

ESD 40.011
Activity 1
Relational Database Concepts
Edited for SQLiteStudio 3 by L. Seligman, V. Sung, and P. Frazier
Edited for SQLiteStudio 3.2.1 By SQ, Gao, Y. Xu

Contents

Basic Concepts	1
Getting Started	2
Relational Database Design.....	2
Unique Table Identifiers.....	3
Using the <i>SELECT</i> Query.....	4
The purpose of the <i>SELECT</i> Query:.....	4
Being selective with the <i>SELECT</i> query:.....	6
Being selective in what fields we view:	6
Being selective in what records we view using the WHERE clause:	6
Sorting data:	7
Using the <i>SELECT</i> query as a calculator:	7
Using other useful functions:.....	8
Using Executable Queries	9
Create a new table using <i>CREATE TABLE ... AS</i> :.....	9
Delete records with the <i>DELETE</i> query:.....	10
Change individual records with the <i>UPDATE</i> query:.....	10
Creating Tables and Manipulating Records.....	10
Define a table and enter data manually:	11
Import data from Microsoft Excel:	12
Copying data to a spreadsheet:.....	13
Review	13

Basic Concepts

You have been introduced to the following concepts:

- Table (data presented in rows and columns)
- Database table (data presented in rows using fixed, pre-specified columns)
- Fields (columns of a database table, with pre-specified attributes)

- Records (rows of data in a database table)
- Data types (typical field types: text, integer, char, single-precision, double-precision, etc.)
- Primary Key (a collection of fields that uniquely define a record)

You should be provided with the following files:

- Chinook_Sqlite.sqlite
- MediaType.csv

You should have completed the exercise called “Get Ready for...SQLite”

Getting Started

1. To receive credit for this activity, complete the activity response document and turn it in. Blanks in this document have matching entries in the activity response document.
2. Create a directory to store your work. If you use the lab machine, be aware that the directory (and all your work) will disappear when the machine is rebooted. We recommend using a USB flash-drive.
3. Open SQLiteStudio 3.2.1.
4. Open an existing database by selecting the option “Add a database” under the “Databases” tab. Download and select the file “Chinook_Sqlite.sqlite” from blackboard. Click on the database, and click the “Connect to the database” icon as outlined in the screenshot below (If the icon is in grey, it indicates that the databased has been connected automatically when downloaded).



Relational Database Design

In a relational database, you should store information in the most efficient way possible by minimizing the duplication of information. For example, if you are logging phone conversations with a customer, you could log his name as “James Bond”, “JBond”, or “Bond, James”. When it comes time to analyze the log, you will have difficulty identifying all entries corresponding to your communication with me, and the database will not recognize all these entries as communication with the same customer. It would be better to have a separate table of customers with unique identifiers for each customer. This way, I would be in the table of customers exactly once, say as CustomerID: 007, Name: Bond, James. This way, in the log file, every time you talk to me, you log it as a conversation with Customer ID 007.

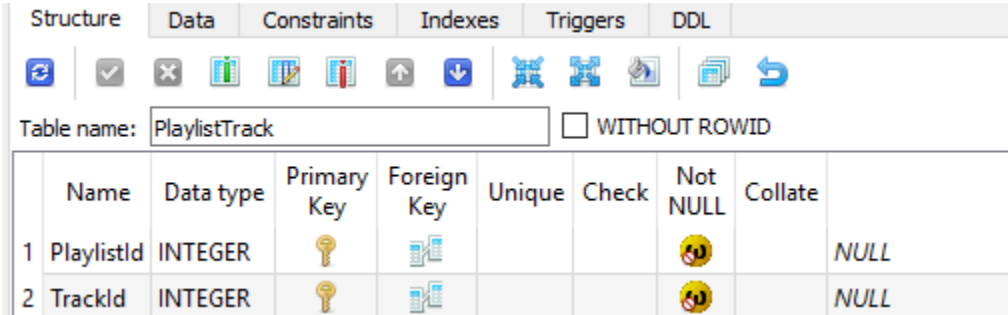
The database created today uses this approach. There are separate tables to describe album, artist, customer, employee, genre, invoice, media type, track etc. Each row in each of these tables has a unique identifier: If you double click a table, you will be brought to the “Structure” view, which would show the **PRIMARY KEY** field.







Unique Table Identifiers

Click on “Tables” to view the tables in this database. You can double click on a table title to view more details about the contents of this table.

1. What is the unique identifier for table “Playlist”? _____
2. What is the unique identifier for table “Track”? _____

Note that the table “PlaylistTrack” refers to data from these tables by these unique identifiers. Double click the table “PlaylistTrack” table to view its structure. Make sure you are viewing the table in the “Structure” tab, as shown in the screen shot below.



	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate
1	PlaylistId	INTEGER						NULL
2	TrackId	INTEGER						NULL

Fields with the “Foreign Key” icon refer to data in other tables. It is good practice in database design to indicate which fields connect to Foreign keys. Double click on a field to view which table it is accessing information from. (Hint: Click “Configure” to view from where this data is being pulled.)

3. List the fields in “PlaylistTrack” that refer to data from other tables, and list the tables where these data are found. (For example, the field “PlaylistId” refers to the unique identifier “PlaylistId” in the table “Playlist”).

4. In the table “Track”, list the fields that refer to data from other tables, and list the tables where these data are found.

Commentary:

1. As a general principle in database design, you will assign a data item to exactly one table. If you need to store a customer’s phone number, make a table “Customers” where each customer corresponds to a row in the table, and include a field in that table to store the phone number. Don’t put the phone number for this customer in any other table.
2. You might think it would be nice to have the customer’s phone number in the order file so that when you look at an order you can view this data, but this way you would store the number in multiple places. If the customer changes their phone number, you would have to correct it in multiple places in your database. Instead, allow other tables to have fields that refer to the unique identifier in the “Customers” table to acquire a customer’s telephone number.

Using the *SELECT* Query

The purpose of the *SELECT* Query:

Defn: The *SELECT* query is the basic tool for getting the data you need from a relational database. Whether you are using SQL for data mining, data manipulation, data analysis, or data reporting, your work starts with the *SELECT* query.

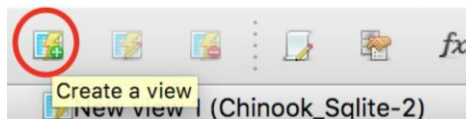
1. Let's create our first query. We want to create a query that would allow us to generate a view of the entries of the "Invoice" table. We can either create this query using the SQL query editor (which can be opened on the top "tool" menu bar), or by creating a view. We will create a view.

*Note the difference between a **query** and a **view**.*

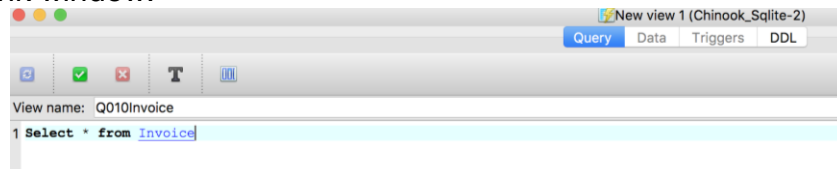
Defn: A *query* is SQL code executed to pull rows from our database.

Defn: A *view* is a saved query. By making a query into a view, we can refer to its results in other queries.

2. Click on the "Create a view" icon in the toolbar.



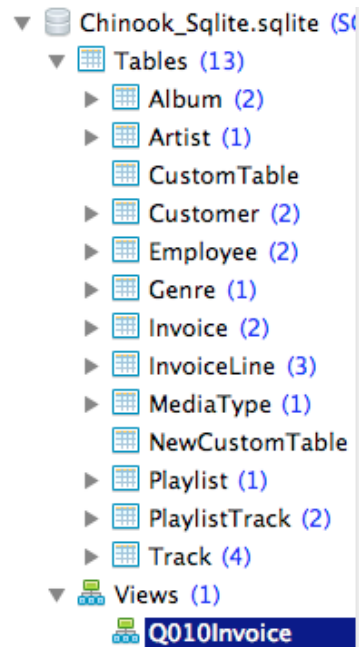
3. Let's name the view "Q010Invoice" and enter the following SQL code in the blank window.



Note the syntax used in the SQL code.

The **asterisk** "*" used in the *SELECT* query gets all the rows and columns from the table specified by the *FROM* query. In other words, the above SQL code "selects all rows and columns from the table called 'Invoice'." Although SQL commands like "SELECT" and "FROM" are often written in all caps, SQL does not require this, and your query would run if you used different capitalization.

4. Click green checkmark to "Commit the view changes".
5. Click "Q010Invoice" under "Views" in the left pane of the window. Whenever you create and save views, they will be listed here.



6. Look at the view generated by this query.

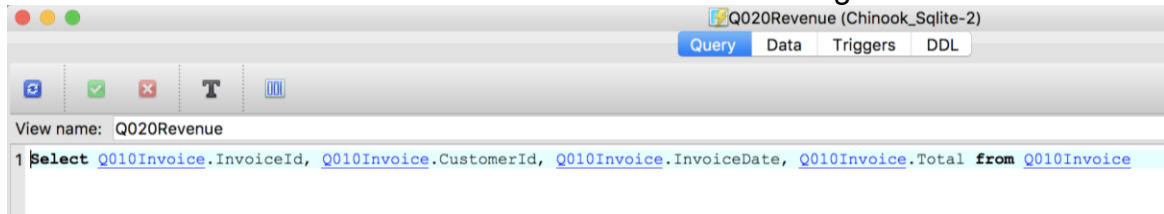
InvoiceId	CustomerId	InvoiceDate	BillingAddress	BillingCity	BillingState	BillingCountry	BillingPostalCode	Total
1	1	2009-01-01 00:00:00	Theodor-Heuss-Straße 34	Stuttgart	NULL	Germany	70174	1.98
2	2	2009-01-02 00:00:00	Ullévalsvæien 14	Oslo	NULL	Norway	0171	3.96
3	3	2009-01-03 00:00:00	Grétrystraat 63	Brussels	NULL	Belgium	1000	5.94
4	4	2009-01-06 00:00:00	8210 111 ST NW	Edmonton	AB	Canada	T6G 2C7	8.91
5	5	2009-01-11 00:00:00	69 Salem Street	Boston	MA	USA	2113	13.86
6	6	2009-01-19 00:00:00	Berger Straße 10	Frankfurt	NULL	Germany	60316	0.99
7	7	2009-02-01 00:00:00	Barbarossastraße 19	Berlin	NULL	Germany	10779	1.98
8	8	2009-02-01 00:00:00	8, Rue Hanovre	Paris	NULL	France	75002	1.98
9	9	2009-02-02 00:00:00	9, Place Louis Barthou	Bordeaux	NULL	France	33000	3.96
10	10	2009-02-03 00:00:00	3 Chatham Street	Dublin	Dublin	Ireland	NULL	5.94
11	11	2009-02-06 00:00:00	202 Hoxton Street	London	NULL	United Kingdom	N1 5LH	8.91
12	12	2009-02-11 00:00:00	Theodor-Heuss-Straße 34	Stuttgart	NULL	Germany	70174	13.86
13	13	2009-02-19 00:00:00	1600 Amphitheatre Parkway	Mountain View	CA	USA	94043-1351	0.99
14	14	2009-03-04 00:00:00	1 Microsoft Way	Redmond	WA	USA	98052-8300	1.98
15	15	2009-03-04 00:00:00	1 Infinite Loop	Cupertino	CA	USA	95014	1.98
16	16	2009-03-05 00:00:00	801 W 4th Street	Reno	NV	USA	89503	3.96
17	17	2009-03-06 00:00:00	319 N. Frances Street	Madison	WI	USA	53703	5.94
18	18	2009-03-09 00:00:00	194A Chain Lake Drive	Halifax	NS	Canada	B3S 1C5	8.91
19	19	2009-03-14 00:00:00	8, Rue Hanovre	Paris	NULL	France	75002	13.86
20	20	2009-03-22 00:00:00	110 Raeburn Pl	Edinburgh	NULL	United Kingdom	E4 1HH	0.99
21	21	2009-04-04 00:00:00	421 Bourke Street	Sydney	NSW	Australia	2010	1.98
22	22	2009-04-04 00:00:00	Calle Lira, 198	Santiago	NULL	Chile	NULL	1.98
23	23	2009-04-05 00:00:00	3,Raj Bhavan Road	Bangalore	NULL	India	560001	3.96
24	24	2009-04-06 00:00:00	Ullévalsvæien 14	Oslo	NULL	Norway	0171	5.94
25	25	2009-04-09 00:00:00	Rua Dr. Falcão Filho, 155	São Paulo	SP	Brazil	01007-010	8.91
26	26	2009-04-14 00:00:00	1 Infinite Loop	Cupertino	CA	USA	95014	13.86
27	27	2009-04-22 00:00:00	5112 48 Street	Yellowknife	NT	Canada	X1A 1N6	0.99
28	28	2009-05-05 00:00:00	Rua da Assunção 53	Lisbon	NULL	Portugal	NULL	1.98
29	29	2009-05-05 00:00:00	Tauentzienstraße 8	Berlin	NULL	Germany	10789	1.98
30	30	2009-05-06 00:00:00	Barbarossastraße 19	Berlin	NULL	Germany	10779	3.96

7. SQL is a non-procedural language. In the days before SQL, you would use a procedural language like FORTRAN to open a file, read the records one at a time using a FOR loop and display or print them. In SQL, you specify what you want the result to look like and the looping is done for you automatically. With SQL, you end up thinking about your data at a more abstract level and you don't have to be a programmer to do useful things with data.

Being selective with the **SELECT** query:

Being selective in what fields we view:

1. One of the powerful features of SQL is that you can base SELECT queries on other saved SELECT queries (views) and in this way build up your queries in simple steps. In other words, we can *build queries on queries*.
2. We will demonstrate this by being selective in what fields we view. Let's be selective about what fields we display in our view "d."
3. We want to exclude the *BillingAddress*, *BillingCity*, *BillingState*, *BillingCountry*, and *BillingPostalCode* fields from our new view because we feel that they aren't relevant to our analysis. In other words, we want to select only the following fields: *InvoiceId*, *CustomerId*, *InvoiceDate*, *Total*.
4. Create a new view called "Q020Revenue" with the following SQL code:



Note that the general syntax used above is "Table/View"."Field". While this gives us the view that we want, there is a shortcut we could have used.

Because we are selecting these fields from only one table, and thus there is no ambiguity*, we can drop the prefix, which specifies which Table/View we are pulling data from. In other words, we could have used the language "SELECT InvoiceId, CustomerId... FROM Invoice"

**In the next recitation, you will create queries that pull data from multiple views and tables that have the same field names. Because of this ambiguity—SQLite won't know which table/view to retrieve the data from—you need to include this prefix.*

5. Edit "Q020Revenue" so that the prefixes are deleted from the field names.
 - a. Select "Q020Revenue" in the left pane where all the table and view names are listed.
 - b. In the "Query" field, change the view name.
6. View the results of your edited query. Your query should generate the same view as in Step 3.

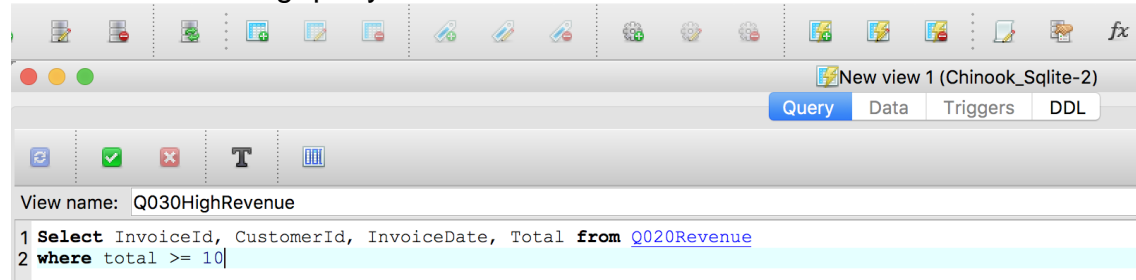
Being selective in what records we view using the **WHERE** clause:

Defn: The **WHERE** clause allows us to specify criteria that dictate which records to display. This clause must be written *after* the FROM clause.

1. Now, let's be selective in what records we retrieve. I want to see orders that generated higher revenue, say at least \$10. In other words, we want

to see records where the field "Total" had values greater than or equal to 10.

2. Create the following query.



3. How many of the invoices satisfy this criteria? _____
4. Furthermore, we want to see only orders with revenues that are at least \$10 and were placed in the year 2013 or later. We want to specify a criterion for the *InvoiceDate* field. This is a bit tricky because SQL doesn't recognize "1/1/2013" as a date. We will need to use **DATETIME("2013-01-01 00:00:00")** as our criterion format.
5. Create a new query Q040RecentHighRevenue in which only invoices that were created in or after 2013 that generated at least \$10 in revenue are listed. (*Hint: build this query on Q030HighRevenue*).
6. Write the WHERE clause you used here:
WHERE _____
7. How many orders satisfy both the criteria (created in 2013 or later, and greater than \$10)?

Sorting data:

Defn: The **ORDER BY** clause allows records to be sorted by either descending or ascending order. This clause must come last in the SQL code. The syntax for this clause is:

ORDER BY FieldName **DESC/ASC**

Note: **DESC** indicates descending order, **ASC** indicates ascending order.

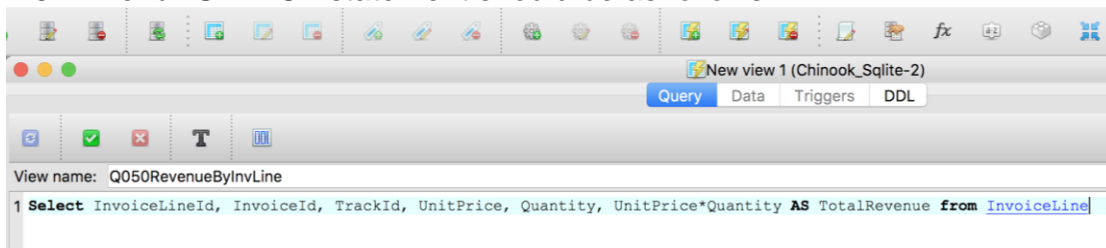
1. We want to view the records from "Q040RecentHighRevenue" that are most recently placed first.
2. Edit "Q040RecentHighRevenue" and write the resulting SQL code here:

Using the *SELECT* query as a calculator:

Defn: The **AS** keyword allows the user to specify the name of a calculated field in the SELECT clause. The syntax for this clause is:

SELECT Field1, Field2, EXPRESSION **AS** FieldName

1. In addition to selecting fields, we can use the SELECT clause to calculate new fields based on the other fields in the field list.
2. We want to create a new query called “Q050RevenueByInvLine” based on the “InvoiceLine” table to calculate how much revenue we generated from each invoice line based on the price and the quantity of each track the customer purchased.
3. Start writing the SQL code by indicating that we want to include the following fields from “U”: *InvoiceLineId*, *InvoiceId*, *TrackId*, *UnitPrice*, *Quantity*.
4. Extend the list of fields by putting a comma “,” after *Quantity* and typing “*UnitPrice*Quantity AS TotalRevenue*”.
5. The full SELECT statement should be as follows:



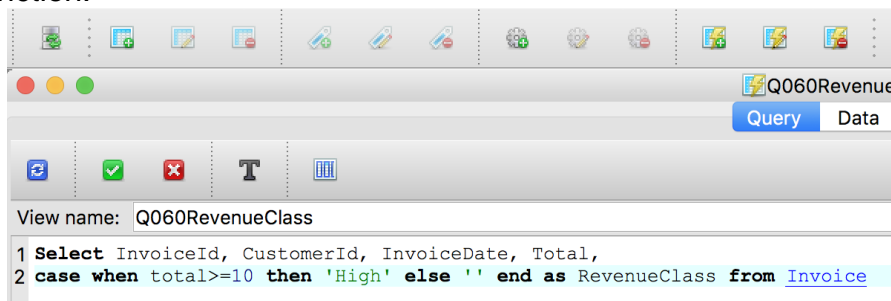
6. Look at the view generated by the query. Unfortunately, all the invoice lines required only a quantity of 1 of each TrackId, so this isn't the most interesting data to analyze.

Using other useful functions:

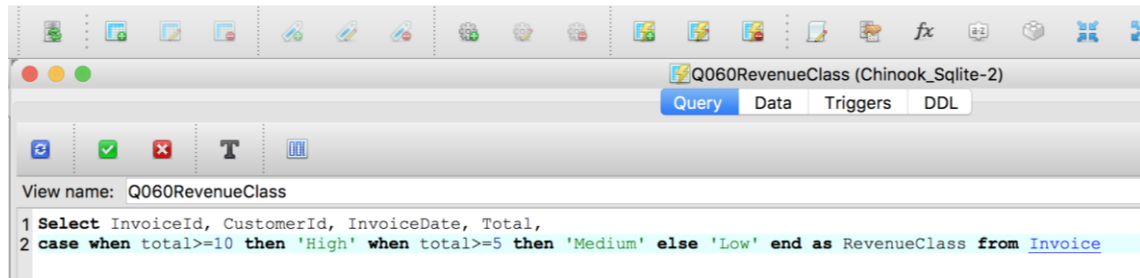
Defn: The **CASE WHEN** function is the IF-THEN-ELSE function in SQLite. The syntax for **CASE WHEN** is as follows:

CASE WHEN [condition] **THEN** [result if true] **ELSE** [result if false] **END**

1. We want to categorize the revenues listed in “Invoice” as “High” if they are at least \$10, “Medium” if they are at least \$5 or “Low.”
2. Let's first categorize only the “High” revenues. Create a new view “Q060RevenueClass” which lists the fields *InvoiceId*, *CustomerId*, *InvoiceDate*, *Total*, and a new field called *RevenueClass*, which will label our invoices as “High,” “Medium,” or “Low” using the CASE WHEN function.



3. In order to label our “Medium” and “Low” revenues, we will need to use a nested CASE WHEN function. In the view “Q060RevenueClass,” edit the ELSE statement in our CASE WHEN function as follows:



4. Create a query based on “Q060RevenueClass” that selects all fields but only records with *RevenueClass*='Medium.'
5. Write the SQL statement for this query here:

6. How many 'Medium' invoices are there? _____

Defn: The *IFNULL(x,y)* function checks if there is missing data in field x and replaces it with default value y if the value is null. Note: there is no change to the underlying table.

1. Create a query “Q070Customers” that checks for missing data in the field *Company* of the “Customer” table. If the field is null, use the phrase “Missing Name” in its place.
2. Write the SQL statement for this query here:

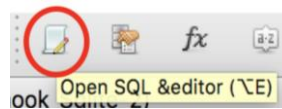
Using Executable Queries

Executable queries cannot be saved as views. Use the query editor (shown below) to create the queries. If you want to save executable queries, you will have to save them to separate files (SQL script files). But you can save sequences of queries that perform complex operations into SQL script files which can be opened and executed again later.

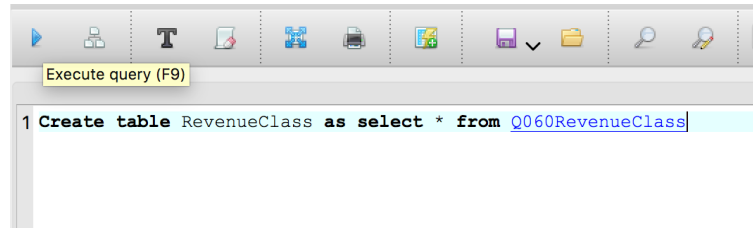
Create a new table using *CREATE TABLE ... AS*:

1. A query, without the results it generates, does not take up very much space, since it is only storing text. However, advanced queries on large datasets may take a long time to run and retrieve the resulting dataset. One way to speed things up is to store your results in a table, so that the query will not need to run every time you wish to view your results. This takes up more space than just storing the query, but it can be faster.
2. Let us now store the results of “Q060RevenueClass” into a table titled “RevenueClass” using an executable query.

- Click the circled icon shown in the screen shot below to “Open SQL & Editor”.



- Execute the following query by typing in the following SQL code and clicking on the blue “Execute query” icon outlined in the screen shot below:



- Make sure that this table was created correctly and populated with the dataset resulting from the query “Q060RevenueClass”. The table should have 412 rows.

Delete records with the *DELETE* query:

- Let’s delete all records from “Track” with a *Genre* of “Sci Fi & Fantasy”, or, equivalently, a *GenreId* of 20.
- View the contents of the “Track” table and note the number of rows.
- Open the SQL Query Editor by clicking on the pencil icon.
- Execute the following query to view the number of rows you should delete:

```
1 SELECT * FROM Track WHERE GenreId=20;
```

- Now execute the following executable query to delete the appropriate rows:

```
1 DELETE FROM Track WHERE GenreId=20;
```

- Re-open the “Track” table and make sure it has 26 fewer rows.

Change individual records with the *UPDATE* query:

- We can use an executable query to update the records in a table.
- Suppose we have decided that we do not like that there are two records in the “Playlist” table with name “Music”. We would like to change the name of Playlist 8 to “Music 2” to avoid confusion.
- Open the SQL Query Editor and execute the following querye

```
1 UPDATE Playlist SET Name="Music 2" WHERE
2 PlaylistId=8;
```

- Open the table “Playlist”. Verify that this change has been made correctly by viewing the data in grid view.

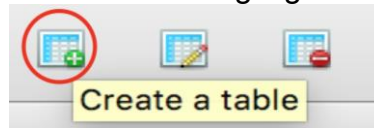
Creating Tables and Manipulating Records

We will consider two ways in which you can import data into your database:

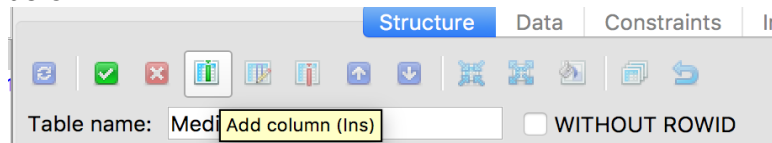
1. Define a table and enter data manually
2. Import data from Microsoft Excel

Define a table and enter data manually:

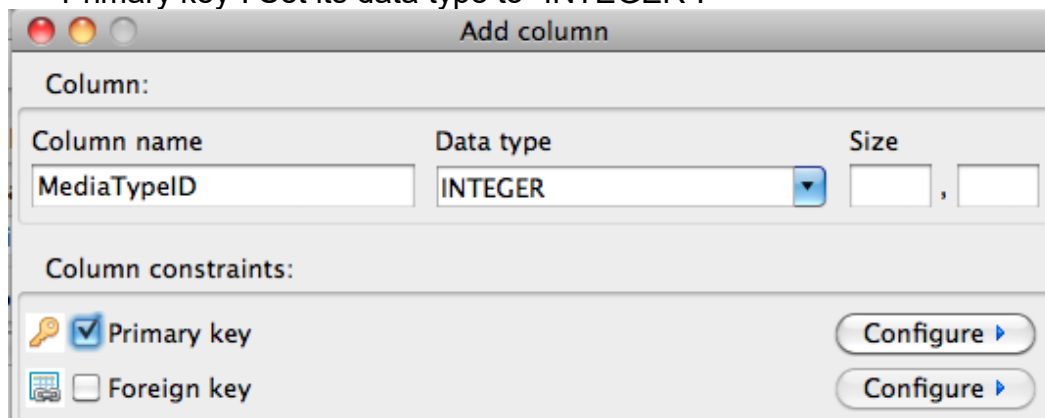
1. Click on the “Create a table” icon as highlighted in the screen shot below:



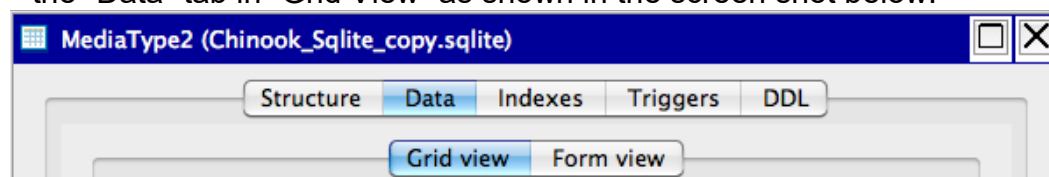
2. Name your table “MediaType2”. Add fields by clicking the “Add Column” button as below:



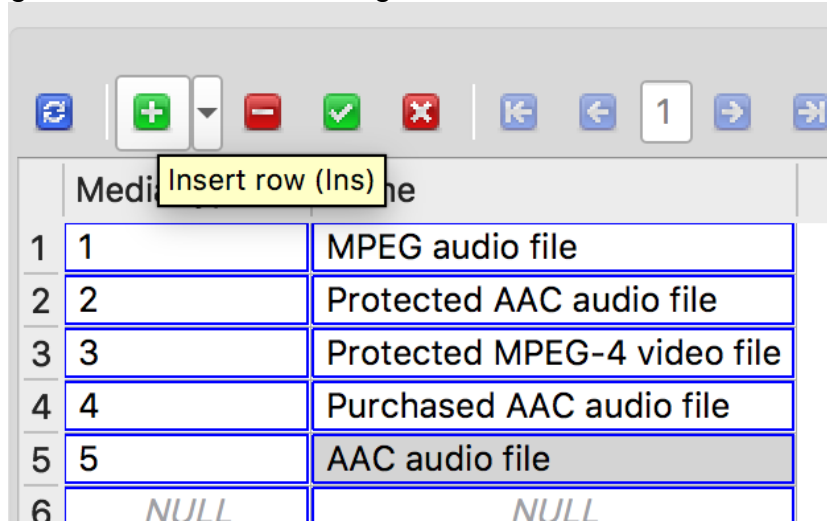
3. Create a column named “MediaTypeId” and check the box to make it the “Primary key”. Set its data type to “INTEGER”.



4. The other boxes (“Foreign Key”, “Unique”, “Not NULL” etc.) allow you to specify other properties you would like to require of the data in this column. For example, if you clicked “Not NULL”, the database would check that all every row entered in this table would have a value (i.e., would not be NULL). We will leave these other boxes unchecked. Go ahead and click “OK” to finish adding this column.
5. Create a column named “Name”, set its data type to “TEXT”, and click “OK”.
6. You do not need to add any constraints. Click the green checkmark to “Commit structure changes”.
7. Double click “MediaType2” under tables to view the contents of the table. The table should have no data in it. Make sure you are viewing the table in the “Data” tab in “Grid View” as shown in the screen shot below:



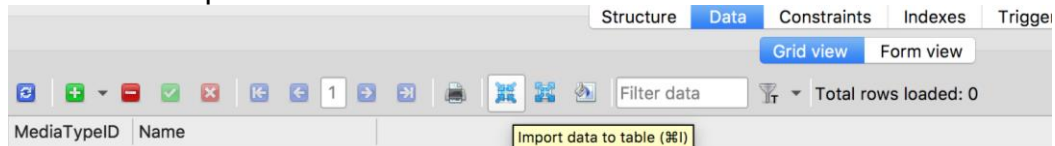
8. Add the following data to the table “MediaType2” by clicking the green plus sign icon as below and filling in the fields.



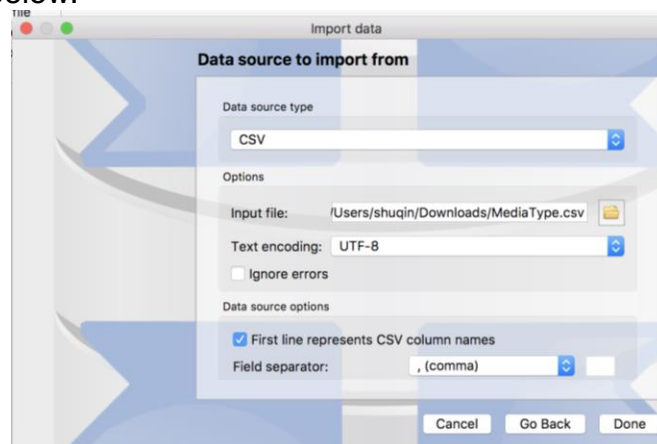
	Media	he
1	1	MPEG audio file
2	2	Protected AAC audio file
3	3	Protected MPEG-4 video file
4	4	Purchased AAC audio file
5	5	AAC audio file
6	NULL	NULL

Import data from Microsoft Excel:

1. We will now add the data from the “MediaType.csv” file you have been provided. Download this file and take note of the folder you put it in.
2. Delete the rows you just added by selecting each row, clicking the red minus sign, and then click the green checkmark to “Commit” the change. Complete this operation on every row.
3. Click the “Import data to table” icon circled in the screen shot below:



4. Make sure that the “Chinook_Sqlite.sqlite” database is selected and you are importing data to the “Existing” table “MediaType2”. Then Click the “Continue” icon.
5. Select the file to import, “MediaType.csv”. In the “data source options” field, check the box “First line represents CSV column names” as in the screen shot below.



6. Click “Done” and verify that the data is correctly imported. If the column names were incorrectly imported, delete this row.
7. Note that this same method of importing data would have worked if we had prepared the data using any other program that can create .csv files.

Copying data to a spreadsheet:

1. Open table “Invoice” and view the data in “Grid View”.
2. Click the “Export table” icon circled in the screen shot below:



3. Make sure the “Table” field is populated by “Invoice” and click “Continue”.
4. For the “Output” field, click the disk icon to select a folder to place your file, name the file in the “Save As” field and then “Save” them. In the “Export format options”, check the “Column names in first row” box. Then click “Done”
5. Find the file in the folder you specified in Step 4 and make sure it contains the desired data.

Review

1. Congratulations! You now know the basics of structured query language (SQL) as implemented by SQLiteStudio. You can now pick up and learn other database packages that use SQL with this basic knowledge.
2. At this stage, you are familiar with the following keywords:
 - **SELECT**
 - **FROM**
 - **WHERE**
 - **ORDER BY**
 - **AS**
 - **CASE WHEN...THEN...ELSE...END**
 - **IFNULL**
 - **CREATE**
 - **DELETE**
 - **UPDATE**
 - **SET**