Practice



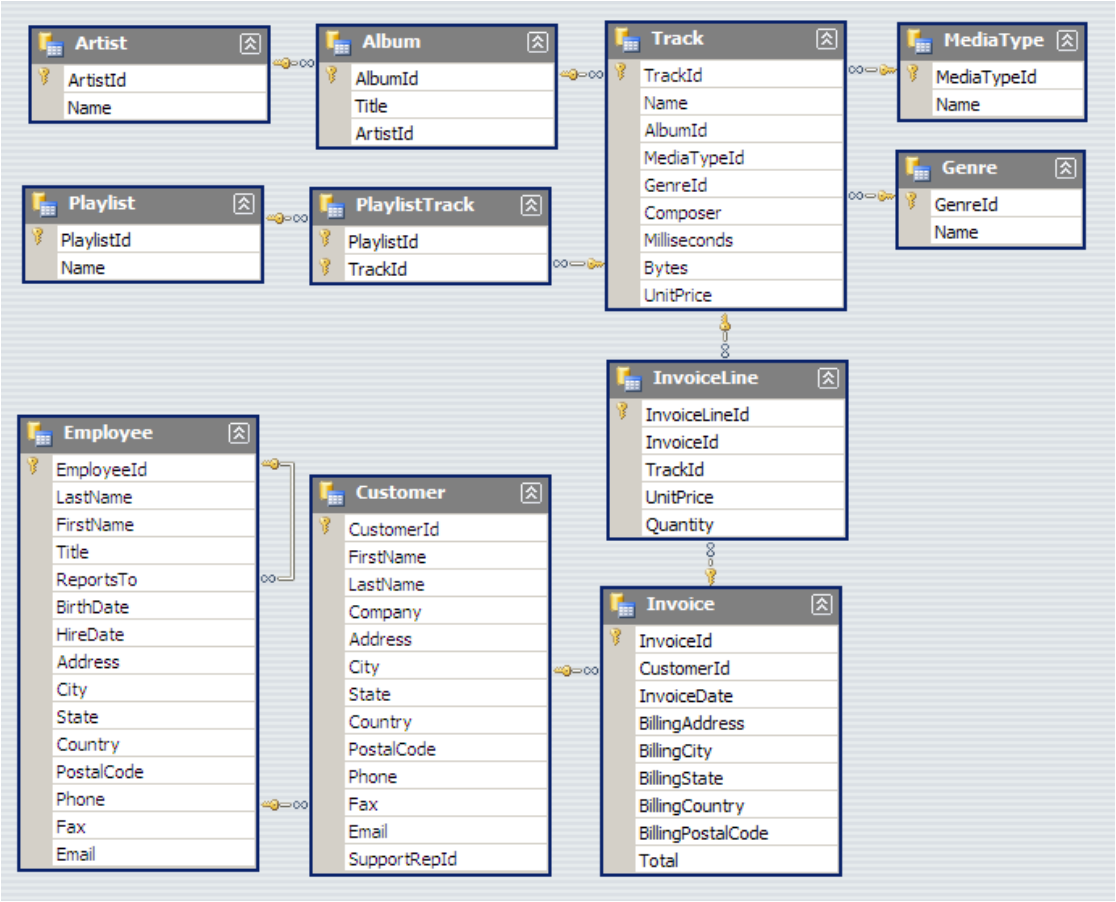
Provide a query showing customers who are not in the US

Practice



Answer:

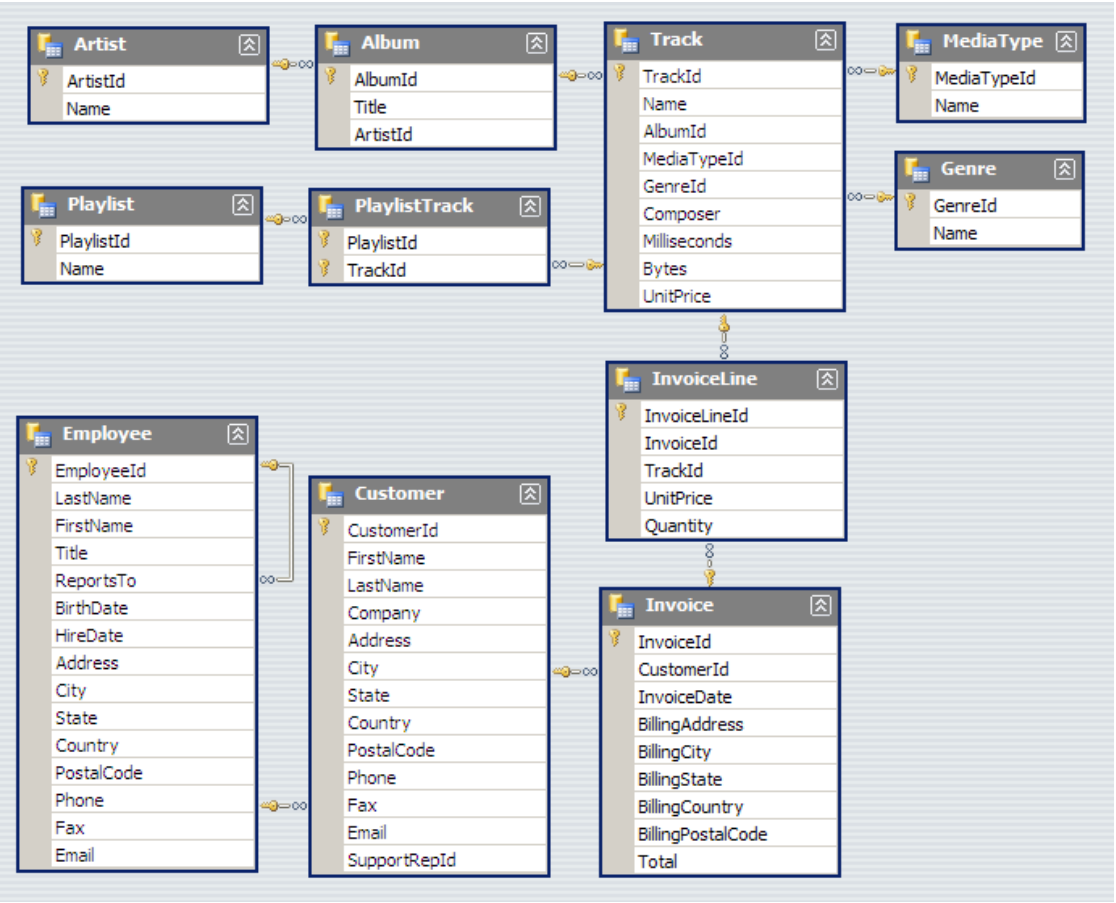
```
SELECT FirstName, LastName, Country
FROM customer
WHERE NOT country = 'USA';
```



Practice

Provide a query showing invoices of customers who are from Brazil

Practice



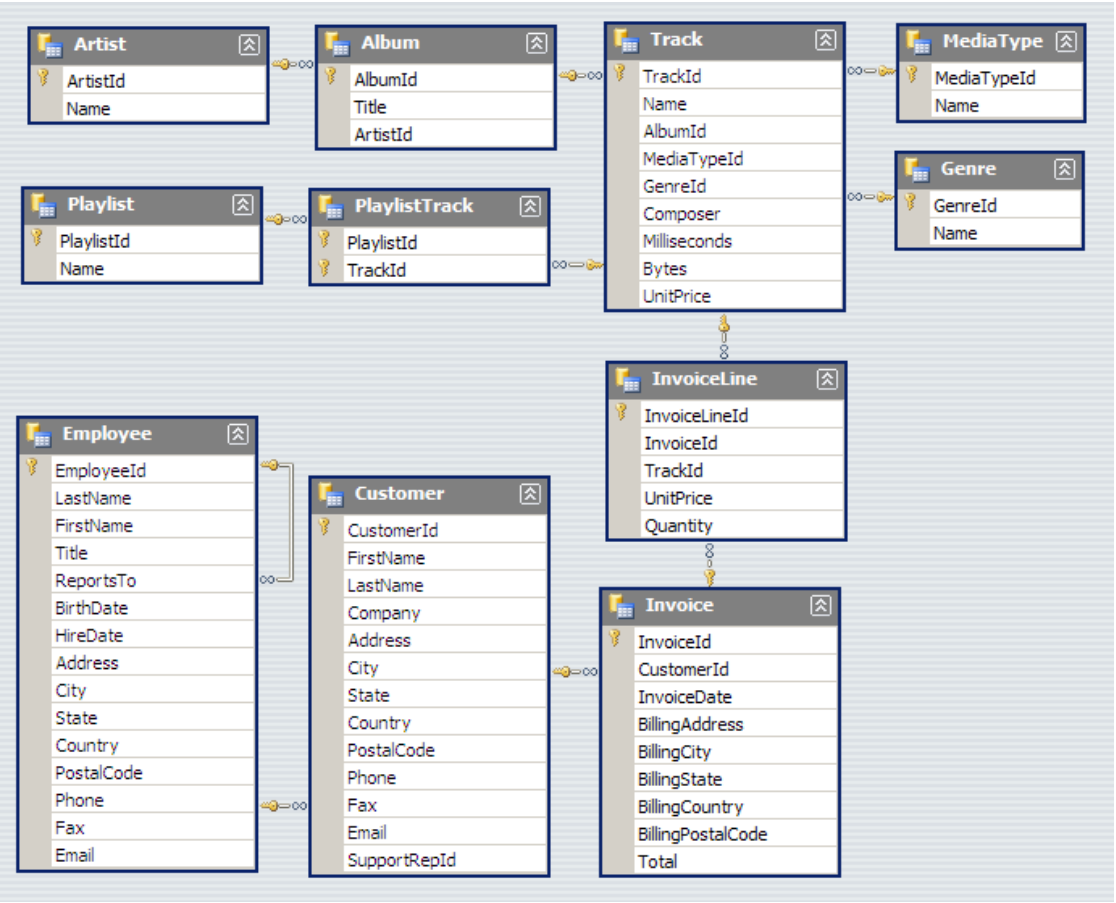
Answer:

SELECT c.firstname, c.lastname, i.invoiceid, i.invoicedate, i.billingcountry

FROM customer AS c INNER JOIN invoice AS i ON c.customerid = i.customerid

WHERE c.country = 'Brazil';

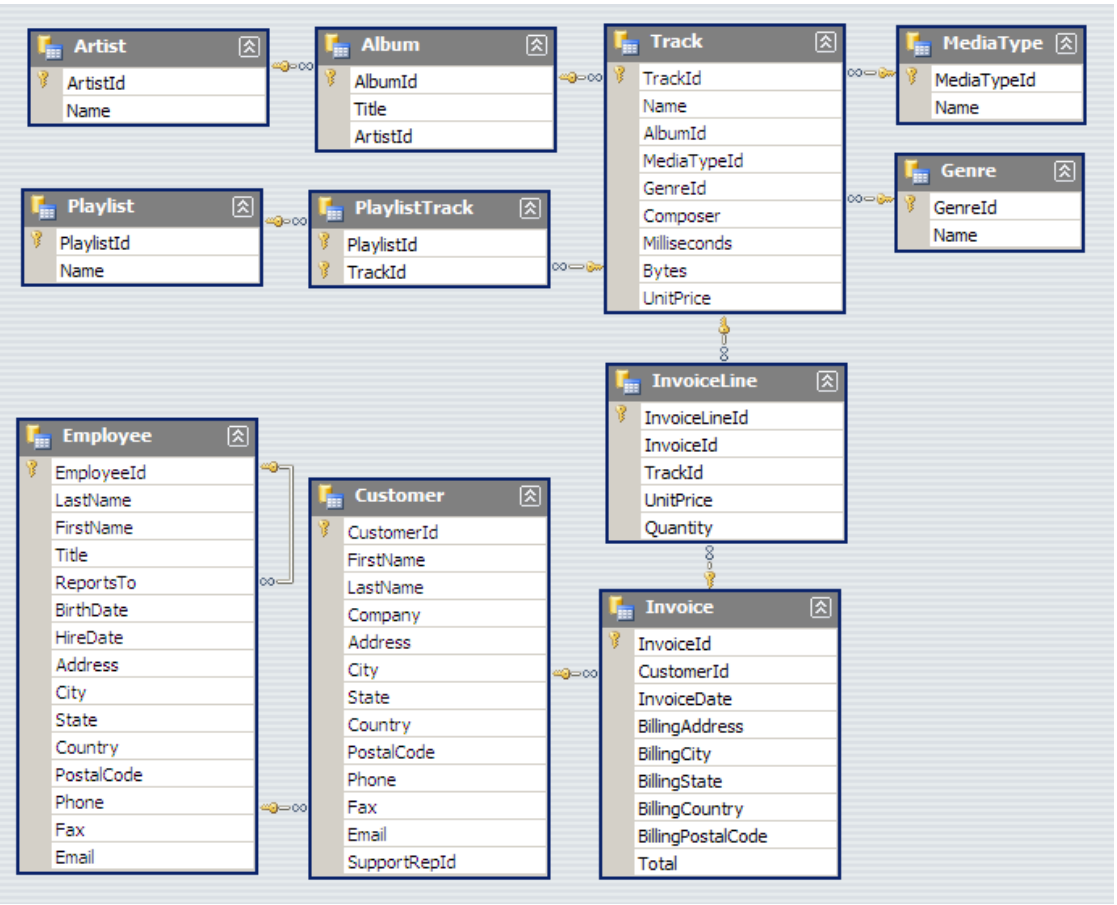
Practice



Answer:

```
SELECT c.firstname, c.lastname,
i.invoiceid, i.invoicedate, i.billingcountry
FROM customer AS c INNER JOIN invoice
AS i ON c.customerid = i.customerid
WHERE c.country = 'Brazil';
```

Practice



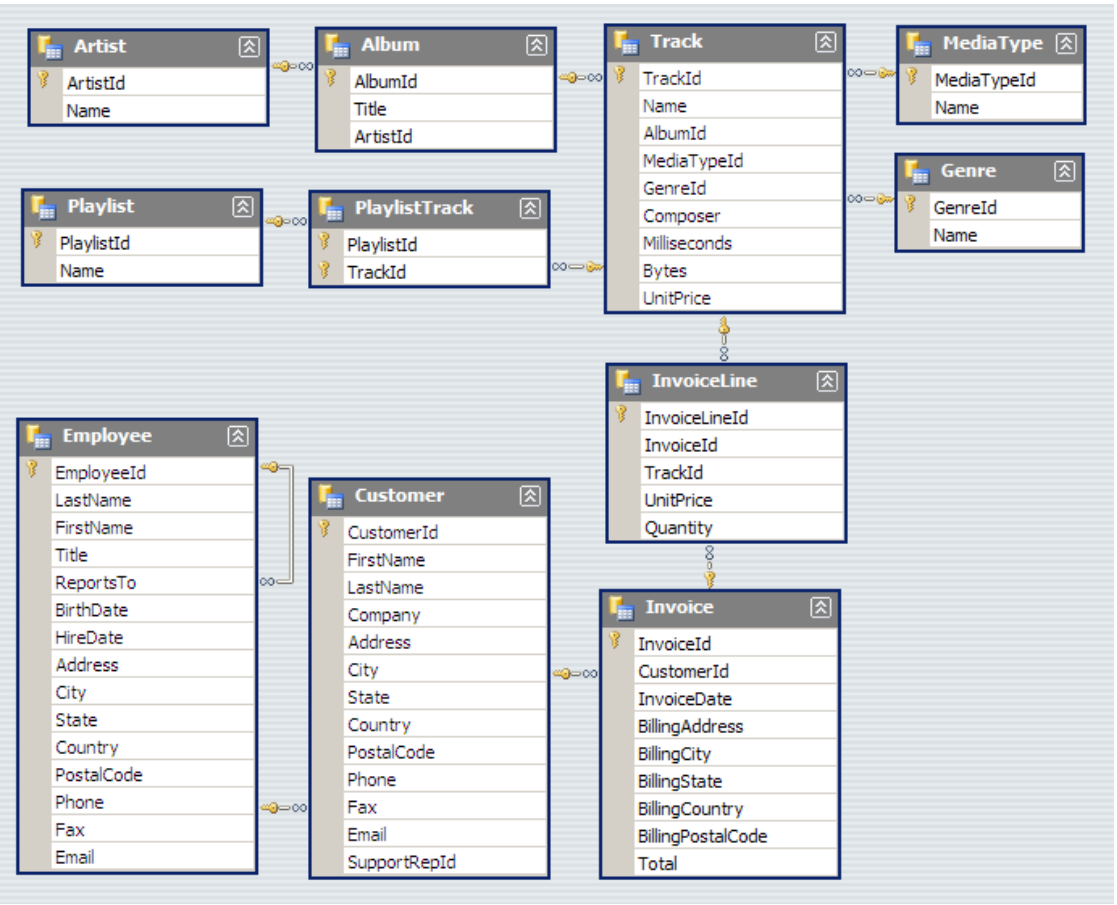
Answer:

SELECT c.firstname, c.lastname, i.invoiceid, i.invoicedate, i.billingcountry

FROM customer AS c INNER JOIN invoice AS i ON c.customerid = i.customerid

WHERE c.country = 'Brazil';

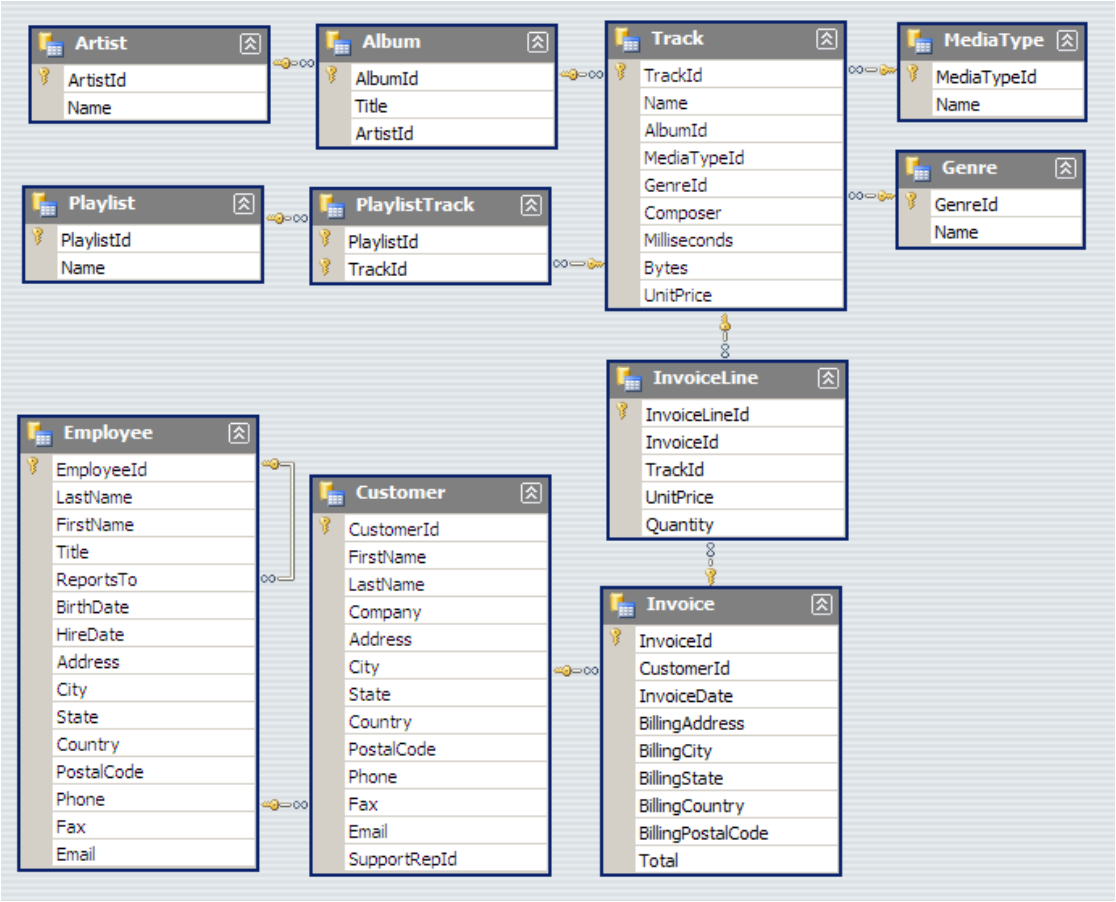
Practice



Answer:

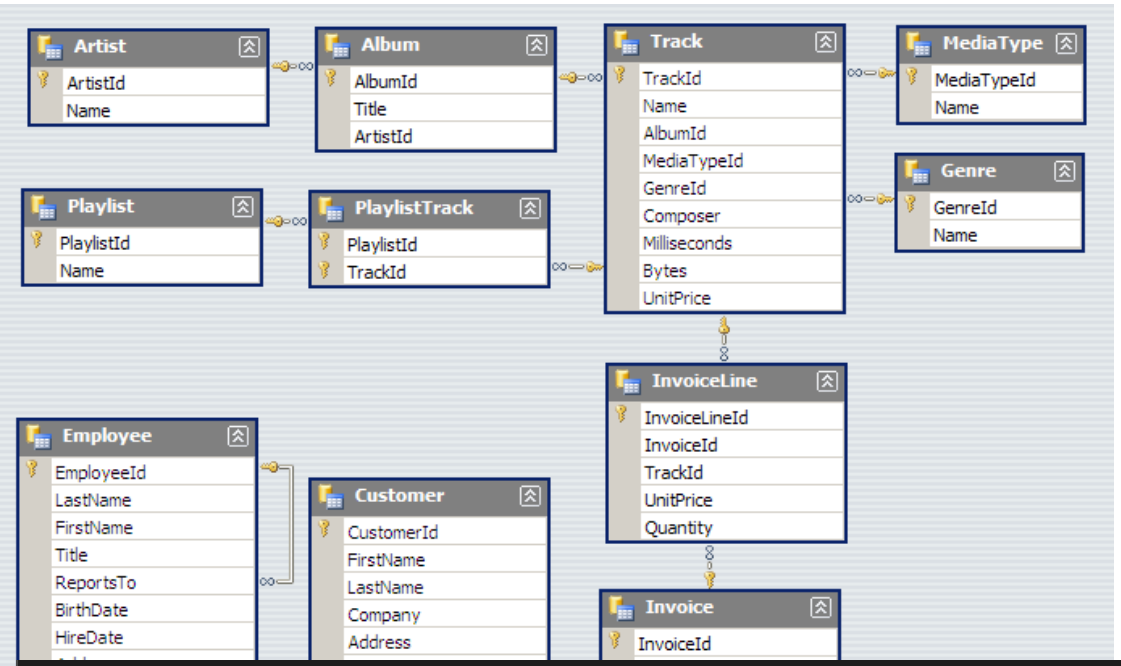
```
SELECT c.firstname, c.lastname,
i.invoiceid, i.invoicedate, i.billingcountry
FROM customer AS c INNER JOIN invoice
AS i ON c.customerid = i.customerid
WHERE c.country = 'Brazil';
```

Practice



Provide a query showing each artist and their albums

Practice



Answer:

```
SELECT art.Name, alb.Title
FROM Artist AS art LEFT OUTER JOIN
Album AS alb
on art.ArtistId = alb.ArtistId;
```

```
Edson, DJ Marky & DJ Patife Featuring Fernanda Porto
Metallica                                                Garage Inc. (Disc 1)
Metallica                                                Black Album
Metallica                                                Garage Inc. (Disc 2)
Metallica                                                Kill 'Em All
Metallica                                                Load
Metallica                                                Master Of Puppets
Metallica                                                ReLoad
Metallica                                                Ride The Lightning
Metallica                                                St. Anger
Metallica                                                ...And Justice For All
Queen                                                    Greatest Hits II
Queen                                                    Greatest Hits I
Queen                                                    News Of The World
Kiss                                                     Greatest Kiss
Kiss                                                     Unplugged [Live]
```

That's it for today

- Questions?