Brendan O’Connell

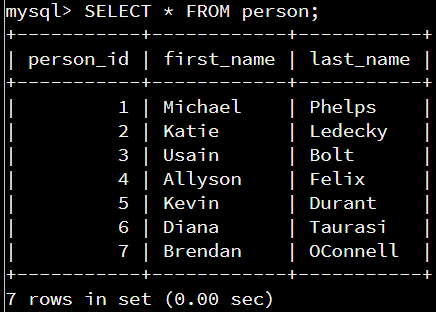
DAD-220-J1547-19EW1

**Final Project**

Prior to completing the tasks, I entered “USE messaging” to enter the *messaging* database

**Task One: Construct the SQL statement to add yourself to the Person table.**

SQL statement:  


Results:  


Explanation:

From the DML acronym “CRUD”, the C stands for CREATE and is represented by the SQL command **INSERT**. To add a new row of values to an existing data table, the INSERT INTO command is used, followed by the table name (person) and then the specific data columns in the order that the data will be entered. In this instance, it’s the (first\_name) and (last\_name) data columns. Once the data columns have been identified, the VALUES are entered. These values must match the data type for the specified columns.

**Task Two: Alter the Person Table**

SQL statement:



Results:

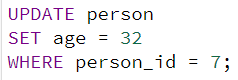


Explanation:

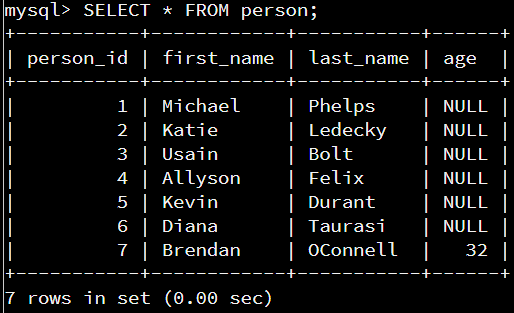
The ALTER TABLE statement is used to add, change, or drop data columns from an existing table. In this instance, the task requires a data column to be added, therefore the ALTER TABLE statement is entered followed by the table name. Then, the ADD statement is entered followed by the new column’s name and its data type, which in this case is an integer of 3 digits.

**Task Three: Update Records in the Person Table**

SQL statement:



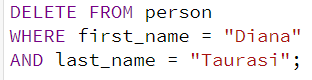
Results:



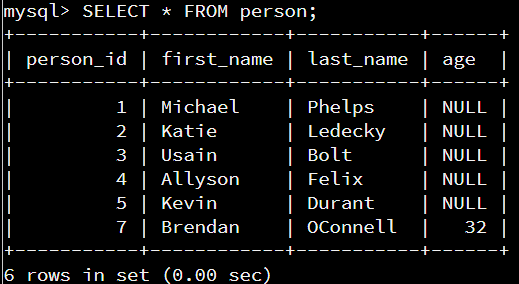
Explanation:

From the DML acronym “CRUD”, the U stands for UPDATE and is represented by the SQL command **UPDATE**. The UPDATE command is followed by the name of the table to be updated. The SET statement is followed by the data column to update, and the “=” operator is followed by the new data to reassign to that specific data column. To limit the change, the WHERE clause is used. In this instance, the parameter of the WHERE clause is the *person\_id* value of “7”.

**Task Four: Delete Records from Person Table**

SQL statement:  


Results:



Explanation:

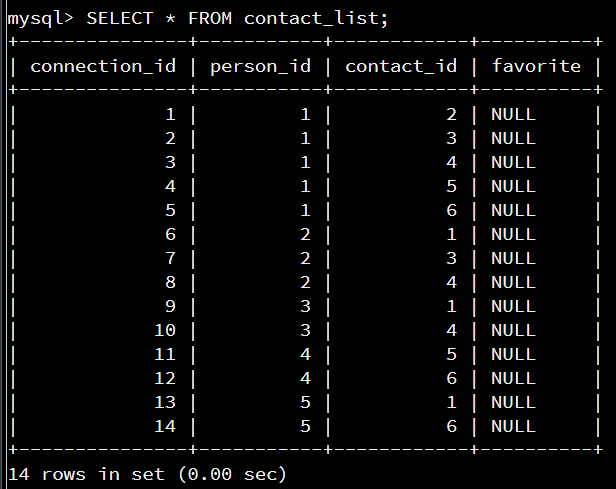
From the DML acronym “CRUD”, the D stands for DELETE and is represented by the SQL command **DELETE**. The command is followed by FROM and then the table name. It’s crucial to include a WHERE clause when deleting a value from a table, otherwise you will inadvertently delete all the data from that table. The WHERE clause here is set to two values by using the AND operator to include the first\_name AND last\_name data columns. Once executed, that specified row of data is no longer in the table.

**Task Five: Alter the Contact List Table**

SQL statement:



Results:

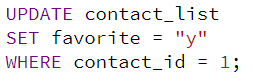


Explanation:

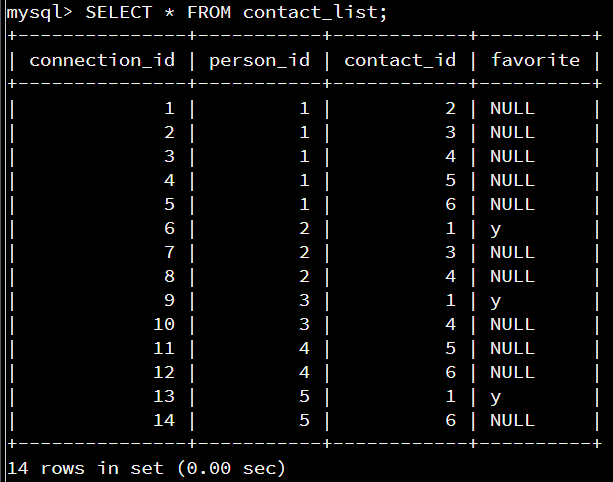
Like Task Two, the ALTER TABLE statement is used to add a column to an existing table, this time to the *contact\_list* table. A variable character data type of up to 10 characters is added, with the column name “favorite”.

**Task Six: Update Records in the Contact List Table**

SQL statement:



Results:

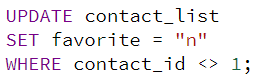


Explanation:

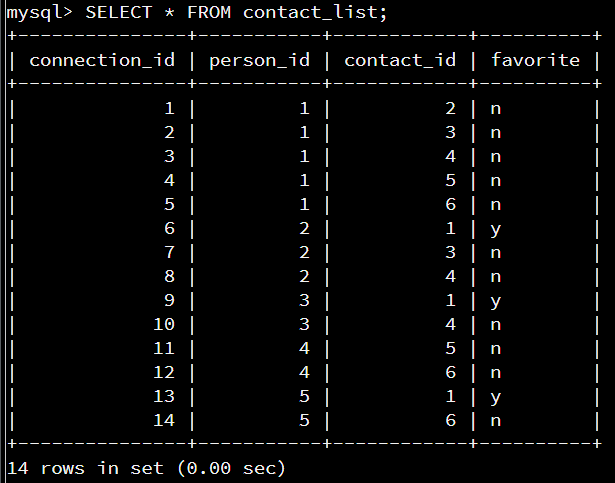
Like Task Two, the UPDATE command is used here to update data, but this time in the *contact\_list* table. The new column that was added in the previous task is being SET so that contact\_id #1 (Michael Phelps) is listed as a favorite contact (value = “y”). To ensure that only Michael Phelps is listed as “y”, the WHERE clause is used to limit this update to his contact\_id.

**Task Seven: Update Records in the Contact List Table**

SQL statement:



Results:

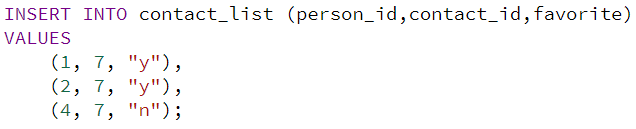


Explanation:

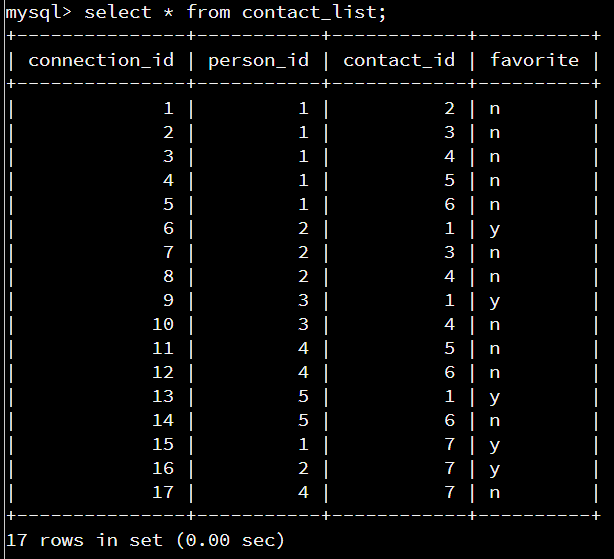
In contrast to the previous task, this task requires all remaining records to be listed as “n” in the *favorite* data column. To do this, the same code as the previous task is used except the SET value is “n” instead of “y”, and the WHERE clause utilizes the <> operator, signifying “not equal to”, and therefore limits the update to all contact\_id’s that are “not equal to” Michael Phelps.

**Task Eight:** **Insert Records to Contact List Table**

SQL statement:



Results:

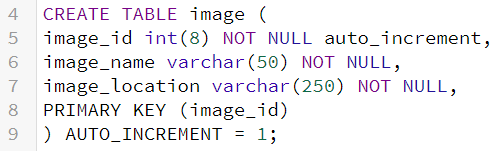


Explanation:

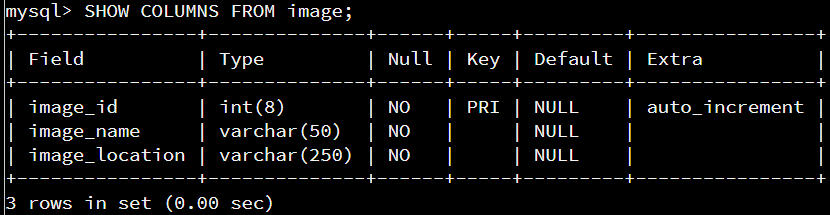
Like Task One, this task requires new values to be added to an existing table. However, this time three new rows of data will be added within the same SQL statement. The *connection\_id* column is set to auto\_increment, therefore doesn’t need to be included in the statement. The *person\_id* column is included and will be different for all three rows of data, whereas the *contact\_id* (which will also be included) will be the same for all three rows of data because the task requires that the *contact\_id* be me for all three. Lastly, the *favorite* column will be included as well, and I get to determine the value I want to enter.

**Task Nine: Create the Image Table**

SQL statement:



Results:

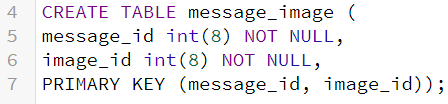


Explanation:

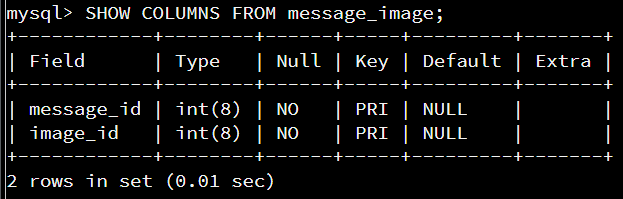
When creating tables, it’s necessary to specify the name of the table, the name of the columns, and the type of data that will be stored in each column. Additional constraints can also be included. For the *image* table, there are three columns created. The *image\_id* column is an 8 digit integer type that is set to auto\_increment starting with the value “1”. This means that for each additional entry (after the first entry), the *image\_id* column will increase by 1 number. For this reason, the *image\_id* data doesn’t need to be maintained when entering new rows of data. The *image\_name* column is a variable character type of 50 characters, and the *image\_location* column is a variable character data type of 250 characters. The PRIMARY KEY constraint is included here, and the *image\_id* is set as the primary key for this table. The primary key constraint is used to identify the individual records of the table, therefore the column assigned as the primary key must have unique values and cannot be NULL.

**Task Ten: Create the Message-Image Intersection Table**

SQL statement:



Results:

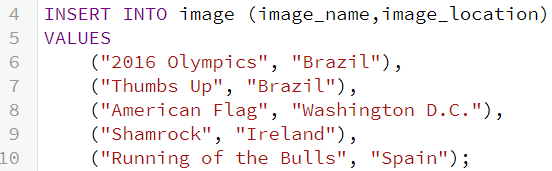


Explanation:

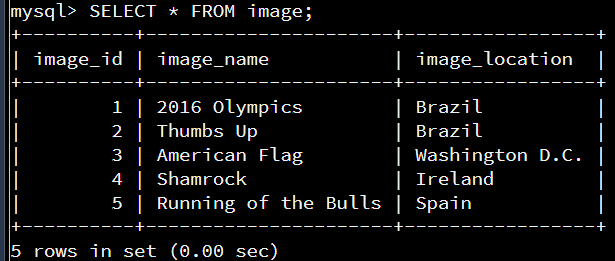
The purpose of this intersection table is essentially to identify messages that have images attached. The *message\_id* from the *message* table is linked here to the *image\_id* from the *image* table. It’s worth noting here that although there is still only 1 primary key, there are 2 columns within the primary key. Therefore, all entries in this table must be unique.

**Task Eleven: Insert Records to Image Table**

SQL statement:



Results:

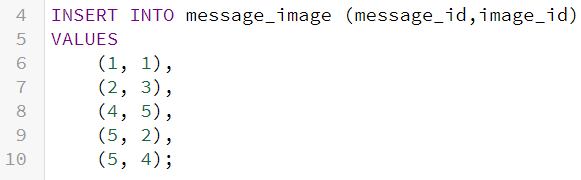


Explanation:

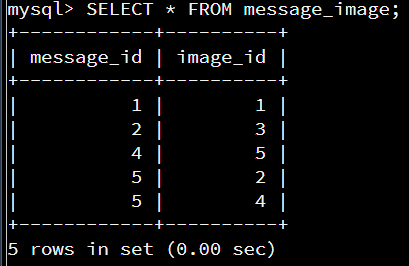
Same as Task Two and Task Eight, this is simply inserting new values into an existing table. The *image* table is what we created in Task Nine, and the VALUES entered here represent the name and location of the images that will eventually be linked to specific messages.

**Task Twelve: Insert Records to Message-Image Table**

SQL statement:



Results:

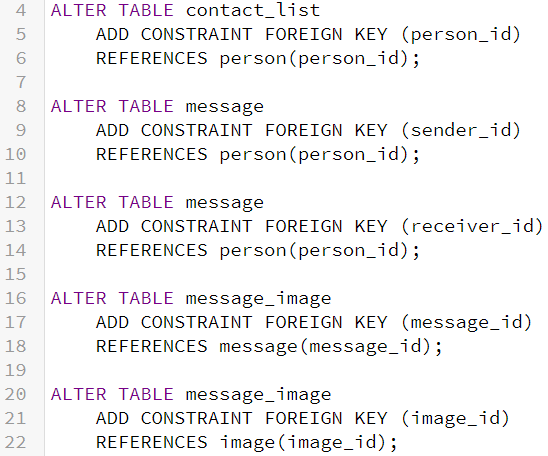


Explanation:

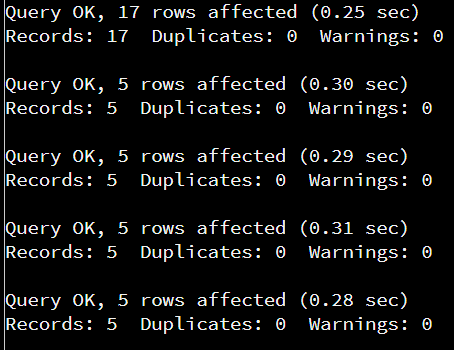
This is where the images inserted into the *image* table in Task Eleven are now linked to specific messages. Notice that all paired values in this table are unique. Although there are two entries for message\_id # 5, both entries have a unique image\_id.

**\*Establishing Foreign Key Constraints Prior to Query Tasks\***

SQL statement:



Results:



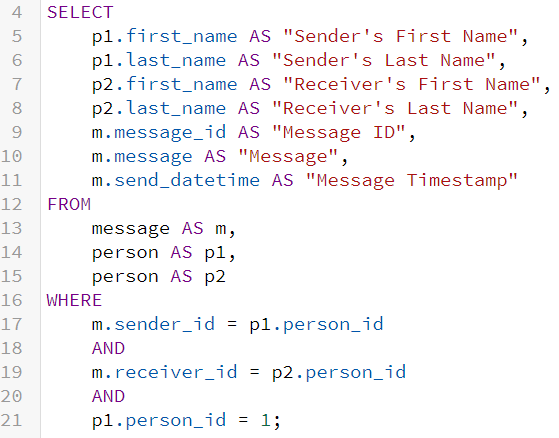
Explanation:

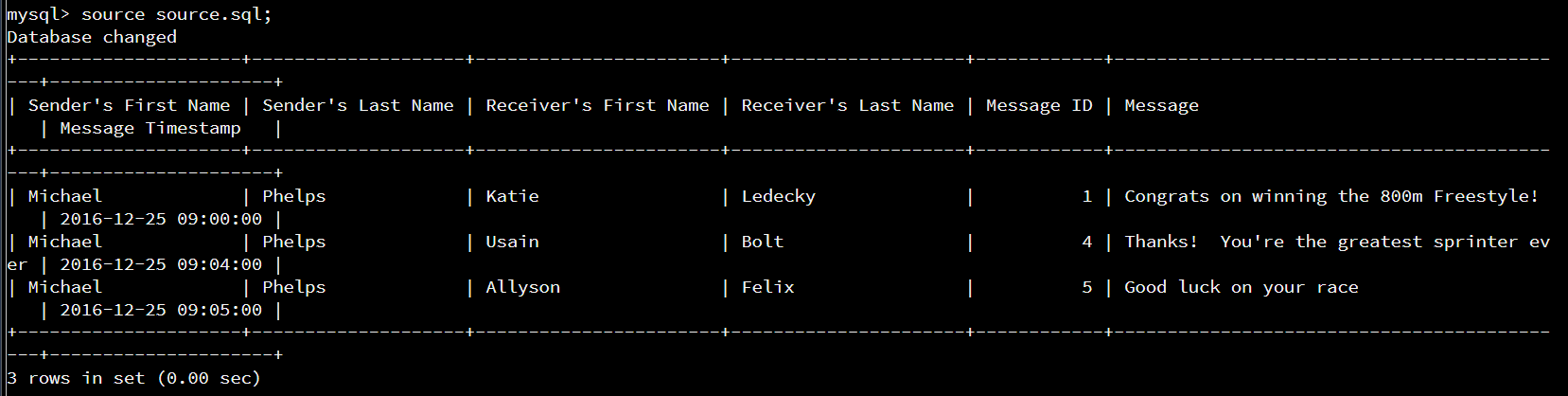
The Foreign Key constraint describes the relationship between data across multiple tables. By establishing these relationships, the reference columns are explicitly named to help illustrate the data columns that share the same data values across two or more tables. Using the database schema provided, I created foreign keys for all the data columns sharing the same values, with one exception. The *contact\_id* data column from the *contact\_list* table is unable to link to the *person\_id* data column from the *person* table. Although they are illustrated as relatable columns in the Final Project Database Schema, the data values changed for the *person* table on Task Four when “Diana Taurasi” was deleted from the table. Although she has been deleted from the *person* table, her i.d. value (6) is still listed in the *contact\_id* data column of the *contact\_list* table, therefore the *contact\_id* column cannot be linked to the *person\_id* data column. Doing so will result in the following error:



**Task Thirteen: Find All of the Messages that Michael Phelps Sent**

SQL statement:



Results:

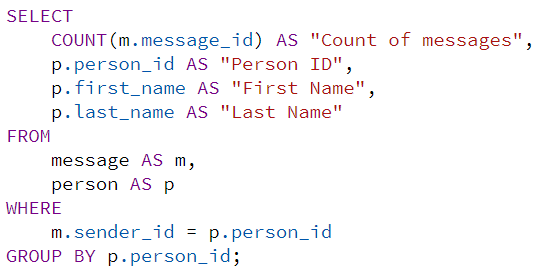
Explanation:

This task is the first of three query tasks, two of which require the WHERE clause to set the conditions. From the DML acronym “CRUD”, the R stands for READ and is represented by the SQL command **SELECT**. All the following queries start with the SELECT command, which will dictate the data columns to display. Within the SELECT command, I utilize the syntax table\_name.column\_name, and I also utilize aliases for both the table names and column names for multiple reasons. The first reason that I utilized aliases for table names is that it’s easier than writing out the table name every time I reference the table. The second reason I utilized aliases for the table names is that the *person* table will need to be referenced for two separate instances, and therefore I required two separate aliases. The reason I utilized aliases for the column names is because the task requested specific column names to be displayed for the query.

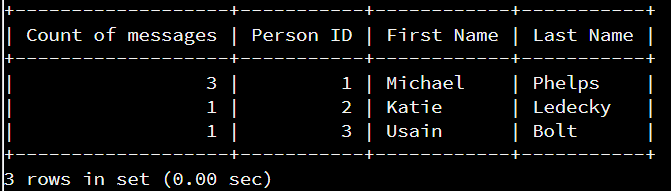
The FROM statement dictates where the data will be pulled from. This is also where I explicitly assigned the table aliases. There are two tables where the data is being pulled from, however, the *person* table will have data pulled for two separate references and therefore I created two separate aliases, p1 and p2. The WHERE clause is where the conditions for the query are set. The first condition describes the relationship between the *sender\_id* column from the *message* table and the *person\_id* column from the *person* table. This is the first reference for the *person\_id* column and therefore is matched with the p1 alias. The second condition describes the relationship between the *receiver\_id* column from the *message* table and the *person\_id* column from the *person* table. This is the second reference for the *person\_id* column and therefore is matched with the p2 alias. The last condition limits the query to only show messages sent by Michael Phelps, person\_id # 1, and since the condition requires the limit be applied to the sender, the p1 table is referenced to match the *sender\_id* established in a previous condition. All three conditions are connected via the AND operator.

**Task Fourteen: Find the Number of Messages Sent for Every Person**

SQL statement:



Results:



Explanation:

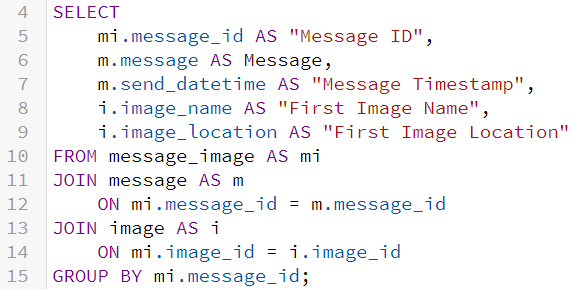
For this query, aggregate functions are now required. The COUNT() function is used to count the number of times a specified condition is met. The condition for this task is the “number of messages sent for every person”. There is a total of 5 messages sent, but only 3 total people sent them.

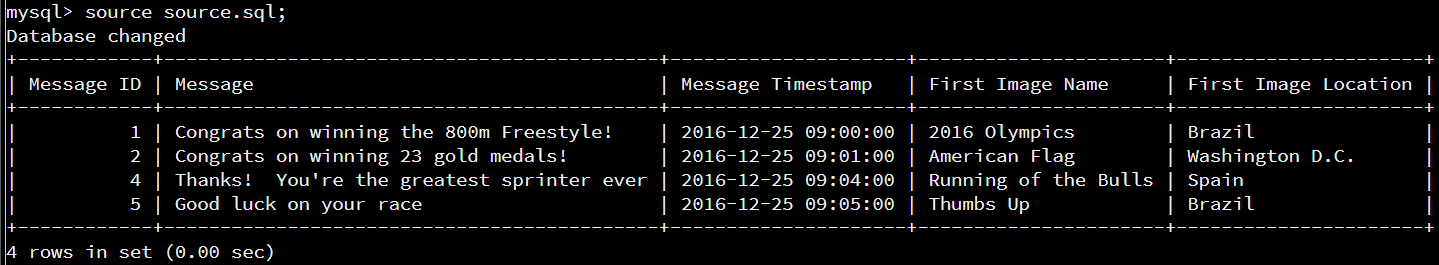
To start the query, the SELECT command is entered followed by the specified columns of data desired for this query. The first column is the aggregate function COUNT(m.message\_id), which will count the unique message id’s (aka all sent messages). An alias is implemented here for the aggregate function as well as the other column names.

The FROM statement determines where the data is being pulled from, and the table aliases are implemented within this statement. The WHERE clause sets the conditions for the query, setting the sender id equal to the person id from the person table. Lastly, the GROUP BY statement is used in conjunction with the aggregate function to group the result. Here, the results are grouped by the person\_id column from the person table.

**Task Fifteen** **: Find All of the Messages that Have At Least One Image Attached Using INNER JOINs**

SQL statement:



Results:

Explanation:

Again, the SELECT command initiates the query by defining the desired columns to be displayed. Each column is given an alias based on the task requirement and references its table alias. The FROM statement dictates the main table to pull the data from and assigns the table’s alias.

In contrast to the previous queries, this query utilizes the JOIN keyword to select data from multiple tables that have common data fields. The foreign keys established prior to the query section of this document describe relationships between specific columns that share the same data values but appear in different data tables. Two such relationships that will be utilized in this query are the *message\_id* columns and the *image\_id* columns.

The JOIN statement is followed by the table name to be joined (and its alias), then the ON statement which initiates the reference portion of the command (foreign keys). To JOIN the *message* table, the reference is described between the *message\_id* columns in the *message\_image* table and the *message* table. To JOIN the *image* table, the reference is described between the *image\_id* columns in the *message\_image* table and the *image* table. Lastly, the GROUP BY statement is used to aggregate the results by the *message­­\_id* field. By aggregating the results, the query only shows the first image for each message id, therefore if there’s a message id with more than 1 attached image we only see the first image. This satisfies the specific request of the task to only show the first image of each message.

**Verification**

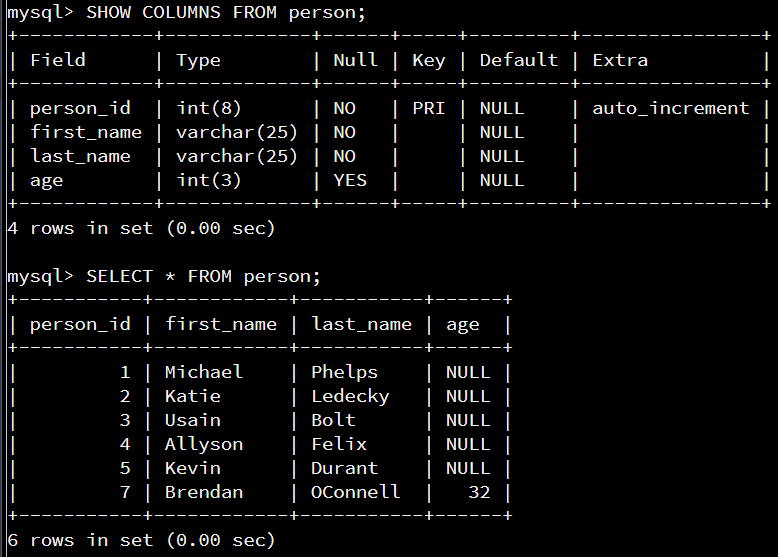
Table Name: person

Table Name: contact\_list

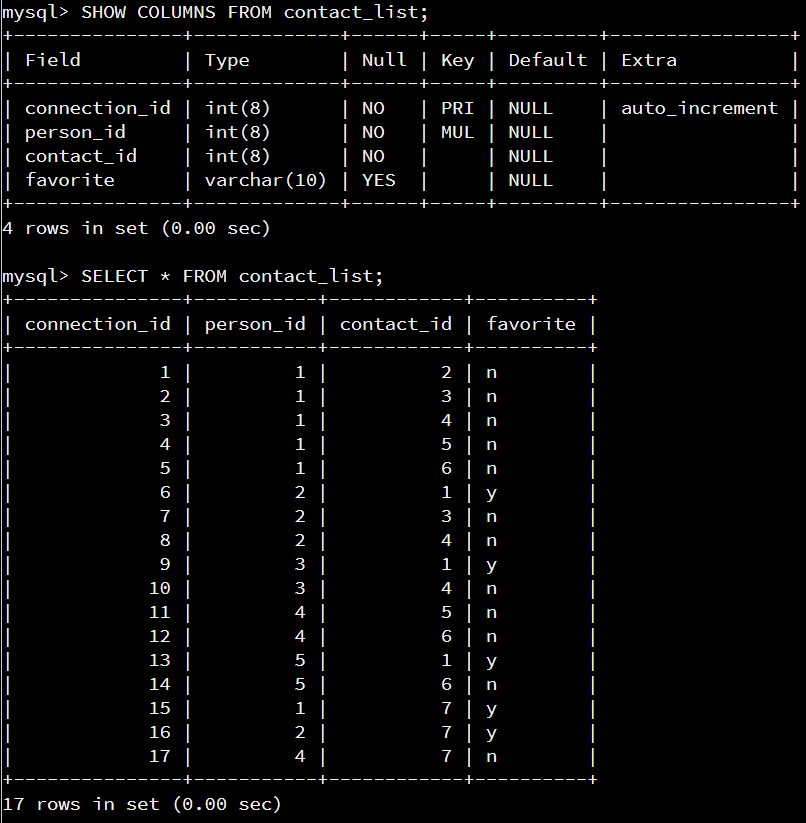


Table Name: message

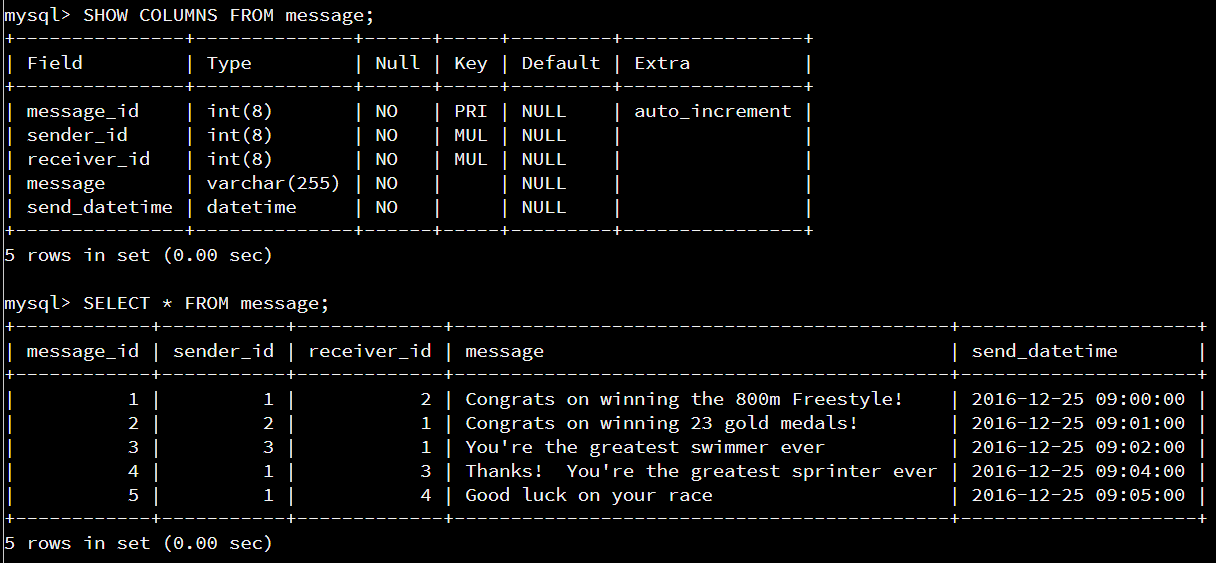


Table Name: image

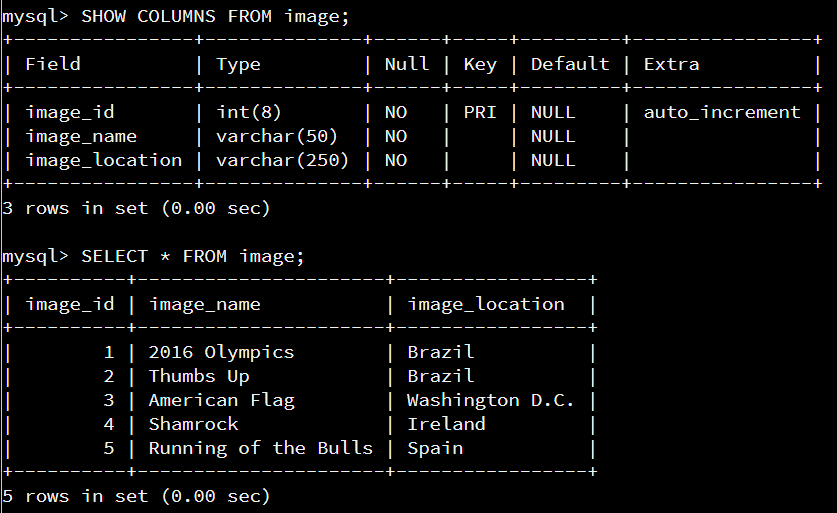
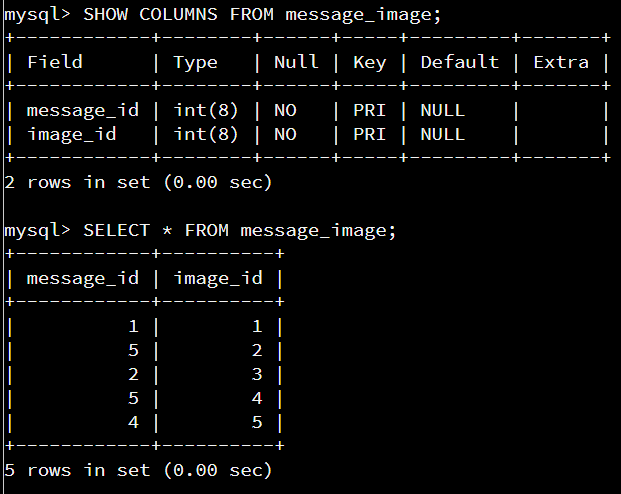


Table Name: message\_image



**References:**

1. CODIO
2. <https://www.w3schools.com/sql/sql_alias.asp>
3. <https://stackoverflow.com/questions/19241183/error-1452-23000-cannot-add-or-update-a-child-row-a-foreign-key-constraint-f?rq=1>