

MLDS-413 Data Management and Information Processing

Homework 5: Schema Creation and Data Manipulation in SQL

Ayush Agarwal

Name 1: _____

scg1143

NetID 1: _____

Lowan(Sydney) Li

Name 2: _____

chr0390

NetID 2: _____

Instructions

You should submit this homework assignment via Canvas. Acceptable formats are word files, text files, and pdf files. Paper submissions are not allowed and they will receive an automatic zero.

As explained during lecture and in the syllabus, assignments are done in groups. The groups have been created and assigned. Each group needs to submit only one assignment (i.e., there is no need for both partners to submit individually the same homework assignment).

Each group can submit solutions multiple times (for example, you may discover an error in your earlier submission and choose to submit a new solution set). We will grade only the last submission and ignore earlier ones.

Make sure you submit your solutions before the deadline. The policies governing academic integrity, tardiness and penalties are detailed in the syllabus.

EntertainmentAgency.sqlite Database (50 points)

- 1) **(10 points)** Using a single query, identify the members that have not been assigned a gender and update their gender to "M". The updated table will be used to answer the subsequent questions in this homework.

123 MemberID	ABC MbrFirstName	ABC MbrLastName	ABC MbrPhoneNumber	ABC Gender
125	Jim	Glynn	555-2531	[NULL]

```
UPDATE
Members
SET Gender = 'M'
WHERE Gender IS NULL;
```

1 row with MemnberID = 125 is updated

- 2) **(10 points)** Using the updated database/table from the previous questions, find the number of male and female members (separate counts for each gender) for each entertainer. The output table should have three columns named EntertainerID, Gender, and GenderCount. The query **must not** use a set operation (i.e., IN, NOT IN, UNION, etc.). *Hint:* GROUP BY can take multiple columns as arguments.

```
SELECT em.EntertainerID , m.Gender , COUNT(*) as GenderCount
FROM
Entertainer_Members em
inner join
Members m
on
em.MemberID = m.MemberID
group by
em.EntertainerID , m.Gender
```

	123 EntertainerID 🔼🔼	ABC Gender 🔼🔼	123 GenderCount 🔼🔼
1	1,001 ↗	F	3
2	1,002 ↗	F	1
3	1,002 ↗	M	1
4	1,003 ↗	F	1
5	1,003 ↗	M	5
6	1,004 ↗	M	1
7	1,005 ↗	F	1
8	1,005 ↗	M	2
9	1,006 ↗	F	1
10	1,006 ↗	M	3
11	1,007 ↗	F	3
12	1,007 ↗	M	2
13	1,008 ↗	F	1
14	1,008 ↗	M	4
15	1,009 ↗	F	1
16	1,010 ↗	F	4
17	1,011 ↗	F	1
18	1,012 ↗	F	1
19	1,013 ↗	F	2
20	1,013 ↗	M	2

- 3) **(10 points)** You want to classify entertainers with more than 10 engagements as Super Bands, and you want to store this information in the database. Write the query that adds a new column to the Entertainers table named “BandRank” that is a string of 50 characters.

```
ALTER TABLE Entertainers
ADD BandRank TEXT(50);
```

- 4) **(10 points)** Using the updated database/table from the previous questions , write the query that populates the BandRank column of the Entertainers table with the text “Super Band” for all entertainers with **more** than 10 engagements.

```
UPDATE
Entertainers
set BandRank = 'Super Band'
where
EntertainerID in (
    SELECT EntertainerID
    FROM
    Engagements e
    group by
    EntertainerID
    having
    COUNT(DISTINCT EngagementNumber) > 10
)
```

3 Entertainers with EntertainerID 1001, 1008 and 1013 are updated. Below is their EngagementCounts

	123 EntainerID 🔼🔼	123 EngagementCounts 🔼🔼
1	1,001 ↗	11
2	1,008 ↗	15
3	1,013 ↗	11

- 5) **(10 points)** Using the updated database/table from the previous questions , write the query that outputs the band rank (the column you added earlier), entertainer ID, entertainer stage name, and number of engagements for all entertainers in descending order of number of engagements.

```

SELECT
    e.EntainerID,
    e.EntStageName,
    e.BandRank,
    COUNT(g.EngagementNumber) AS NumberOfEngagements
FROM Entainers e
LEFT JOIN Engagements g ON e.EntainerID = g.EntainerID
GROUP BY e.EntainerID
ORDER BY NumberOfEngagements DESC;

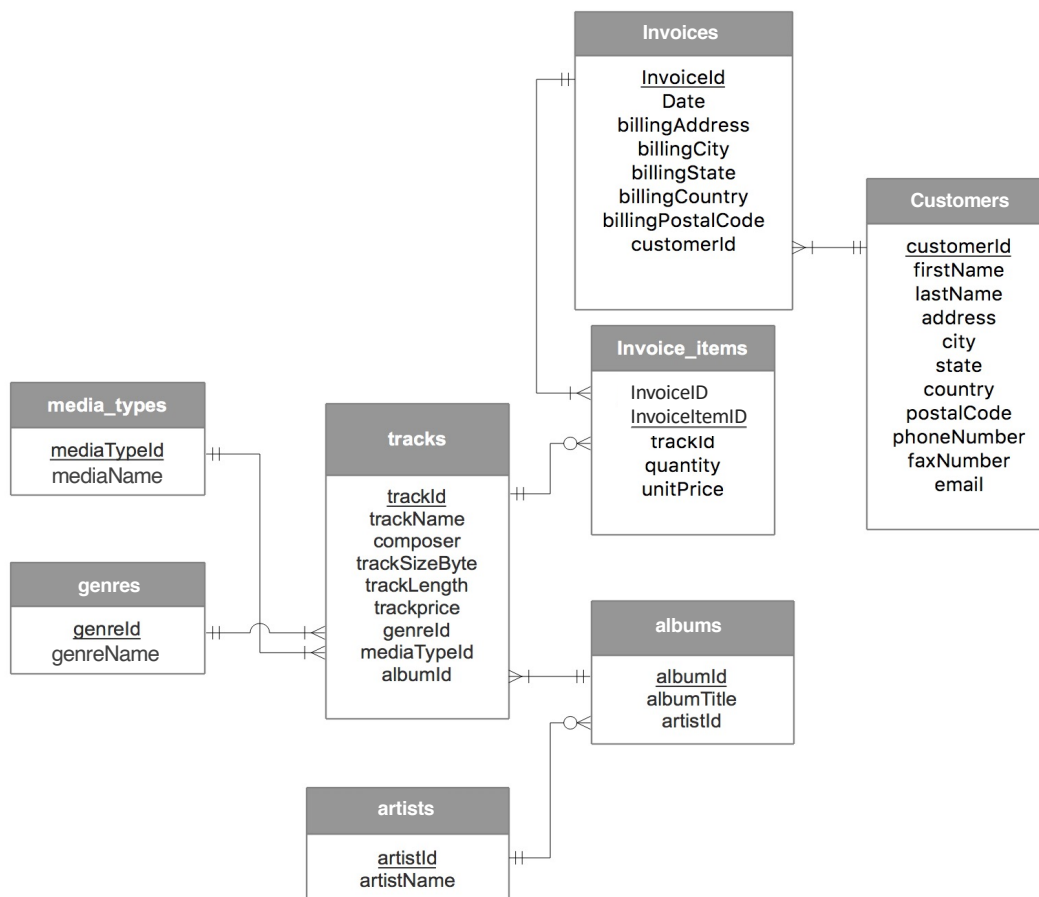
```

	123 EntainerID 🔼🔼	ABC EntStageName 🔼🔼	ABC BandRank 🔼🔼	123 NumberOfEngagements 🔼🔼
1	1,008	Country Feeling	Super Band	15
2	1,001	Carol Peacock Trio	Super Band	11
3	1,013	Caroline Coie Cuartet	Super Band	11
4	1,003	JV & the Deep Six	[NULL]	10
5	1,006	Modern Dance	[NULL]	10
6	1,004	Jim Glynn	[NULL]	9
7	1,010	Saturday Revue	[NULL]	9
8	1,007	Coldwater Cattle Company	[NULL]	8
9	1,011	Julia Schnebly	[NULL]	8
10	1,002	Topazz	[NULL]	7
11	1,005	Jazz Persuasion	[NULL]	7
12	1,012	Susan McLain	[NULL]	6
13	1,009	Katherine Ehrlich	[NULL]	0

Online Music Store Database (50 points)

- 6) (30 points). The hw5_original.csv file is a database of an on-line music store in a comma-separated file format. In order to remove redundancy and inefficiencies, we normalized it according to the following rules:
- Artist names, customer last names and invoice IDs are unique
 - Each artist can have zero or more albums
 - Each album is made by exactly one artist
 - Each track appears in exactly one album. Note that some tracks with the same name from the same composer may appear in different albums; in that case, however, they have different lengths, so they are considered different tracks (i.e., the tuple <trackName, trackLength> is unique for each track)
 - Each album has at least one track
 - Each track belongs to exactly one genre
 - Each genre is represented by at least one track
 - Each track is stored in exactly one media type
 - Each media type is used by at least one track
 - Each invoice item is for exactly one track and part of exactly one invoice
 - Each invoice has at least one invoice item
 - Some tracks may have never been sold (so there are no invoices for them)
 - Each invoice is issued to exactly one customer
 - Each customer has been issued at least one invoice
 - The following columns always have a value: media name, genre name, artist name, album title, customer first and last name, customer email, track name, track price, track length, invoice date, invoice item unit price, and invoice item quantity.

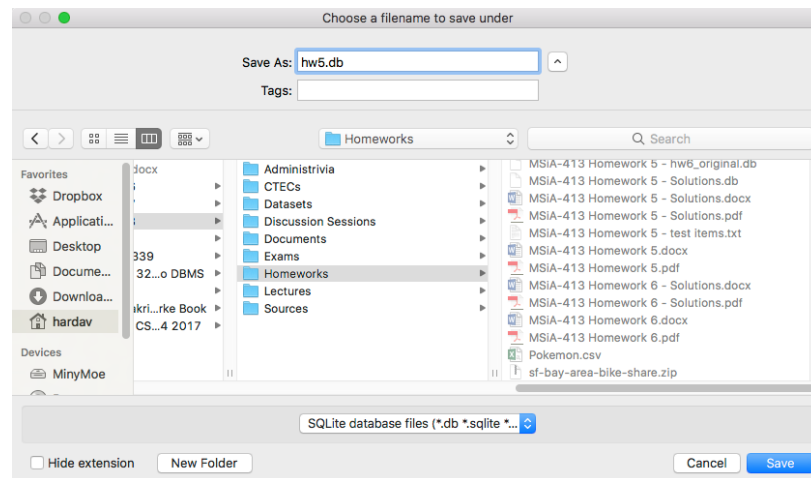
The corresponding normalized database diagram is given below:



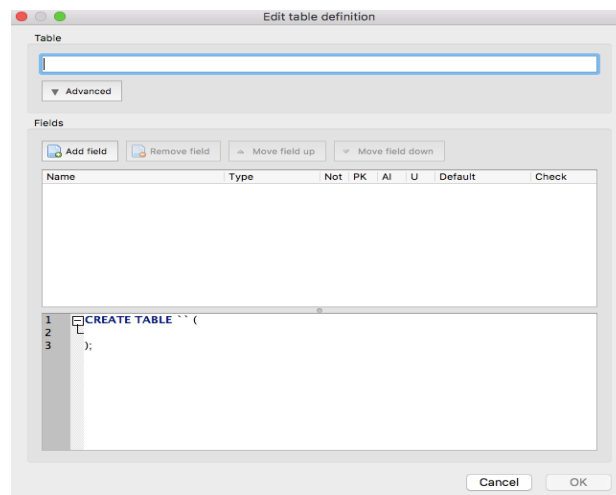
We now want to create a SQLite database that implements the normalized database. The first step is to create a new database with just one table that has the same data as the CSV file. Here is how to do that:

(0 points) Part A: Creating a new database and importing a CSV file

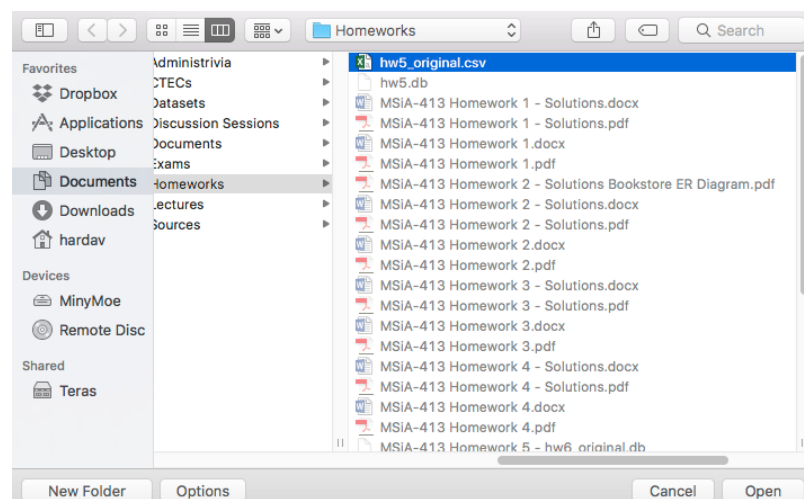
Start DB Browser for SQLite. Create a new database (File → New Database). A window will pop up like the one below. Specify a name and path to your new database file and click “save” to create the file.



Then, another window will pop up like the one below that asks for a table definition. Click cancel.



Then, create and populate one table with the data from the hw5_original.csv file. The easiest way to do this is to use File → Import → Table from CSV file. Select the hw5_original.csv file (download it from Canvas) and click open.



Then, a window will pop up in which you specify how the CSV data will be imported. Make sure “Column names in first line” is checked and the UTF-8 encoding is selected, as in the snapshot below. Click OK.

Table name:

Column names in first line: ☒

Field separator:

Quote character:

Encoding:

Trim fields? ☐

	TrackName	Composer	TrackLength	TrackSizeBytes	TrackPrice	Genre	MediaT
1	For Those A...	Angus Youn...	343719	11170334	0.99	Rock	MPEG au
2	Balls to the ...		342562	5510424	0.99	Rock	Protecter
3	Balls to the ...		342562	5510424	0.99	Rock	Protecter
4	Fast As a Sh...	F. Baltes, S. ...	230619	3990994	0.99	Rock	Protecter
5	Restless and...	F. Baltes, R....	252051	4331779	0.99	Rock	Protecter

This process automatically creates a single table (with the default name `hw5_original`) with as many columns as the CSV file, and with the column names extracted from the first line of the CSV file. Now, it is time to create the tables of the normalized database and populate them with data.

(30 points) Part B: Creating the normalized database

With the single-table database above as the starting point, create the database tables that follow the normalized database diagram shown earlier. Then, issue SQL queries against the original table you created (`hw5_original`) to populate these tables with data. Include in your answer:

- All the SQL commands that you used to create the tables and populate the data.
- The resulting SQLite database (submit it as a separate file through canvas). Remember to click “Write Changes” to save your data and tables. Please do not drop the `hw5_original` table (the initial table you created when uploading the data from the CSV file); it needs to be in your database for debugging and grading purposes.
- Please ensure tables and columns are named exactly as pictured in the diagram to facilitate grading.

Please note that in the SQL command that creates a table you must explicitly specify the primary and foreign keys and any `UNIQUE` and `NOT NULL` constraints. Your SQL commands should ensure that you implement all constraints in the list above except (e), (g), (i), (k), and (n); these constraints require a more complex treatment and cannot be implemented using the `CREATE TABLE` interface.

As a starting point, below are the SQL commands to create the table for media types. Use these SQL statements in your answer, and proceed with creating and populating the remaining tables. The commands should appear in the exact order that are to be issued to the database. We assume that the imported data from the CSV file were stored in a table with name `hw5_original`.

```
drop table if exists media_types;
create table media_types (
    mediaTypeId integer not null primary key autoincrement,
    mediaName nvarchar(20) not null
);
insert into media_types (mediaName)
select distinct mediaType from hw5_original;
```

CUSTOMERS TABLE

```
CREATE TABLE Customers (  
    customerId INTEGER PRIMARY KEY AUTOINCREMENT,  
    CustomerFirstName VARCHAR(8) NOT NULL,  
    CustomerLastName VARCHAR(8) NOT NULL,  
    CustomerAddress VARCHAR(32),  
    CustomerCity VARCHAR(16),  
    CustomerState VARCHAR(2),  
    CustomerCountry VARCHAR(16),  
    CustomerPostalCode VARCHAR(16),  
    CustomerPhone VARCHAR(32),  
    CustomerFax VARCHAR(32),  
    CustomerEmail VARCHAR(32) NOT NULL  
);
```

```
INSERT INTO Customers (CustomerLastName,  
                        CustomerFirstName,  
                        CustomerAddress,  
                        CustomerCity,  
                        CustomerState,  
                        CustomerCountry,  
                        CustomerPostalCode,  
                        CustomerPhone,  
                        CustomerFax,  
                        CustomerEmail)
```

```
SELECT DISTINCT(CustomerLastName) ,  
               CustomerFirstName ,  
               CustomerAddress ,  
               CustomerCity ,  
               CustomerState ,  
               CustomerCountry ,  
               CustomerPostalCode ,  
               CustomerPhone ,  
               CustomerFax ,  
               CustomerEmail  
FROM  
hw5_original ho  
where  
CustomerFirstName is not NULL and CustomerLastName is not NULL and CustomerEmail is not NULL  
and  
length(CustomerFirstName) != 0 and length(CustomerLastName) != 0 and length(CustomerEmail) != 0;
```

ARTISTS TABLE

```
CREATE TABLE artists (  
    artistId INTEGER PRIMARY KEY AUTOINCREMENT,  
    ArtistName VARCHAR(32) NOT NULL  
);
```

```
INSERT INTO artists (ArtistName)  
SELECT DISTINCT ArtistName from hw5_original  
where ArtistName is not NULL and LENGTH(ArtistName) != 0;
```


GENRES TABLE

```
CREATE TABLE genres (  
    genreId INTEGER PRIMARY KEY AUTOINCREMENT,  
    genreName VARCHAR(8) NOT NULL  
);
```

```
INSERT INTO genres (genreName)  
SELECT DISTINCT ho.Genre  
FROM  
hw5_original ho  
where  
ho.Genre is not NULL and LENGTH(ho.Genre) != 0;
```

MEDIA TYPES TABLE

```
CREATE TABLE media_types (  
    mediaTypeId INTEGER PRIMARY KEY AUTOINCREMENT,  
    mediaName VARCHAR(32) NOT NULL  
);
```

```
INSERT INTO media_types (mediaName)  
SELECT DISTINCT MediaType  
from  
hw5_original  
where  
MediaType is not NULL and length(MediaType) != 0
```

ALBUMS TABLE

```
CREATE TABLE albums (  
    albumId INTEGER PRIMARY KEY AUTOINCREMENT,  
    albumTitle VARCHAR(64) NOT NULL,  
    artistId INT NOT NULL,  
    FOREIGN KEY (artistId) REFERENCES artists(artistId)  
);
```

```
INSERT INTO albums (albumTitle, artistId)  
SELECT DISTINCT ho.AlbumTitle, a.artistId  
FROM  
hw5_original ho  
JOIN  
artists a  
ON ho.ArtistName = a.ArtistName;
```

INVOICES TABLE

```
CREATE TABLE invoices (  
    invoiceId INTEGER PRIMARY KEY,  
    Date DATE NOT NULL,  
    billingAddress VARCHAR(32),  
    billingCity VARCHAR(16),  
    billingState VARCHAR(2),
```

```

        billingCountry VARCHAR(16),
        billingPostalCode VARCHAR(16),
        customerId INTEGER,
        FOREIGN KEY (customerId) REFERENCES customers(customerId)
    );

```

```

INSERT                                INTO                                invoices
(invoiceId,Date,billingAddress,billingCity,billingState,billingCountry,billingPostalCode,customerId)
SELECT
    DISTINCT ho.InvoiceId,
    ho.InvoiceDate ,
    InvoiceBillingAddress ,
    InvoiceBillingCity ,
    InvoiceBillingState ,
    InvoiceBillingCountry ,
    InvoiceBillingPostalCode ,
    c.customerId
FROM hw5_original ho
inner join
Customers as c
on ho.CustomerLastName = c.CustomerLastName
where
LENGTH(ho.InvoiceId) != 0;

```

TRACKS TABLE

```

CREATE TABLE tracks(
    trackid INTEGER PRIMARY KEY AUTOINCREMENT,
    trackName VARCHAR(64) NOT NULL,
    composer VARCHAR(128),
    trackSizeByte INTEGER,
    trackLength INTEGER NOT NULL,
    trackprice REAL NOT NULL,
    genreId INTEGER NOT NULL,
    mediaTypeId INTEGER NOT NULL,
    albumId INTEGER NOT NULL,
    FOREIGN KEY (genreId) REFERENCES genres(genreId),
    FOREIGN KEY (mediaTypeId) REFERENCES media_types(mediaTypeId),
    FOREIGN KEY (albumId) REFERENCES albums(albumId)
);

```

```

INSERT                                INTO                                tracks
(trackName,composer,trackSizeByte,trackLength,trackprice,genreId,mediaTypeId,albumId)
SELECT  trackName,composer,trackSizeBytes,trackLength,trackprice,  g.genreId  ,  mt.mediaTypeId  ,
a.albumId
FROM
hw5_original ho
inner join
genres g
on Genre = g.genreName
inner join
media_types mt

```

```

on ho.MediaType = mt.mediaName
inner join
albums a
on
a.albumTitle = ho.AlbumTitle
group by
ho.TrackName || '-' || ho.TrackLength;

```

INVOICE_ITEMS TABLE

```

CREATE TABLE invoice_items (
    InvoiceItemId INTEGER PRIMARY KEY ,
    InvoiceId INTEGER NOT NULL,
    trackId INTEGER NOT NULL,
    quantity INTEGER NOT NULL,
    unitPrice REAL NOT NULL,
    FOREIGN KEY (InvoiceId) REFERENCES invoices(invoiceId),
    FOREIGN KEY (trackId) REFERENCES tracks(trackId)
);

INSERT INTO invoice_items (InvoiceId,trackId,quantity,unitPrice)
SELECT ho.InvoiceId , t.trackId, ho.InvoiceItemQuantity , ho.InvoiceItemUnitPrice
FROM
hw5_original ho
inner join
invoices i
on
ho.InvoiceId = i.invoiceId
inner join
tracks t
on
ho.TrackName || '-' || ho.TrackLength = t.trackName || '-' || t.trackLength
where
length(ho.InvoiceId) != 0;

```

- 7) **(10 points)** Find the best-selling artist and how much customers spent in buying this artist's songs, based on the normalized database that you created and populated in the previous questions.

```

SELECT a.ArtistName , COUNT(it.invoiceId) as tracks_sold, SUM(it.quantity * it.unitPrice) as money_spent
FROM
artists a
inner join
albums a2
on a.artistId = a2.artistId
inner join
tracks t
on a2.albumId = t.albumId
inner join
invoice_items as it
on t.trackId = it.trackId
group by
a.artistId
order by

```

```
tracks_sold  
DESC  
limit 1;
```

Best Selling Artist : Iron Maiden
Customer Spent : 138.6

	ABC ArtistName	123 tracks_sold	123 money_spent
1	Iron Maiden	140	138.6

- 8) **(10 points)** Instead of creating and loading the database through the GUI as explained in Problem (6) Part A, write your own Python program to do it. Your program should open the CSV file, interface with the SQLite database, and issue SQL commands to the database to create and populate the `hw5_original` table with data from the CSV file.

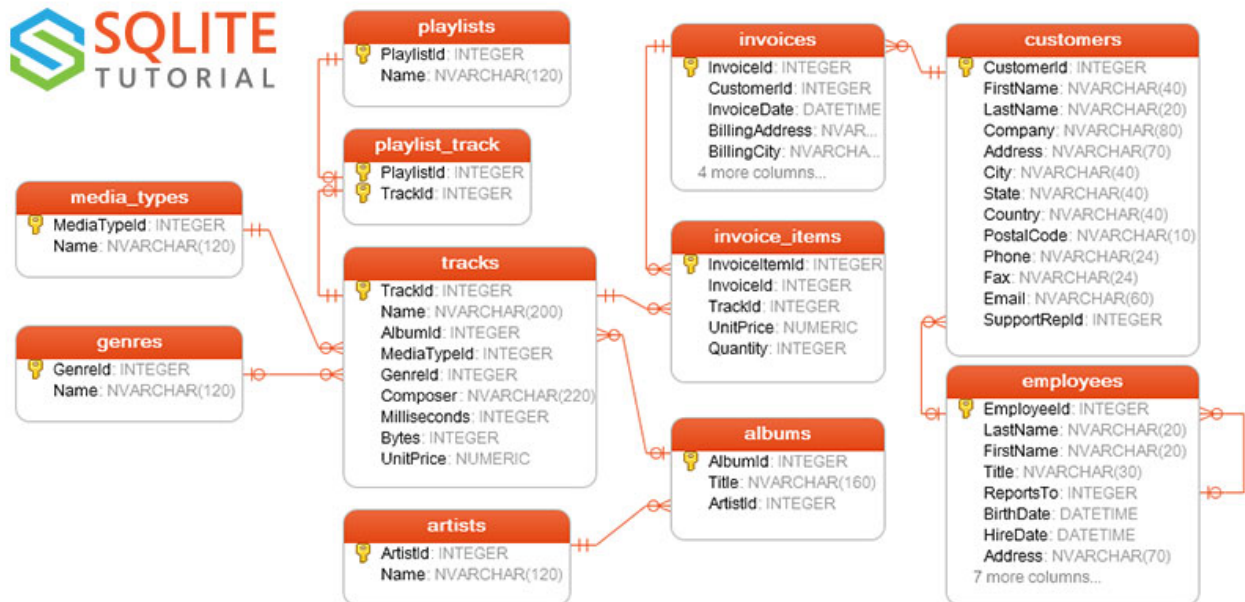
For this problem, you will need to do some self-learning. There are several tutorials online on the programming interfaces supported by SQLite. For example, <http://www.sqlitetutorial.net/> has tutorials for PySQLite and APSW which allow Python to interface with SQLite. Similarly, you can find tutorials on the SQLite-Python interface at <https://www.tutorialspoint.com/sqlite/>. There are many other tutorials online that may give more details on interfacing Python to SQLite, and also expand the choices to other programming languages (e.g., R, Matlab, ML, C, C++). Feel free to find a tutorial that best fits your needs and utilize it.

To submit your answer, submit your program's source code by uploading it as a separate file on Canvas.

Reference: creating a CSV dataset through sqlite3

This part is written as a reference of how I created the dataset and stored it in a CSV file for problems 5–8. It is not needed to solve this homework, but it is given here for your perusal and future reference.

- I took the database from <http://www.sqlitetutorial.net/sqlite-sample-database/>



- Then, I joined these tables together into a single result table using the following commands on SQLite's command line interface (sqlite3):

```
sqlite> .open hw5_original.db
sqlite> .headers on
sqlite> .mode csv
sqlite> .output hw5_original.csv
sqlite> select tracks.Name as TrackName,
        tracks.composer,
        tracks.milliseconds as TrackLength,
        tracks.bytes as TrackSizeBytes,
        tracks.unitPrice as TrackPrice,
        genres.name as Genre,
        media_types.name as MediaType,
        albums.Title as AlbumTitle,
        artists.Name as ArtistName,
        invoice_items.Quantity as InvoiceItemQuantity,
        invoice_items.UnitPrice as InvoiceItemUnitPrice,
        invoices.invoiceId,
        invoices.InvoiceDate,
        invoices.BillingAddress as InvoiceBillingAddress,
        invoices.BillingCity as InvoiceBillingCity,
        invoices.BillingState as InvoiceBillingState,
        invoices.BillingCountry as InvoiceBillingCountry,
        invoices.BillingPostalCode as InvoiceBillingPostalCode,
        customers.FirstName as CustomerFirstName,
        customers.LastName as CustomerLastName,
        customers.Address as CustomerAddress,
        customers.City as CustomerCity,
        customers.State as CustomerState,
        customers.Country as CustomerCountry,
        customers.PostalCode as CustomerPostalCode,
        customers.Phone as CustomerPhone,
        customers.Fax as CustomerFax,
        customers.Email as CustomerEmail
from tracks
left join invoice_items on tracks.TrackId = invoice_items.TrackId
left join media_types on media_types.MediaTypeId = tracks.MediaTypeId
```

```
left join genres on genres.GenreId = tracks.GenreId
left join albums on albums.AlbumId = tracks.AlbumId
left join artists on artists.ArtistId = albums.ArtistId
left join invoices on invoice_items.InvoiceId = invoices.InvoiceId
left join customers on invoices.CustomerId = customers.CustomerId;
```

- Then, I exported the resulting table as a CSV file with name `hw5_original.csv`.