Project Summary

The program starts by first determining whether the JDBC Driver class exists on your system, throwing a ClassNotFoundException if it doesn't.

Because you will have a different password, username, and database to add to, a .properties file has been created ("config.properties). This file must first be edited to include your own database connection, e.g. localhost:3306/nameofdatabase, user name, and password for your MySQL account. Do not remove the key values, i.e. "database", "dbuser", and "dbpassword" as these are used to obtain this information.

The contents of the .properties file are then read into a Hashmap, containing the key value, i.e. "database", "dbuser", and "dbpassword", and the subsequent values you will need to enter.

Then, this hashmap is passed into a method that sits within the Connector.java class titled open(). The .open() method establishes the connection with the MySQL database, concatenating the variables from the config.properties file with the connection string, "jdbc:mysql://" and "?useSSL=false". The latter is needed as establishing an SSL connection without server's identify verification is not recommended, so this is set to false.

The Connection instance is then returned and passed to the Execute() class, where it will be used to execute SQL queries and updates.

For the first file, "Person.data", the control character is set to ",". The createHeader() method is then used so that the column names for the insert statement can be added without having to create two different methods for the different files. The column names, "PERSON_ID", "FIRST_NAME", "LAST_NAME", "STREET", and "CITY", are all passed into an array list of Strings. This array list of column names, the name of the .data file, the control character and the name of the table in the database are then passed into the addToDatabase() method. This is where the contents of the .data file are read into a table and inserted into the database.

The Read class(), which was used to read the contents of the .properties file, also contains a method to read from the data file. It uses the Scanner class to read the contents line by line. The line is passed into a String and then separated using the control character. These separated strings are passed into an array of Strings, called record. Then, for each field in record, if the field is not empty and not null, then it is added to the 2D array list titled tableFromFile at the next row. This continues until there are no more lines to be read in.

The getTableFromFile() method is then used to get the 2D array list from the Read class into the Program class. Then, the .insert() method from the Execute class is passed this 2D array list (table), along with the list of column names, and the name of the table in the database. The latter, for this program, is either “PERSON”, or “`ORDER`”. Order is a keyword in SQL, so this needs to be escaped using the ` character.

In order to test this program, I had to make a CREATE / DROP SQL script in a test database. I noticed in the document that it said PERSON_ID is in both tables, which I assumed meant that PERSON_ID in `ORDER` was a foreign key that referenced the PERSON_ID in PERSON. Thus, I added this to the create / drop script. However, because of this, a foreign key constraint would occur whenever the Order with PERSON_ID 10 was inserted into the database. This is because I wouldn’t have someone in the PERSON column with the ID 10 unless this was added in. I did subsequently add this to my database, but later changed the create / drop script so that PERSON_ID in order wasn’t a foreign key. This was just one of the considerations I was deliberating when designing the program.

In the Execute class, the insert() method is used to insert the contents from the .data file into the corresponding table in the MySQL database. First, the question marks are generated depending on the number of columns to be added. These question marks are used within prepared statements, i.e. “INSERT INTO table_name (column_name1, column_name2) VALUES (?, ?);”. These question marks act as placeholders which are passed the values corresponding to that particular column.

Thus, a table with 5 columns will generate 5 question marks. Moreover, the columns need to be turned into a string so that they can be passed into the SQL statement also. Now, the

SQL String is created, concatenating with the table name, columns, and number of question marks. For the Person.data file, the SQL statement will look like "INSERT INTO PERSON (PERSON_ID, FIRST_NAME, LAST_NAME, STREET, CITY) VALUES (?,?,?,?,?);". Noticeably, there are 5 question marks as there are 5 columns to be edited.

A try…catch statement is then used to determine whether the SQL statement is valid. It will try to prepare the statement based on the database connection, and if the statement is in fact valid, then the parameters for each column are set using the .setString() method. Noticeably, the program loops through 2D array list, table, and executes the update to the database. Now, the contents of the file have been added to the database.

The same is done for the Order.data file, except in this instance, the control character is a bar ("|"). As this is a character used in regex, this needs to be escaped, so the control character becomes "\\|". Then, as before, the column names are created and then the contents from the file are read into a table and inserted into the corresponding table in the database, i.e. \`ORDER\`.

Then, the queries were executed to print a table to the terminal. In the Execute class, I had a method for returning people with at least one order, and a method for returning all orders with first name of corresponding person (if available). For the latter, I perceived this as returning orders where the first name is available, i.e. the FIRST_NAME column is not null. Therefore, the script is:
"SELECT ORDER_ID, ORDER_NUMBER, FIRST_NAME FROM PERSON INNER JOIN \`ORDER\` ON PERSON.PERSON_ID = \`ORDER\`.PERSON_ID WHERE FIRST_NAME IS NOT NULL;"

Both methods were similar in that they returned a 2D array list of Strings. The ResultSet class is used to execute the query, and if there is a row (resultSet.next()) then the columns are generated (using the generateTopRow() method) and the rows are added to the 2D array list until there are no more rows to add. If there is no row when the query executes, then "Empty set" is added to the 2D array list and returned. Again, if the SQL is invalid, an SQLException will be thrown.

Finally, the Display class was used to print the table to the console. This was formatted in a way that made it look presentable.

In addition, the rows are deleted from the database following completion of the program. This was used for debugging so that the program could be run multiple times without having to go into the MySQL database and delete the rows manually. This is included in the final program so that the results don’t change the database, but if this was sent out properly then this wouldn’t be included.

For unit testing, the Junit framework was used. The unit testing can be seen in the Read and Execute classes. There are main methods in those classes, so if you build and run whilst in these classes, the tests should run. I didn’t use any test suites; I used the assertEquals method. Furthermore, a Test.data file was created so that, in most cases, the original .data files didn’t need to be used. The .data files did, however, need to be used for testing the insert statements as the table names (i.e. PERSON and `ORDER`) need to exist.

Moreover, as this program was created in the Eclipse IDE, the external libraries for the JDBC driver and JUnit were added to the build path. The .jar files are included in the project directory in the “lib” folder.