# Task 1

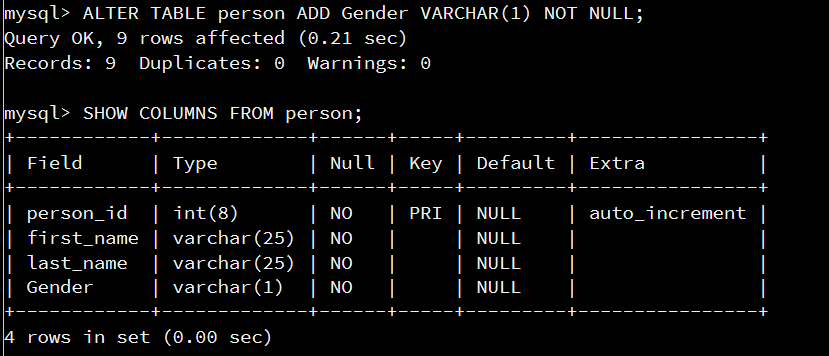
For this task, we are inserting the first and last name of the person table. With the records found after “Values”. Since we specified the first\_name, last\_name columns they will be inserted in that order.

**INSERT** **INTO** person **(**first\_name**,** last\_name**)** **VALUES** **(**'Austin'**,** 'Franklin'**),** **(**'Bob'**,** 'Johnson'**),** **(**'Billy'**,** 'Mallek'**);**



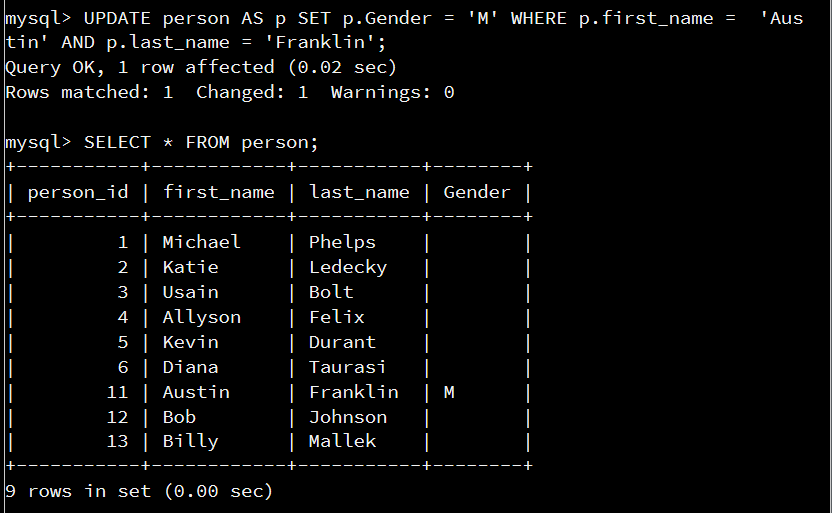
# Task 2

For this task we are altering the table “person” itself. In my case, we are adding “Gender” columns to the table that is required, cannot be null. It is a single character varchar requiring M or F.

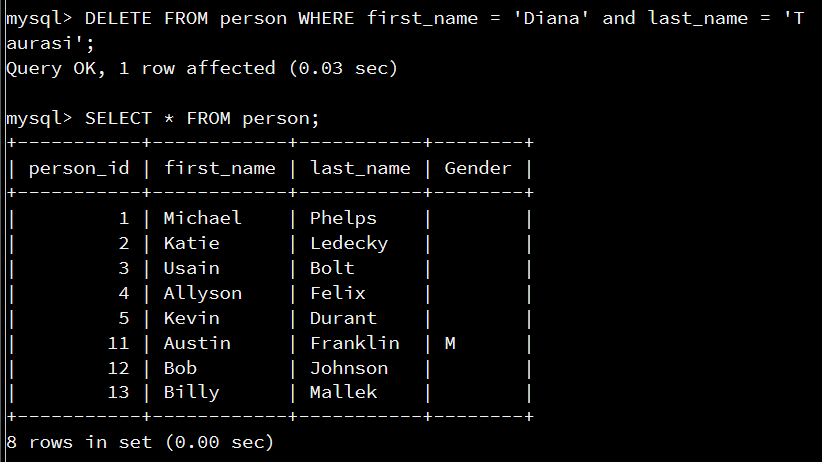
**ALTER** **TABLE** person **ADD** Gender **VARCHAR(**1**)** **NOT** **NULL;**

# Task 3

For this task we are updating the records on the “person” table. Setting the gender of my record to “M” for male.

**UPDATE** person **AS** p **SET** p**.**Gender **=** 'M' **WHERE** p**.**first\_name **=** 'Austin' **AND** p**.**last\_name **=** 'Franklin'**;**

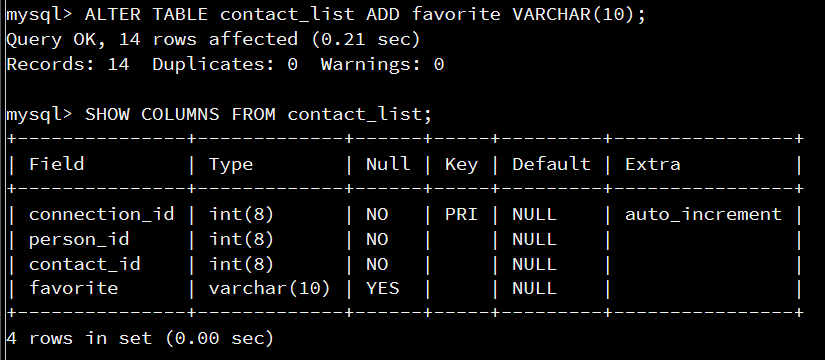
# Task 4

For this task, we are deleting the record from the “person” table where the first and last name are specified in the WHERE clause. This removes the record completely from the table.  
**DELETE** **FROM** person **WHERE** first\_name **=** 'Diana' **and** last\_name **=** 'Taurasi'**;**

# Task 5

In this task, we are again modifying a table itself. But we are altering the “contact\_list” table this time and adding a new column called favorite. This new column is not required and can be null. Also only supports characters up to 10 characters.

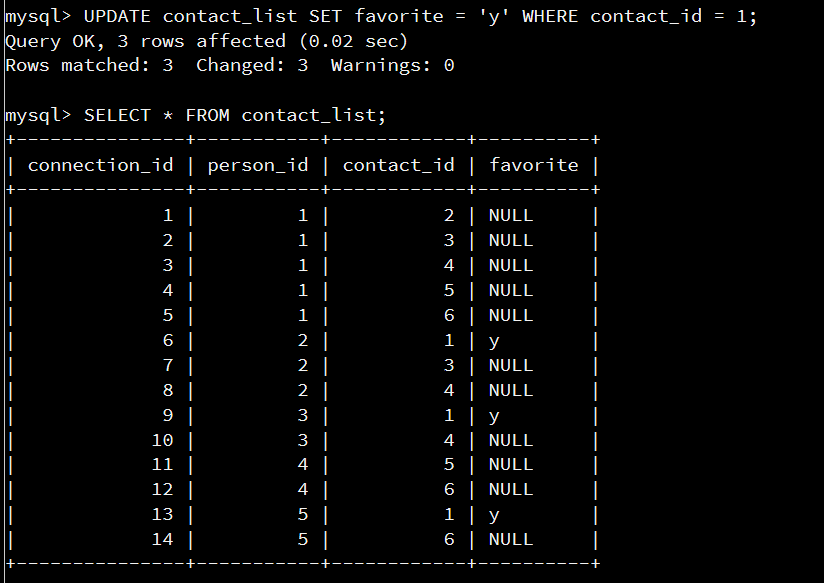
**ALTER** **TABLE** contact\_list **ADD** favorite **VARCHAR(**10**);**



# Task 6

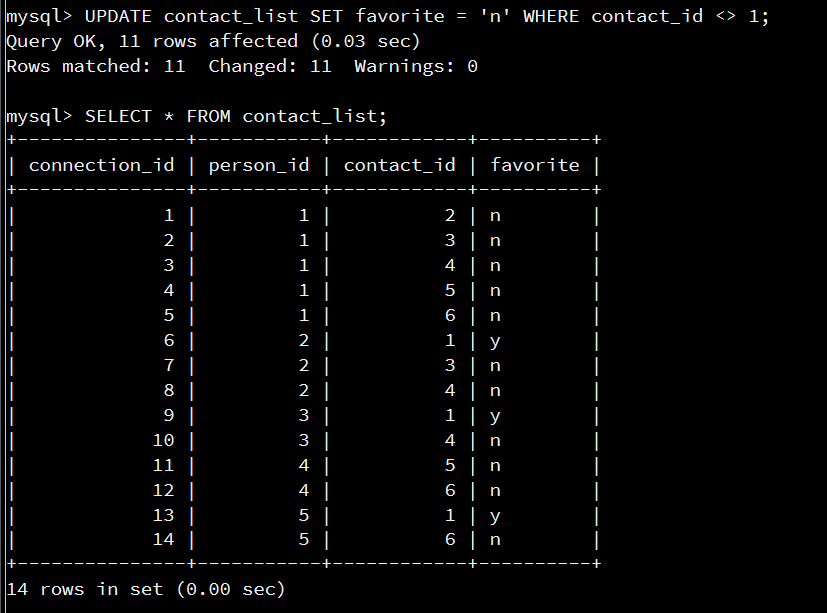
In this task we are updating the records in the “contact\_list” table and setting the favorite column to ‘y’ where the contact\_id on the record is 1.

**UPDATE** contact\_list **SET** favorite **=** 'y' **WHERE** contact\_id **=** 1**;**



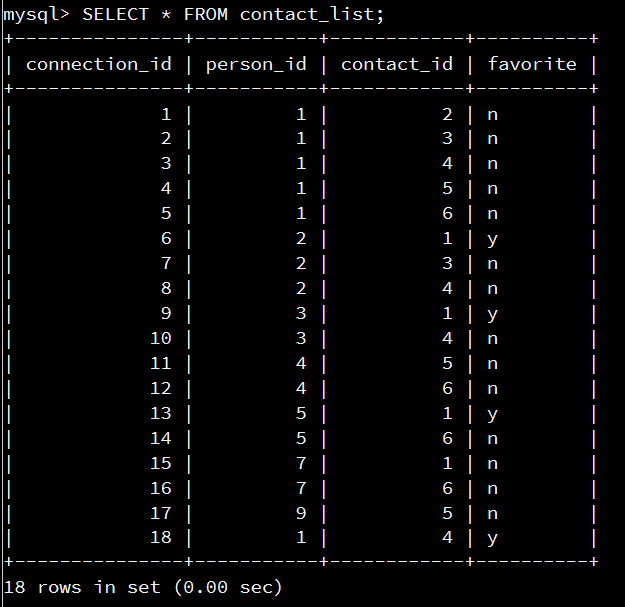
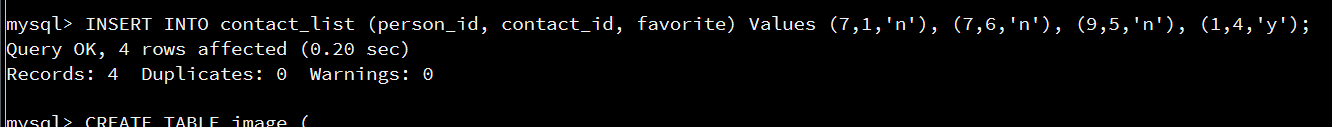
# Task 7

For this task, we are going onto the next step from the last update. Updating the same table but setting all of the other records where the countact\_id is not 1 to ‘n’. This updates all other records from the last step to ‘n’ in their favorite column.

**UPDATE** contact\_list **SET** favorite **=** 'n' **WHERE** contact\_id **<>** 1**;**

# Task 8

For this task, we are inserting records into the contact\_list table. Creating more person to contact ids. Since we created a favorite column in this table, we should to fill this value as well even if it isn’t required.

**INSERT** **INTO** contact\_list **(**person\_id**,** contact\_id**,** favorite**)** **Values** **(**7**,**1**,**'n'**),** **(**7**,**6**,**'n'**),** **(**9**,**5**,**'n'**),** **(**1**,**4**,**'y);

# Task 9

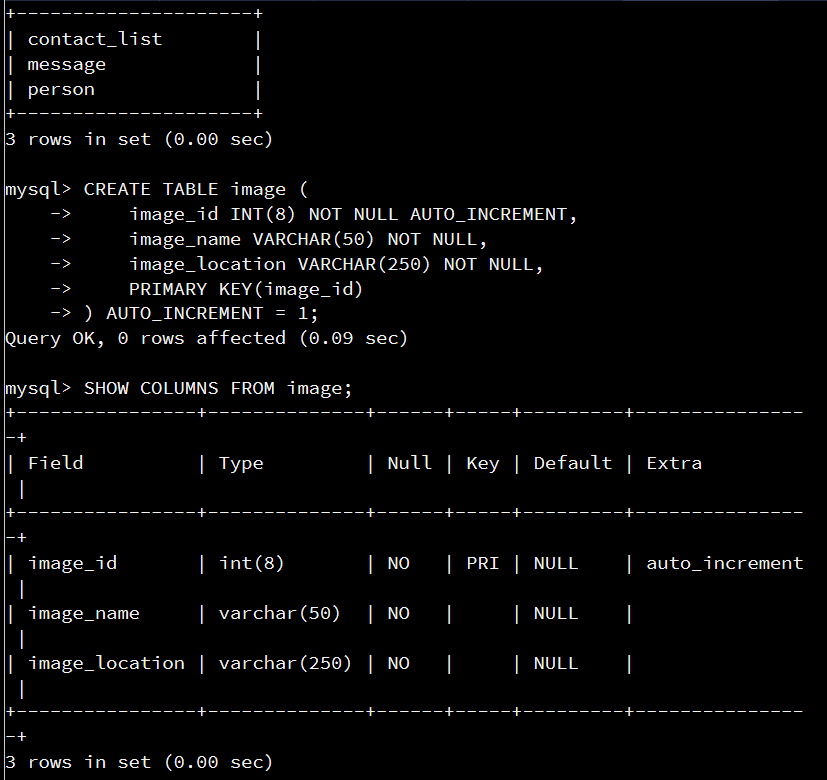
For this task we are creating a new table. We are specifying that the columns are image\_id, image\_name, and image\_location. We are automatically incrementing on each record insert into this table based on the primary key “image\_id”. None of the columns can be null. They are all images based on varchars.  
**CREATE** **TABLE** image **(**

image\_id **INT(**8**)** **NOT** **NULL** AUTO\_INCREMENT**,**

image\_name **VARCHAR(**50**)** **NOT** **NULL,**

image\_location **VARCHAR(**250**)** **NOT** **NULL,**

**PRIMARY** **KEY(**image\_id**)  
)** AUTO\_INCREMENT **=** 1**;**



# Task 10

Now that we created the image table, we create the transaction image table which maps between the image table and message table. We are creating the message\_image table which has message\_id and image\_id columns. These columns respectively have foreign keys pointing to the message table primary key and the image table primary key. This will prevent issues with invalid data being inserted into the tables.

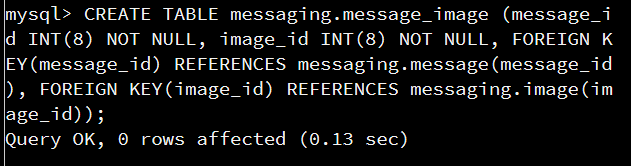
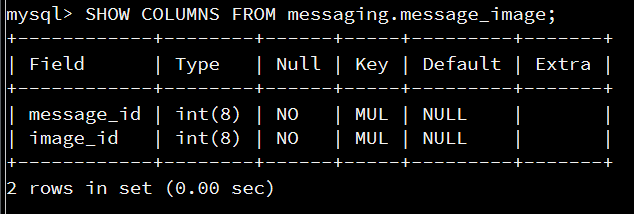
**CREATE** **TABLE** messaging**.**message\_image **(**

message\_id **INT(**8**)** **NOT** **NULL,**

image\_id **INT(**8**)** **NOT** **NULL,**

**FOREIGN** **KEY** **(**message\_id**)** **REFERENCES** messaging**.**message**(**message\_id**),**

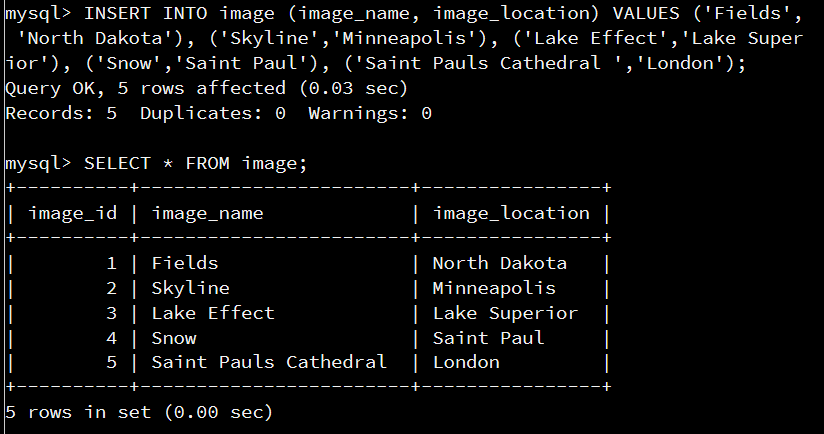
**FOREIGN** **KEY** **(**image\_id**)** **REFERENCES** messaging**.**image**(**image\_id**));**

# Task 11

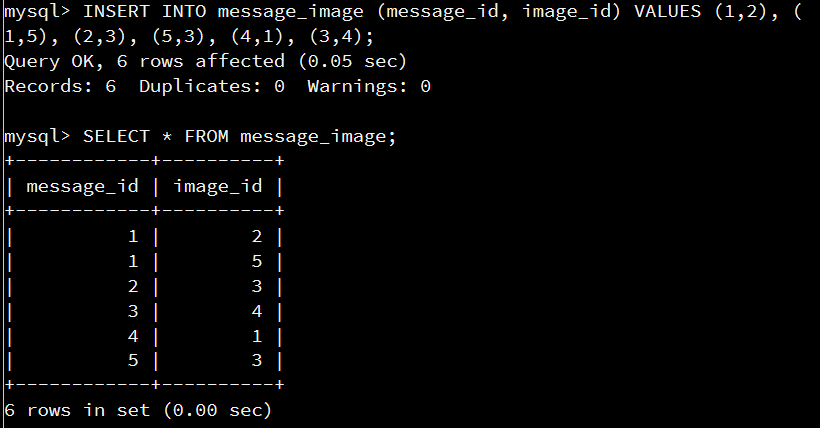
Now that we have our transaction table, we are inserting our images into the image table we created. This will provide data that we can then map to the message table through the message\_image table. Create 5 different locations, being the image name then the image location.

**INSERT** **INTO** image **(**image\_name**,** image\_location**)**   
**VALUES** **(**'Fields'**,** 'North Dakota'**),**   
**(**'Skyline'**,**'Minneapolis'**),**   
**(**'Lake Effect'**,**'Lake Superior'**),**   
**(**'Snow'**,**'Saint Paul'**),**   
**(**'Saint Pauls Cathedral '**,**'London'**);**



# Task 12

We are inserting records into the message\_image table. This is where the mapping of the message\_id on the message table to the image\_id on the image table is done. Now that this is complete we can query our data in the next tasks. Also see that message\_id = 1 has duplicates. Has two image\_ids associated.

**INSERT** **INTO** message\_image **(**message\_id**,** image\_id**)** **VALUES** **(**1**,**2**),** **(**1**,**5**),** **(**2**,**3**),** **(**5**,**3**),** **(**4**,**1**),** **(**3**,**4**);**

# Task 13

We are selecting all of the aliased records from the sender (person table), message, and receiver(person table again since those records are used for both sender and receiver). We are querying the records that are equal to Michael Phelps. We are joining these tables together based on the corresponding ids.

**SELECT**

sender**.**first\_name **AS** "SENDER\_FIRST\_NAME"**,**

sender**.**last\_name **AS** "SENDER\_LAST\_NAME"**,**

receiver**.**first\_name **AS** "RECEIVER\_FIRST\_NAME"**,**

receiver**.**last\_name **AS** "RECEIVER\_LAST\_NAME"**,**

message**.**message\_id **AS** "MESSAGE\_ID"**,**

message**.**message **AS** "MESSAGE"**,**

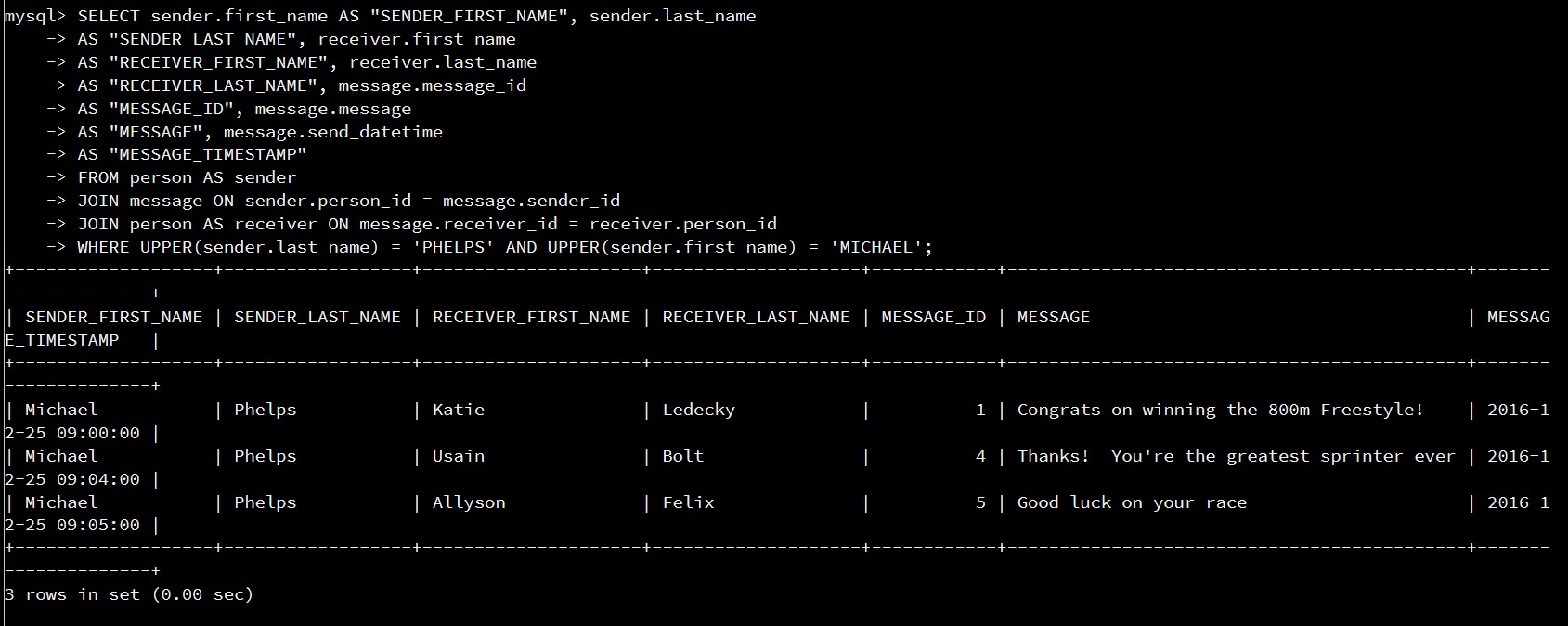
message**.**send\_datetime **AS** "MESSAGE\_TIMESTAMP"

**FROM** person **AS** sender

**JOIN** message **ON** sender**.**person\_id **=** message**.**sender\_id

**JOIN** person **AS** receiver **ON** message**.**receiver\_id **=** receiver**.**person\_id

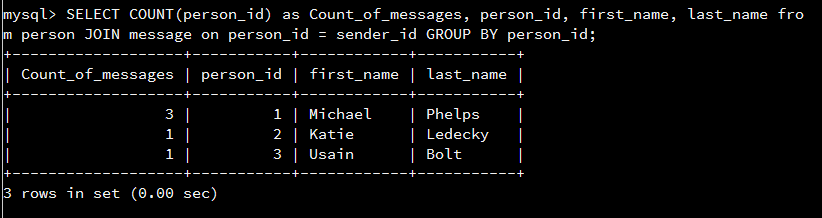
**WHERE** **UPPER(**sender**.**last\_name**)** **=** 'PHELPS' **AND** **UPPER(**sender**.**first\_name**)** **=** 'MICHAEL'**;**



# Task 14

We need to find the number of messages sent for every person.  
So we are doing a count on the person\_id record and defining that as Count\_of\_messages alias. If we do not aggregate these values, we will only get the number of messages sent total, not aggregated by the person sending it. So the returned records would be incorrect. Count of messages would be 5 (total messages on the tables), and return only Michael Phelps. Once we aggregate by the person\_id that we counted on, we will see the number of messages for each person id. Due to each message being a record, we run the function count which returns the number of records found.

**SELECT** **COUNT(**person\_id**)** **as** Count\_of\_messages**,** person\_id**,** first\_name**,** last\_name **from** person **JOIN** message **on** person\_id **=** sender\_id **GROUP** **BY** person\_id**;**



# Task 15

Now that we found the number of messages per person. We are then finding the messages that have at least one image attached. We are joining together the message table to the image table and the transaction table between. Since we are finding the messages with images between, our joins will query and return values that have images. So there is no need for a where clause as we are returning all messages with an image. Again we are joining by the id columns of each table required.

To show the duplicate images for single message:  
**SELECT** message**.**message\_id **AS** "MESSAGE\_ID"**,**

message**.**message **AS** "MESSAGE"**,**

message**.**send\_datetime **AS** "MESSAGE\_TIMESTAMP"**,**

image**.**image\_name **AS** "FIRST\_IMAGE\_NAME"**,**

image**.**image\_location **AS** "FIRST\_IMAGE\_LOCATION"

**FROM** person **AS** person **JOIN** message

**ON** person**.**person\_id **=** message**.**sender\_id **JOIN** message\_image **AS** message\_image

**ON** message**.**message\_id **=** message\_image**.**message\_id **JOIN** messaging**.**image

**ON** message\_image**.**image\_id **=** image**.**image\_id;

For the requested “Note:

For messages with multiple images, display only the first image fo

r the message

“ We need to aggregate by the message\_id on the message table. This will only show the first image found for each message\_id. Which essentially hides the additional images per message\_id. This query is shown below.

**SELECT** message**.**message\_id **AS** "MESSAGE\_ID"**,**

message**.**message **AS** "MESSAGE"**,**

message**.**send\_datetime **AS** "MESSAGE\_TIMESTAMP"**,**

image**.**image\_name **AS** "FIRST\_IMAGE\_NAME"**,**

image**.**image\_location **AS** "FIRST\_IMAGE\_LOCATION"

**FROM** person **AS** person **JOIN** message

**ON** person**.**person\_id **=** message**.**sender\_id **JOIN** message\_image **AS** message\_image

**ON** message**.**message\_id **=** message\_image**.**message\_id **JOIN** messaging**.**image

**ON** message\_image**.**image\_id **=** image**.**image\_id

**GROUP** **BY** message**.**message\_id**;**

