INFO90002 Week 4 Tutorial Solutions

Objectives:

- Create a Physical Model using MySQL Workbench (25 minutes)
- Learn SQL by example (25 minutes)

Section 1. Data Modelling

Creating the model

MySQL Workbench has three sections: the default window, the modelling window and the migration window.

1.1: Launch MySQL Workbench

When you launch MySQL Workbench you will be in the default "MySQL Connections" view. This is circled in red below:
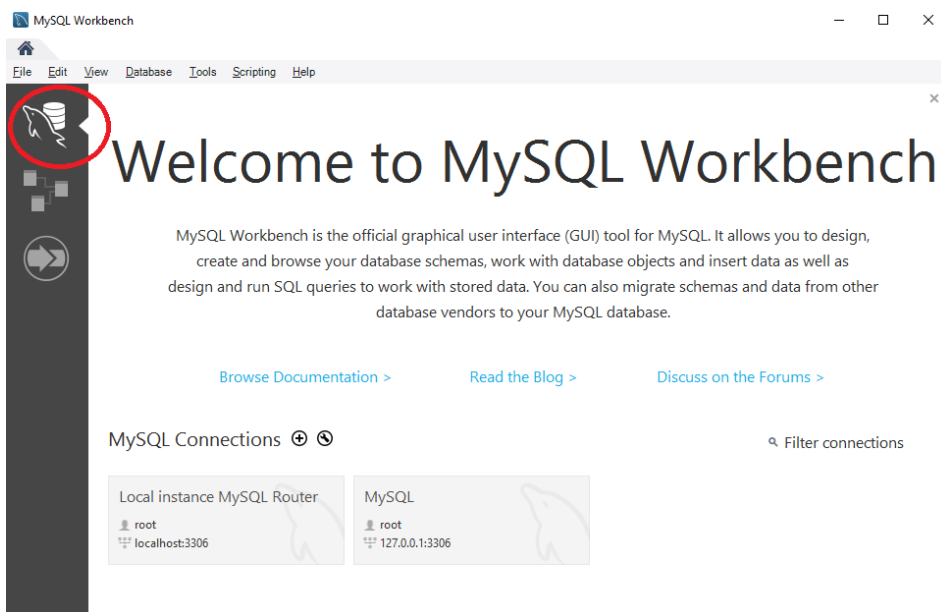

Figure 1: The default MySQL Workbench window

You will need to select the "Models" view:

Figure 2: The Models icon
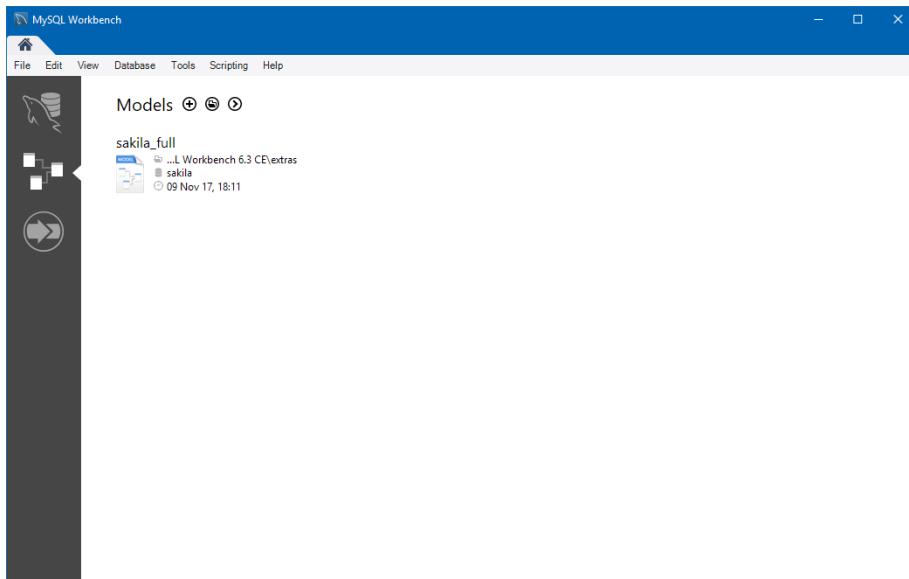
Your window will have changed to a different view:

Figure 3: The full screen of the Models view

1.2 Click the Add model (+) symbol next to the word "Models" at the top left of the MySQL

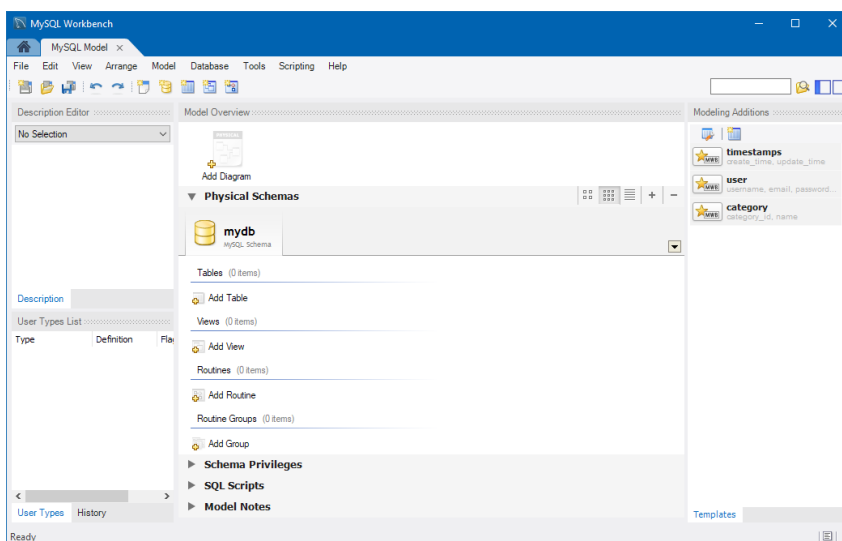Workbench window.

This will launch a new modelling window:

Figure 4: The new modelling window

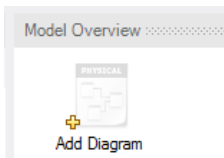**1.3:** Click the "Add Diagram" icon to add a new diagram:

Figure 5: The add diagram icon

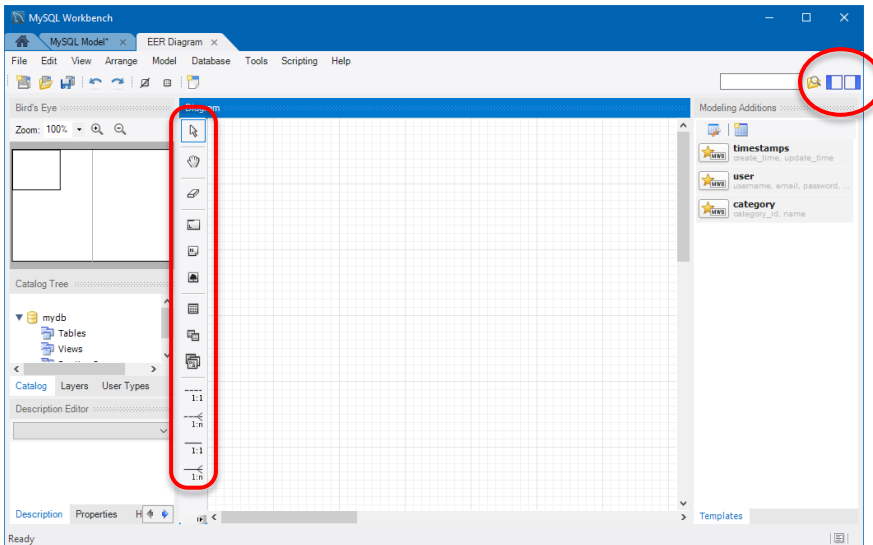This will bring up a second tab containing an empty diagram window:

Figure 6: The diagram with modelling tools on the left-hand side of the canvas. Note the buttons to hide the side panels in the top-right corner of the canvas.

1.4: Make more space for your diagram. Hide the left- and right-hand panels of your modelling

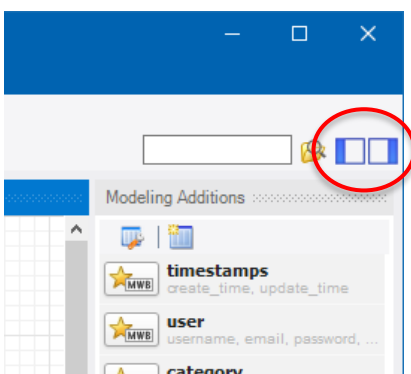canvas using the buttons in the top-right corner:

Figure 7: Highlight of the icon to minimize the right and left columns of the modelling canvas

Adding tables to your model

1.5 Add a new table to your model by clicking on the table icon, then clicking anywhere in your diagram.
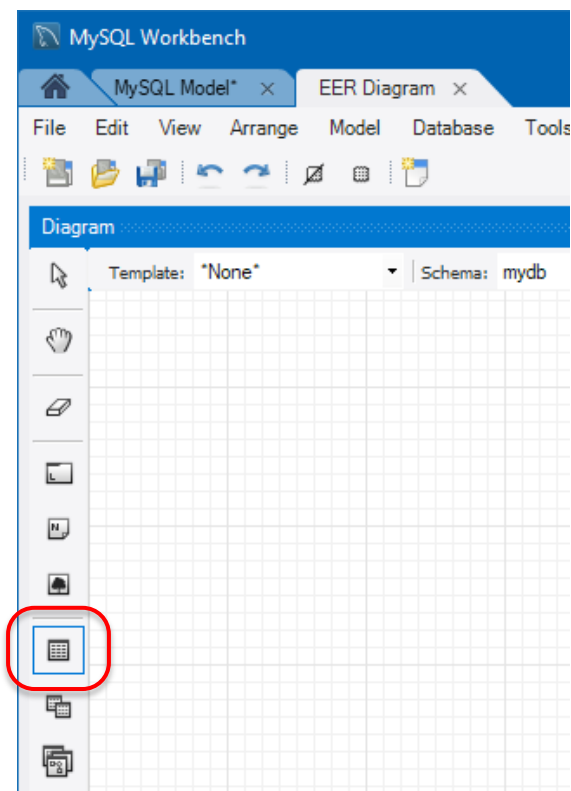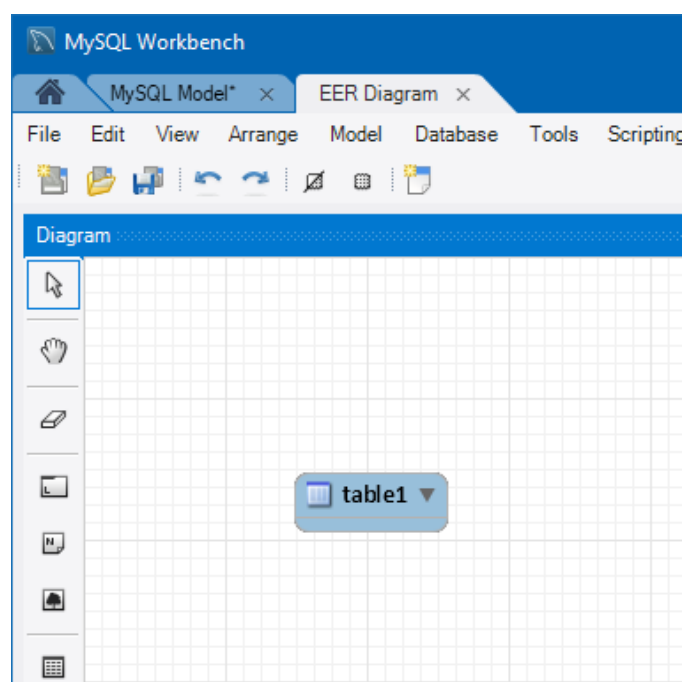

Figure 8: The add table tool

A new table will appear:

Figure 9: The table has now been placed on the diagram canvas

1.6: Double click the table. The table editor appears at the bottom of the window. The table name is highlighted.
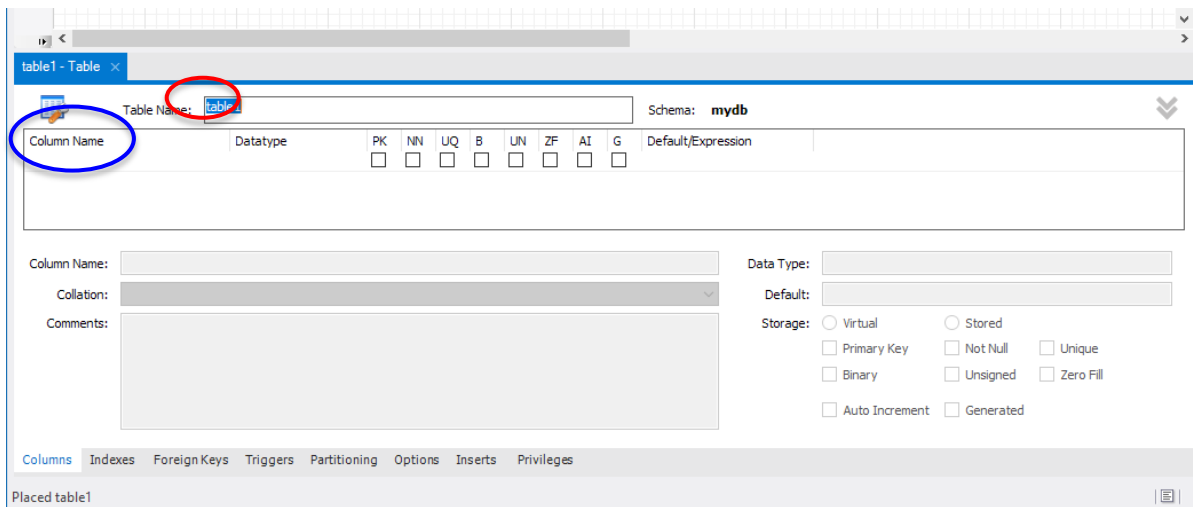

Figure 10: The table editor

1.7: Change the name of the table from 'table1' to 'Song'.

1.8 Now double-click the area under 'Column Name' and enter the following information for each column:

Song Table:

Column Name	Data Type	Check Box (select)
SongID	INT	PK, NN *
SongTitle	VARCHAR(45)	NN
Artist	VARCHAR(45)	NN
AlbumID	INT	
LastPlayed	DATETIME	
Genre	VARCHAR(20)	

Table 1: The Music table column names and data types

* Note: PK stands for "primary key", and NN is short for "NOT NULL" (see below).

To change the column's data type, double-click in the data type column and select from the drop-down list. For VARCHAR you need to type a number in the brackets. This number is the maximum length of the text that can be stored in this column. For DATETIME you **do not** need to type a number in the brackets. If brackets are shown, simply delete them.

The finished table should look like this in the table editor:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
🔑 SongID	INT	☑	☑	☐	☐	☐	☐	☐	☐	
◇ SongTitle	VARCHAR(45)	☐	☑	☐	☐	☐	☐	☐	☐	
◇ Artist	VARCHAR(45)	☐	☑	☐	☐	☐	☐	☐	☐	
◇ AlbumID	INT	☐	☐	☐	☐	☐	☐	☐	☐	
◇ LastPlayed	DATETIME	☐	☐	☐	☐	☐	☐	☐	☐	
◇ Genre	VARCHAR(20)	☐	☐	☐	☐	☐	☐	☐	☐	
		☐	☐	☐	☐	☐	☐	☐	☐	

Figure 11: The Song table column information in the Table Wizard

And the Song table in the diagram should now look like this:
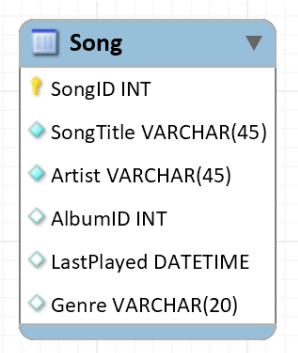
Figure 12: The Song table as drawn in the diagram

The yellow 'key' indicates that the SongID column is a Primary Key. The solid blue diamond indicates that this column must be populated for every row in the table and can not be empty (or "null"). The blue outlined diamonds indicate the column can contain null or empty values for a row.

1.9 Repeat Tasks 5 to 8 to add the Album and RecordCompany tables to the diagram:

Album Table

Column Name	Data Type	Check Box (select)
AlbumID	INT	PK, NN
AlbumTitle	VARCHAR(45)	NN
Rating	INT	
RecordCompanyID	INT	NN

Table 2: The Album table column names and data types

RecordCompany Table

Column Name	Data Type	Check Box (select)
RecordCompanyID	INT	PK, NN
RecordCompanyName	VARCHAR(45)	NN
Country	VARCHAR(45)	NN

Table 3: The RecordCompany table column names and data types

After completing Task 10, your diagram should look similar to below:
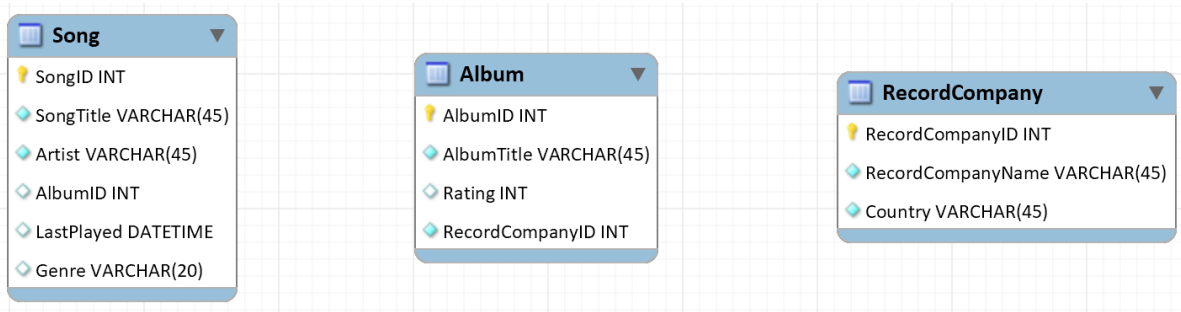
Figure 13: The three tables (Song, Album and RecordCompany) in your diagram, without relationships

1.10 Save your changes (**File** > **Save**).

Relating tables

Now that you have created the tables, it is time to join them with relationships.

When joining tables to each other, make sure you use the "eyedropper" tool that allows you to use existing columns.
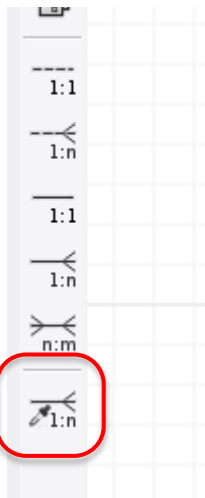
Figure 14: The "Place a Relationship Using Existing Columns" or "eyedropper" tool. This tool allows you to define relationships between two entities using an existing column in each entity.
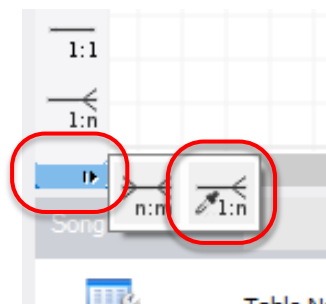
Figure 15: On Windows, this tool might be located in the "overflow" popout at the bottom of the toolbar.

Mac users note: If you cannot find this tool, you might need to close the Table editor section of the window.

In the Music table, the AlbumID column is the attribute that will identify which album each song is found on.

In the following tasks, you will learn how to use the "eyedropper" tool to link the two tables together.

> *HINT: When drawing relationships between two tables, there must be a column of the same data type representing the same information in each entity.*

1.11: Select the "eyedropper" tool and select the AlbumID column in the Song table.
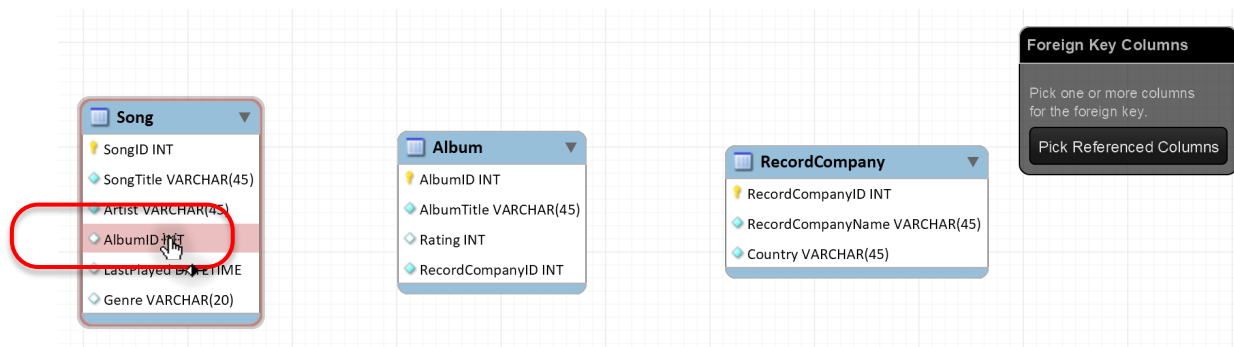
Figure 16: Select the AlbumID column of the Song table.

1.12: Then Select the AlbumID column in the Album table.
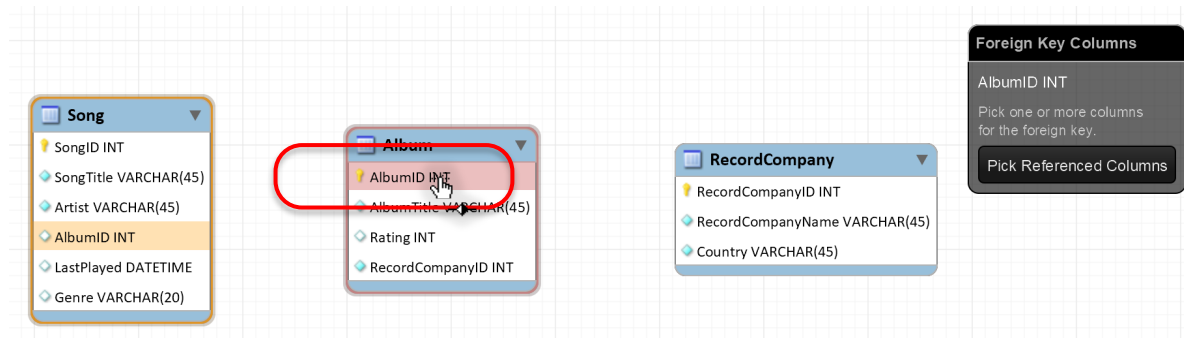
Figure 17: Select the Primary Key of the Album table.

Note: The two attributes used to create a relationship between the two tables must have exactly the same data type. However, they may have different names.
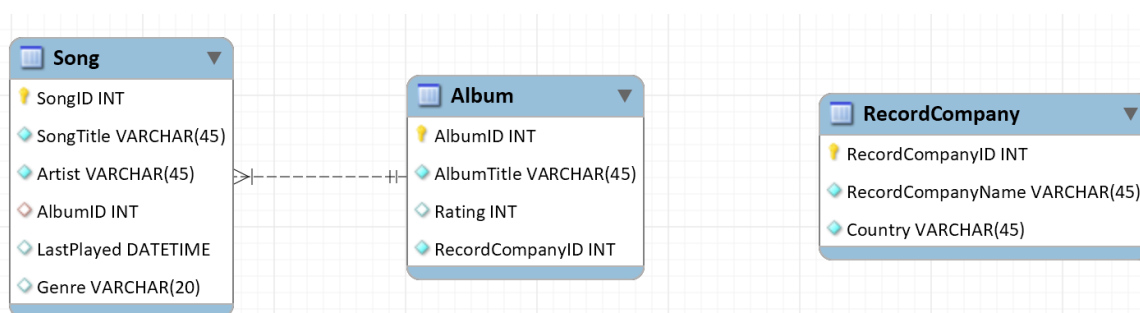
Figure 18: A relationship is now defined between the AlbumID in the Song table and the AlbumID in the Album table.

Note that the AlbumID diamond in the Song table has changed colour from blue to red. This indicates that this column is a "foreign key", used in linking the two tables.

1.13 Now repeat this step task by joining the RecordCompanyID column in the Album table to the RecordCompanyID column in the RecordCompany table.

Your end result should look like this:
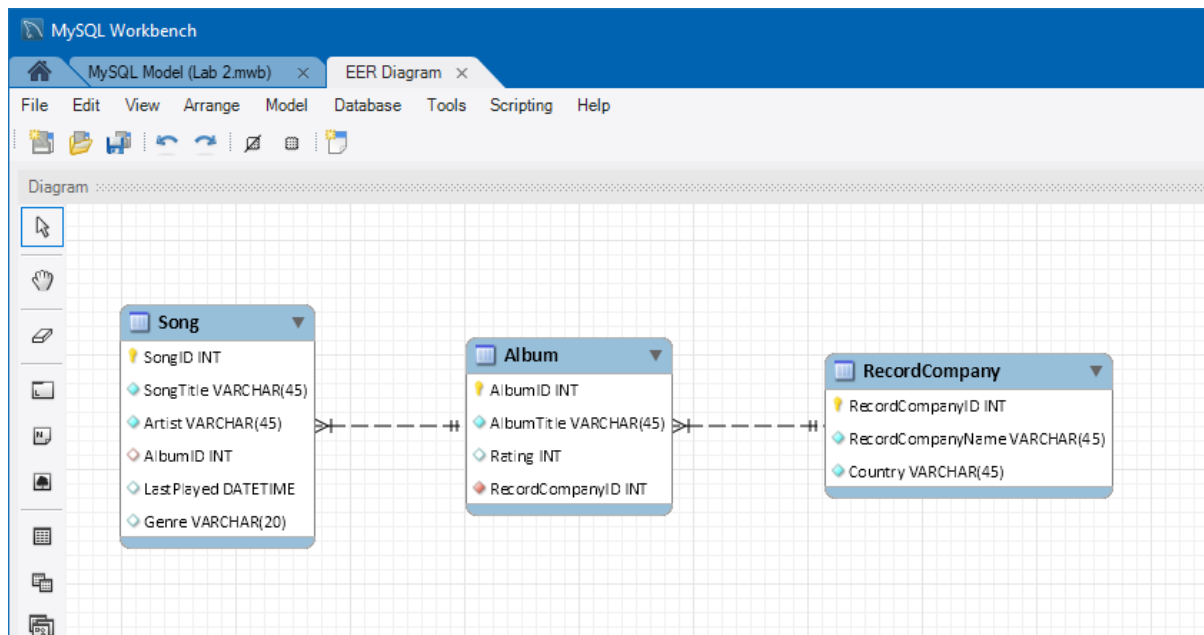
Figure 19: The finished relationships between the Song, Album and RecordCompany tables

1.14: Save your changes.

Adding relationship labels

We are now going to label the relationship between each table. Relationship labels substantially improve the readability of the models.

1.15: From the menu, select **Model** > **Model Options**.
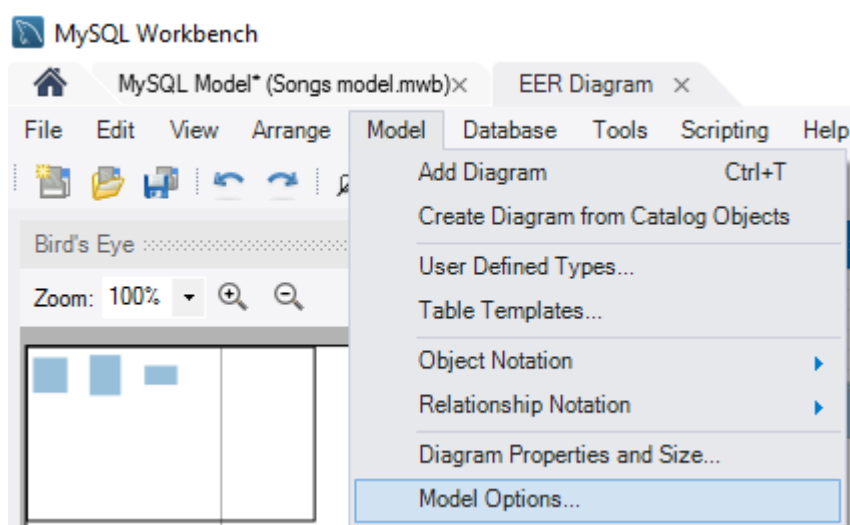
Figure 20: The Model Options menu

1.16: In the Model Options window, select "Diagram" and uncheck the "Use defaults from global settings" checkbox at the bottom of the window.
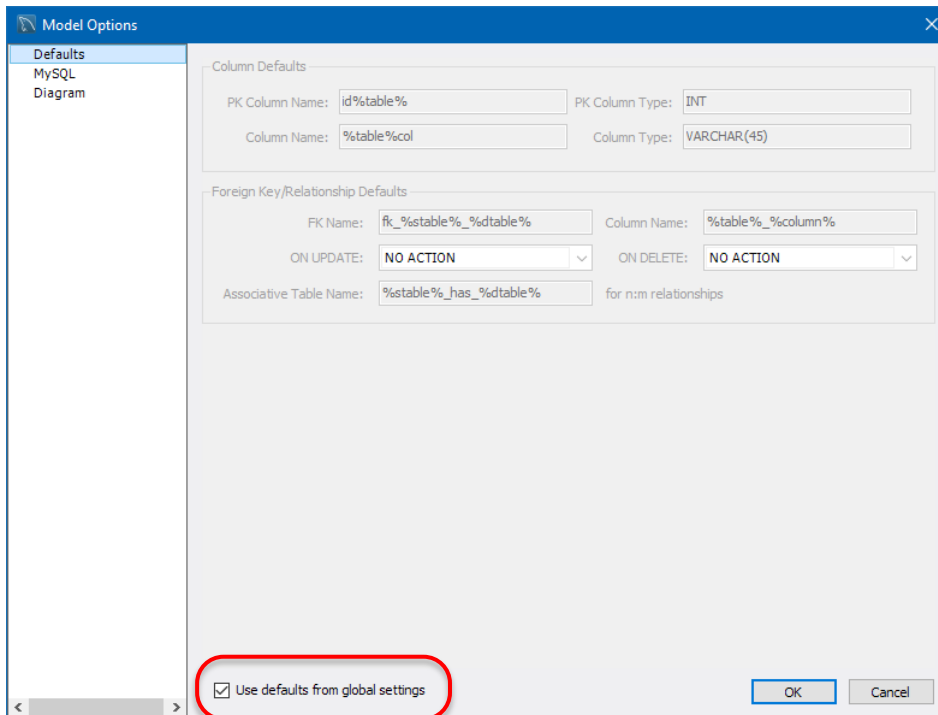
Figure 21: The Diagram menu in Model Options
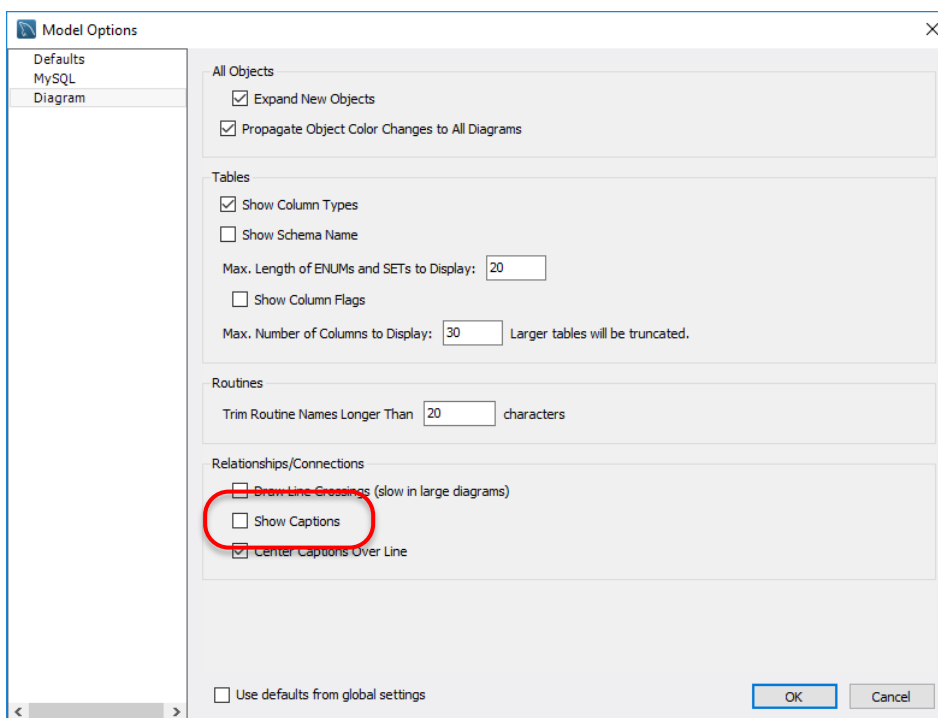
Figure 22: After unchecking the "Use defaults from global settings" checkbox, all the diagram options are now available to be altered.

1.17: Check the "Show Captions" checkbox in the "Relationship Connections" section of the

Diagram Model preferences. Then click OK.

The relationship connections are now labelled with default labels.
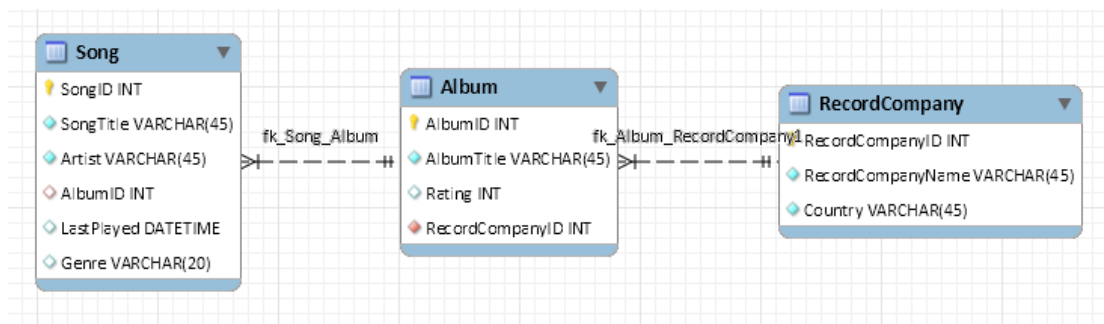
Figure 23: The relationships are labelled fk_Song_Album and fk_Album_RecordCompany1.

1.18: Double-click on the fk_Song_Album relationship label. The Relationship editor appears.
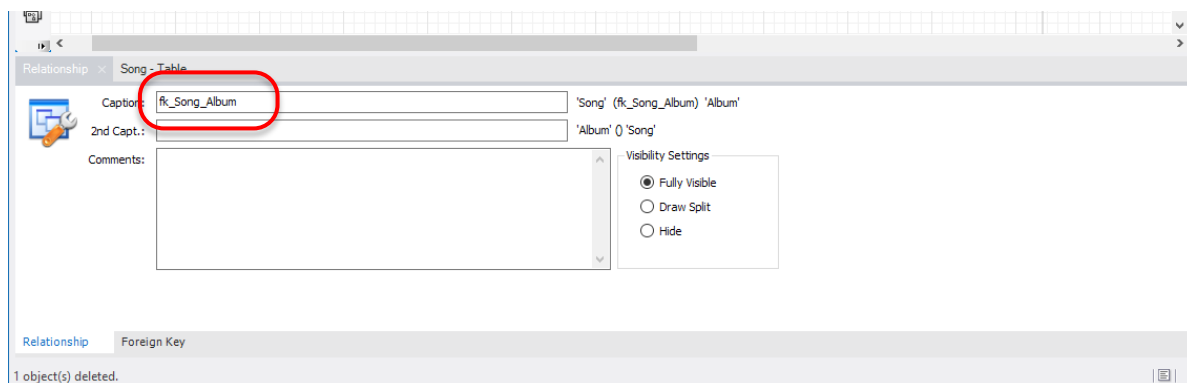
Figure 24: The Relationship editor

1.19: Change the "Caption" (circled in red in Figure 27) from 'fk_Song_Album' to 'recorded on' and close the Relationship editor.

The relationship between the Song and Album tables has been renamed.

1.20: Repeat the process for the other relationship, changing the name to 'released by'.

Your final diagram should look like this:
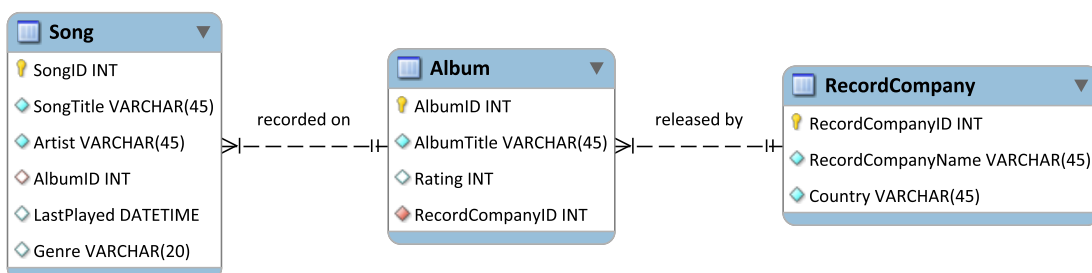
Figure 25: The final Music model

You have just modelled a simple music-related scenario with three entities (tables).

The model is not quite complete – it lacks participation constraints (an indication of whether the relationship is mandatory or optional). This will be covered in next week's lab.

1.21: Save your changes to your diagram. <u>You will need this model again next week</u>, so save it in a safe place!

If you are working on a lab PC, save your file to a location where you can retrieve it later, such as your H: drive. Files saved to the lab PC's C: drive are deleted on machine reboot.

Section 2 Learning SQL by example

Objectives

- Learn how to use the following SQL keywords: ORDER BY, LIMIT, GROUP BY, FORMAT, ROUND
- Learn how to join two tables together with INNER JOIN and NATURAL JOIN
- Learn to use maths Operators
- Learn how to use SQL Functions such as MAX, MIN, COUNT

ORDER BY

ORDER BY will return the rows in the order you specify not in the order they are stored.

```
SELECT DepartmentID, Name, Floor
FROM department;
```

However, if we wish to order by the floor in ascending order, we can do this

```
SELECT DepartmentID, Name, Floor
FROM department
ORDER BY Floor;
```

DepartmentID	Name	Floor
2	Books	1
6	Navigation	1
3	Clothes	2
7	Recreation	2
4	Equipment	3
5	Furniture	4
1	Management	5
8	Accounting	5
9	Purchasing	5
10	Personnel	5
11	Marketing	5
NULL	NULL	NULL

The default order is always in ascending order (0-9, A-Z) if you wish to reverse the default order use DESC keyword.

```
SELECT DepartmentID, Name, Floor
FROM department
ORDER BY Floor DESC;
```

DepartmentID	Name	Floor
1	Management	5
8	Accounting	5
9	Purchasing	5
10	Personnel	5
11	Marketing	5
5	Furniture	4
4	Equipment	3
3	Clothes	2
7	Recreation	2
2	Books	1
6	Navigation	1
NULL	NULL	NULL

LIMIT

We can also reduce the results displayed to users by using the LIMIT keyword. This will only return the *n* number of rows specified after LIMIT;

```
SELECT DepartmentID, Name, Floor
FROM department
ORDER BY Floor
LIMIT 2;
```

DepartmentID	Name	Floor
6	Navigation	1
2	Books	1
NULL	NULL	NULL

2.1 List the first name, last name and salary of the five highest salary earners across the Department store.

```
SELECT FirstName, LastName, Salary
FROM employee
ORDER BY Salary DESC
LIMIT 5;
```

OPERATORS and FUNCTIONS

Functions are mathematical and scientific calculations that are performed automatically by the database engine. There are several function types across all database data types. The most common functions we use are COUNT, MAX, MIN. The full list of Functions you can use in MYSQL are found [here in the MySQL reference manual](#)

To find out how many departments there are we can use the COUNT() function. Functions must be given something to act on which can be a column, or all columns using the wild card *

E.G.

```
SELECT COUNT(*)
FROM department;
```

	COUNT(*)
	11

```
SELECT COUNT(Name)
FROM department;
```

COUNT(Name)
11

```
SELECT  CONCAT(FirstName,' ',LastName , ' works in the ' ,
department.Name, ' Department') AS INFO
FROM employee NATURAL JOIN department;
```

*Note we did a **join** between two tables Employee and Department*
More about joining tables in the next lab.

INFO
Alice Munro works in the Management Department
Rita Skeeter works in the Books Department
Gigi Montez works in the Clothes Department
Maggie Smith works in the Clothes Department
Paul Innit works in the Equipment Department
James Mason works in the Equipment Department
Pat Clarkson works in the Furniture Department
Sanjav Patel works in the Navigation Department
Mark Zhang works in the Recreation Department
Todd Beamer works in the Accounting Department
Nancy Cartwright works in the Accounting Department
Brier Patch works in the Purchasing Department
Sarah Fergusson works in the Purchasing Department
Sophie Monk works in the Personnel Department
Ned Kelly works in the Marketing Department
Andrew Jackson works in the Marketing Department
Clare Underwood works in the Marketing Department

2.2 Type the SQL query to find the total number of employees in the employee table

Your result set should look like this

count(*)
17

```
SELECT COUNT(*)
FROM employee;
```

Alternatively:

```
Select COUNT(LastName)
FROM employee;
```

Group By

Sometimes we want to group the function by a particular attribute. For example to find out the number of each departments on each floor of the department store we would type:

```
SELECT Floor, COUNT(DepartmentID)
FROM department
GROUP BY Floor;
```

Floor	count(DepartmentID)
5	5
1	2
2	2
3	1
4	1

We use the GROUP BY keyword when aggregate functions are with a column that does not aggregate the rows. We must group by the non aggregated column or columns to ensure the full result set is returned. Thus in the above example we GROUP BY *floor.*

Try this: Remove the GROUP BY keyword and notice the difference in the query output

Alias

We can also alias the columns to make the output make more sense to the reader. Then use that alias within the query

```
SELECT Floor as DEPT_FLOOR, COUNT(DepartmentID) AS DEPT_COUNT
FROM department
GROUP BY DEPT_FLOOR
ORDER BY DEPT_FLOOR;
```

	DEPT_FLOOR	DEPT_COUNT
	1	2
	2	2
	3	1
	4	1
	5	5

2.3 Type the SQL query to find how many employees work in each department

	DepartmentID	Count(EmployeeID)
▶	1	1
	2	1
	3	2
	4	2
	5	1
	6	1
	7	1
	8	1
	9	3
	10	1
	11	3

```sql
SELECT DepartmentID, Count(EmployeeID)
FROM employee
GROUP BY DepartmentID;
```

2.4 Type the SQL query to find each department's average salary?

	DepartmentID	AVG(Salary)
	1	125000.000000
	2	45000.000000
	3	46000.000000
	4	43000.000000
	5	45000.000000
	6	45000.000000
	7	45000.000000
	8	68000.000000
	9	70333.333333
	10	75000.000000
	11	64000.000000

```sql
SELECT DepartmentID, AVG(Salary)
FROM employee
GROUP BY DepartmentID;
```

2.5 Type the SQL query that finds what department has the highest salary?

	DepartmentID	MAX(Salary)
	1	125000.00

```sql
SELECT DepartmentID, MAX(Salary)
FROM employee
GROUP BY DepartmentID
```

```
ORDER BY MAX(Salary) DESC
LIMIT 1;
```

2.6 Type the SQL query that finds the department with the lowest average salary?

DepartmentID	MIN(Salary)
4	41000.00

```
SELECT DepartmentID, MIN(Salary)
FROM employee
GROUP BY DepartmentID
ORDER BY MIN(Salary)
LIMIT 1;
```

Formatting & Rounding your results

FORMAT(X,D) and ROUND(X,D) are functions you can use to improve the readability of a query result. Round will round the argument – what is in the parenthesis "X" to D decimal places. Format will format the argument to D decimal places and include commas.

```
SELECT AVG(Salary) AS AVG_SAL
FROM employee;


60529.411765


SELECT FORMAT(AVG(SALARY),2) AS AVG_SAL
FROM employee;


60,529.41


SELECT ROUND(AVG(SALARY),2) AS AVG_SAL
FROM employee;


60529.41
```

Format and Round while producing the same output are subtly different. Format converts the output into a STRING (hence the 60<comma>529), whereas round keeps the result as a NUMBER

2.7 Find the first and last names of all the employees

```
SELECT FirstName, LastName
FROM employee
ORDER BY LastName;
```

FirstName	LastName
Todd	Beamer
Nancy	Cartwright
Pat	Clarkson
Sarah	Fergusson
Paul	Innit
Andrew	Jackson
Ned	Kelly
James	Mason
Sophie	Monk
Gigi	Montez
Alice	Munro
Brier	Patch
Sanjay	Patel
Rita	Skeeter
Maggie	Smith
Clare	Underwood
Mark	Zhang

JOINS

2.8 List each employee's full name and the department name they work in. Order the result by department name

This query requires you to join the Department table to the Employee table. Use the Physical data model to work out if you need to do a NATURAL JOIN or an INNER JOIN

```
SELECT Name, FirstName, LastName
FROM department NATURAL JOIN employee
ORDER by Name;
```

Alternatively, you could format the query for better readability

```
SELECT Name as Department_name,concat(FirstName, ' ',lastname) as
Employee_name
FROM department NATURAL JOIN employee
ORDER by Name;
```

Alternatively using an INNER JOIN

```
SELECT Name as Department_name,concat(FirstName, ' ',lastname) as
Employee_name
FROM department  INNER JOIN employee
ON department.DepartmentID = employee.DepartmentID
ORDER by Name;
```

Department_name	Employee_name
Accounting	Todd Beamer
Accounting	Nancy Cartwright
Books	Rita Skeeter
Clothes	Gigi Montez
Clothes	Maggie Smith
Equipment	Paul Innit
Equipment	James Mason
Furniture	Pat Clarkson
Management	Alice Munro
Marketing	Ned Kelly
Marketing	Andrew Jackson
Marketing	Clare Underwood
Navigation	Sanjay Patel
Personnel	Sophie Monk
Purchasing	Brier Patch
Purchasing	Sarah Fergusson
Recreation	Mark Zhang

2.9 Type a query to find the first and last name of all the employees in in the Management department. Then test your written SQL in MySQL Workbench

```
SELECT FirstName, LastName
FROM employee NATURAL JOIN department
WHERE Name = 'Management';
```
Your result set should look like this:

FirstName	LastName
Alice	Munro

2.10 List the Supplier name and the number of deliveries made to the department store

```
SELECT supplier.Name, count(delivery.DeliveryID) as numDeliveries
FROM supplier NATURAL JOIN delivery
GROUP by supplier.Name;
```

Name	Deliveries
All Points Inc.	3
All Sports Manufacturing	2
Global Books & Maps	3
Nepalese Corp.	3
Sao Paulo Manufacturing	4
Sweatshops Unlimited	1

In this SQL we have prefaced each column with its table name for readability. This format is always

`TABLENAME.COLUMNNAME`

END of LAB

Appendix New Department Store Physical ER Model

DeliveryItem
- DeliveryId INT
- ItemId SMALLINT
- DepartmentID SMALLINT
- Quantity TINYINT
- WholesalePrice DECIMAL(9,2)

Delivery
- DeliveryID INT
- SupplierID SMALLINT
- DeliveryDate DATE

Supplier
- SupplierID SMALLINT
- Name VARCHAR(25)
- Phone CHAR(10)

Department
- DepartmentID SMALLINT
- Name VARCHAR(50)
- Floor TINYINT
- Phone CHAR(10)
- ManagerID SMALLINT

Item
- ItemID SMALLINT
- Name VARCHAR(50)
- Type CHAR(1)
- Colour VARCHAR(20)
- ItemPrice DECIMAL(9,2)

Employee
- EmployeeID SMALLINT
- FirstName VARCHAR(50)
- LastName VARCHAR(50)
- Salary DECIMAL(8,2)
- DepartmentID SMALLINT
- BossID SMALLINT
- DateOfBirth DATE

SaleItem
- SaleId INT
- ItemId SMALLINT
- Quantity TINYINT

Sale
- SaleID INT
- DepartmentID SMALLINT
- SaleDate DATE

INFO90002
Department Store ER Model
V1.4 8th January 2018