# CST-345 Activity 4 Guide

## Contents

# Part 1 - Tutorial

## Objectives

1. Review the concepts taught in previous lessons.

## Instructions

1. Sign into your account in SoloLearn.
2. Join the Introduction to SQL course.
3. Complete the section titled "Challenges."
4. Take a screenshot of the progress report when you finish the section. Paste the screenshot into a Microsoft Word document for credit.

## Deliverables

Submit a Word document with a screenshot showing the completion status of the lesson. You may combine the deliverables from all parts of this activity in a single Word document.

---

# Part 2 – Database Design Concepts

## Objectives

1. Execute a JOIN statement.
2. Write SQL select statements that satisfy business conditions.
3. Write SQL select statements that summarize a data set.
4. Redesign a set of related tables in order to conform to first, second, and third normal form design principles.

## Instructions

1. Read the required textbook readings for this topic.
2. Write select statements using the antique book store database (from Chapter 16) to produce the following data sets:
    a. Use a select statement to fetch the maximum, minimum, and average: sale_total_amount from the sale table.
    b. Start with the following select statement, which will return all of the book data for items that were sold.

```
SELECT work.title, book.isbn, volume.sale_id, volume.selling_price
FROM  work
JOIN book ON book.work_numb = work.work_numb
JOIN volume ON book.isbn = volume.isbn
```

3. Modify the statement to return only the sales that were above the average selling price.
4. Modify the following select statement, which returns all of the sales and shows the name of the customer.

```
SELECT customers.customer_numb,
```

```
    customers.first_name,
    customers.last_name,
    sale.sale_id,
    sale.sale_total_amt
FROM customers
JOIN sale ON customers.customer_numb = sale.customer_numb
```

| customer_numb | first_name | last_name | sale_id | sale_total_amt |
|---|---|---|---|---|
| 1 | Janice | Jones | 1 | 510 |
| 1 | Janice | Jones | 2 | 125 |
| 1 | Janice | Jones | 3 | 58 |
| 4 | Jane | Doe | 4 | 110 |
| 6 | Janice | Smith | 5 | 110 |
| 12 | Franklin | Hayes | 6 | 505 |
| 8 | Helen | Jerry | 7 | 80 |
| 5 | Jane | Smith | 8 | 90 |
| 8 | Helen | Jerry | 9 | 50 |
| 11 | Edna | Hayes | 10 | 125 |
| 9 | Mary | Collins | 11 | 200 |
| 10 | Peter | Collins | 12 | 200 |
| 2 | Jon | Jones | 13 | 25.95 |
| 6 | Janice | Smith | 14 | 80 |
| 11 | Edna | Hayes | 15 | 75 |
| 2 | Jon | Jones | 16 | 130 |
| 1 | Janice | Jones | 17 | 100 |
| 5 | Jane | Smith | 18 | 100 |
| 6 | Janice | Smith | 19 | 95 |
| 2 | Jon | Jones | 20 | 75 |

Figure 1 *Results of selecting all sales. Some customers, such as Janice Jones, have more than one sale.*

5. Modify the select statement to show each customer ID and first and last name with **a total amount (sum) of money** that the customer has spent at the store.

| customer_numb | first_name | last_name | sum(sale.sale_total_amt) |
|---|---|---|---|
| 1 | Janice | Jones | 793 |
| 2 | Jon | Jones | 230.95 |
| 4 | Jane | Doe | 110 |
| 5 | Jane | Smith | 190 |
| 6 | Janice | Smith | 285 |
| 8 | Helen | Jerry | 130 |

Figure 2 *Results of select statement that sums the entire amount of money each customer spent. Notice that on line one of the output, Janice Jones spent a total of $793 on all her sales.*

6. View "Learn Database Normalization - 1NF, 2NF, 3NF, 4NF, 5NF."
7. Create the following in a Microsoft Word document.
    a. Give an example of a table that violates the "First Normal Form" principle. You may invent your own example or find an example online. Cite your reference if you

borrow someone else's work. Draw a Logical Entity Resource diagram (define table names and column names) of the database. Explain why it is incorrect. Show a corrected version of the ER diagram. Explain what problem the correction solves.

b. Give an example of a database design that violates the "Second Normal Form" principle. You may invent your own example or find an example online. Cite your reference if you borrow someone else's work. Draw a Logical Entity Resource diagram of the database. Explain why it is incorrect. Show a corrected version of the diagram. Explain what problem the correction solves.

c. Give an example of a database that violates the "Third Normal Form" principle. You may invent your own example or find an example online. Cite your reference if you borrow someone else's work. Draw a Logical Entity Resource diagram of the database. Explain why it is incorrect. Show a corrected version. Explain what problem the correction solves.

### Deliverables

Submit a Word document that contains:

1. The queries used to solve each of the statements in this section. Show both the SQL statements and the output.
2. The three examples of database normalization errors, the explanations for why each is designed incorrectly, and a solution for each.

---

## Part 3 - Build a Music App

### Objectives:

1. Write SQL select statements that satisfy business conditions.
2. Create a GUI front end for a SQL database.
3. Connect an application to a SQL database.

### Foreign Key

In this section, we are going to add another table to the database. The second table will store the track information. A track is an individual song on an album.

Here is a preview of the database that we are going to see.

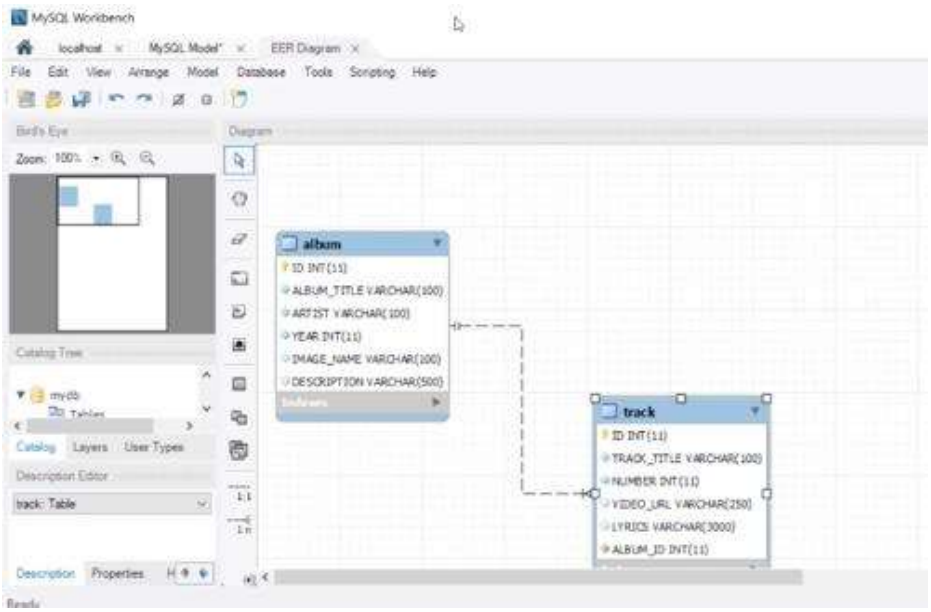


Figure 3 *Foreign key relationship between two tables.*

To create a second table and connect them, we are going to use the program MySQL workbench.

## About MySQL Workbench

MySQL workbench does not contain a database. It is simply a client for the MAMP server. The Workbench tool has administrative and design features that make it very easy to manage a database in MySQL. For workbench to work correctly, MAMP must be running.


Figure 4 *MAMP main screen.*

MySQL workbench allows you to create new connections.

1. On the main screen, click the very small + next to the words MySQL connections.


Figure 5 *Adding a new connection with MySQL workbench.*

2. Recall that the settings for the database have a specific port number.


Figure 6 *Port number being used in MAMP.*

3. When you create a connection in workbench, you must specify the host name password and port.

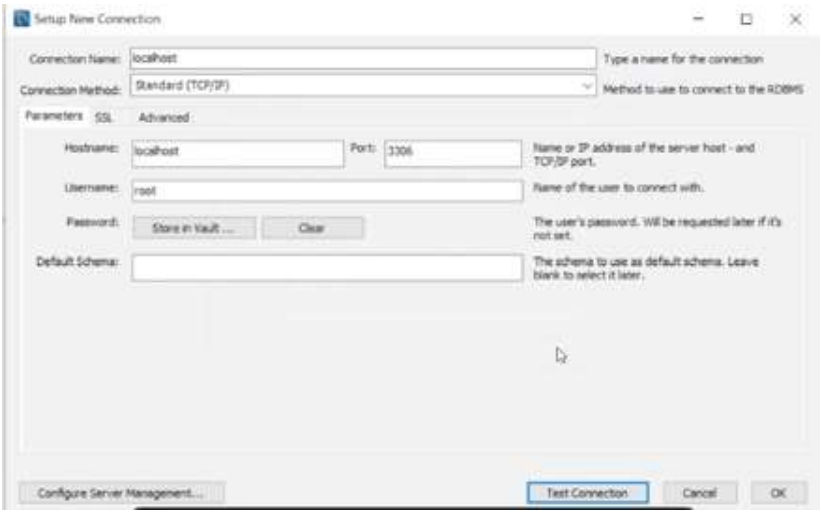


Figure 7 *Setting up a new connection in MySQL Workbench.*

4. Click the test connection button to see if the connection is valid.

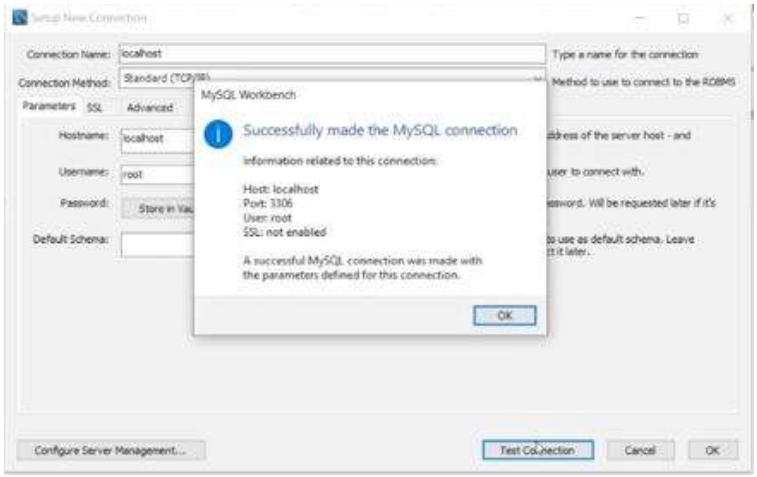


Figure 8 *Successful connection to MySQL message.*

5. Click OK to save the connection.
6. Click on the Gray square and the connection will be made. You should see the database schemas that are configured in MAMP.


Figure 9 *Saved connection.*

7. Expand each branch to see the tables and column names.


Figure 10 *Music2 schema shown in the navigator.*

8. Next, we will import the data and structure of the database. Under the database menu, choose Reverse Engineer.

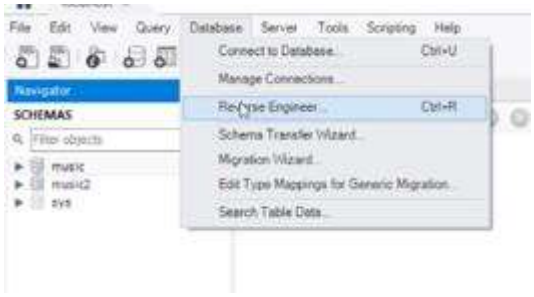


Figure 11 *Reverse engineer option.*

9. You will be asked to connect to the database.

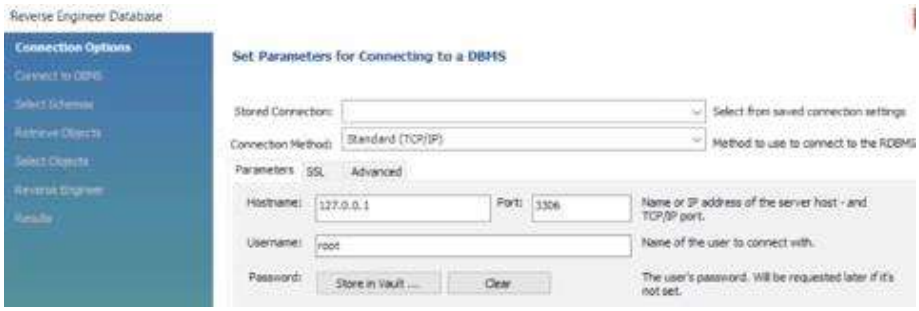


Figure 12 *Connecting to MySQL.*

10. Select the music2 schema.

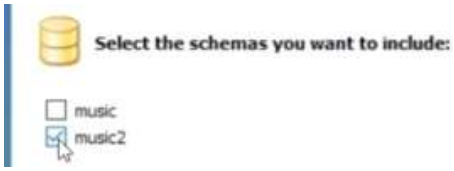


Figure 13 *Selecting Music2.*

11. Click through a few screens until you see the albums table appear on a grid. This grid is used to assemble the tables and relate them to each other.

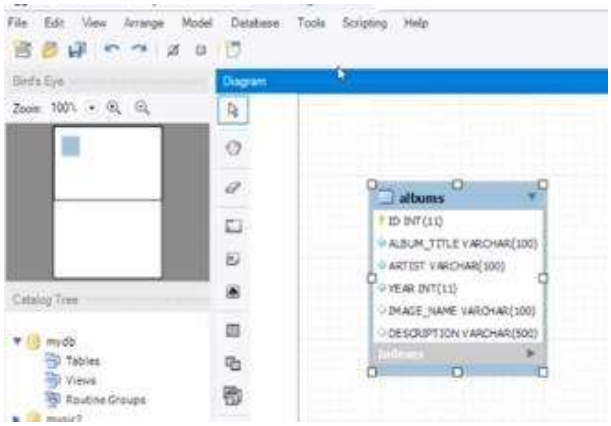


Figure 14 *The music2 database currently has one table.*

12. Let's create a new table. Click the new table icon on the left side of the screen.

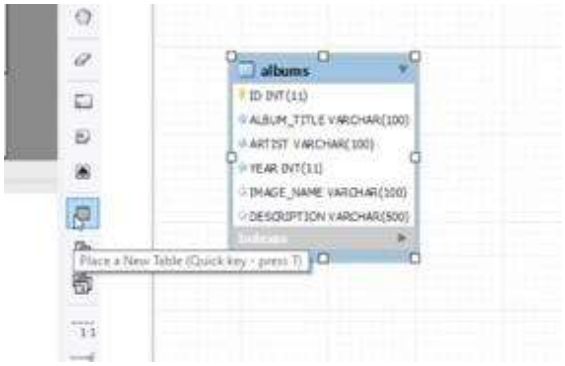


Figure 15 *Adding a new table.*

13. Table1 should now appear on the grid. Double-click Table1 to edit it.


Figure 16 *New table added but not defined.*

14. Rename the table to tracks.
15. Add the following columns to the table. Notice that ID is listed as auto increment and a primary key.

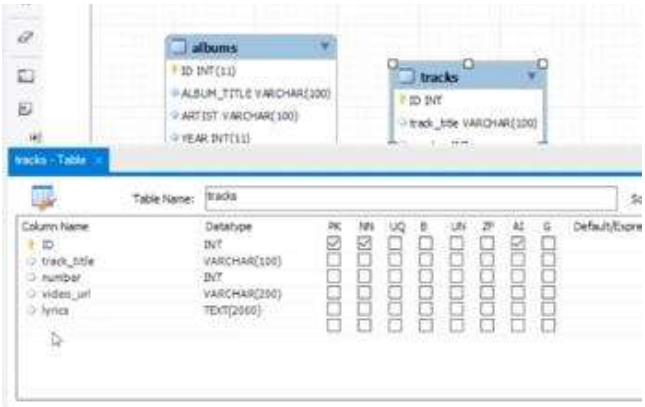


Figure 17 *Renaming and adding columns to the tracks table.*

16. Connections are made between the tables using relationships. Select the one to many relationship and then click on each table. This is a three-click operation: First, click the link icon; second, click one of the tables; third, click the final table.


Figure 18 *Selecting the one to many button.*

## First Attempt

You may do the order backwards by mistake. The following picture shows that the tables were linked **incorrectly**. This relationship shown is backwards. It shows that each track has many albums. The relationship should be that **each album has many tracks**.

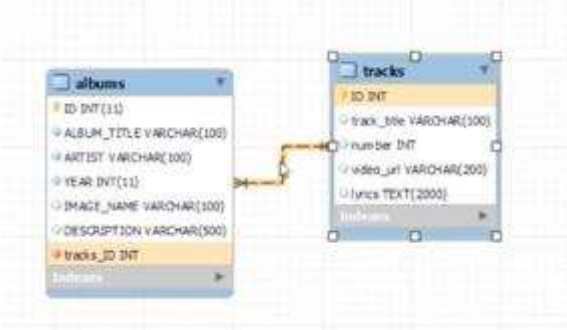


Figure 19 *Foreign key created backwards.*

1. If you created this relationship backwards, then right-click on the link and choose delete.

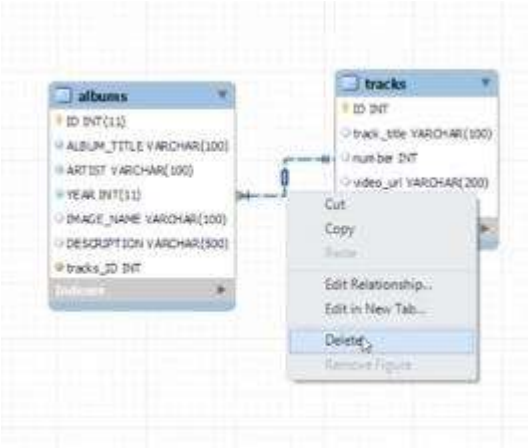


Figure 20 *Deleting the incorrect foreign key.*

2. Recreate the relationship using the opposite order. This picture shows the correct relationship between albums and tracks.

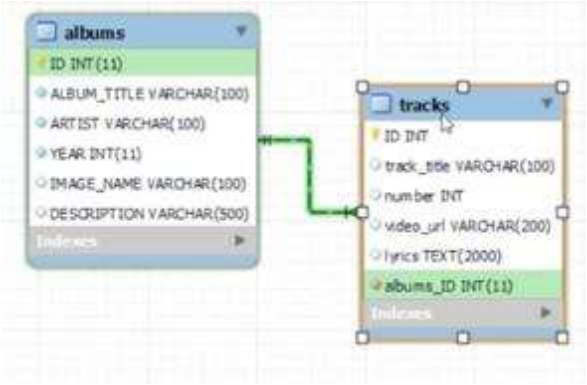


Figure 21 *Correct foreign key.*

3. A new field in the tracks table shows that there is a relationship between the ID numbers of the albums and each track. When we create a new track, this will be better understood.

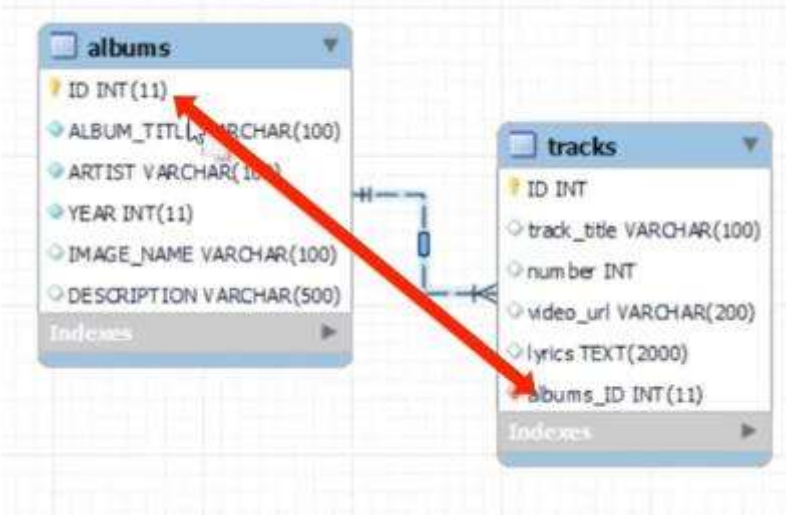


Figure 22 *Foreign key relationship is defined by the ID and albums_ID fields.*

4. Now that we have modified the schema of the database, we need to synchronize it with the MAMP server. Under the database menu choose synchronize model.

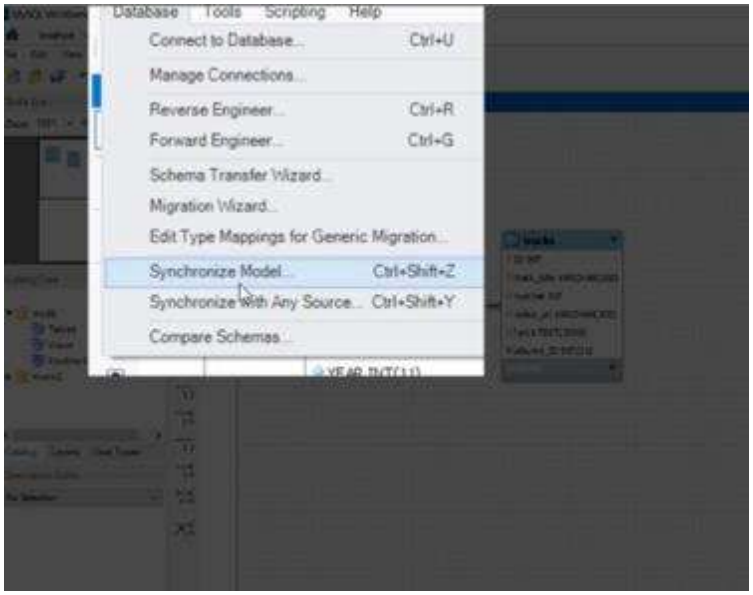


Figure 23 *Synchronize the database with MySQL to save modifications.*

5. You will be asked to connect to the MAMP server again.

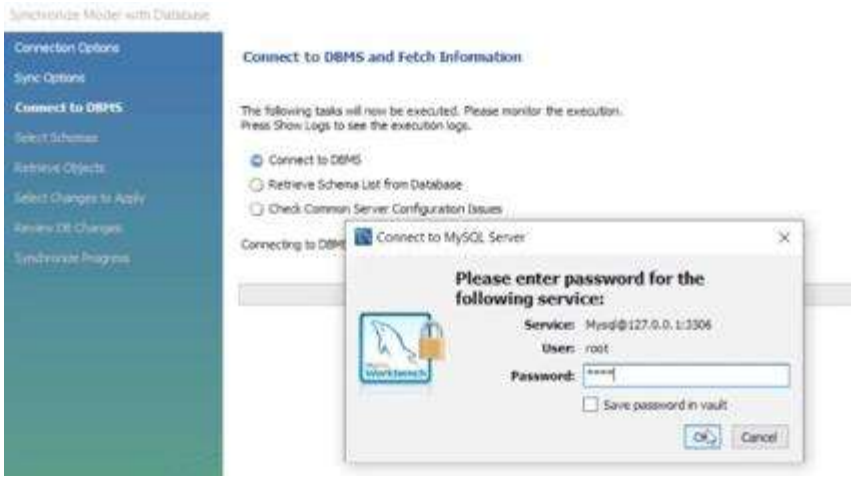


Figure 24 *Logging into MySQL in order to make changes.*

6. Select the music2 schema to synchronize.

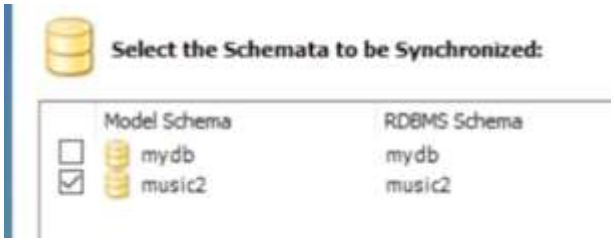


Figure 25 *Selecting music2 and ignoring mydb.*

7. You will be shown a preview of the changes that are about to be enacted. This shows that the tracks table will be added to the database.


Figure 26 *Preview of pending changes.*

8. The SQL commands for creating the table have been written for you in the text below.

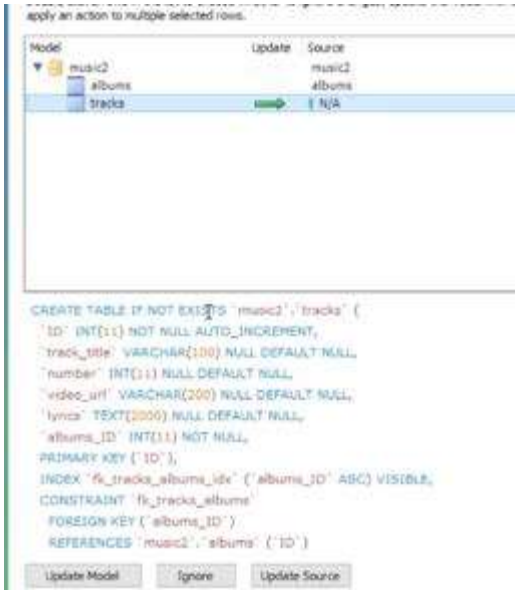


Figure 27 *SQL statement preview.*


Figure 28 *Incorrect SQL statement that will cause an error because of line 20, which contains the word "visible." This keyword only works in MySQL version 8. MAMP is running an older version of MySQL.*

9. You may experience an error when you try to execute the changes. This error is caused because of a difference in SQL versions between MAMP and workbench.


Figure 29 *Error in synchronizing the database in MAMP.*

10. Under the edit menu, choose preferences.

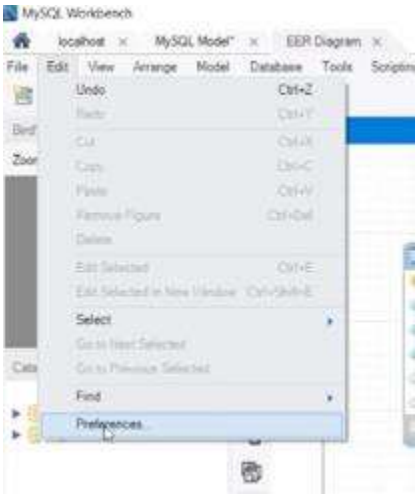


Figure 30 *Updating the preferences.*

11. Select modeling.
12. Select MySQL. Notice that the default target SQL version is version 8.

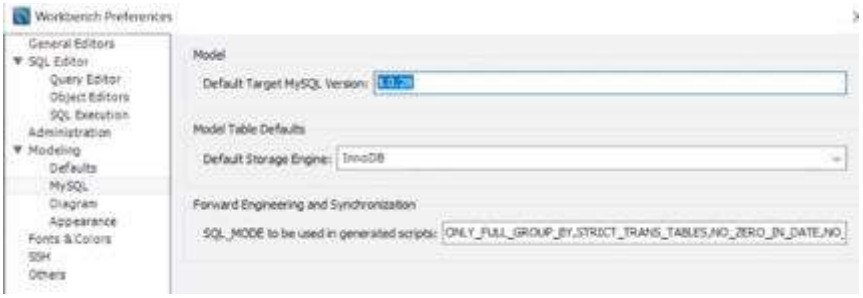


Figure 31 *Updating the SQL version to match the MAMP server.*

13. Let's check the version in the MAMP server. Under the server tab, you may see a MySQL version of 5.2.24.

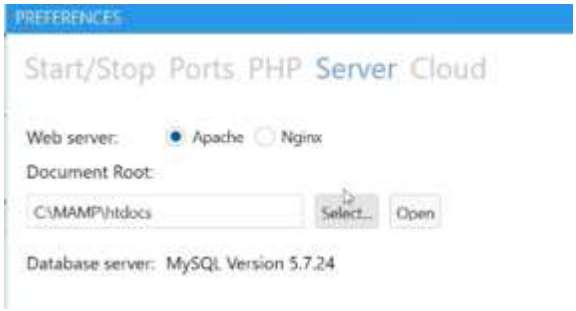


Figure 32 *MySQL 5.7.24 is being used by MAMP.*

14. Change the SQL version in workbench to 5.2.

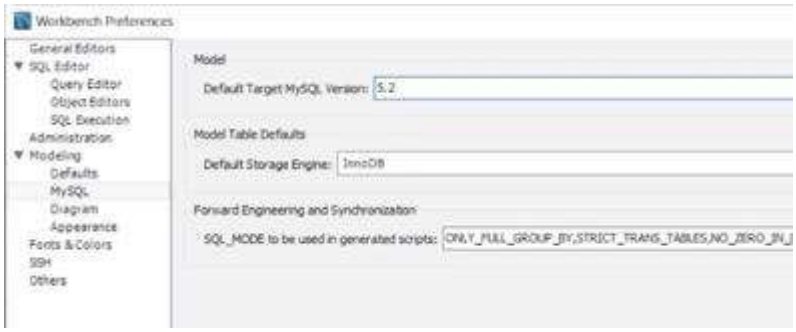


Figure 33 *Setting the SQL version in MySQL Workbench.*

15. Rerun the synchronization process.

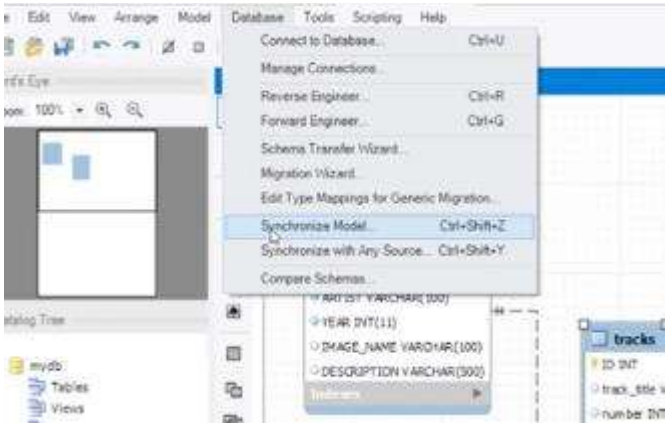


Figure 34 *Attempting to synchronize again.*

16. This time when the SQL code is generated, you will see one small difference between this version and the one that failed. Can you spot the difference?


Figure 35 *Next attempt at SQL statement.*

17. This time the synchronization should be successful.

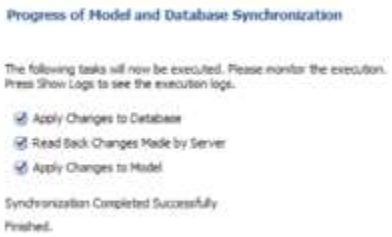


Figure 36 *Successful database update.*

## Add Items to the Tracks Table

In this section, we're going to add some new tracks to the database.


Figure 37 *Reload the database in phpMyAdmin.*

1. Refresh the PHP admin screen. You should see the new tracks table appear in the schema window.

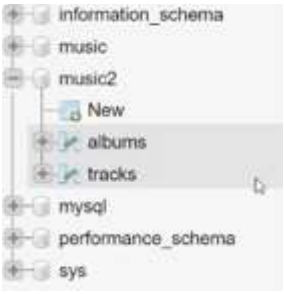


Figure 38 *New changes are visible.*

2. Let's create some information for our new track. Open a YouTube page to get the URL for any song of your choice.


Figure 39 *YouTube page for a song.*

3. Copy the URL from the top of the browser.
4. In the tracks table, choose the insert tab.
5. Add the track title, the song number, and the URL from YouTube.

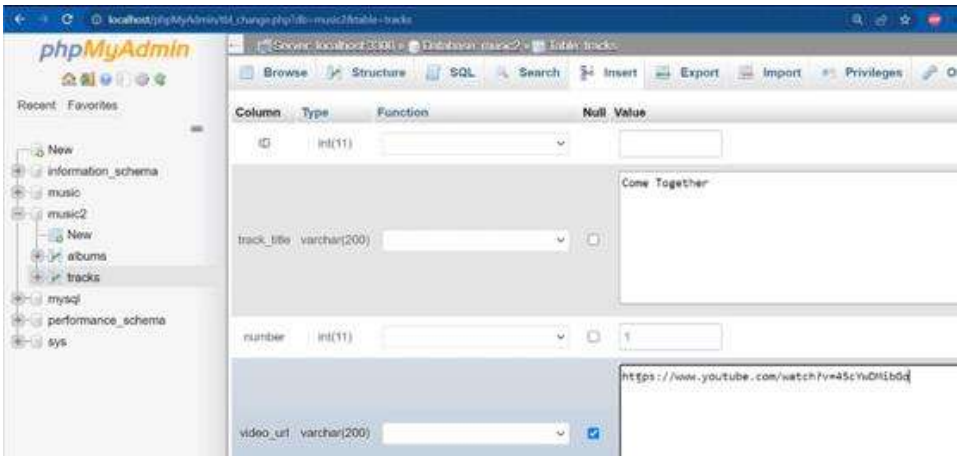


Figure 40 *Adding the URL for the video.*

6. Search for the lyrics on the internet and copy and paste into the lyrics field.

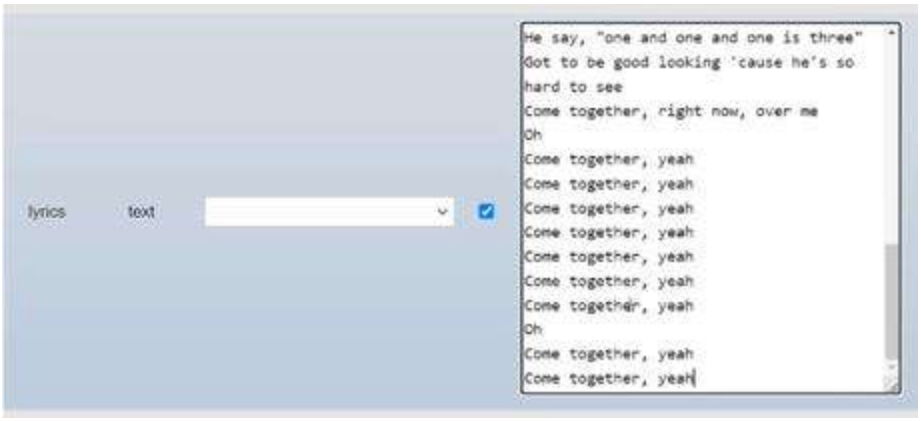


Figure 41 *Adding lyrics.*

7. Finally, we will choose an album to associate this song with. Click the album choice. Notice that this is a **choice menu** instead of a fill in the black field. The foreign key relationship will force you to pick from one of the albums that is already in the database. If this menu does not appear as shown, it means that there is something wrong with the foreign key.

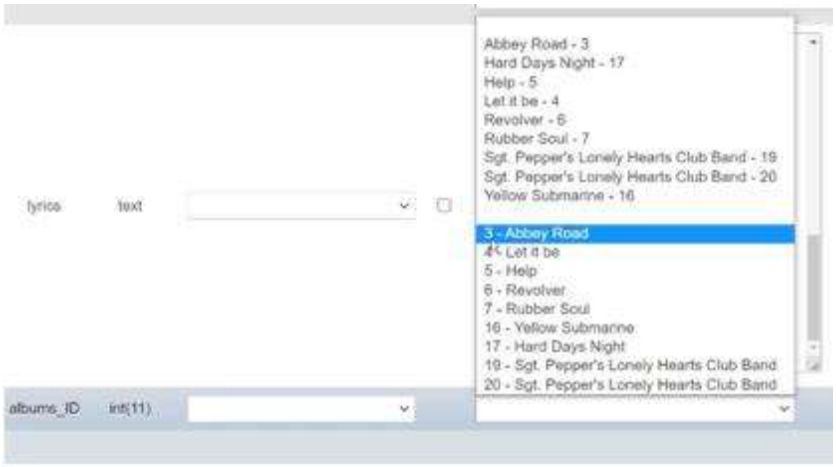


Figure 42 *Adding the album ID number. If this selection box does not appear in your database, it means the foreign key was not created correctly. Go back and fix the errors in MySQL workbench.*

8. Click the go button and the insert command will be executed. You can see the insert SQL command after it runs.

```
INSERT INTO `tracks` (`ID`, `track_title`, `number`, `video_url`, `lyrics`,
`albums_ID`) VALUES (NULL, 'Come Together', '1', 'https://www.youtube.com/watch?
v=45cYwDMibGo', 'Here come old flat top\r\nHe come grooving up slowly\r\nHe got
joo joo eyeball\r\nHe one holy roller\r\nHe got hair down to his knee\r\nGot to
```

Figure 43 *SQL insert statement preview.*

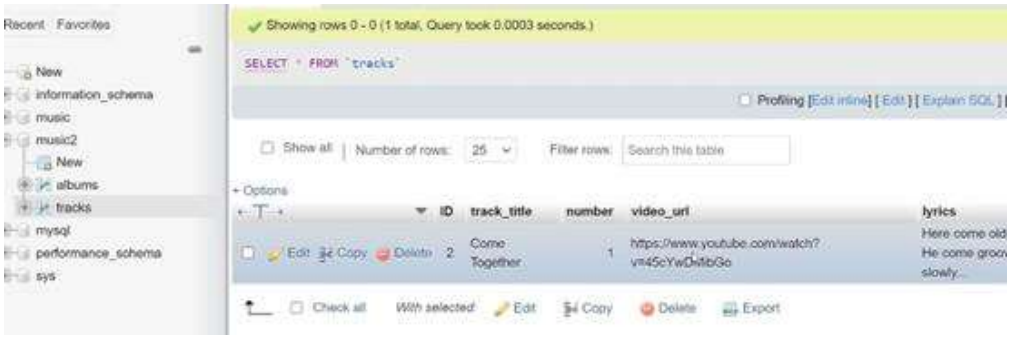


Figure 44 *SQL insert success.*

## Show Tracks for Each Album

In this section, we are going to add a second grid to the Visual Studio Program. In this grid, we will show each track is associated with the album.

Here is a preview of what we will create:


Figure 45 *Showing the tracks associated with album ID = 5.*

## Add a Second Data Grid

1. In the form designer, drag a new label onto the form. Place the new data grid view beneath the first one.

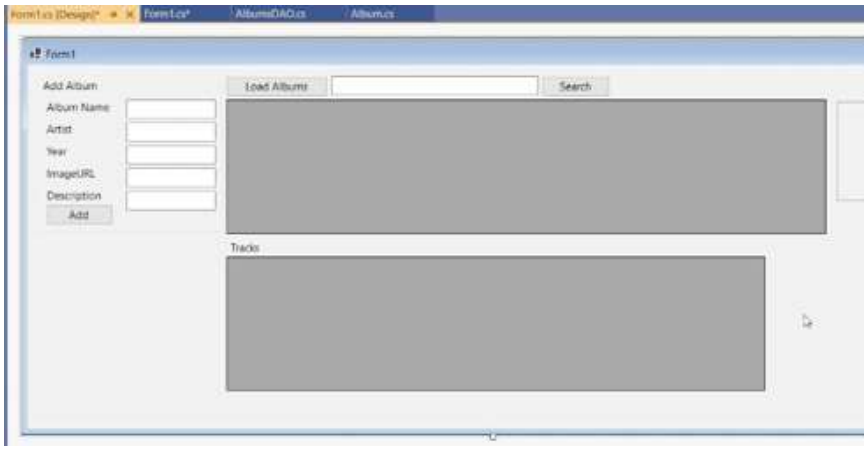


Figure 46 *Position of second grid view.*

2. Copy and paste the code for getAllAlbums.
3. Change the name of the new method to getTracksForAlbum. We will change the internals of this function to work with tracks.


Figure 47 *Modify code from the albums dao.*

4. The track class does not exist yet causing an error. Right-click on the word track and create a new class.


Figure 48 *Adding the track class.*

5. Set the following properties for the track class: ID, Name, Number, VideoURL, and Lyrics.

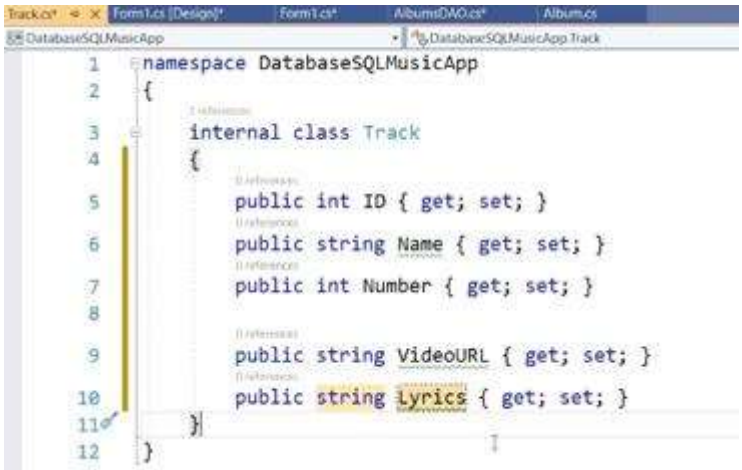


Figure 49 *Properties for the track class.*

6. You should see the new class in the solution explorer.

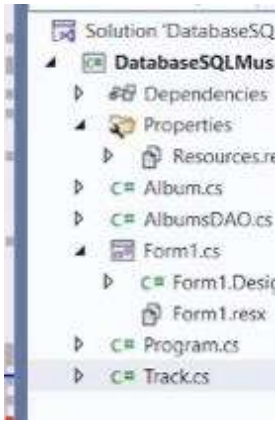


Figure 50 *Track class appears in the solution explorer.*

7. Modify the rest of the contents of the function to retrieve all of the tracks for a specific album number.

```
122     public List<Track> getTracksForAlbum(int albumID)
123     {
124         // start with an empty list
125         List<Track> returnThese = new List<Track>();
126
127         // connect to the mysql server
128
129         MySqlConnection connection = new MySqlConnection(connectionString);
130         connection.Open();
```

```
133
134         // define the sql statement to fetch all albums
135         MySqlCommand command = new MySqlCommand();
136         command.CommandText = "SELECT * FROM TRACKS  WHERE albums_ID = @albumid";
137         command.Parameters.AddWithValue("@albumid", albumID);
138         command.Connection = connection;
139
140         using (MySqlDataReader reader = command.ExecuteReader())
141         {
142
```

```
143             while (reader.Read())
144             {
145                 Track t = new Track
146                 {
147                     ID = reader.GetInt32(0),
148                     Name = reader.GetString(1),
149                     Number = reader.GetInt32(2),
150                     VideoURL = reader.GetString(3),
151                     Lyrics = reader.GetString(4),
152                 };
153
154                 returnThese.Add(t);
155             }
156         }
157         connection.Close();
158
159
160
161         return returnThese;
162     }
```

Figure 51 *Code to fetch tracks.*

8. At the top of form1, add a second binding source. This binding source will serve the track gridview.

```
3   public partial class Form1 : Form
4   {
5       BindingSource albumbBindingSource = new BindingSource();
6       BindingSource tracksBindingSource = new BindingSource() {
7       public Form1()
8       {
9           InitializeComponent();
10      }
```

Figure 52 *Adding a second binding source.*

9. In the click function for button one, add the following code that will update the dataGridView2 control.
    a. Get the row number clicked.
    b. Get the imageURL from column 4.
    c. Load the imageULR into the pictureBox.
    d. Get the tracks list for the album clicked.
    e. Set the tracks list as the contents of dataGrid2.

```
43              // get the row number clicked
44
45              int rowClicked = dataGridView.CurrentRow.Index;
46             // MessageBox.Show("You clicked row " + rowClicked);
47
48             String imageURL = dataGridView.Rows[rowClicked].Cells[4].Value.ToString();
49
50
51            //  MessageBox.Show("URL=" + imageURL);
52            pictureBox1.Load(imageURL);
53
54              AlbumsDAO albumsDAO = new AlbumsDAO();
55
56              tracksBindingSource.DataSource = albumsDAO.getTracksForAlbum((int)
                  dataGridView.Rows[rowClicked].Cells[0].Value);
57
58              dataGridView2.DataSource = tracksBindingSource;
59
60              |
61          }
```

Figure 53 *Click event for showing the tracks associated with an album.*

10. Run the program and test it. You should see all the tracks associated with the album that you click on. In Figure 54, I clicked on the help album. There are three tracks associated with the help album: Help, The Night Before, and Ticket to Ride.


Figure 54 *Tracks are displayed for "Help" album.*

## Select Joins

In this section, we are going to write a join statement in SQL.

The join statement will select multiple columns from the two tables in our database and return the results as a single list.

A join statement requires a foreign key to link two tables together.

Here is a preview of the app after we do the join. The join statement does not affect the user interface in any way. The changes that we affect will all be in the background.

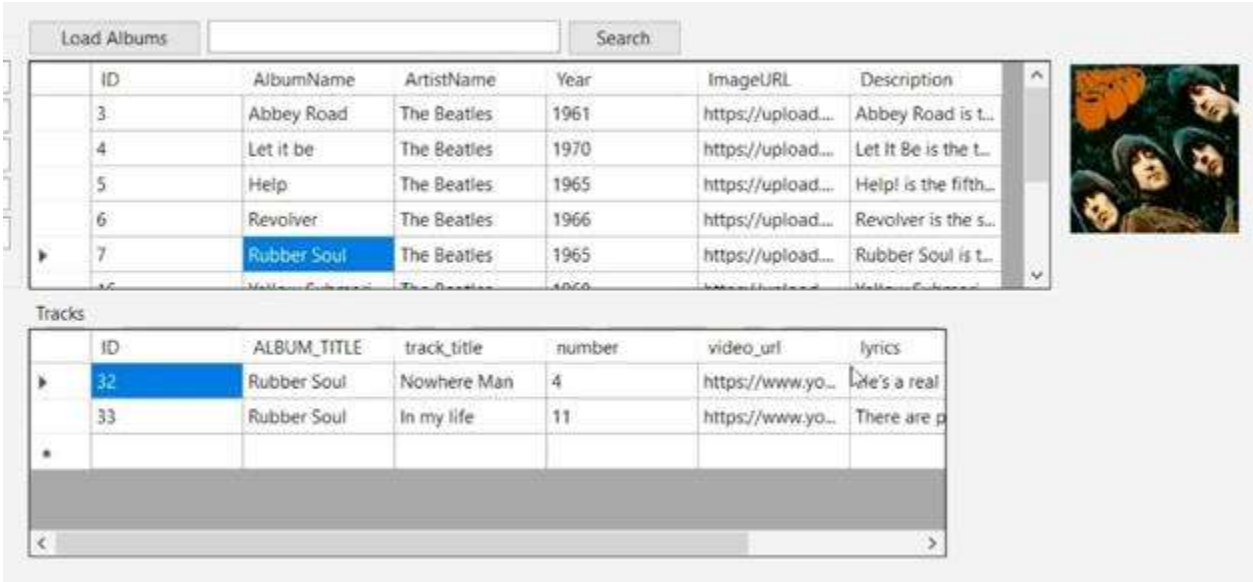


Figure 55 *Result of a SELECT statement.*

Recall from the workbench tool that we created a foreign key between the albums table and the tracks table.

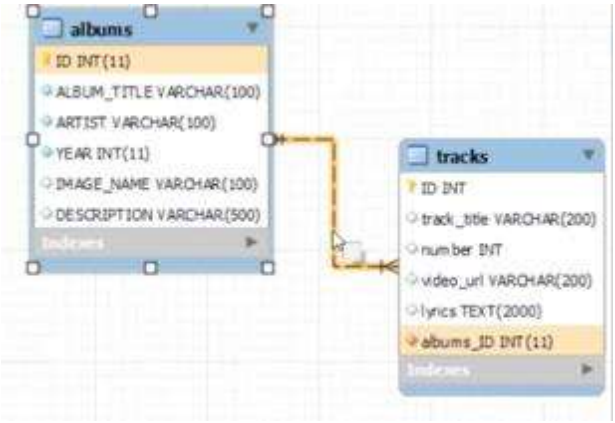


Figure 56 *Previously created foreign key relationship.*

The numbers that are stored in the ID column and the albums ID column have corresponding matches.

1. Open the MyAdmin tool and select the SQL tab.
2. Click the select button at the bottom of the input window. A select statement will be created for you.

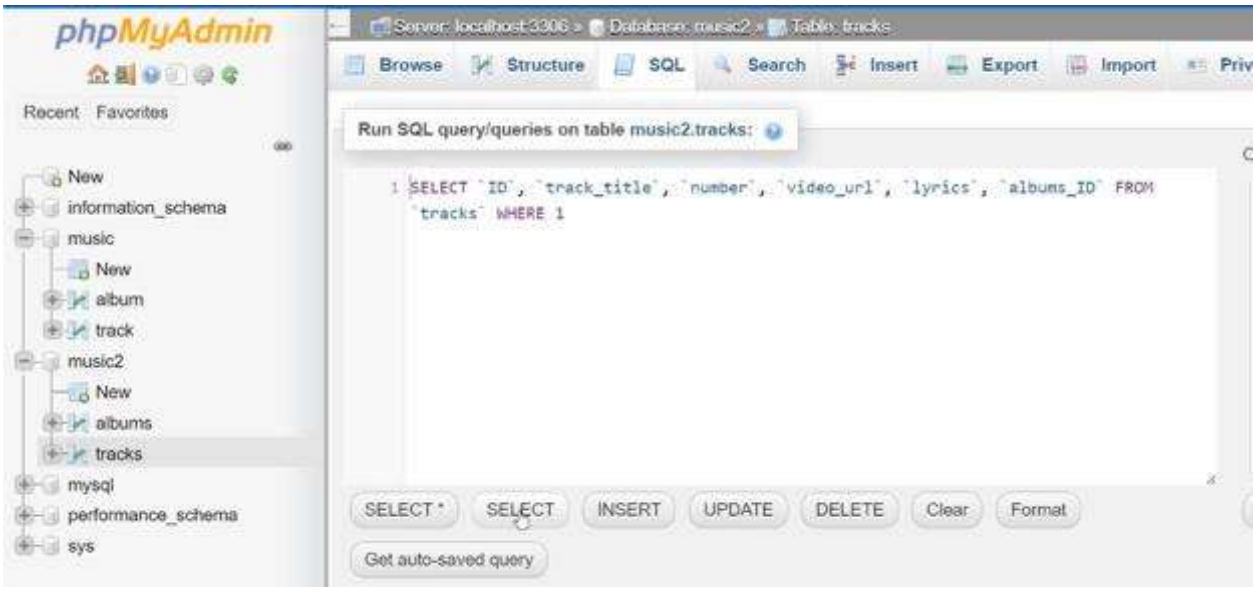


Figure 57 *Starting a SELECT statement in PHP MyAdmin.*

3. Modify the statement to add a join statement. We will join the table on the album_ID and ID column. You can translate this SQL into plain English as "find every instance where these two numbers match."

```
SELECT `ID`, `track_title`, `number`, `video_url`, `lyrics`, `albums_ID` FROM
`tracks` JOIN albums ON albums_ID = albums.ID
```

Figure 58 *Modify the SELECT statement to join two tables.*

4. When we execute this statement, we receive an error. It says that **ID is ambiguous**. That means that there are two columns that have the same name.


Figure 59 *Ambiguous column name error.*

5. Notice that the albums table has an ID column and the tracks table has an ID column. We must distinguish these two column names.

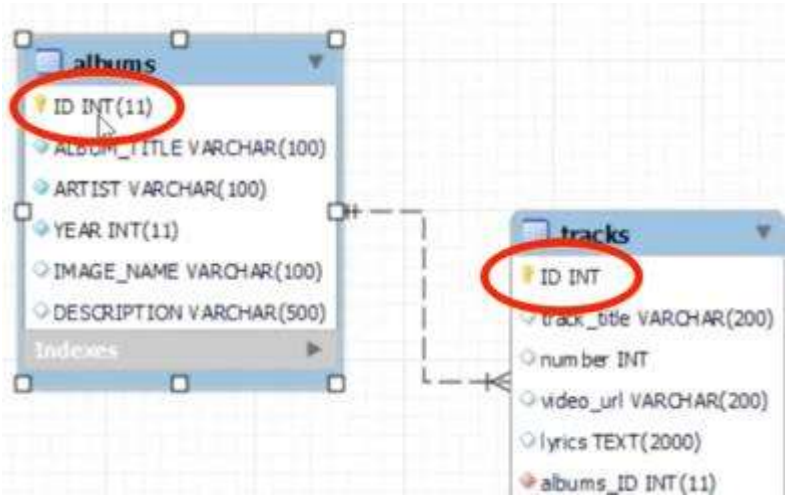


Figure 60 *ID is a column name in both tables.*

6. Rewrite the SQL statement and notice that the first two columns are specified as coming from the albums table and the tracks table. This will remove the ambiguity.

```
SELECT albums.ID, tracks.ID, `track_title`, `number`, `video_url`, `lyrics`,
`albums_ID` FROM `tracks` JOIN albums ON albums_ID = albums.ID
```

Figure 61 *Specifying the table name along with the ID column name.*

7. Run the SQL statement. Notice that there are two columns with the word ID at the top. This is still ambiguous to the user, but the computer can tell the difference.


Figure 62 *ID is shown twice in the result set.*

8. Let's rename each of these ID columns so we can distinguish them in the results. Use the word "as" to rename each of the ID columns.

```
SELECT albums.ID as albumID, tracks.ID as trackID, `track_title`, `number`,
`video_url`, `lyrics`, `albums_ID` FROM `tracks` JOIN albums ON albums_ID =
albums.ID
```

Figure 63 *Renaming the ID results to be more specific.*

9. Now the search results show distinct names for each column, including the first two columns.


Figure 64 *Result set shows separate ID names.*

10. Now that we have a successful join, we can select columns from both tables in the database.
11. Add the album title to the select statement.

```
SELECT albums.ID as albumID, tracks.ID as trackID, albums.ALBUM_TITLE,
`track_title`, `number`, `video_url`, `lyrics`, `albums_ID` FROM `tracks` JOIN
albums ON albums_ID = albums.ID
```

Figure 65 *AlbumTitle property is now part of the SELECT statement.*

12. The search results show the album title for each track.

| albumID | trackID | ALBUM_TITLE | track_title | number | video_url |
|---|---|---|---|---|---|
| 3 | 2 | Abbey Road | Come Together | 1 | https://www.youtube.com/wat v=45cYwDMibGo |
| 5 | 27 | Help | Help | 1 | https://www.youtube.com/wat v=2O_ZzBGPdqE |

Figure 66 *Results of the SELECT query.*

13. In order to simplify the results, remove some of the columns.

```
SELECT tracks.ID as trackID, albums.ALBUM_TITLE, `track_title`, `video_url`,
`lyrics` FROM `tracks` JOIN albums ON albums_ID = albums.ID
```

Figure 67 *Fewer columns have been selected.*

| trackID | ALBUM_TITLE | track_title | video_url | lyrics |
|---|---|---|---|---|
| 2 | Abbey Road | Come Together | https://www.youtube.com/watch?v=45cYwDMibGo | Here come old fl… He c… |
| 27 | Help | Help | https://www.youtube.com/watch?v=2O_ZzBGPdqE | Intro: John Lenn… (Help!) I need so… (Hel… |

Figure 68 *Only trackID, ALBUMS_TITLE, track_title, video_url and lyrics have been selected.*

14. Add a *where* clause to the statement so that we fetch all of the songs from only one of the albums. I am going to choose album #3 because I know that exists in my table. You can choose a different number than three.

```
SELECT tracks.ID as trackID, albums.ALBUM_TITLE, `track_title`, `video_url`,
`lyrics` FROM `tracks` JOIN albums ON albums_ID = albums.ID WHERE albums_ID = 3
```

Figure 69 *Select statement where only album_ID 3 will be selected.*

15. In the AlbumsDAO file, copy the get tracks method and paste it.

16. Rename it as get tracks using join.

```
169     public List<Track> getTracksUsingJoin(int albumID)
170     {
171         // start with an empty list
172         List<Track> returnThese = new List<Track>();
173
174         // connect to the mysql server
175
176         MySqlConnection connection = new MySqlConnection(connectionString);
177         connection.Open();
178
```

Figure 70 *New method called getTracksUsingJoin has been added to the albumsDAO file.*

17. Replace the existing SQL statement with the SQL join statement. Notice that the parameter for AlbumID replaces the number 3.

```
169     public List<Track> getTracksUsingJoin(int albumID)
170     {
171         // start with an empty list
172         List<Track> returnThese = new List<Track>();
173
174         // connect to the mysql server
175
176         MySqlConnection connection = new MySqlConnection(connectionString);
177         connection.Open();
178
179
180
181         // define the sql statement to fetch all albums
182         MySqlCommand command = new MySqlCommand();
183         command.CommandText = "SELECT tracks.ID, albums.ALBUM_TITLE, `track_titl
              `number`, `video_url`, `lyrics`, `albums_ID` FROM `tracks` JOIN albums
              albums_ID = albums.ID WHERE albums.ID = @albumid";
184         command.Parameters.AddWithValue("@albumid", albumID);
```

Figure 71 *@albumid has replaced "3" from the previous select statement.*

18. In the form one module, change the data source inside of the button one click method. The datasource is now albumsDAO.getTracksUsingJoin( ).

```
50          String imageURL = dataGridView.Rows[rowClicked].Cells[4].Value.ToString();
51
52
53          //  MessageBox.Show("URL=" + imageURL);
54          pictureBox1.Load(imageURL);
55
56           AlbumsDAO albumsDAO = new AlbumsDAO();
57
58           tracksBindingSource.DataSource = albumsDAO.getTracksUsingJoin((int)
               dataGridView.Rows[rowClicked].Cells[0].Value);
59
60           dataGridView2.DataSource = tracksBindingSource;
61
62
63      }
```

Figure 72 *New data source applied to the trackBidingSource variable.*

19. Run the program. You will likely see an error when the code tries to parse a track from the database. The reason for this error is that the number of columns returned from the database has changed.

```
190
191          while (reader.Read())
192          {
193              Track t = new Track
194              {
195                  ID = reader.GetInt32(0),
196                  Name = reader.GetString(1),
197                  Number = reader.GetInt32(2),
198                  VideoURL = reader.GetString(3),
199                  Lyrics = reader.GetString(4),
200              };
201
202              returnThese.Add(t);
```

Figure 73 *Runtime error caused by column numbers having been changed in the SELECT statement.*

20. To fix this error, we are going to use a new object. Change the type of data to JObject (JSON object). JObject does not exist in the program yet.

21. Right-click on JObject and choose install package.

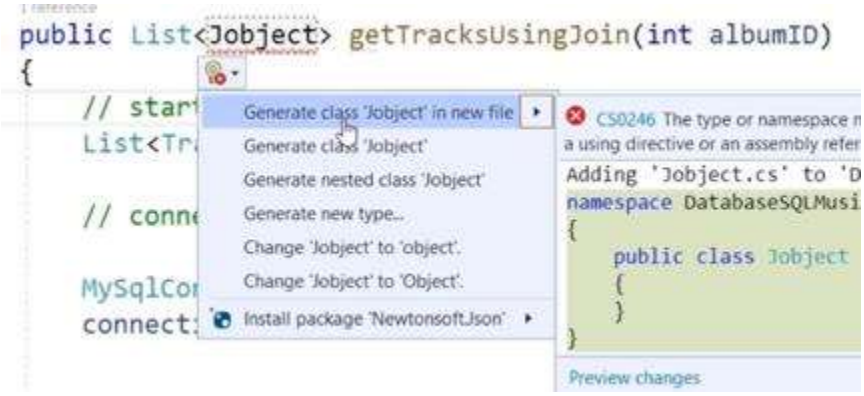


Figure 74 *Installing the NewtonsoftJson dependency.*

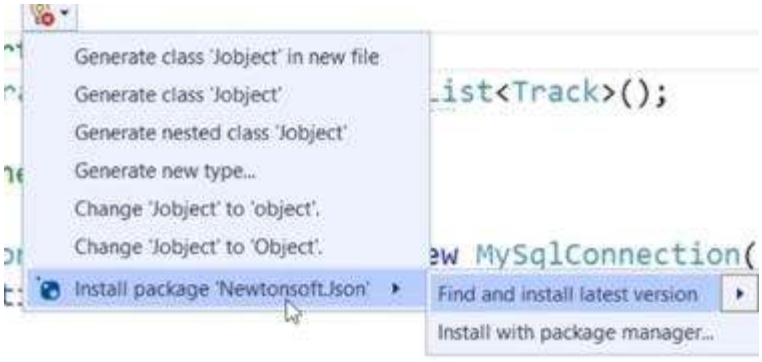


Figure 75 *NewGet will Find and install the latest version of NewtownsoftJSON.*

22. Open the package manager to confirm that the correct object has been installed. You should see a package from Newtown soft.

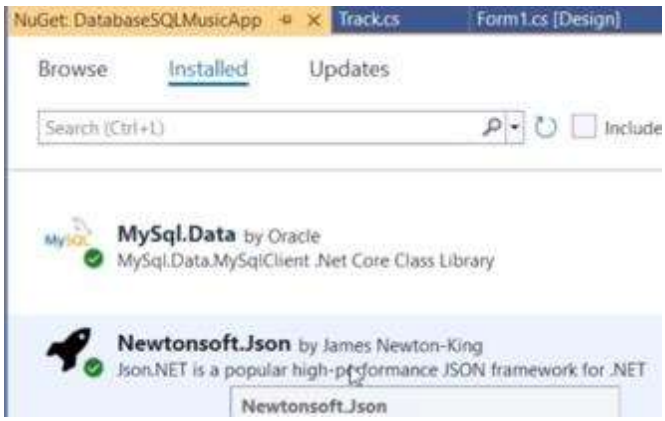


Figure 76 *Newtownsoft_json is now visible in the list of installed Nuget packages.*

## About JObject

A JObject is much more flexible. The object can be assigned with multiple columns, and the columns do not need to be defined before use.

This allows us to modify the number of columns coming from the database without causing an error. It will require us to change a bit of the code to make it work properly.

1. Change any references in the code from track to JObject.


Figure 77 *Track data type in the List<Track> has been changed to List<Jobject>.*

2. Inside of the while loop, change the loop to account for a flexible number of fields. There is a property called field count.

```
188
189         using (MySqlDataReader reader = command.ExecuteReader())
190         {
191
192
193             while (reader.Read())
194             {
195                     JObject newTrack = new JObject();
196
197                     for (int i = 0; i < reader.FieldCount; i++)
198                     {
199                         newTrack.Add(reader.GetName(i).ToString(), reader.GetValue
                        (i).ToString());
200                     }
201
202                     returnThese.Add(newTrack);
203             }
204         }
205         connection.Close();
206
```

Figure 78 *The "for" loop is set to count from 0 to reader.FieldCount which will work regardless of the number of columns used in the SELECT statement.*

3. Run the program again. This time it should successfully retrieve the tracks and have a flexible amount of column names. Notice that the names now match the statement that we created using a join of the two tables.


Figure 79 *App is running. The album "Rubber Soul" has been selected. Two tracks for this album are displayed in the Track table.*

## App Extensions

Let's talk for a moment about extensions to this application. If we were to add new features, we would have to update both the classes in the C# program as well as the tables in the database.

Let's assume that we wanted to track users that log into the application, as well as comments they make for each song. Let's also try to add an artist to each album.

As a result of these features, you may have classes in the program that look similar to Figure 80. This is a UML diagram for classes. The new classes include artist, user, and comments.

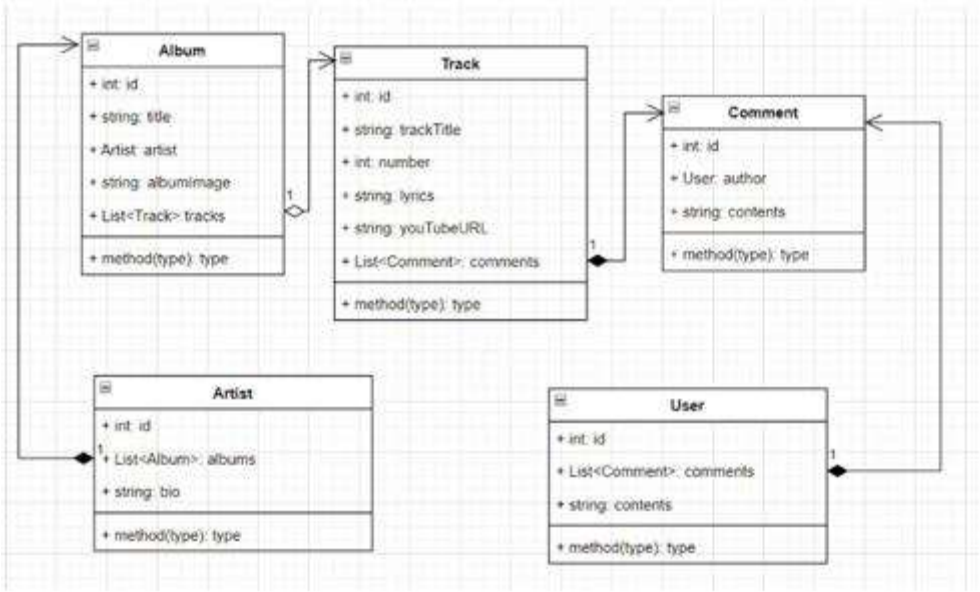


Figure 80 *An example UML diagram showing that the app design could be extended to allow for user accounts, artist categories, and comments for each track.*

In addition to the new classes, the database would also change.

We would have to add a new table for artists comments and users.

The foreign key relationships would look something like this:

1. Each artist can be associated with multiple albums. We create a one-to-many relationship between the artists and albums tables.
2. If you wanted to create a many-to-many relationship, that is each album could be associated with many artists and each artist could be associated with many albums, we would need to create an intermediate table between albums and artists.
3. Each track can have multiple comments. We create a foreign key between the tracks table and the comments table.
4. Each user can have multiple comments.

These descriptions are shown in the example design in Figure 81.

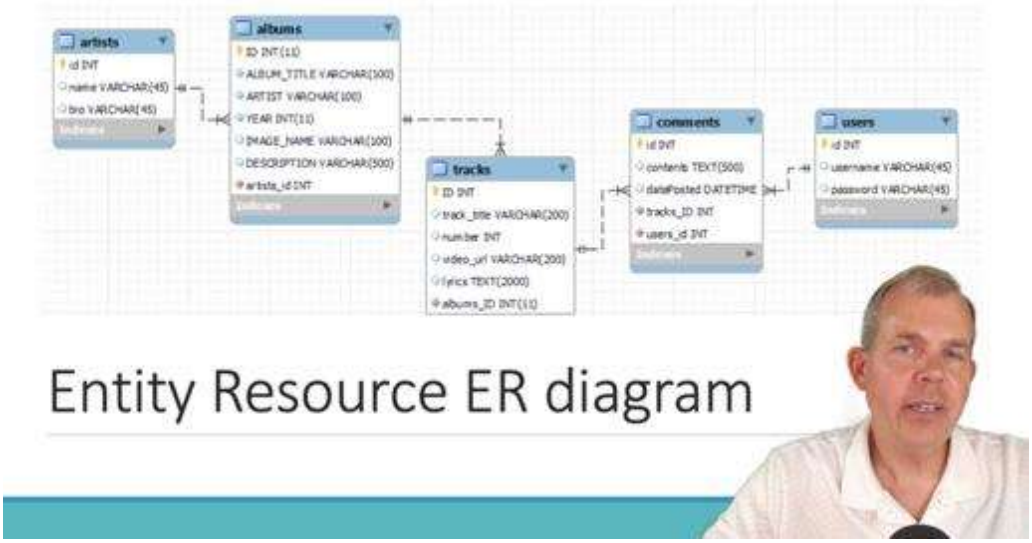


Figure 81 *Five tables with foreign key relationships as described above.*

Of course, this design only affects the back end. The other half of the work is to add new forms to accommodate the new features.

In this tutorial, we will not create these improvements. However, you could use this design for a project in one of your other classes. At Grand Canyon University, each senior student is required to create a capstone project. Usually, a capstone project is like the projects that have been done in classes taken during the first three years; however, new features and more extensive design is usually required.

## Compound Queries

In this section, we are going to create compound queries. This will involve running one query inside of another using the results of the first query. In this case, we are going to fetch all of the tracks associated with an album at the same time we are fetching the album. As a result, we will only connect to the database one time during the albums load phase. The tracks will be collected along with the album titles.

We will now update the album model to include related tracks.

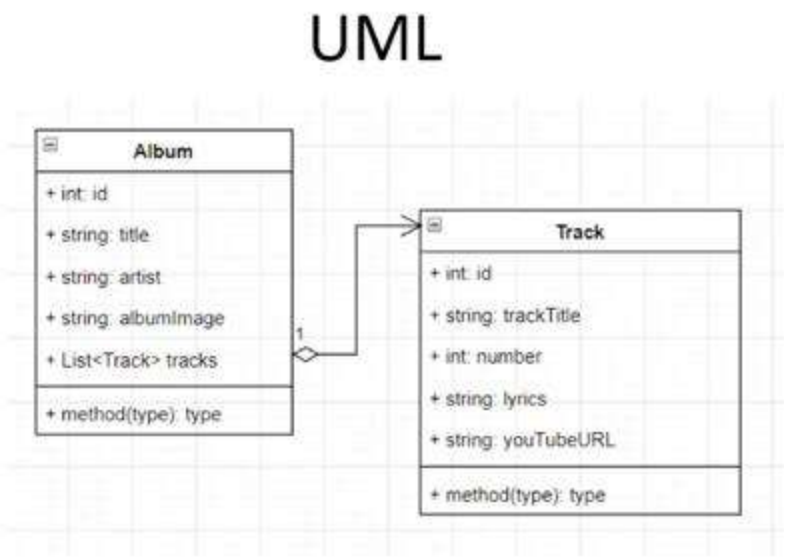


Figure 82 *UML diagram that we will implement. Notice the List<Track> tracks property in the Album class.*

1. In the album class, add a new property, a list of type track.

```
9   internal class Album
10  {
11      public int ID { get; set; }
12      public String AlbumName { get; set; }
13      public String ArtistName { get; set; }
14      public int Year { get; set; }
15      public String ImageURL { get; set; }
16      public String Description { get; set; }
17
18      // later make a List<Track> songs
19      public List<Track> Tracks { get; set; }
20  }
21  }
22
```

Figure 83 *The Album class has a new property of a List<Track> type.*

2. Let's review the **albumsDAO** module and all the functions that are in it.

```
13    internal class AlbumsDAO
14    {
15        string connectionString =
            "datasource=localhost;port=3306;username=root;password=root;database=music2;
16
17        public List<Album> getAllAlbums()...
52
53
54        public List<Album> searchTitles(String searchTerm)...
96
97        internal int addOneAlbum(Album album)...
123
124
125       public List<Track> getTracksForAlbum(int albumID)...
166
167
168   public List<JObject> getTracksUsingJoin(int albumID)...
209
210   }
```

Figure 84 *The AlbumsDAO class has the following methods: getAllAlbums, searchTitles, addOneAlbum, getTracksForAlbum and getTracksUsingJoin..*

3. In the get all albums function, let's add a new feature. After creating the album, use the function gettracksforalbum to collect all the tracks.

```
29
30            using (MySqlDataReader reader = command.ExecuteReader())
31            {
32
33                while (reader.Read())
34                {
35                    Album a = new Album
36                    {
37                        ID = reader.GetInt32(0),
38                        AlbumName = reader.GetString(1),
39                        ArtistName = reader.GetString(2),
40                        Year = reader.GetInt32(3),
41                        ImageURL = reader.GetString(4),
42                        Description = reader.GetString(5)
43                    };
44
45                    a.Tracks = getTracksForAlbum(a.ID);
46
47
48                    returnThese.Add(a);
49                }
```

Figure 85 *The Tracks property of the Album class is populated using the method called getTracksForAlbum.*

4. In Form1, add a new variable at the top of the script.

```
3 references
public partial class Form1 : Form
{
    BindingSource albumbBindingSource = new BindingSource();
    BindingSource tracksBindingSource = new BindingSource();

    List<Album> albums = new List<Album>();
1 reference
public Form1()
```

Figure 86 *The Form1 class has three variables: albumsBindingSource, tracksBindingSource and albums.*

5. In the button click event, change the data source for the tracks grid to refer to the albums list. See line number 56.

```
42          DataGridView dataGridView = (DataGridView)sender;
43
44          // get the row number clicked
45
46          int rowClicked = dataGridView.CurrentRow.Index;
47          // MessageBox.Show("You clicked row " + rowClicked);
48
49          String imageURL = dataGridView.Rows[rowClicked].Cells[4].Value.ToString();
50
51
52          //  MessageBox.Show("URL=" + imageURL);
53          pictureBox1.Load(imageURL);
54
55
56          tracksBindingSource.DataSource = albums[rowClicked].Tracks;
57          dataGridView2.DataSource = tracksBindingSource;
58      }
59
```

Figure 87 *Line 56 shows the tracksBindingSource.DataSource property is related to the Tracks property.*

## Video Player

In this section, we are going to add a video player to the application.

Here is a preview of what we will create. Notice the video player in the bottom right corner.

This control is really just a web browser contained inside of a rectangle. This control relies on an embedded version of the Edge internet browser.


Figure 88 *The application will contain a video player section in the bottom right corner of the form.*

1. In order to use this control, we need to install another dependency. Open the new get package manager and choose WebView2 from Microsoft.

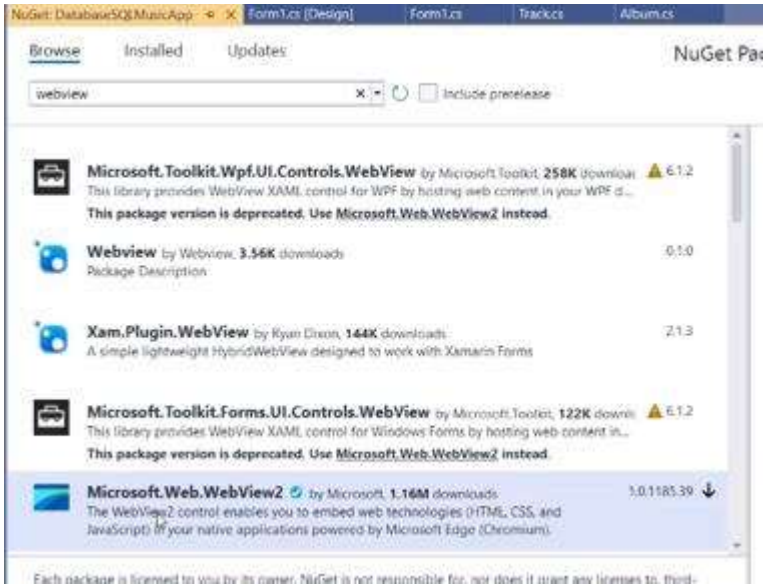


Figure 89 *WebView2 is found in the NuGet dependencies list by searching for WebView.*

2. The description of the package shows that it is an embedded version of Edge.


Figure 90 *Description of the NuGet plugin WebView2 is shown when installing the dependency.*

3. After installing the package, you should be able to find WebView2 in the toolbox.


Figure 91 *WebView2 now appears as part of the Visual Studio Toolbox after adding the NuGet package to the solution.*

4. Drag a WebView2 control onto the form.

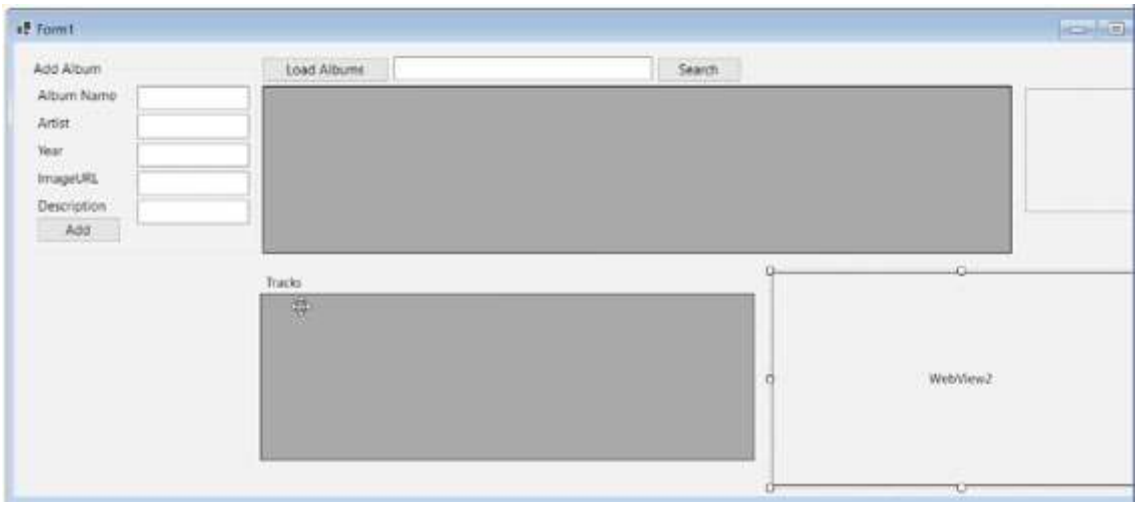


Figure 92 *WebView2 control is placed in the lower-right corner of the form.*

5. The goal here is to update the contents of WebView2 with the URL taken from the video URL column of the track table.

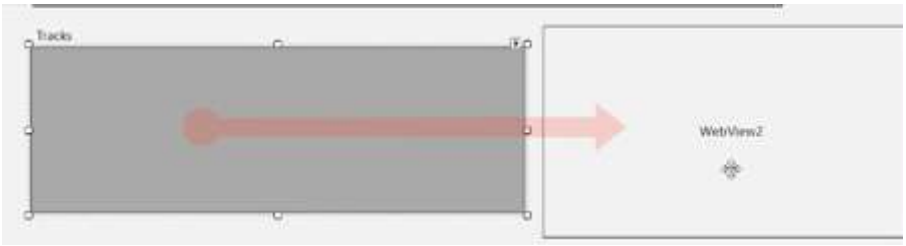


Figure 93 *The URL property from the GridView control will be used to load the webpage of WebView2.*

6. Now let's add a click event to the second gridview. Select gridview2.
7. In the properties of gridview2 find the mouse events.
8. Select the cell click event.

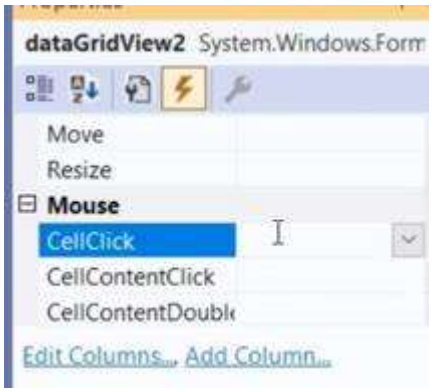


Figure 94 *Editing the CellClick event of dataGridView2.*

9. Double-click in the event property. A new click event will appear inside of the form one code. This event will trigger every time you click a cell in data grid 2.


Figure 95 *New code is automatically generated for the dataGridView2_CellClick event after double-clicking the CellClick property.*

10. Add the following code in Figure 96 to handle an update of the WebView2 control.
    a. Get a reference to the dataGridView.
    b. Fetch the URL string from the gridView.
    c. Set the data source of webView2 to the YouTube video associated with the track.

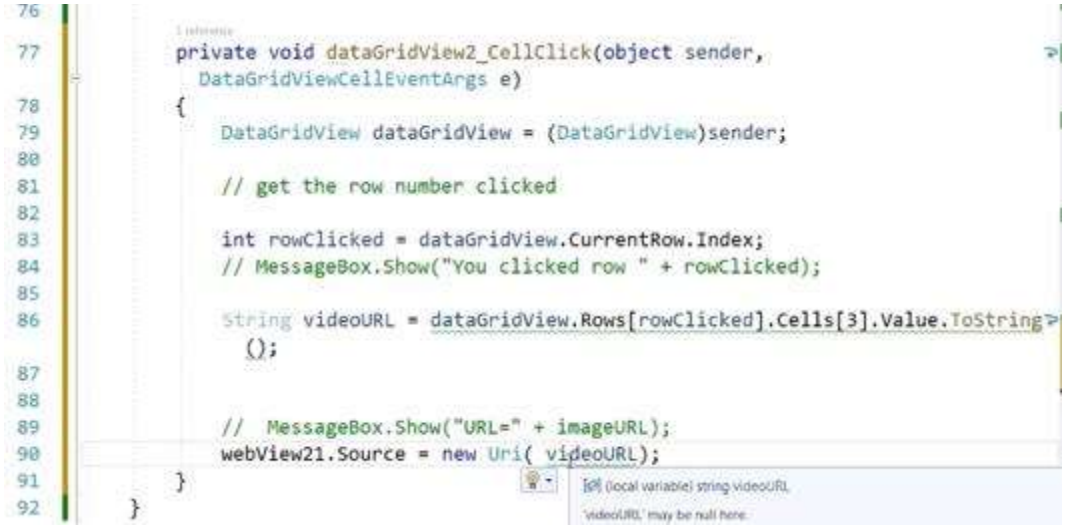


Figure 96 *New code for the CellClick method will display the video URL in the WebView2 component.*

11. Test the program. You should be able to see a YouTube video when you click on a track title.

## Delete Items

In this section, we're going to add a delete button so we can remove a track from the database.

Here is a preview of what we will create. If a user clicks the delete button, a pop-up will show the result of how many rows were deleted. It should say one row deleted if successful.


Figure 97 *Message will confirm that an item has been deleted.*

There are two parts to this section. First, we will create the delete button, then we will create a method in the DAO to update the database.

1. In Form1, add a new button below grid 2.
2. Set the text property to delete selected track.

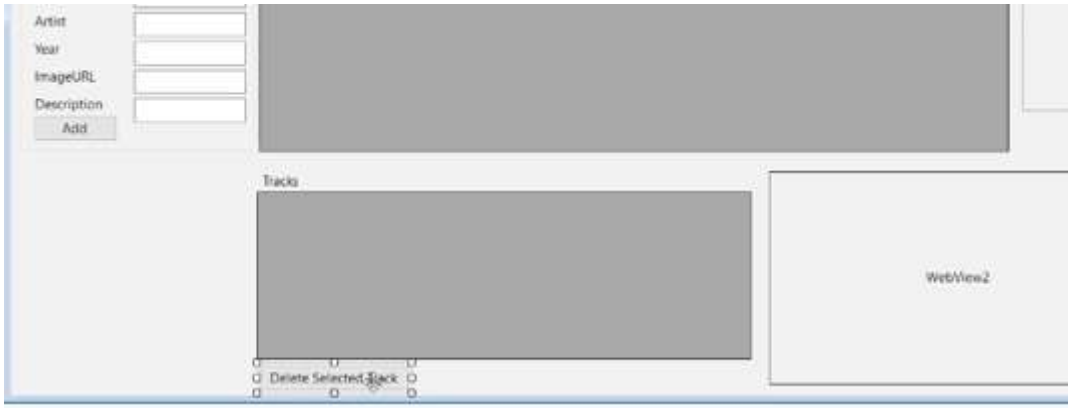


Figure 98 *New button placed at the bottom of the form with the button text set to "Delete Selected Track."*

3. Double-click the button to create an event.
4. Use the code in Figure 99 to test whether the click event is working.
    a. Get the integer value of the row number clicked.
    b. Show a message box to the user confirming the row number.
    c. Get the ID value of the track that was clicked.
    d. Show a message box to the user confirming the row number.

```
93          private void button4_Click(object sender, EventArgs e)
94          {
95              // get the row number clicked
96
97              int rowClicked = dataGridView2.CurrentRow.Index;
98              MessageBox.Show("You clicked row " + rowClicked);
99              int trackID = (int) dataGridView2.Rows[rowClicked].Cells[0].Value;
100             MessageBox.Show("ID of track " + trackID);
101         }
102     }
103 }
```

Figure 99 *New code for button4 click event that implements the features mentioned in the instructions.*

6. Run the program. You should see two pop-up messages. The first will tell you which row number you clicked. The second pop-up will tell you the ID number of that track that was selected. In this case, you can see that row #2 is the third track. Ticket to Ride is considered to be track #30 in the database.


Figure 100 *The first message shows that row 2 was clicked. Row 2 is the third row in the table.*

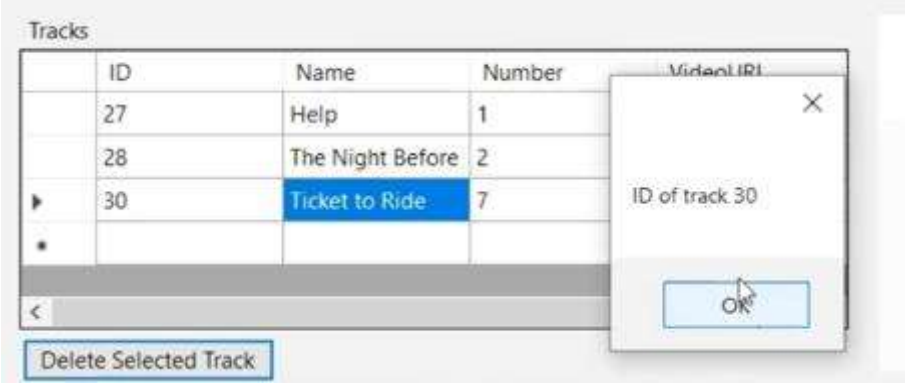


Figure 101 *The second message shows that the ID number of the track was 30.*

7. Update the button click event to communicate with the DAO and call a delete statement.
    a. Get the row number clicked.
    b. Get the track ID number that was clicked.
    c. Create an instance of the albumsDao class.
    d. Call the deleteTrack method for the ID number of the Track to delete.

```
93      private void button4_Click(object sender, EventArgs e)
94      {
95          // get the row number clicked
96
97          int rowClicked = dataGridView2.CurrentRow.Index;
98          MessageBox.Show("You clicked row " + rowClicked);
99          int trackID = (int) dataGridView2.Rows[rowClicked].Cells[0].Value;
100         MessageBox.Show("ID of track " + trackID);
101
102         AlbumsDAO albumsDao = new AlbumsDAO();
103
104         int result = albumsDao.deleteTrack(trackID);
105
106         MessageBox.Show("REsult " + result);
107
108
109     }
```

Figure 102 *Code for button4 that deletes the selected Track according to the instructions.*

8. The delete statement does not yet exist. Right-click on the text and choose generate method.

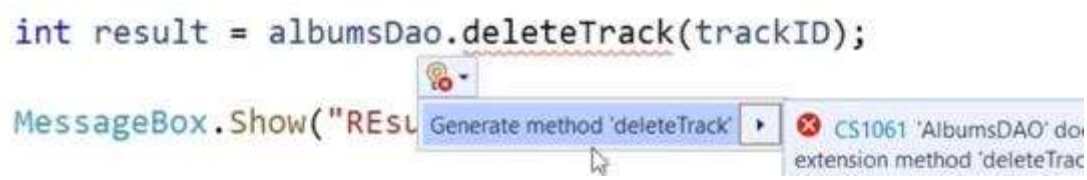


Figure 103 *Visual Studio is suggesting that we generate a new method called "deleteTrack."*

9. You should see a new method in the DAO.


Figure 104 *Delete track is added as a new method for the AlbumsDAO class.*

10. The delete method will only require the track ID as an input parameter.
    a. Create a connection to the database.
    b. Run the AddWithValue SQL command.
    c. Save the results.
    d. Close the database connection.
    e. Return the results.

```
128         {
129             // connect to the mysql server
130
131             MySqlConnection connection = new MySqlConnection(connectionString);
132             connection.Open();
133
134             // define the sql statement to fetch all albums
135             MySqlCommand command = new MySqlCommand("", connection);
136
137             command.Parameters.AddWithValue("@tr@ckID", trackID);
138
139
140             int result = command.ExecuteNonQuery();
141             connection.Close();
142
143
144             return return;
145         }
146
```

Figure 105 *The contents of DeleteTrack method.*

11. Run the PHP admin program.
12. Open the tracks table.
13. Choose the delete button for a track.
14. You should see a SQL statement.
15. Copy the statement.


Figure 106 *PHPMyAdmin shows the SQL syntax for a delete command. Copy this example and use it in the music player code.*

16. Paste the SQL statement in the command string of the DAO.

```
128         {
129             // connect to the mysql server
130
131             MySqlConnection connection = new MySqlConnection(connectionString);
132             connection.Open();
133
134             // define the sql statement to fetch all albums
135             MySqlCommand command = new MySqlCommand("DELETE FROM `tracks` WHERE
                  `tracks`.`ID` = 43;", connection);
136
137             command.Parameters.AddWithValue("@trackID", trackID);
138
139
140             int result = command.ExecuteNonQuery();
141             connection.Close();
142
143
144             return return;
145         }
```

Figure 107 *Delete SQL command has been added to the C# app. "43" is a placeholder for changes we will make soon.*

17. Replace the track number with a parameter called track ID.

```
128         {
129             // connect to the mysql server
130
131             MySqlConnection connection = new MySqlConnection(connectionString);
132             connection.Open();
133
134             // define the sql statement to fetch all albums
135             MySqlCommand command = new MySqlCommand("DELETE FROM `tracks` WHERE
                  `tracks`.`ID` = @trackID;", connection);
136
137             command.Parameters.AddWithValue("@trackID", trackID);
138
139
140             int result = command.ExecuteNonQuery();
141             connection.Close();
142
143
144             return result;
145         }
```

Figure 108 *@TrackID is the parameter used to identify which track will be deleted. @TrackID replaced "43" from the example statement.*

18. In the button click, update the code so that it uses the new delete track method.

```
92
93      private void button4_Click(object sender, EventArgs e)
94      {
95          // get the row number clicked
96
97          int rowClicked = dataGridView2.CurrentRow.Index;
98          MessageBox.Show("You clicked row " + rowClicked);
99          int trackID = (int) dataGridView2.Rows[rowClicked].Cells[0].Value;
100         MessageBox.Show("ID of track " + trackID);
101
102         AlbumsDAO albumsDao = new AlbumsDAO();
103
104         int result = albumsDao.deleteTrack(trackID);
105                                                  (local variable) int trackID
106         MessageBox.Show("REsult " + result);
107
108         dataGridView2.DataSource = null;
109         albums = albumsDao.getAllAlbums();
110
```

Figure 109 *DeleteTrack with trackID is used to update the database.*

19. Run the program. You should have a pop-up that says result one if the delete option worked properly.

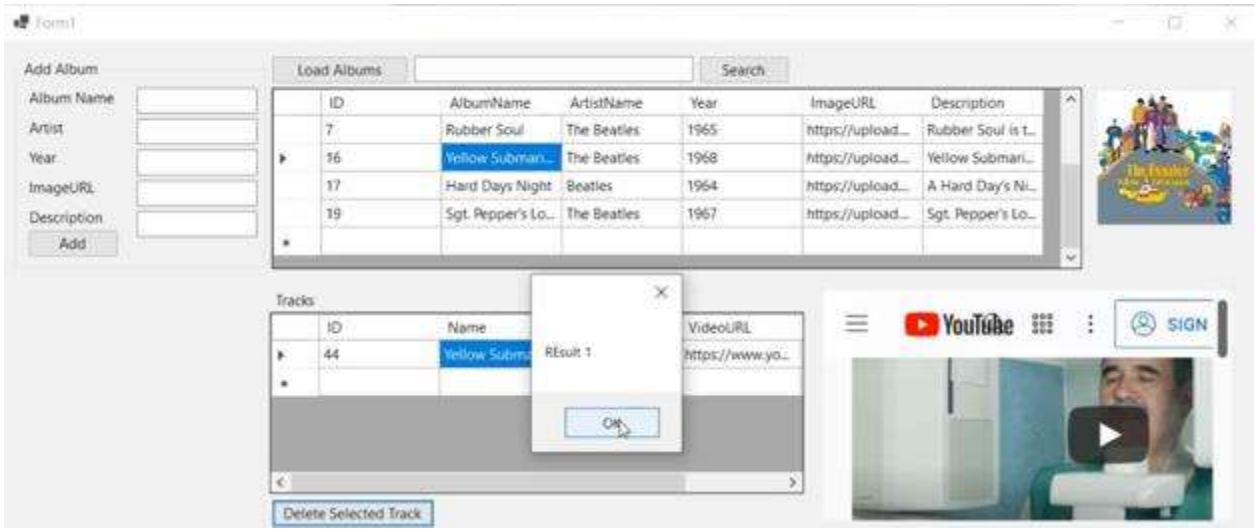


Figure 110 *Result 1 in the message box indicates that one row was deleted from the table.*

## Deliverables

1. Submit a Microsoft Word document with screenshots of the application being run. Show each screen of the output and include a caption under each picture explaining what is being demonstrated.
    a. Show the tracks associated with an album when the album data is clicked.
    b. Show the video player displaying the YouTube video when a track is clicked.
2. In the same document, write a one-paragraph summary of the key concepts that were demonstrated in this lesson.

3. Submit a ZIP file of the project file. In order to save space, you can delete the bin and the obj folders of the project. These folders contain the compiled version of the application and are automatically regenerated each time the build or run commands are executed.

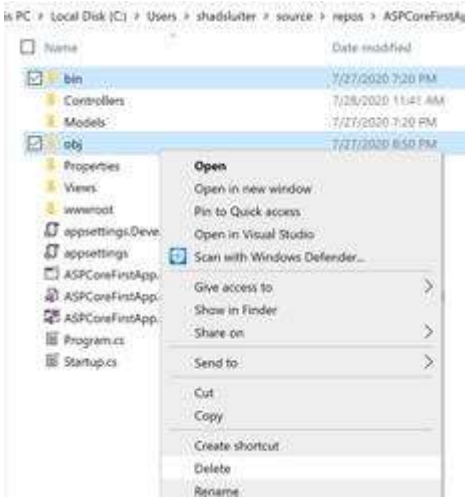


Figure 111 *Bin and obj folders.*