MLDS-413 Data Management and Information Processing

Homework 5: Schema Creation and Data Manipulation in SQL

Name 1: _____

NetID 1: _____

Name 2: _____

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.

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

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 EntertainerID,
       Gender,
       COUNT(*) AS GenderCount
FROM Members JOIN Entertainer_Members
     ON Members.MemberID = Entertainer_Members.MemberID
GROUP BY EntertainerID, Gender;
```

Output:

```
"1001"      "F"   "3"
"1002"      "F"   "1"
"1002"      "M"   "1"
"1003"      "F"   "1"
"1003"      "M"   "5"
"1004"      "M"   "1"
"1005"      "F"   "1"
"1005"      "M"   "2"
"1006"      "F"   "1"
"1006"      "M"   "3"
"1007"      "F"   "3"
"1007"      "M"   "2"
"1008"      "F"   "1"
"1008"      "M"   "4"
"1009"      "F"   "1"
"1010"      "F"   "4"
"1011"      "F"   "1"
"1012"      "F"   "1"
"1013"      "F"   "2"
"1013"      "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 COLUMN BandRank varchar(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 Entertainers.EntertainerID
                        FROM Entertainers JOIN Engagements
                        ON Entertainers.EntertainerID = Engagements.EntertainerID
                        GROUP BY Entertainers.EntertainerID
                        HAVING COUNT(*) > 10);
```

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 BandRank,
      Entertainers.EntertainerID,
      EntStageName,
      COUNT(*) AS NumEngagements
FROM Entertainers JOIN Engagements
      ON Entertainers.EntertainerID = Engagements.EntertainerID
GROUP BY Entertainers.EntertainerID
ORDER BY NumEngagements DESC;
```

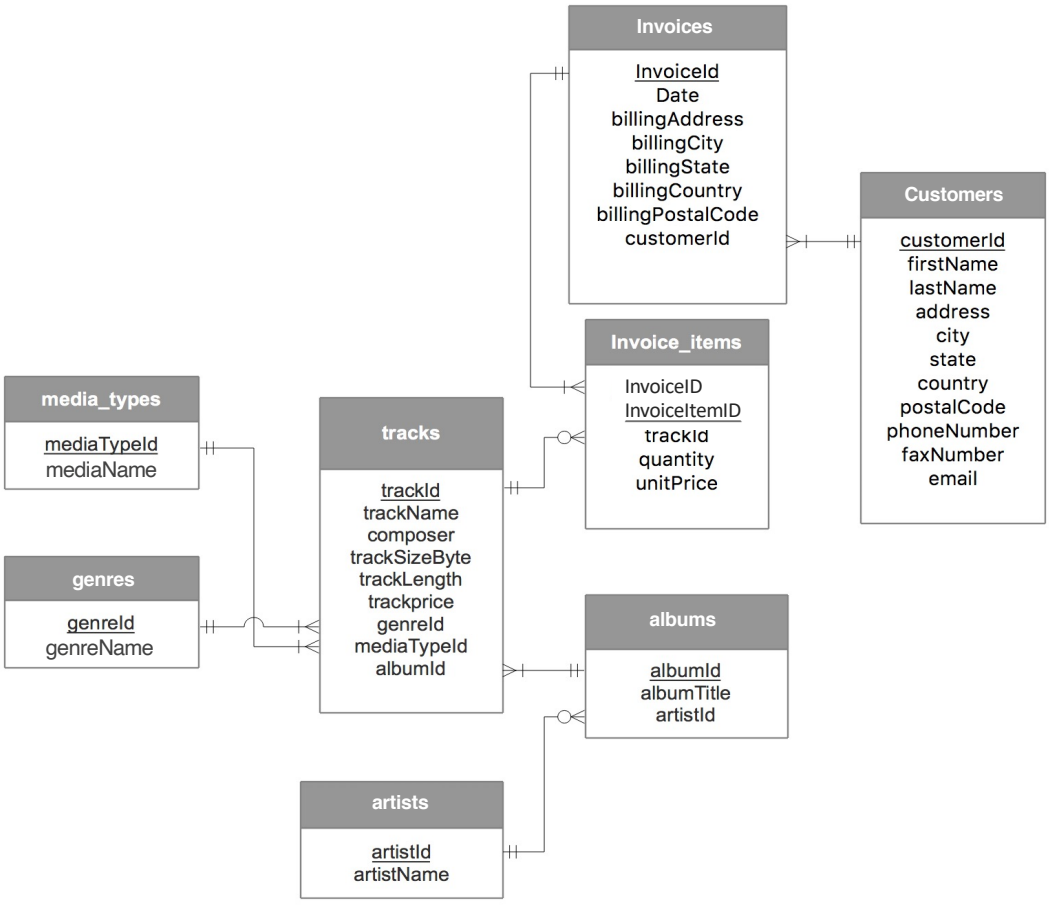
Output:

```
Super Band  1008  Country Feeling              15
Super Band  1013  Caroline Coie Cuartet        11
Super Band  1001  Carol Peacock Trio           11
            1006  Modern Dance                 10
            1003  JV & the Deep Six            10
            1010  Saturday Revue               9
            1004  Jim Glynn                    9
            1011  Julia Schnebly               8
            1007  Coldwater Cattle Company     8
            1005  Jazz Persuasion              7
            1002  Topazz                       7
            1012  Susan McLain                 6
```

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:
   a. Artist names, customer last names and invoice IDs are unique
   b. Each artist can have zero or more albums
   c. Each album is made by exactly one artist
   d. 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)
   e. Each album has at least one track
   f. Each track belongs to exactly one genre
   g. Each genre is represented by at least one track
   h. Each track is stored in exactly one media type
   i. Each media type is used by at least one track
   j. Each invoice item is for exactly one track and part of exactly one invoice
   k. Each invoice has at least one invoice item
   l. Some tracks may have never been sold (so there are no invoices for them)
   m. Each invoice is issued to exactly one customer
   n. Each customer has been issued at least one invoice
   o. 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.

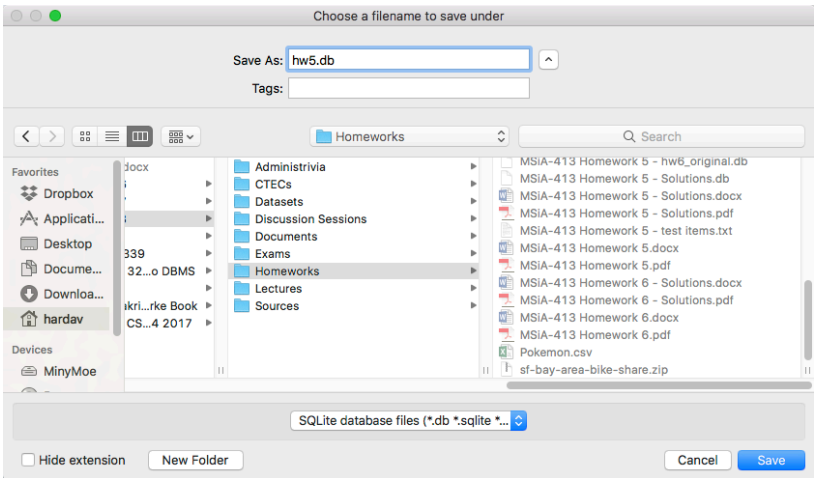
The corresponding normalized database diagram is given below:



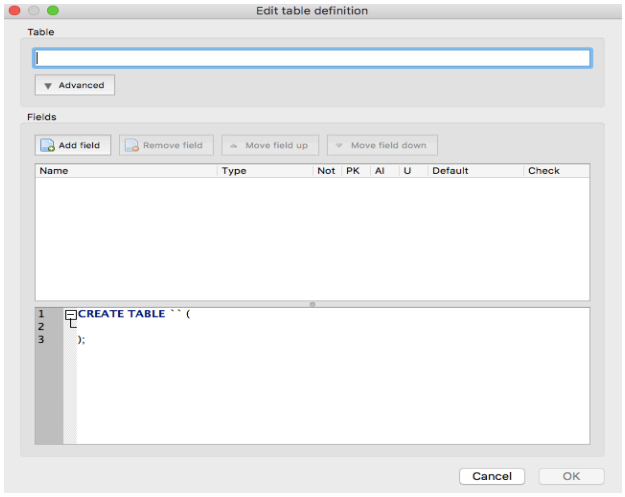
We now want to create a SQLite database that implements the normalized database. The first step is to create a new database with just one table that has the same data as the CSV file. Here is how to do that:

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

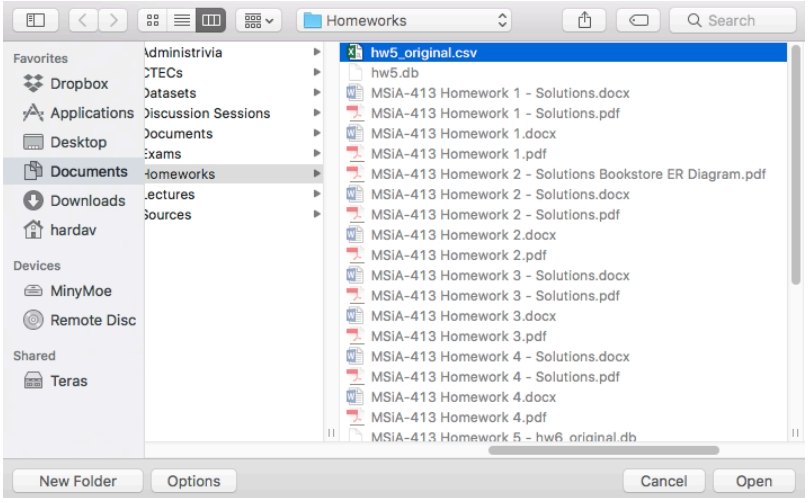
Start DB Browser for SQLite. Create a new database (`File → New Database`). A window will pop up like the one below. Specify a name and path to your new database file and click "save" to create the file.



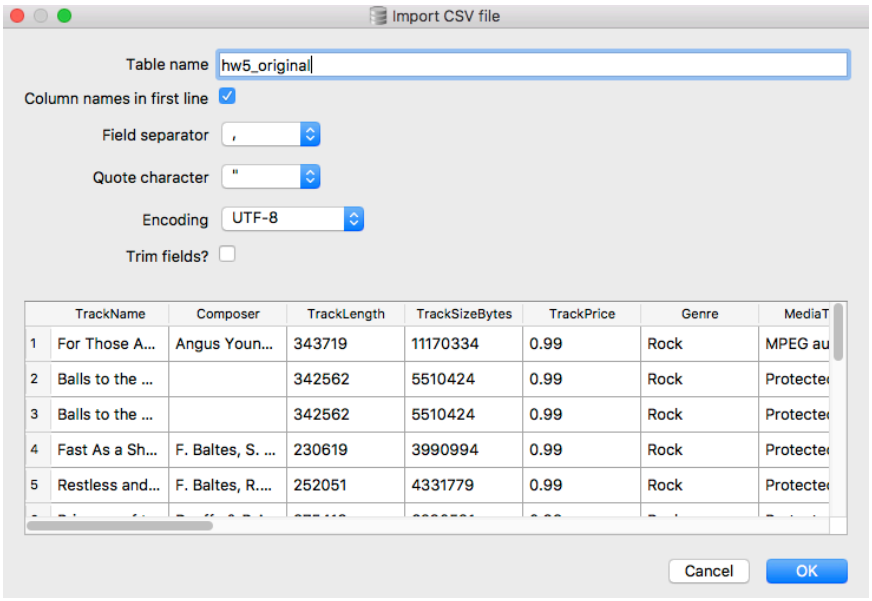
Then, another window will pop up like the one below that asks for a table definition. Click cancel.



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.



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;
```

Answer:

You have two choices when designing the `invoices` table: use `invoiceID` from `hw5_original` as a primary key, or create your own `invoiceID`. If you use `invoiceID` from `hw5_original`, populating the database tables is easier. However, the primary key cannot be declared `autoincrement`. Thus, when you use the database in the future to insert new invoices, you must explicitly specify the value of `invoiceID` for each new invoice, which is cumbersome and error prone. It would be better to declare `invoiceID` as an `autoincrement` primary key. The solution below assumes the latter design: it keeps the old `invoiceID` on a separate column for backwards compatibility (if needed) and to assist populating the tables with the correct data. Both designs will be considered correct.

```
-- drop tables in the inverse hierarchy order. This is just to simplify debugging.
drop table if exists invoice_items;
drop table if exists tracks;
drop table if exists invoices;
drop table if exists customers;
drop table if exists albums;
drop table if exists artists;
drop table if exists genres;
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;
-------------------------------------------------------------------------------
--

create table genres (
      genreId integer not null primary key autoincrement,
      genreName nvarchar(20) not null
);
insert into genres (genreName)
select distinct genre from hw5_original;
-------------------------------------------------------------------------------
--

create table artists (
      artistId integer not null primary key autoincrement,
      artistName text not null,
      unique(artistName)
);
insert into artists (artistName)
select distinct artistName from hw5_original;
-------------------------------------------------------------------------------
--

create table albums (
      albumId integer not null primary key autoincrement,
      albumTitle nvarchar(200) not null,
      artistId integer not null references artists(artistid)
);
insert into albums (albumTitle, artistId)
select distinct albumTitle, artistId
from hw5_original
   join artists on artists.artistName = hw5_original.artistName;
-------------------------------------------------------------------------------
--
```

```
create table customers (
      customerId integer not null primary key autoincrement,
      firstName nvarchar(200) not null,
      lastName nvarchar(200) not null,
      address nvarchar(300),
      city nvarchar(100),
      state nvarchar(10),
      country nvarchar(20),
      postalCode nvarchar(20),
      phoneNumber nvarchar(30),
      faxNumber nvarchar(30),
      email nvarchar(80) not null,
      unique(lastName)
);
insert into customers
      (lastName, firstName,
       address, city, state,
       country, postalCode,
       phoneNumber, faxNumber, email)
select distinct customerLastName, customerFirstName,
       customerAddress, customerCity, CustomerState,
       CustomerCountry, CustomerPostalCode,
       CustomerPhone, CustomerFax, CustomerEmail
       from hw5_original
         where customerLastName is not null;
-------------------------------------------------------------------------------
--

create table invoices (
      invoiceId integer not null primary key autoincrement,
      Date datetime not null,
      billingAddress nvarchar(300),
      billingCity nvarchar(100),
      billingState nvarchar(10),
      billingCountry nvarchar(20),
      billingPostalCode nvarchar(20),
      customerId integer not null references customers(customerId),
      originalInvoiceId integer
);
insert into invoices (originalInvoiceId, Date,
                      billingAddress, billingCity, billingState,
                      billingCountry, billingPostalCode, customerId)
      select distinct invoiceId, invoiceDate,
             invoiceBillingAddress, invoiceBillingCity, invoiceBillingstate,
             invoiceBillingCountry, invoiceBillingPostalCode, customerId
      from hw5_original
           join customers on customers.firstName = customerFirstName
                             and customers.lastName = customerLastName;
-------------------------------------------------------------------------------
--

create table tracks (
      trackId integer not null primary key autoincrement,
      trackName nvarchar(200) not null,
      composer nvarchar(200),
      trackSizeByte integer,
      tracklength integer not null,
      trackprice numeric not null,
      genreId integer not null references genres(genreId),
      mediaTypeId integer not null references media_types(mediaTypeId),
      albumId integer not null references albums(albumId),
      unique(trackName,trackLength)
);
insert into tracks (trackName, composer, trackSizeByte, tracklength, trackprice,
```

```
                    genreid, mediaTypeId, albumId)
select distinct trackName, Composer, TrackSizeBytes, TrackLength, TrackPrice,
       genreId, mediaTypeId, albumId
from hw5_original join genres on genres.genreName = hw5_original.Genre
     join albums on albums.albumTitle = hw5_original.AlbumTitle
      join media_types on media_types.mediaName = hw5_original.MediaType;
-------------------------------------------------------------------------------
--

create table invoice_items (
      invoiceItemId integer not null primary key autoincrement,
      invoiceId integer not null references invoices(invoiceId),
      trackId integer not null references tracks(trackId),
      quantity integer not null,
      unitPrice numeric not null
);
insert into invoice_items (invoiceid, trackid, quantity, unitPrice)
select       invoices.invoiceId,       tracks.trackid,       invoiceItemQuantity,
invoiceItemUnitPrice
from hw5_original
join tracks on tracks.trackName = hw5_original.TrackName
            and tracks.trackLength = hw5_original.TrackLength
join invoices on invoices.originalInvoiceId = hw5_original.invoiceId;
```

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 artists.artistName,
      SUM(invoice_items.Quantity * invoice_items.unitPrice) AS artistInvoiceSum
FROM tracks NATURAL JOIN albums NATURAL JOIN artists NATURAL JOIN invoice_items
GROUP BY artists.ArtistId
ORDER BY artistInvoiceSum DESC LIMIT 1;
```

```
Output
"Iron Maiden"    "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.

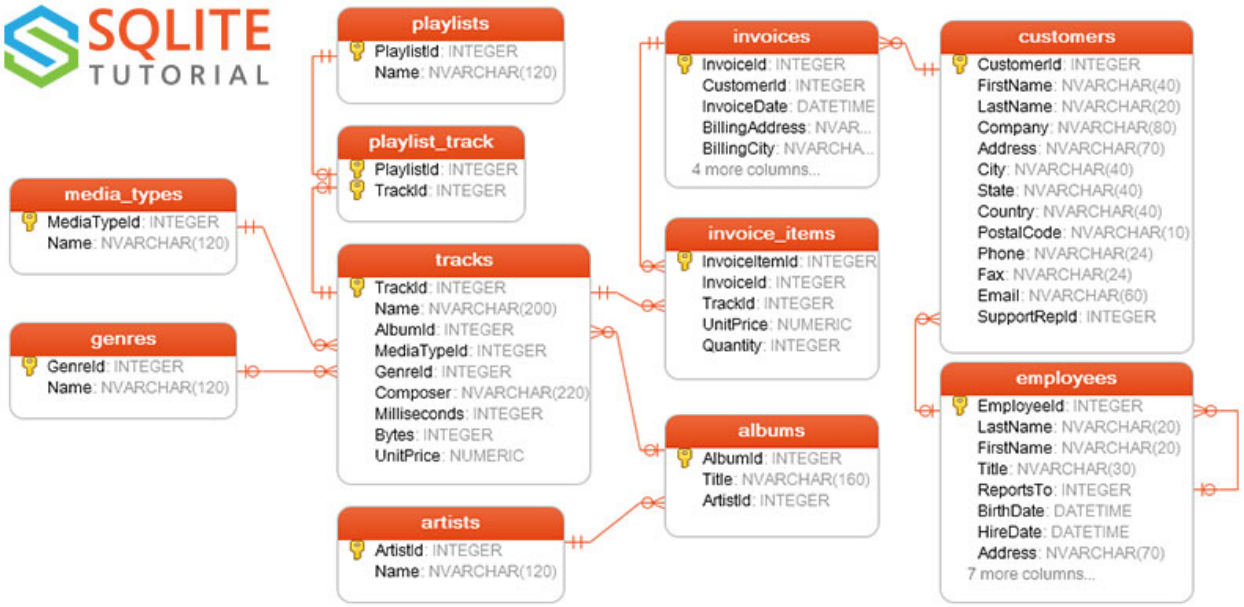
For this problem, you will need to do some self-learning. There are several tutorials online on the programming interfaces supported by SQLite. For example, http://www.sqlitetutorial.net/ has tutorials for PySQLite and APSW which allow Python to interface with SQLite. Similarly, you can find tutorials 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`.