# MSiA-413 Data Management and Information Processing

### Homework 5: Advanced SQL Queries and Data Creation

Name 1: Kristiyan Dimitrov

NetID 1: ktd5131

Name 2: Laurie Merrell

NetID 2: lmr8733

# 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.

**Due Date: Thursday November 21, 11:59 pm**

**EntertainmentAgency.sqlite Database (60 points)**

1. **(5 points)** Find the EntertainerID of the entertainers that have no engagements. You **must** use EXCEPT for full credit.

**Query:**

**﻿SELECT E.EntertainerID**

**FROM Entertainers AS E**

**EXCEPT SELECT EN.EntertainerID**

**FROM Engagements AS EN;**

**Output:**

**1009**

1. **(10 points)** Find the EntertainerID **and stage name** of the entertainers that have no engagements. Your answer must be a single query with no subqueries. You **must not** directly use the result of question (1) above.

**Query:**

**﻿SELECT E.EntertainerID, E.EntStageName**

**FROM Entertainers AS E**

**LEFT JOIN Engagements AS EN ON EN.EntertainerID = E.EntertainerID**

**GROUP BY E.EntertainerID**

**HAVING COUNT(EN.EngagementNumber) = 0**

**;**

**OR- EQUIVALENT**

**SELECT Ent.EntertainerID, Ent.EntStageName**

**FROM Entertainers AS Ent**

**LEFT JOIN Engagements AS Eng**

**ON Ent.EntertainerID = Eng.EntertainerID**

**WHERE EngagementNumber IS NULL**

**;**

**Output:**

|  |  |
| --- | --- |
| **1009** | **Katherine Ehrlich** |

1. **(10 points)** For all customers that have less than 10 engagements, list the customer ID, full name (i.e., a string containing the customer’s first name followed by the last name with a space in between), and number of engagements, in ascending order of number of engagements. Your answer must be a single query with no subqueries.

**Query:**

**﻿SELECT E.CustomerID, C.CustFirstName || " " || C.CustLastName as Cust\_FullName, COUNT(\*)**

**FROM Customers AS C**

**NATURAL JOIN Engagements as E**

**GROUP BY E.CustomerID**

**HAVING COUNT(E.EngagementNumber) < 10**

**ORDER BY COUNT(\*);**

**Output:**

|  |  |  |
| --- | --- | --- |
| **10013** | **Estella Pundt** | **6** |
| **10003** | **Peter Brehm** | **7** |
| **10007** | **Liz Keyser** | **7** |
| **10012** | **Kerry Patterson** | **7** |
| **10015** | **Carol Viescas** | **7** |
| **10001** | **Doris Hartwig** | **8** |
| **10005** | **Elizabeth Hallmark** | **8** |
| **10009** | **Sarah Thompson** | **8** |
| **10006** | **Matt Berg** | **9** |

1. **(5 points)** Using a single query, identify the members that have not been assigned a gender and update their gender to male. The updated table will be used to answer later questions in this homework.

**Query:**

**﻿UPDATE Members**

**SET Gender = "M"**

**WHERE Gender IS NULL;**

**(NO OUTPUT but Member ID 125, Jim Glynn has now been updated with Gender = “M”)**

1. **(10 points)** Using the updated database/table from question (4), find the number of male and female members (separate counts for each gender) for each entertainer. The output table should have the columns EntertainerID, Gender, and GenderCount. The query **must** use the UNION operator.

**Query:**

**﻿SELECT EM.EntertainerID, M.Gender, COUNT(M.MemberID) AS GenderCount**

**FROM Entertainer\_Members AS EM**

**JOIN Members AS M ON EM.MemberID = M.MemberID**

**WHERE M.Gender = "M"**

**GROUP BY EntertainerID**

**UNION**

**SELECT EM.EntertainerID, M.Gender, COUNT(M.MemberID) AS GenderCount**

**FROM Entertainer\_Members AS EM**

**JOIN Members AS M ON EM.MemberID = M.MemberID**

**WHERE M.Gender = "F"**

**GROUP BY EntertainerID;**

**Output:**

|  |  |  |
| --- | --- | --- |
| **1001** | **F** | **3** |
| **1002** | **F** | **1** |
| **1002** | **M** | **1** |
| **1003** | **F** | **1** |
| **1003** | **M** | **5** |
| **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** |

1. **(10 points)** Write a query to answer question (5) above, but this time the query **must not** use a set operation and it **must** use a natural join. *Hint:* GROUP BY can take multiple columns as arguments.

**Query:**

**﻿SELECT E.EntertainerID, M.Gender, COUNT(M.Gender) AS GenderCount**

**FROM Entertainer\_Members AS E**

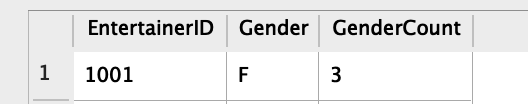
**NATURAL JOIN Members AS M**

**GROUP BY E.EntertainerID, M.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** |

**Screenshot with column names:**

****

1. **(10 points)** You want to classify each entertainer as follows:

* Super Band (if it has more than 10 engagements)
* Regular Band (if it has more than 7 but no more than 10 engagements)
* Support Band (if it has at least one engagement, but no more than 7), and
* Amateur Band (if it has no engagements)

Write the query that makes this classification and returns the class of the entertainer, the entertainer’s stage name, and the number of engagements, with the entertainers appearing in descending rank (i.e., super bands first, followed by regular bands, then support bands, and amateurs at the bottom). Your answer must be a single query with no subqueries.

**QUERY**

**﻿SELECT E1.EntStageName, COUNT(DISTINCT(E2.EngagementNumber)) AS Num\_Engagements, CASE**

**WHEN COUNT(DISTINCT(E2.EngagementNumber)) > 10 THEN "Super Band"**

**WHEN COUNT(DISTINCT(E2.EngagementNumber))> 7 THEN "Regular Band"**

**WHEN COUNT(DISTINCT(E2.EngagementNumber))> 0 THEN "Support Band"**

**WHEN COUNT(DISTINCT(E2.EngagementNumber))= 0 THEN "Amateur Band"**

**ELSE "Unclassified"**

**END AS Class**

**FROM Entertainers AS E1**

**LEFT JOIN Engagements AS E2**

**ON E1.EntertainerID = E2.EntertainerID**

**GROUP BY E1.EntertainerID**

**ORDER BY Num\_Engagements DESC;**

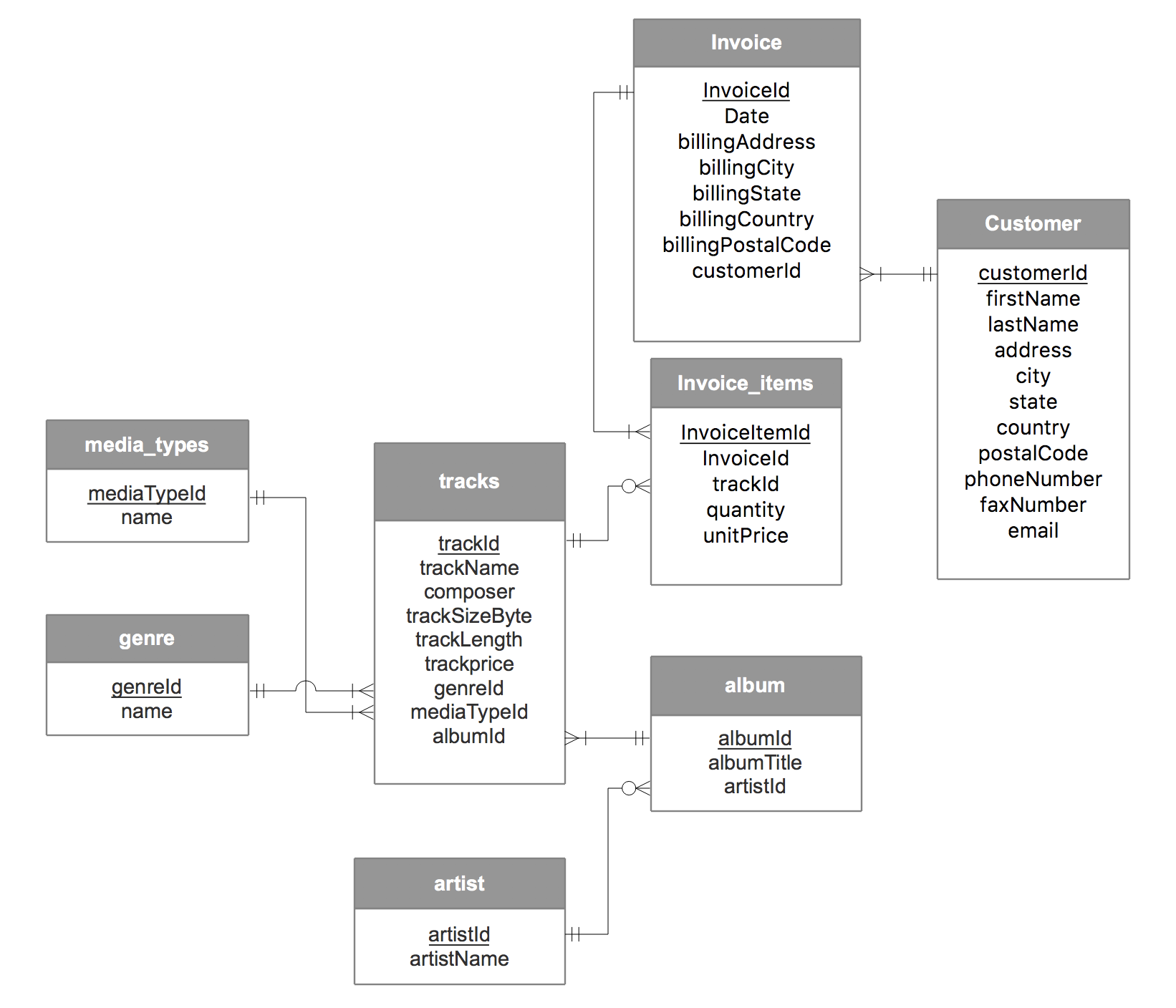
**RESULT**

|  |  |  |
| --- | --- | --- |
| **Country Feeling** | **15** | **Super Band** |
| **Carol Peacock Trio** | **11** | **Super Band** |
| **Caroline Coie Cuartet** | **11** | **Super Band** |
| **JV & the Deep Six** | **10** | **Regular Band** |
| **Modern Dance** | **10** | **Regular Band** |
| **Jim Glynn** | **9** | **Regular Band** |
| **Saturday Revue** | **9** | **Regular Band** |
| **Coldwater Cattle Company** | **8** | **Regular Band** |
| **Julia Schnebly** | **8** | **Regular Band** |
| **Topazz** | **7** | **Support Band** |
| **Jazz Persuasion** | **7** | **Support Band** |
| **Susan McLain** | **6** | **Support Band** |
| **Katherine Ehrlich** | **0** | **Amateur Band** |

**Online Music Store Database (40 points + 10 bonus points)**

1. **(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:
2. Artist names, customer last names and invoice IDs are unique
3. Each artist can have zero or more albums
4. Each album is made by exactly one artist
5. 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)
6. Each album has at least one track
7. Each track belongs to exactly one genre
8. Each genre is represented by at least one track
9. Each track is stored in exactly one media type
10. Each media type is used by at least one track
11. Each invoice item is for exactly one track and part of exactly one invoice
12. Each invoice has at least one invoice item
13. Some tracks may have never been sold (so there are no invoices for them)
14. Each invoice is issued to exactly one customer
15. The following columns always have a value: 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.

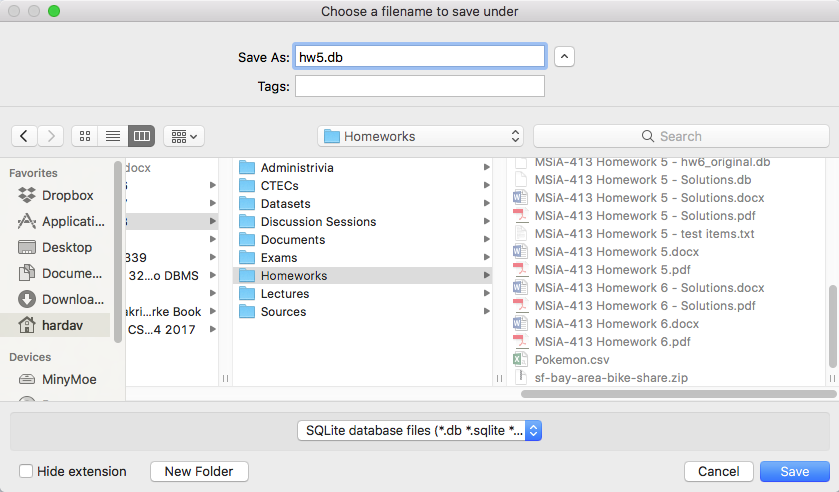
The corresponding normalized database diagram is given below:

****

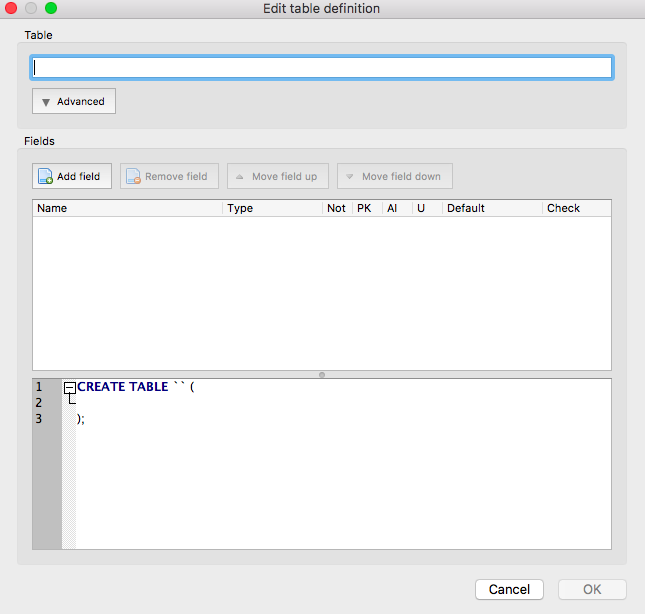
We now want to create a SQLite database that implements the normalized database. The first step is to create a new database with just one table that has the same data as the CSV file. Here is how to do that:

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

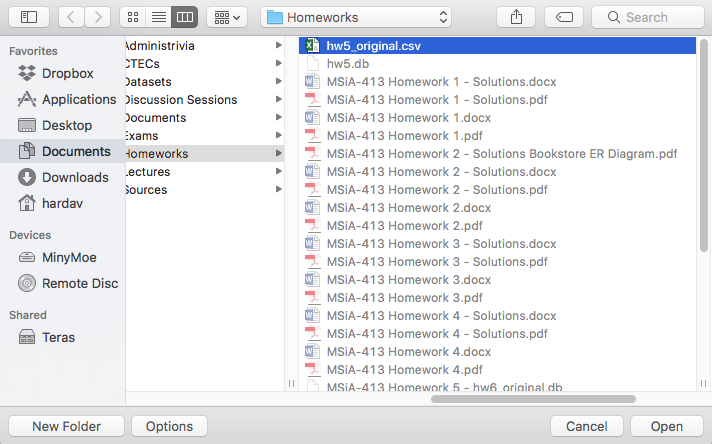
Start DB Browser for SQLite. Create a new database (File 🡪 New Database). A window will pop up like the one below. Specify a name and path to your new database file and click “save” to create the file.



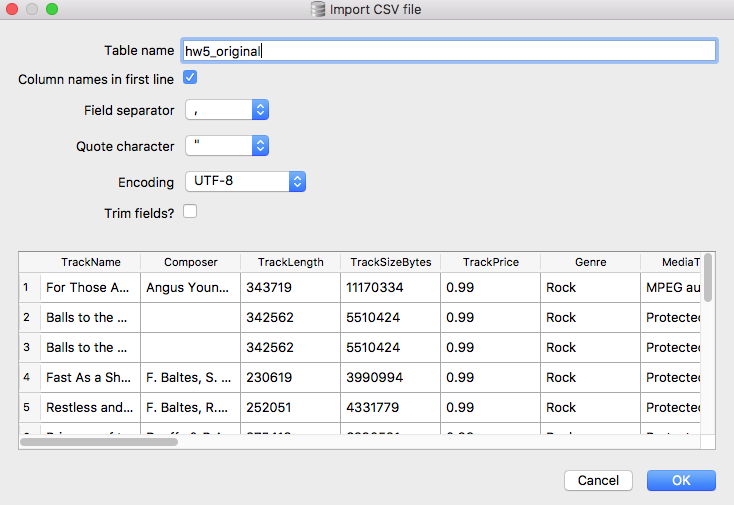
Then, another window will pop up like the one below that asks for a table definition. Click cancel.



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 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.



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). Please leave it in your database for debugging and grading purposes.

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.
* When you import a row from a CSV file into a table, an empty column may have as a value the empty string. An empty string is different than a NULL string. This means that if you want to check whether a field has a value, you may need to compare against the empty string and not against a NULL value (i.e., instead of writing WHERE customerLastName IS NOT NULL you may need to write WHERE customerLastName != "").

As a starting point, below are the SQL commands to create the table for media types. Use these SQL statements as the first statements in your answer, and proceed with creating and populating the remaining tables. We assume that the imported data from the CSV file were stored in a table with name hw5\_original.

**Notes:**

* **Added some buffer on the max lengths for data types to accommodate potential future data insertions**
* **Broke out steps for each table based on complexity (simple tables have steps combined, larger tables have more sections in their creation/population)**

**CREATE TABLE media\_types ( mediaTypeId INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,**

**mediaName nvarchar(20));**

**​**

**CREATE TABLE genre ( genreId INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,**

**genreName nvarchar(25));**

**​**

**CREATE TABLE artist ( artistId INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,**

**artistName nvarchar(90) UNIQUE);**

**​**

**CREATE TABLE Customer (customerId INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,**

**firstName nvarchar(20) NOT NULL,**

**lastName nvarchar (20) NOT NULL UNIQUE,**

**address nvarchar(50),**

**city nvarchar(25),**

**state nvarchar(10),**

**country nvarchar(20),**

**postalCode nvarchar(20),**

**phoneNumber nvarchar(40),**

**faxNumber nvarchar(40),**

**email nvarchar(40) NOT NULL);**

**​**

**​**

**CREATE TABLE Invoice (InvoiceId INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,**

**Date nvarchar(50) NOT NULL,**

**billingAddress nvarchar(50),**

**billingCity nvarchar(30),**

**billingState nvarchar(20),**

**billingCountry nvarchar(30),**

**billingPostalCode nvarchar(20),**

**customerId INTEGER NOT NULL REFERENCES Customer(customerId));**

**​**

**​**

**CREATE TABLE album (albumId INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,**

**albumTitle nvarchar(100) NOT NULL,**

**artistId INTEGER NOT NULL REFERENCES artist(artistId)**

**);**

**​**

**​**

**CREATE TABLE tracks (trackId INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,**

**trackName nvarchar(130) NOT NULL,**

**composer nvarchar(200),**

**trackSizeByte INTEGER,**

**trackLength INTEGER NOT NULL,**

**trackprice REAL NOT NULL,**

**genreId INTEGER NOT NULL REFERENCES genre(genreId),**

**mediaTypeId INTEGER NOT NULL REFERENCES media\_types(mediaTypeId),**

**albumId INTEGER NOT NULL REFERENCES album(albumId),**

**UNIQUE (trackName, trackLength)**

**);**

**​**

**​**

**​**

**CREATE TABLE Invoice\_items (InvoiceItemId INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,**

**InvoiceId INTEGER REFERENCES Invoice(InvoiceId),**

**trackId INTEGER NOT NULL REFERENCES tracks(trackId),**

**quantity INTEGER NOT NULL,**

**unitPrice REAL NOT NULL**

**);**

**​**

**​**

***-- ^^^^^^^^^^ INSERTING VALUES ^^^^^^^^^***

**INSERT INTO media\_Types (mediaName)**

**SELECT DISTINCT MediaType FROM hw5\_original;**

**​**

**INSERT INTO genre (genreName)**

**SELECT DISTINCT Genre FROM hw5\_original;**

**​**

**INSERT INTO artist(artistName)**

**SELECT DISTINCT ArtistName FROM hw5\_original;**

**​**

**INSERT INTO Customer(firstName, lastName, address, city, state, country,postalCode, phoneNumber, faxNumber,email)**

**SELECT DISTINCT CustomerFirstName, CustomerLastName, CustomerAddress, CustomerCity, CustomerState, CustomerCountry, CustomerPostalCode, CustomerPhone, CustomerFax, CustomerEmail**

**FROM hw5\_original WHERE CustomerFirstName IS NOT NULL;**

**​**

**INSERT INTO album(albumTitle, artistId)**

**SELECT DISTINCT HW*.AlbumTitle*, A*.artistId***

**FROM hw5\_original AS HW JOIN artist AS A ON A*.artistName* = HW*.ArtistName*;**

**​**

**INSERT INTO Invoice(Date, billingAddress, billingCity, billingState, billingCountry, billingPostalCode, customerId)**

**SELECT DISTINCT HW*.InvoiceDate*, HW*.InvoiceBillingAddress*, HW*.InvoiceBillingCity*, HW*.InvoiceBillingState*, HW*.InvoiceBillingCountry*, HW*.InvoiceBillingPostalCode*, C*.customerId***

**FROM Customer AS C JOIN hw5\_original AS HW ON C*.lastName* = HW*.CustomerLastName*;**

**​**

**INSERT INTO tracks(trackName, composer,trackSizeByte, trackLength, trackprice, genreId, mediaTypeId, albumId)**

**SELECT DISTINCT HW*.TrackName*, HW*.Composer*, HW*.TrackSizeBytes*, HW*.TrackLength*, HW*.TrackPrice* , genre*.genreId*, media\_types*.mediaTypeId*, album*.albumId***

**FROM hw5\_original AS HW JOIN album ON album*.albumTitle* = HW*.AlbumTitle* JOIN genre ON genre*.genreName* = HW*.Genre* JOIN media\_types ON media\_types*.mediaName* = HW*.MediaType***

**;**

**​**

**INSERT INTO Invoice\_items(InvoiceId,trackId,quantity,unitPrice)**

**SELECT DISTINCT I*.InvoiceId*, T*.TrackId*, H*.InvoiceItemQuantity*, H*.InvoiceItemUnitPrice***

**FROM hw5\_original AS H**

**JOIN Customer as C**

**ON (H*.CustomerLastName* = C*.lastName*)**

**JOIN Invoice as I**

**ON (H*.InvoiceDate* = I*.date*) AND (H*.InvoiceBillingAddress* = I*.billingAddress*) AND (I*.customerId* = C*.CustomerId*)**

**JOIN Tracks as T**

**ON (H*.trackName* = T*.trackName*) AND (H*.trackLength* = T*.trackLength*);**

1. **(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 question (8).

**Collaboration note: Checked answer with Kris N. and Parth**

﻿

**Query:**

**SELECT SUM(I.quantity \* I.unitPrice) as Total\_Sales, A2.artistName**

**FROM invoice\_items as I**

**JOIN tracks as T**

**ON I.trackId = T.trackId**

**JOIN album as A1**

**ON T.albumId = A1.albumId**

**JOIN artist as A2**

**ON A1.artistId = A2.artistId**

**GROUP BY A2.artistId**

**ORDER BY Total\_Sales DESC**

**LIMIT 1;**

**Output:**

|  |  |
| --- | --- |
| **138.6** | **Iron Maiden** |

1. **(10 bonus points)** Instead of creating and loading the database through the GUI as explained in Problem (8) Part A, write your own program to do it. Your program can be written in any language (e.g., C, C++, Java, Python, Pearl). 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.

There are several tutorials online on the programming interfaces supported by SQLite. For example, at the bottom of <http://www.sqlitetutorial.net/> you can find tutorials for SQLite interfaces with Python, Java, PHP and node.js. Similarly, at the bottom of the left pane at <https://www.tutorialspoint.com/sqlite/> you can find tutorials of SQLite interfacing with C, C++, Java, Python, Pearl, and PHP. There are many other tutorials online that expand the choices to other programming languages (e.g., R, Matlab, ML, etc). Feel free to find a tutorial that best fits your needs and utilize it.

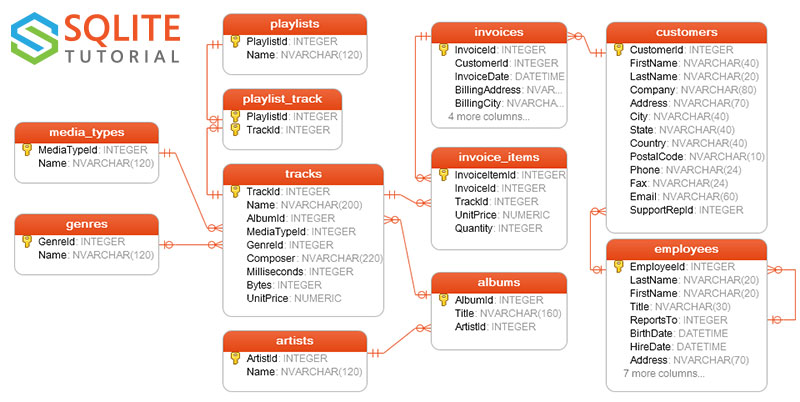
To receive full credit in this bonus problem you must submit your answer to this problem as a tarball (e.g., .tar, .zip, .gz or .bz2 file) with your program’s source code modules, and instructions on how to install the necessary libraries, how to compile your code and how to run it. You must also provide pointers to the tutorial(s) or any other online sources you used.

Because of the free choice of programming languages we give you, **we will not provide TA/instructor support for the bonus problem on all occasions**. Remember this is a bonus problem, not a required one. You can get full credit without answering Problem 10.

**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 8-10. 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/**](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.**