Final

Tobias J. O’Brien

Southern New Hampshire University

Author Note

This was prepared for DAD-220-Q4933 Introduction to SQL taught by Patrick Nieland

Final

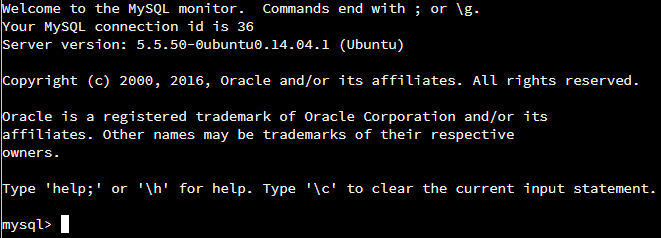
My name is Tobias O’Brien and I am a database engineer for SMCorp, a new startup offering social media applications. This documentation will cover the enhancement of the existing messaging application database to support two new features. The first feature will allow SMCorp to capture additional information about the users which will enable SMCorp to provide better targeted advertisements. The second feature will allow users to send one or more pictures in their messages. Along with these two features, new queries will be developed to provide SMCorp insight into how the new enhancements are being utilized.

Connecting to the Server

The first step is to open a connection to the MySQL server. This is done by running the following command:



It should be noted that if the server was remote you would also need to provide host information via the -h *hostname* option. Depending upon your MySQL server configuration, you may also need to provide authentication information via the -u *username* and -p *password* options. Once you are connected you should she information like the below. As seen, the server version, your connection id, and some basic commands are also displayed. (\h for help; \c to clear input)



Using the Database

To utilize the messaging database, we must change into the database by issuing the following command.



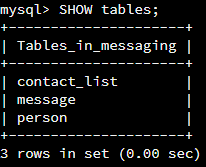
The result of the command shows that the database was changed.



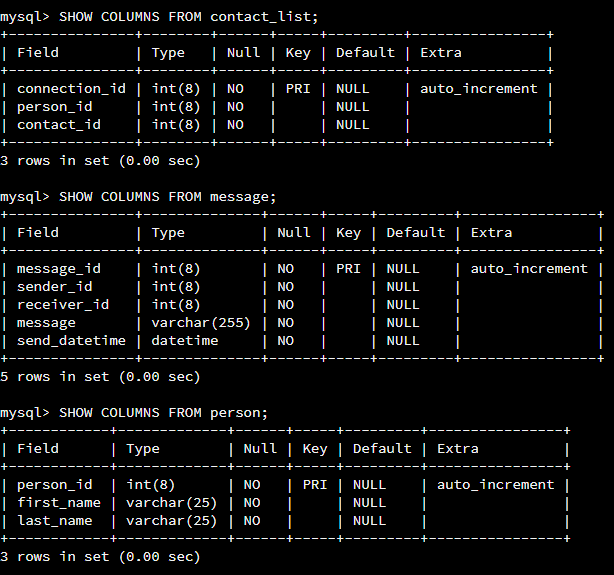
It should be noted that with UNIX type systems database and table names are case sensitive.

Reviewing the Database

To view the existing tables that are in the messaging database we can run the following command.



As seen in the output above there are three existing tables, contact\_list, message, and person. To evaluate the tables, the SHOW COLUMNS FROM command will be used as shown below. It should also be noted that the DESCRIBE <*tablename>* could also have been used as a quicker method. The difference between the two is that the SHOW COLUMNS FROM method has some additional options like filtering for specific columns. One additional option would be the SHOW CREATE TABLE <*tablename>* which will show the CREATE TABLE statement that creates the named table.

Insert Record to the Person Table

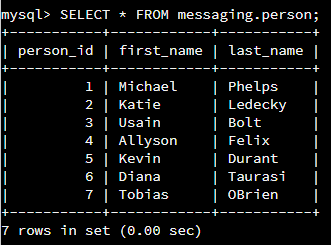
Per the requirements we need to insert a record containing our own personal information into the person table. We learned from the SHOW COLUMNS FROM person; output that the table has three required fields, person\_id, which will be auto assigned, frist\_name, and last\_name. Both the first\_name and last\_name fields are varchar and have a max limit of 25 characters. The below statement was utilized to add my personal information to the person table.



The results show that the insert was successful, took 0.03 sec, 1 row was affected and there were no warnings.



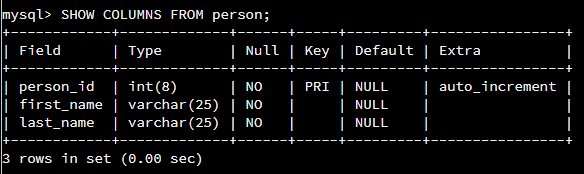
To see the results, the following statement was used to display the records.



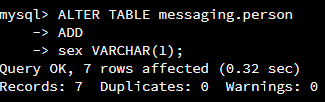
The command shows the records in the table, their values, and the time the command ran. As seen in the above my personal information was added as person\_id 7.

Altering the Person Table

Now that we have reviewed the Person table and added our own personal record, we need to add an additional column of our choice. The column that I have chosen is sex, the column will have a data type of VARCHAR, a max size of 1, and will not be required. As a refresher, the below show the existing person table setup.

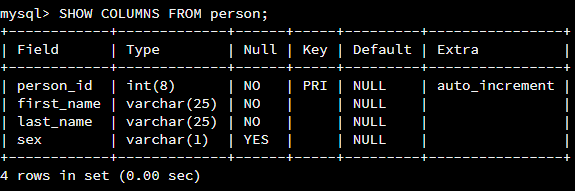


The below command was used to add the new column with the constraints that were noted.

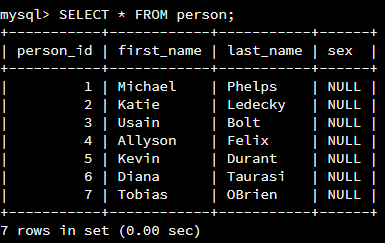


In the statement above the ALTER command was used to add the sex column to the messaging.person TABLE with a data type of VARCHAR and a size of 1. The results show that 7 rows were affected, the command took 0.32 sec to run and there were not any warnings.

Running the SHOW COLUMNS FROM statement shows the new column was created.

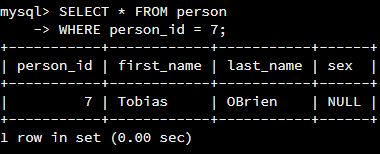


We can also see this by inspecting the table records with the SELECT statement as seen below.

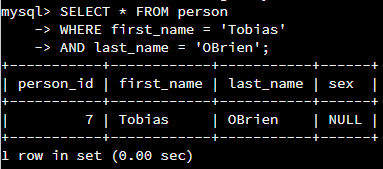


Update Records in the Person Table

Our requirements now state that we need to update the new column (sex) for our personal record that we created earlier. We will use the UPDATE and SET statement to focus on the correct table and column and will use the WHERE statement to ensure we are updating the correct record. As seen in the SEELCT \* FROM person; statement above we could utilize the person\_id or first\_name and last\_name fields in the WHERE statement. A safe method of ensuring you are utilizing the WHERE statement correctly is to first use the SELECT statement as show below.

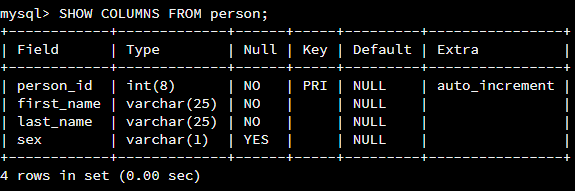


The above is the example of utilizing the person\_id column.

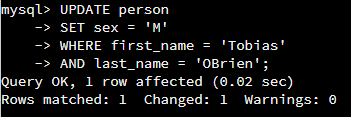


The above example utilizes the first\_name and last\_name columns.

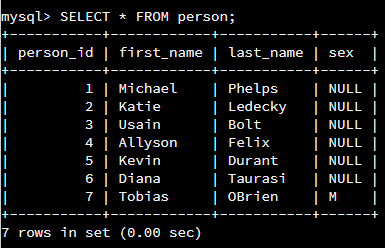
Now that we are sure our WHERE filter is correct we can write the complete statement to UPDATE the sex column for our personal record. In reviewing the SHOW COLUMNS FROM statement below we know that the sex column is VARCHAR and should have a size of 1.



Our statement to UPDATE the record is shown below.

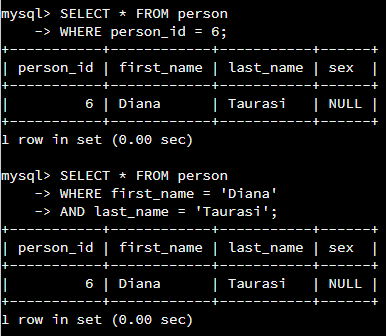


We can see from the results that the results were successful, 1 record was updated, the command took 0.02 sec to complete and there were no warnings. Running the SELECT command against the table show that the record was updated as well.

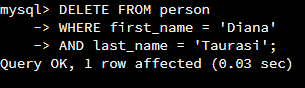


Delete Records in the Person Table

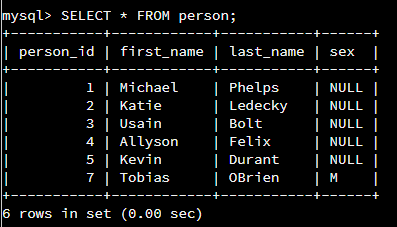
We are now required to DELETE records from the person table where the first name is Diana and the last name is Taurasi. To ensure we are deleting the correct records, similar filtering to the above will be utilized. As shown in the SELECT \* FROM person statement above we could either utilize person\_id or the first\_name and last\_name columns. First, we will utilize the SELECT and WHERE commands to ensure we are filtering for the correct records.



Now that we know our filtering is correct we can write the statement to DELETE the record. For the purposes of this exercise we will use the first\_name and last\_name columns.

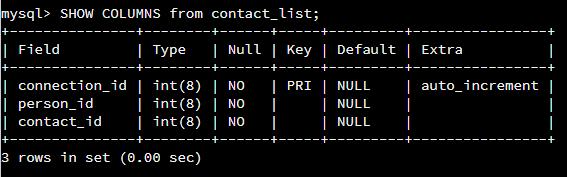


We can see from the results that the results were successful, 1 record was affected, the command took 0.03 sec to complete and there were no warnings. Running the SELECT command against the table show that the record was deleted.

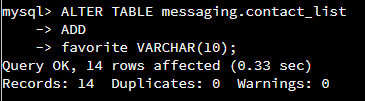


Alter the Contact List Table

We are now required to ALTER the contact\_list table to add a new column. The new column will be called favorite, will have a data type of VARCHAR, a max size of 10, and will not be required. As a refresher, the below show the existing contact\_list table setup.

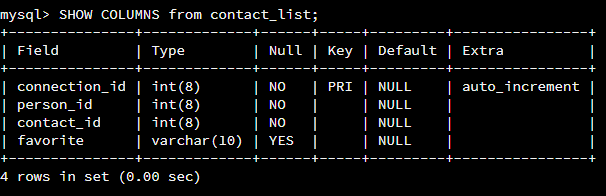


The below command was used to add the new column with the constraints that were noted.

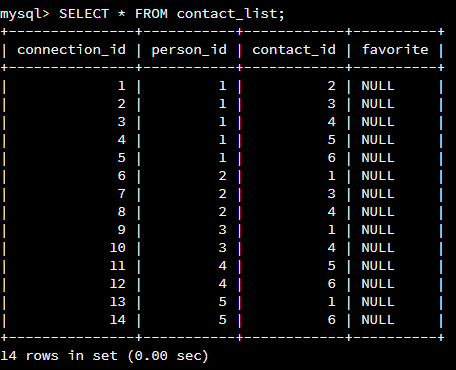


In the statement above the ALTER command was used to add the favorite column to the messaging.contact\_list TABLE with a data type of VARCHAR and a size of 10. The results show that 14 rows were affected, the command took 0.33 sec to run and there were not any warnings.

Running the SHOW COLUMNS FROM statement shows the new column was created.

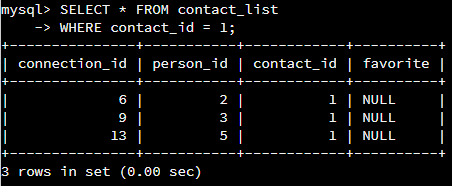


We can also see this by inspecting the table records with the SELECT statement as seen below.

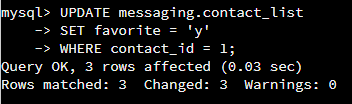


Update Records in the Contact List Table

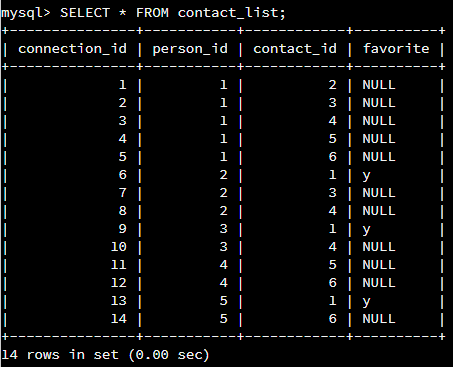
We are now required to UPDATE records in the contact\_list table to use the new favorite column that was just created. We will be setting records with Michael Phelps contact\_id (contact\_id = 1) to have favorite = y. We will use the UPDATE and SET statement to focus on the correct table and column and will use the WHERE statement to ensure we are updating the correct record. Like before, we can ensure we have the correct filtering in place by testing with the SELECT command.



Now that we know the WHERE filter is correct we can execute the statement to UPDATE the records.

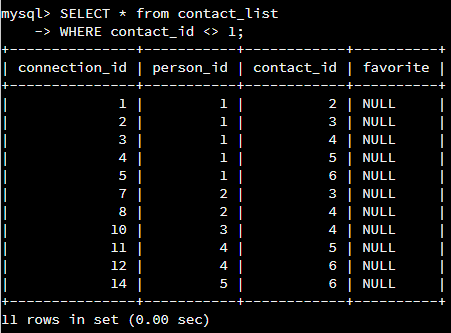


We can see from the above that the statement executed without error, 3 rows were affected, there were no warnings, and it took 0.03 sec. Using the SELECT command we can see that the three records were updated correctly.

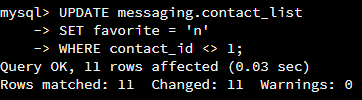


Update Records in the Contact List Table

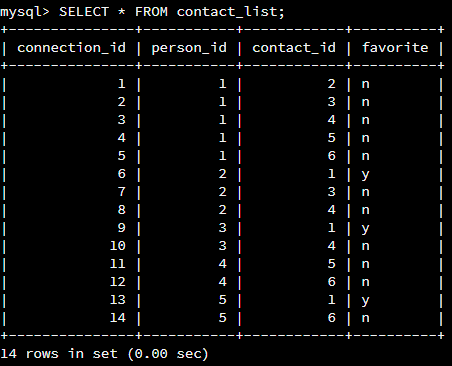
We are now required to UPDATE the remaining records in the contact\_list table to use the new favorite column that was just created. We will be setting records without Michael Phelps contact\_id (contact\_id <> 1) to have favorite = n. We will use the UPDATE and SET statement to focus on the correct table and column and will use the WHERE statement to ensure we are updating the correct record. Like before we can ensure we have the correct filtering in place by testing with the SELECT command.



Now that we know the WHERE filter is correct we can execute the statement to UPDATE the records.

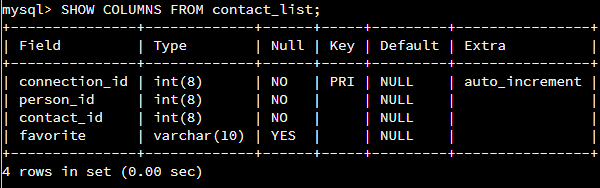


We can see from the above that the statement executed without error, 11 rows were affected, there were no warnings, and it took 0.03 sec. Using the SELECT command we can see that the three records were updated correctly.

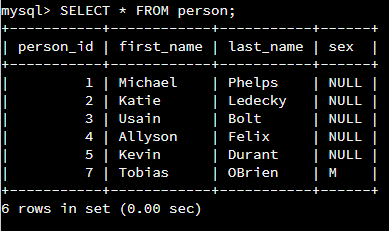


Insert Records in the Contact List Table

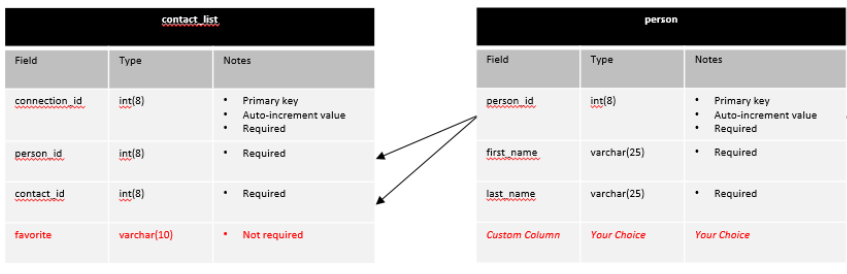
We are now required to add at least three new records into the contact\_list table. The records must be our personal record that was created and must include the favorite column that was recently created. In reviewing the contact\_list table using the SHOW COLUMNS FROM <*tablename>* statement as seen below we know that the connection\_id column will auto increment, the person\_id and contact\_id columns are data type INT and have a MAX size of 8, and that the favorite column is data type VARCHAR with a max size of 10.



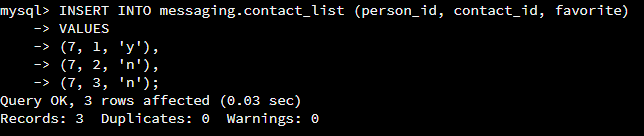
We can now review the person table to ensure we understand the data prior to inserting new records into the contact\_list table. This can be accomplished by using the SELECT command.



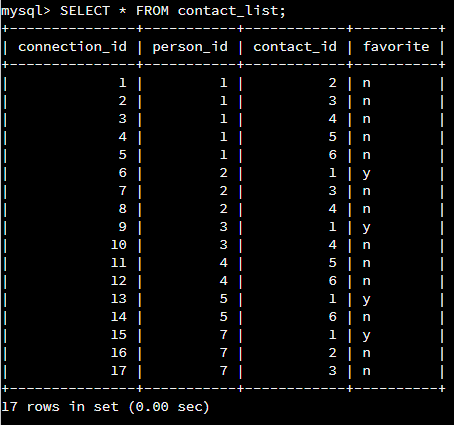
The records we would like to create will come from person\_id 7 as this is my personal records and will be sent to person\_id 1,2, and 3. As seen in the database schema below the person\_id column is used for both the person\_id and contact\_id columns in the contact\_list table. We will also set the record where Michael Phelps (person\_id/ contact\_id 1) to have a value of y in the favorite columns. All other records that will be created will have value of n for favorite.



The below command was used to insert the records as described above.

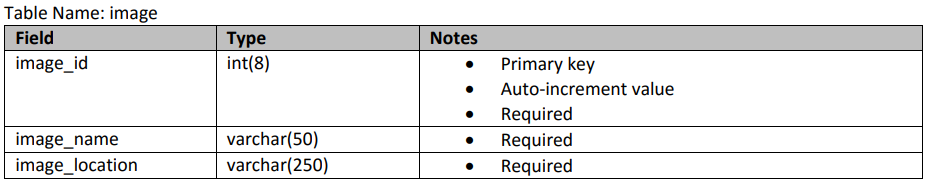


We can see from the above that the statement executed without error, 3 rows were affected, there were no warnings, and it took 0.03 sec. Using the SELECT command we can see that the three records were inserted correctly.

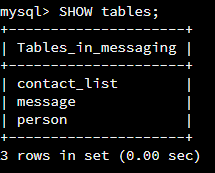


Create the Image Table

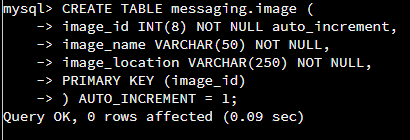
We are now required to create the image table according to the specification noted below. As seen in the table we are being asked to create three columns. The first column is image\_id and will be of data type INT, have a max size of 8, is required, and will be the primary key. The second column is image\_name, and will be of data type VARCHAR, have a max size of 50, and is required. The last column is image\_location and will have a data type of VARCHAR, have a max size of 250, and is required.



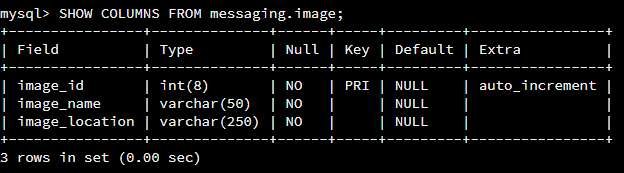
To ensure the table doesn’t exist already the following command was ran to see the table structure.



To create the table with the required specification the following statement was used.

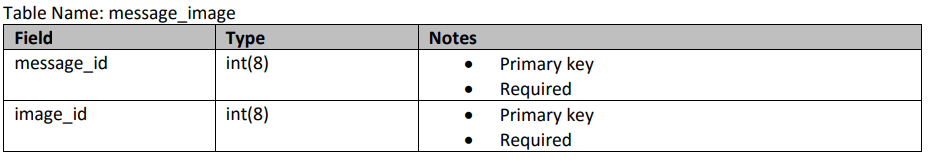


We can see from the above that the statement executed without error, 0 rows were affected, there were no warnings, and it took 0.09 sec. Using the SHOW COLUMNS FROM messaging.image statement we can confirm that the table was created with the correct specifications.

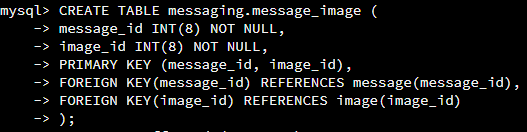


Create the Message-Image Intersection Table

To fulfill the next requirement, we need to create the message-image intersection table. This table will hold both the message\_id and image\_id columns.



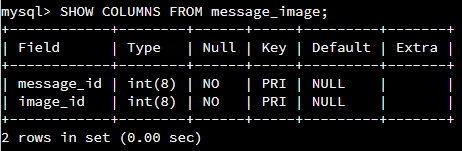
To create the table, the following statement was executed.



In the statement above, the main CREATE command is used to create the message\_image table and the additional statements are utilized to create the columns and define their specifications. As seen, the message\_id and image\_id columns were specified to be the first column and second column, have a data type of integer (INT), a max size of 8 digits, to NOT be NULL (or is required). The PRIMARY KEY (message\_id, image\_id) statements define message\_id and image\_id as the primary keys for the table. Along with the two primary keys, FOREGIN KEYS were used and referenced the PRIMARY KEYS in the message and image tables. This resulting intersection table assist in the defining of the many to many relationships that may exist between the message and image tables. The results of the above, shown below, show that the execution time was 0.20 secs and that the query did not produce any errors.

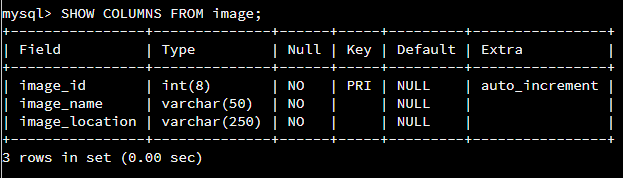


To verify the columns and specifications were created as specified, the SHOW COLUMNS FROM message\_image statement was executed.

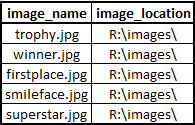


Insert Records to Image Table

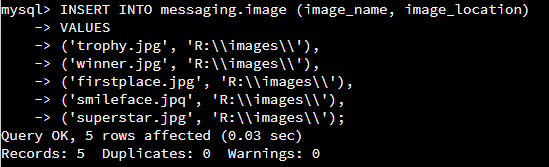
We are now required to add 5 new records into the image table. In reviewing the image table using the SHOW COLUMNS FROM <*tablename>* statement as seen below we know that the image\_id column will auto increment, the image\_name and image\_location columns are data type VARCHAR and have a MAX size of 50 and 250.



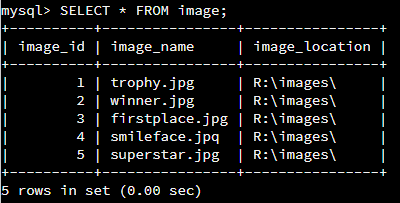
The following records will be created.



The below command was used to INSERT the records as described above.

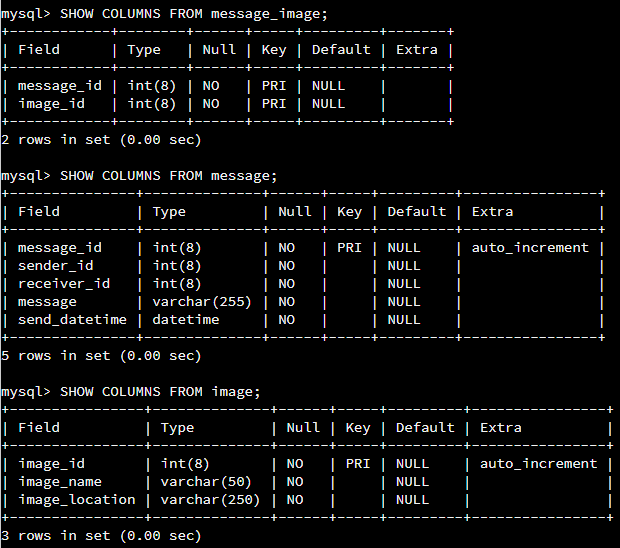


We can see from the above that the statement executed without error, 5 rows were affected, there were no warnings, and it took 0.03 sec. Using the SELECT command we can see that the three records were inserted correctly. As seen above the escape character ‘\’ was used to allow for the usage of the backslash in the image\_location column.

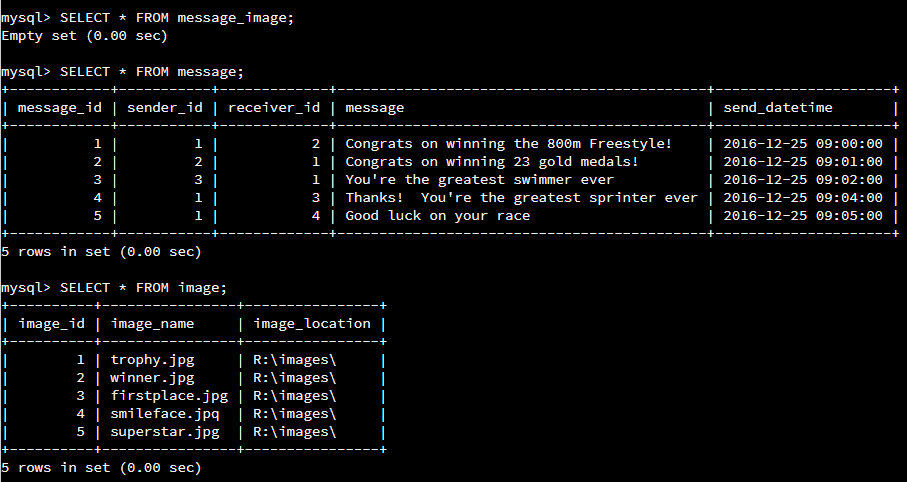


Insert Records to Message-Image Table

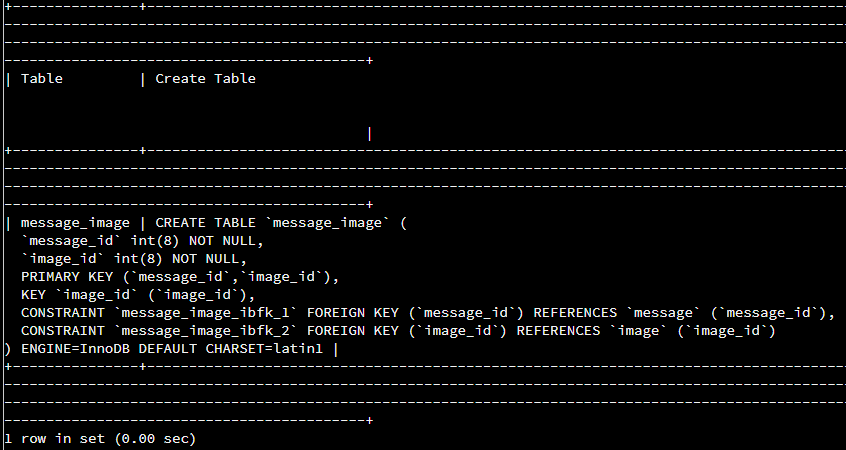
Our next task is to insert 5 new records in the message\_image table. One of the records must include Michael Phelp’s and at least one image and at least one record must include multiple images. As a refresher the table information is shown below.



SELECT statements were also ran to review table data prior to completing the task.



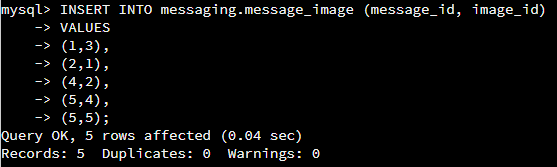
As seen in the SHOW COLUMNS FROM message\_image statement the two columns that need to be added are message\_id and image\_id. Both columns are required and have a data type of INT, and a MAX size of 8. Both message\_id and image\_id have FOREIGN KEY references as well. This can be seen by running the SHOW CREATE TABLE message\_image statement as shown below.



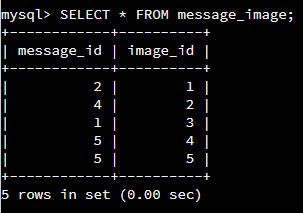
The following is the plan for the records that will be added. As seen in the table there is a message from Michael Phelps and a message with multiple images.

|  |  |  |
| --- | --- | --- |
| **message\_id** | **Image\_id** | **Notes** |
| 1 | 3 | MP Message w/ Image |
| 2 | 1 |  |
| 4 | 2 |  |
| 5 | 4 | Message w/ multi-image |
| 5 | 5 | Message w/ multi-image |

The following statement was utilized to create the records.



As seen from the above the statement was successful, 5 rows were affected, there were no warnings and the execution took 0.04 sec. Running the SELECT statement below show the records were created.

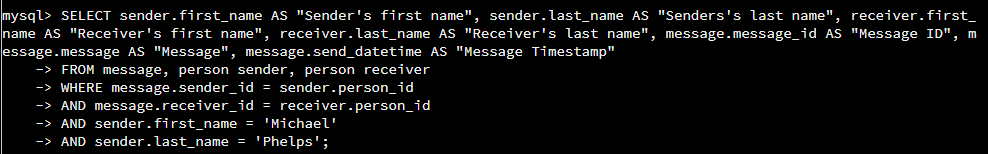


Find all the Records that Michael Phelps Sent

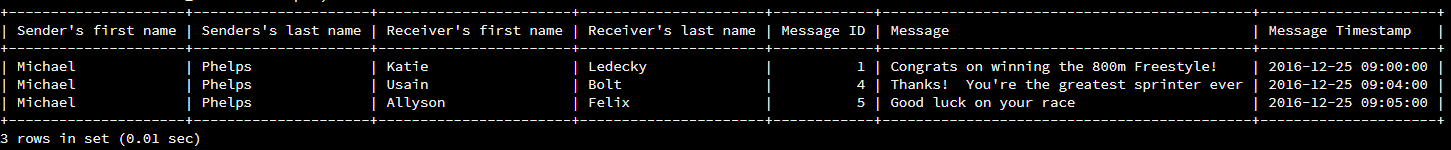
Our next requirement is to SELECT all the messages that Michael Phelps sent. We are also required to utilize the WHERE clause to set the conditions for the query. As part of the output we need to include the following columns.

* Sender's first name
* Sender's last name
* Receiver's first name
* Receiver's last name
* Message ID
* Message
* Message Timestamp

Below is the statement that was used to fulfill the requirements.



As seen above, I used an alias for the person table to ensure I was able to lookup both the sender and receiver information. I also utilized AS in the main SELECT statement to rename the columns. The main WHERE clause enusres that sender\_id is the same in the message and person table so that we can filter for sender.first\_name = ‘Michael’ AND sender.last\_name = ‘Phelps’. The results of the above are shown below.

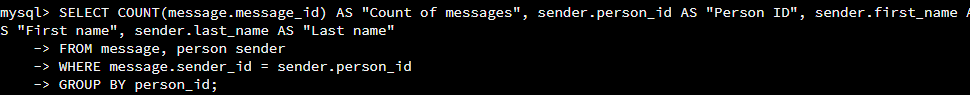


Find the Number of Messages Sent for Every Person

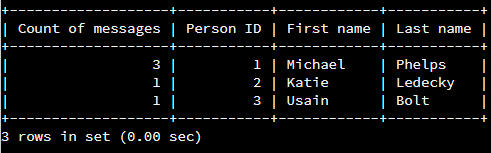
For this task we are asked to COUNT the number of messages sent for every person. Like the last task we must use the WHERE clause to set the conditions for the query. As part of the output we need to display the following columns.

* Count of messages
* Person ID
* First Name
* Last Name

Below is the statement that was used to fulfill the requirements.



As seen above, I used an alias for the person table to make the code more easily understood. I also utilized AS in the main SELECT statement to rename the columns. The main WHERE clause enusres that sender\_id is the same in the message and person table. I also used GROUP BY person\_id to ensure I was looking at things from th point of view of the person that sent the message. The results of the above are shown below.

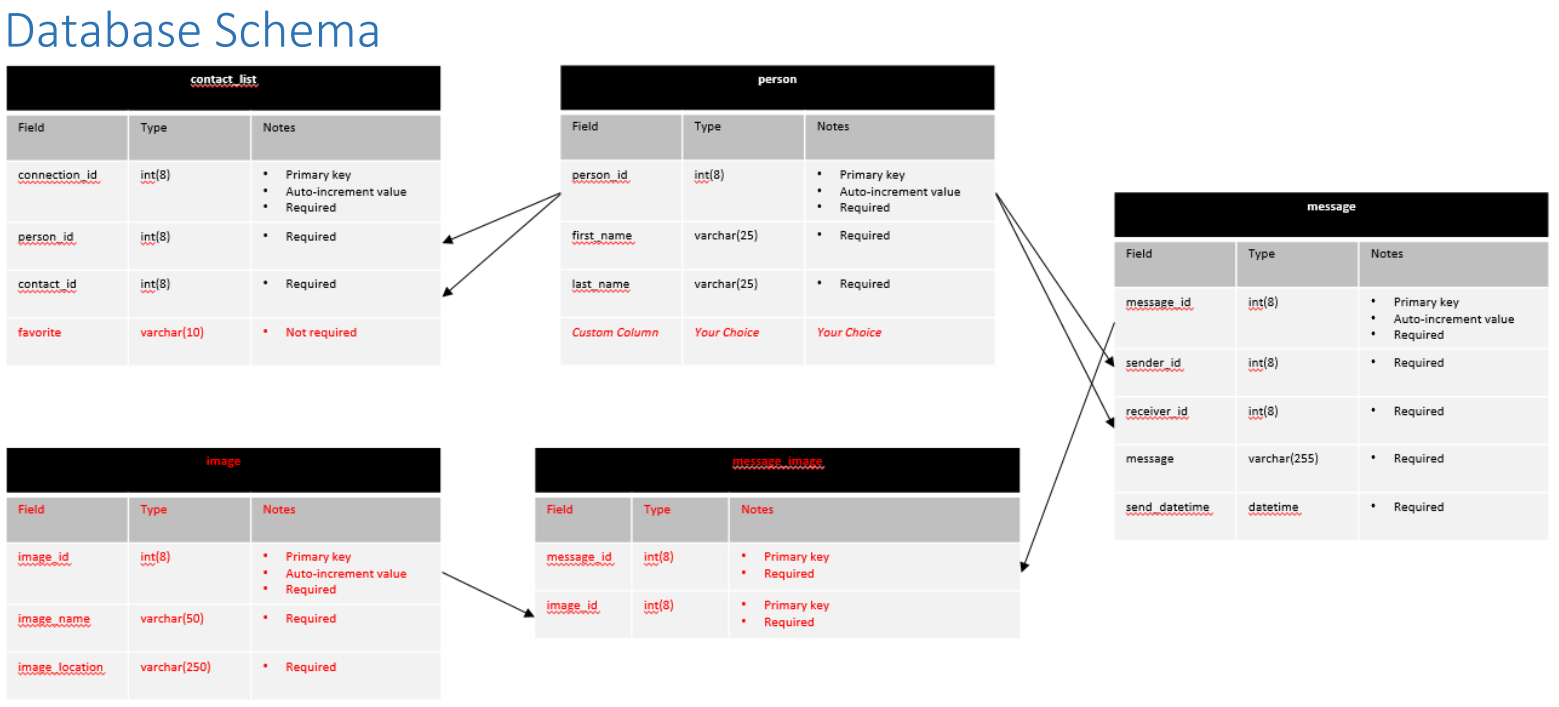


Find All of the Messages that Have at Least One Image Attached

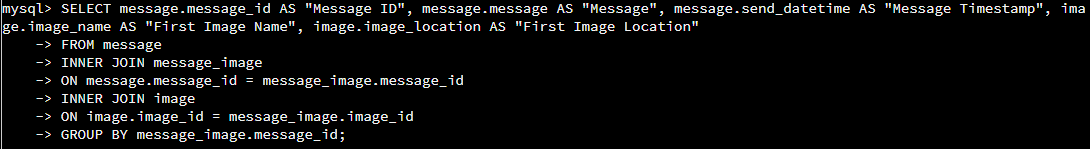
The final task asks us to find all messages with images attached. If the message has more than one image we are to only display the information for the first message. As part of the requirements we must also utilize INNER JOINs to complete the task. The output of the statement should include the following columns.

* Message ID
* Message
* Message Timestamp
* First Image Name
* First Image Location

As a refresher the database schema is included below. This is useful when defining the INNER JOINs to be used.



For this task we are concerned with three of the 5 tables, message, message\_image, and image. In reviewing the above schema, we can see that message can be joined with message\_image via the message\_id column and image and message\_image can be joined via the image\_id column. The below statement was utilized to provide the requested output.



The results of the statement are below.

