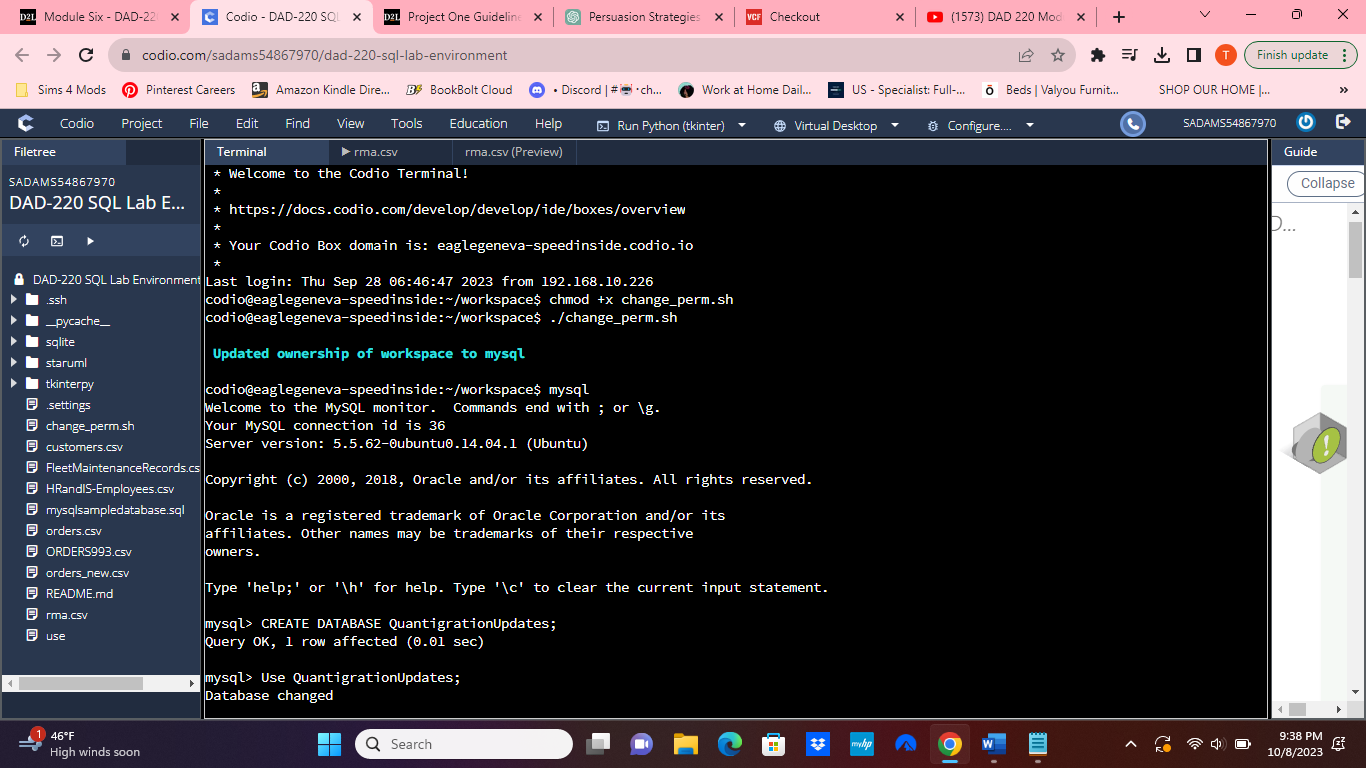
# DAD 220 Database Documentation Template

## Step One: Create a Database

1. Navigate to your online integrated development environment (IDE). List and record the SQL commands that you used to complete this step here:



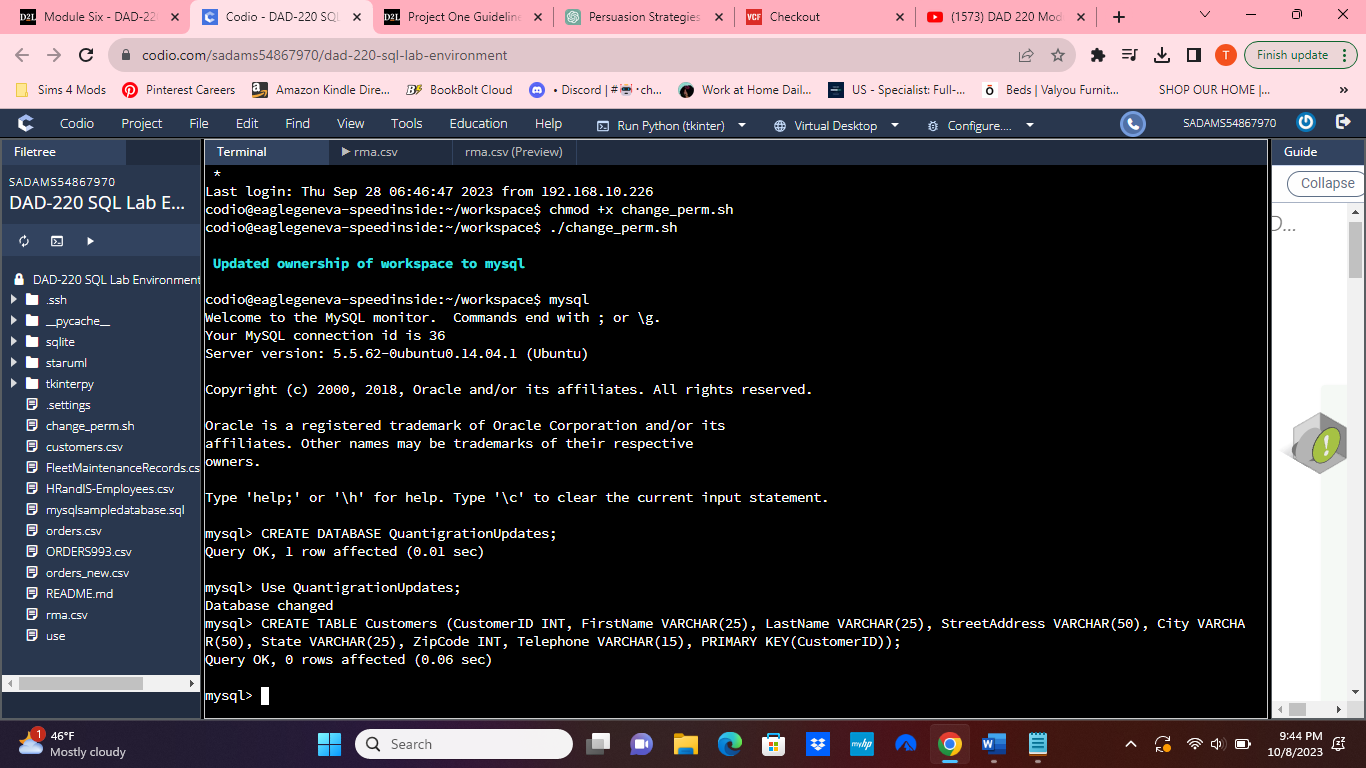
Commands to change the ownership of Mysql:

Chmod +x change\_perm.sh

./change\_perm.sh

Mysql

1. Create a database schema called QuantigrationUpdates. List out the database name. Provide the SQL commands you ran against MySQL to successfully complete this in your answer:



A computer screen shot of a black screen

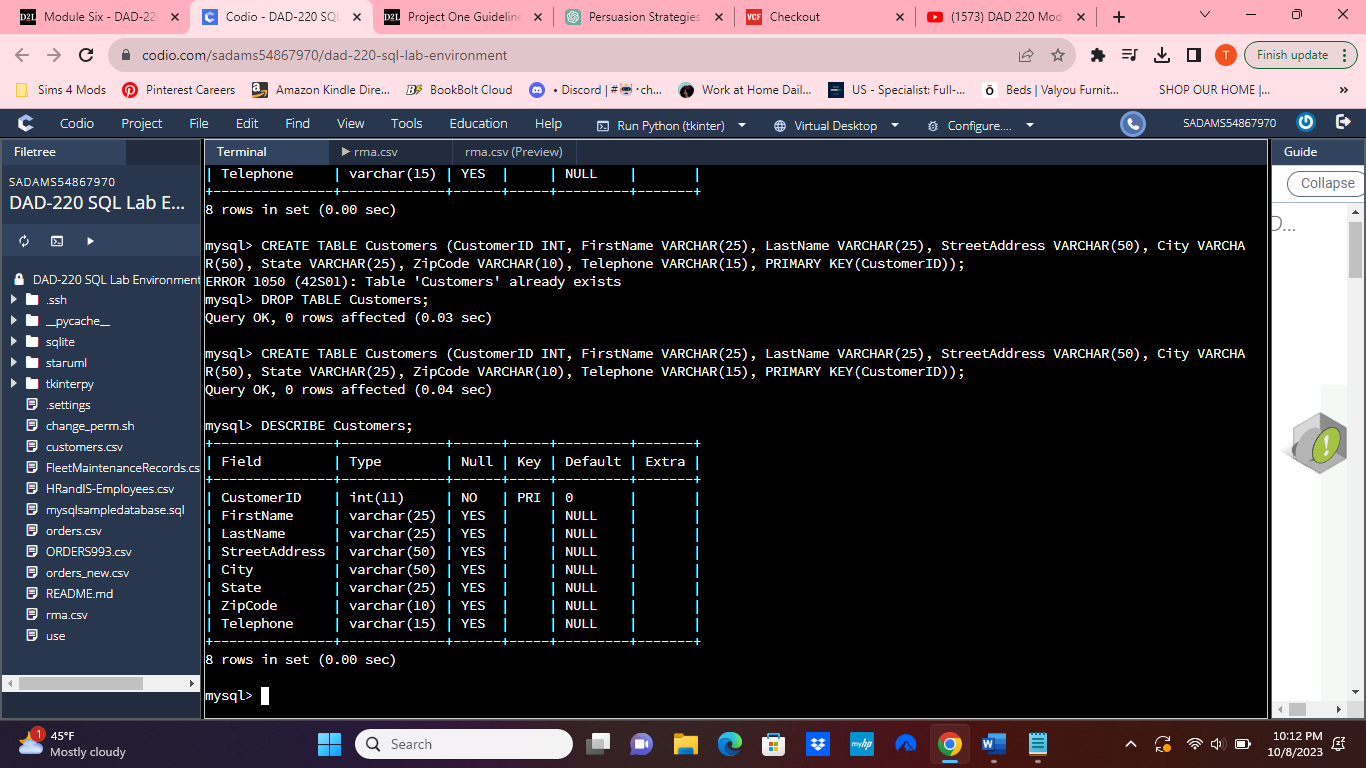
Description automatically generated

CREATE DATABASE QuantigrationUpdates;

SHOW DATABASES;

The "CREATE DATABASE" statement is used to initiate the creation of a new database in a DMS. After executing, the system created a new and empty database under the name "QuantigrationUpdates." The "SHOW DATABASES" command simply returns all existing databases in the current database server.

1. Using the entity relationship diagram (ERD) as a reference, create the following tables with the appropriate attributes and keys:
   1. A table named **Customers** in the QuantigrationUpdates database, as defined on the project ERD. Provide the SQL commands you ran against MySQL to complete this successfully in your answer:



A screenshot of a computer

Description automatically generated

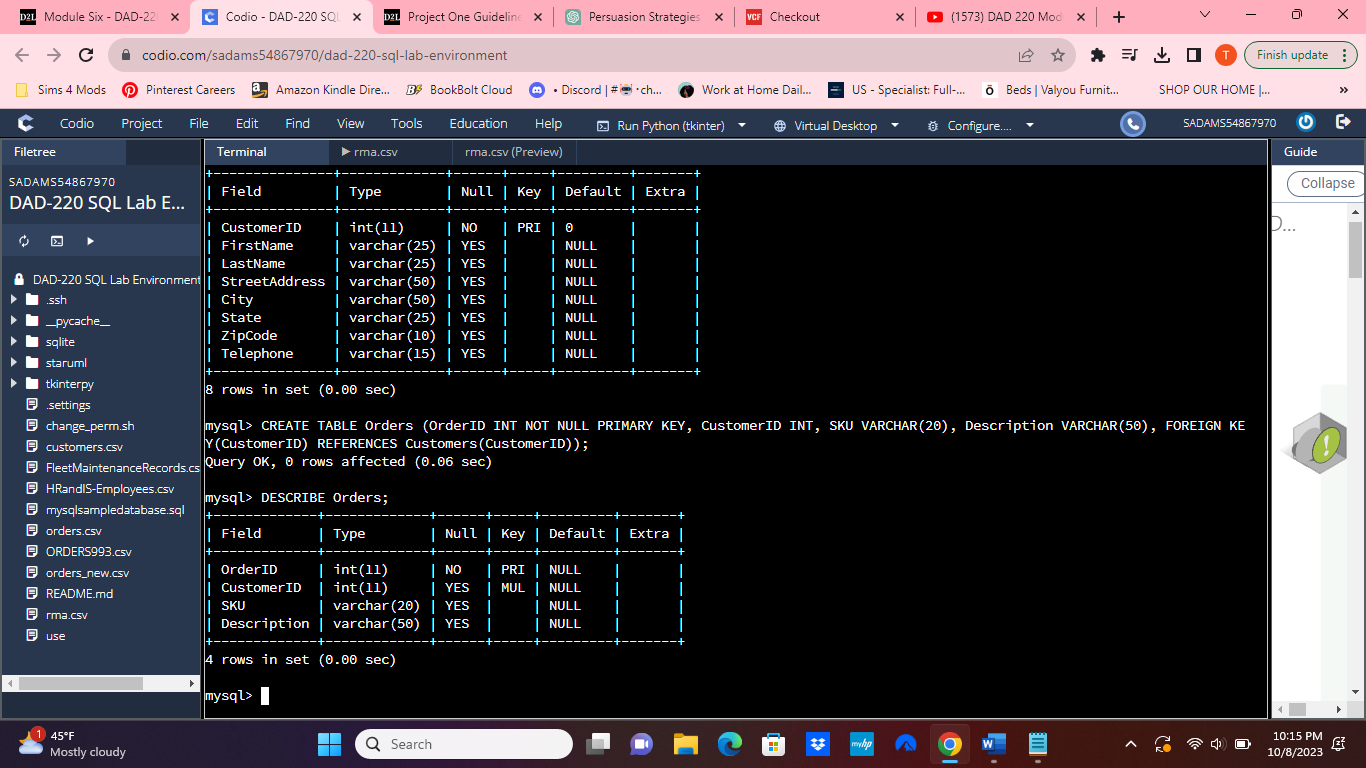
CREATE TABLE Customers (CustomerID INT, FirstName VARCHAR(25), LastName VARCHAR(25), StreetAddress VARCHAR(50), City VARCHAR(50), State VARCHAR(25), ZipCode VARCHAR(10), Telephone VARCHAR(15), PRIMARY KEY(CustomerID));

DESCRIBE Customers;

The initial SQL command established a fresh table called "Customers" within the database. This table incorporates numerous columns, each assigned its specific data type and size. The inclusion of PRIMARY KEY(CustomerID) in the statement's conclusion designates the "CustomerID" column as the table's primary key. This primary key guarantees the uniqueness of each entry in the "CustomerID" column and acts as a pivotal reference for distinguishing individual customer records.

The subsequent SQL command offers an insight into the configuration of the "Customers" table, presenting details regarding its column composition and the respective data types. Upon executing this command, the Database Management System (DBMS) presented details regarding the table's columns, encompassing their names, data types, and any related constraints. This action served as a confirmation of the successful implementation of the preceding SQL command.

* 1. A table named **Orders** in the QuantigrationUpdates database, as defined on the project ERD. Provide the SQL commands you ran against MySQL to complete this successfully in your answer:



A screenshot of a computer

Description automatically generated

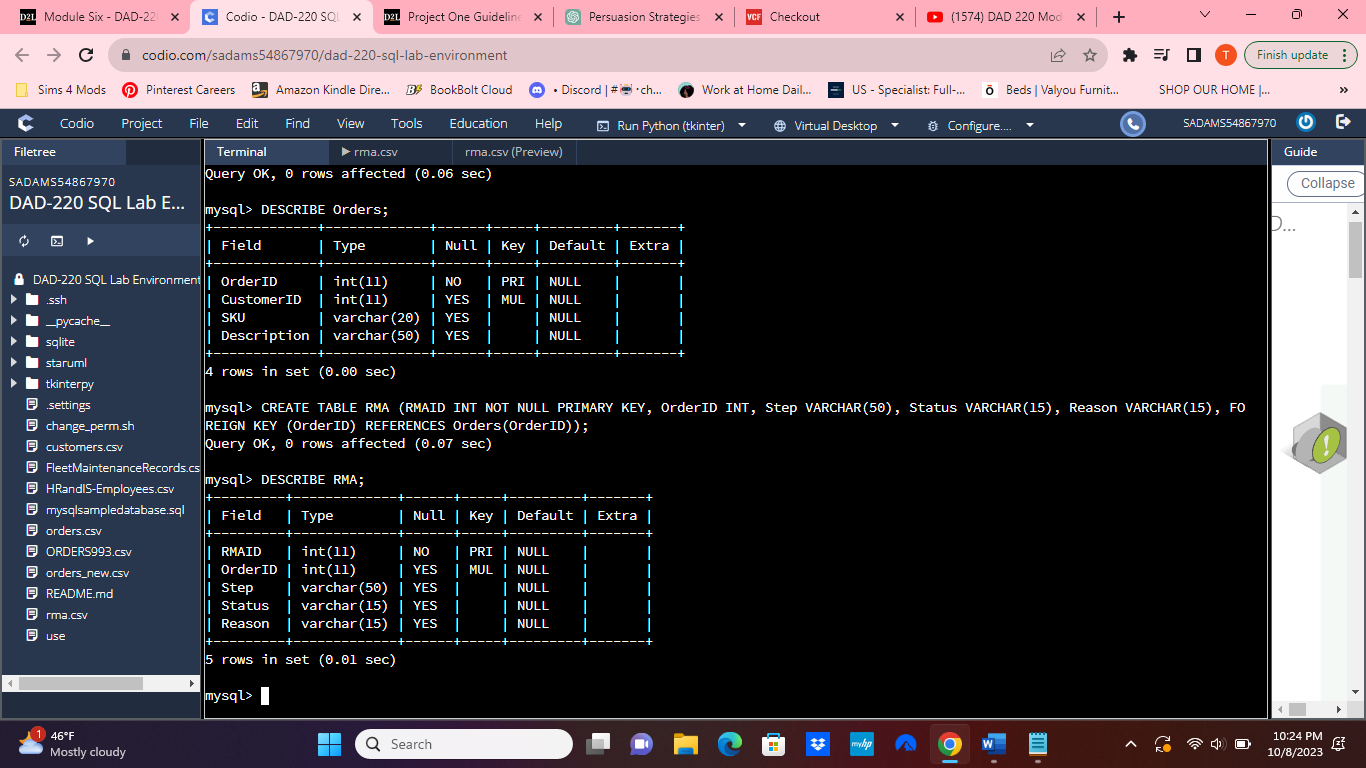
CREATE TABLE Orders (OrderID INT NOT NULL PRIMARY KEY, CustomerID INT, SKU VARCHAR(20), Description VARCHAR(50), FOREIGN KEY(CustomerID) REFERENCES Customers(CustomerID));

DESCRIBE Orders;

The initial SQL command generated a database table named "Orders," which consists of four rows. Within this table, the "FOREIGN KEY" constraint was implemented to establish a connection between the "Orders" table and the "Customers" table. This linkage is formed by associating the "CustomerID" column in the "Orders" table with the "CustomerID" column in the "Customers" table, thereby establishing a foreign key relationship.

The subsequent SQL command furnishes an overview of the structural composition of the "Orders" table, presenting details regarding its columns and their respective data types. When this command was executed, the Database Management System (DMS) responded by listing the table's columns, inclusive of their names, data types, and any accompanying constraints. This action served as confirmation of the successful implementation of the preceding SQL command.

* 1. A table named **RMA** in the QuantigrationUpdates database, as defined on the project ERD. Provide the SQL commands you ran against MySQL to complete this successfully in your answer:



CREATE TABLE RMA (RMAID INT NOT NULL PRIMARY KEY, OrderID INT, Step VARCHAR(50), Status VARCHAR(15), Reason VARCHAR(15), FOREIGN KEY (OrderID) REFERENCES Orders(OrderID));

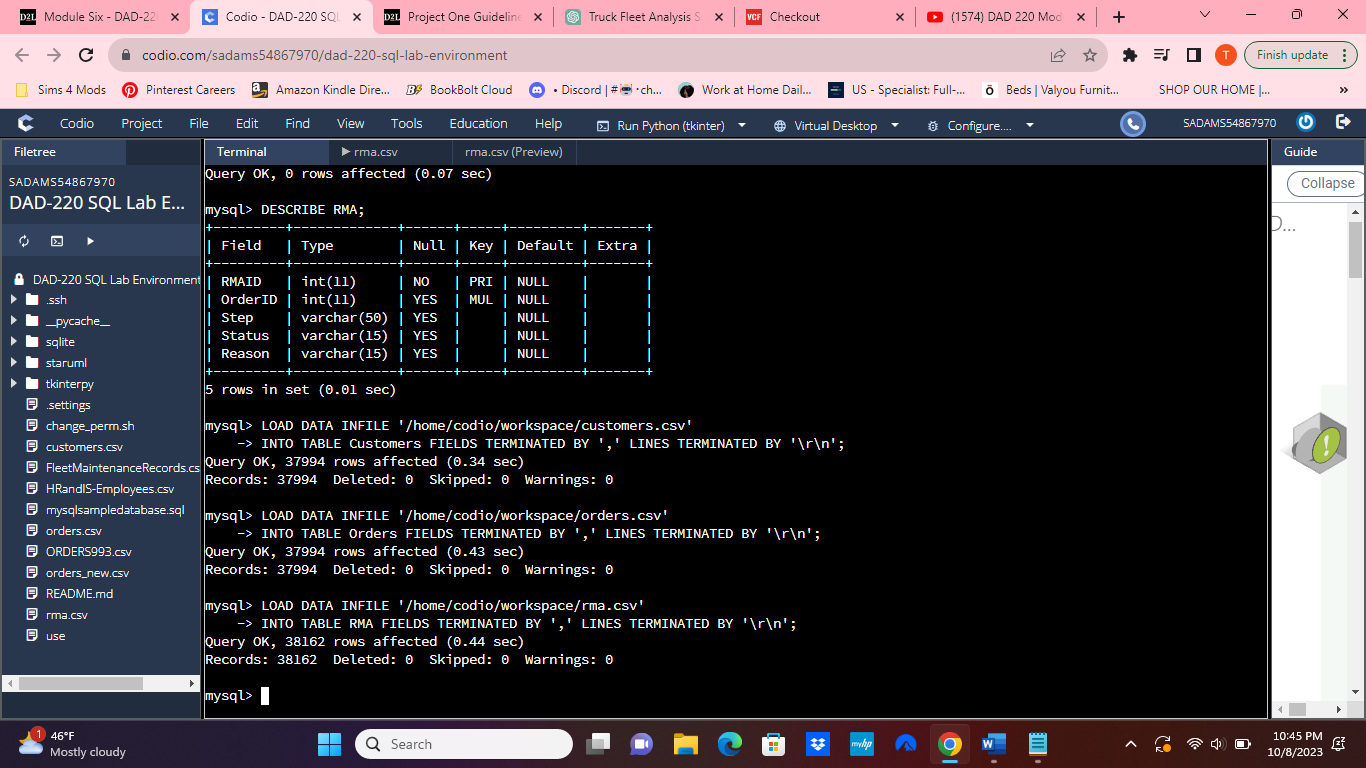
DESCRIBE RMA;

The initial SQL command was used to create the "RMA" table within the database. This table incorporates several essential columns. First, the "RMAID" column serves as a repository for unique identifiers assigned to each Return Merchandise Authorization (RMA). Notably, the "NOT NULL" constraint is applied to this column, ensuring that all entries within it must contain valid and non-null values. Additionally, the "PRIMARY KEY" constraint designates the "RMAID" column as the primary key for this table, guaranteeing the uniqueness of each RMA identifier. The "OrderID" column is designed to store identifiers that establish links between each RMA and the respective order it pertains to. Furthermore, the table includes columns such as "Step," "Status," and "Reason," each of which accommodates specific attributes and data relevant to the RMA process. Collectively, these columns collectively form the structural foundation of the "RMA" table, facilitating the systematic organization and management of information associated with Return Merchandise Authorizations.

The subsequent SQL command offers an overview of the structural configuration of the "RMA" table. It presents comprehensive insights into the attributes of the table, encompassing details regarding the names of individual columns, their respective data types, and any constraints linked to them. The execution of this command elicited a response from the Database Management System (DMS), which diligently provided a concise account of the table's structural components. This process served to corroborate the successful implementation of the preceding SQL command, affirming the accurate establishment of the "RMA" table within the database.

## Step Two: Load and Query the Data

1. **Import the data from each file into tables.** 
   * Use the QuantigrationUpdates database, the three tables you created, and the three CSV files preloaded into Codio.
   * Use the import utility of your database program to load the data from each file into the table of the same name. You will perform this step three times, once for each table.



LOAD DATA INFILE '/home/codio/workspace/customers.csv'

INTO TABLE Customers FIELDS TERMINATED BY ',' LINES TERMINATED BY '\r\n';

LOAD DATA INFILE '/home/codio/workspace/orders.csv'

INTO TABLE Orders FIELDS TERMINATED BY ',' LINES TERMINATED BY '\r\n';

LOAD DATA INFILE '/home/codio/workspace/rma.csv'

INTO TABLE RMA FIELDS TERMINATED BY ',' LINES TERMINATED BY '\r\n';

The SQL code provided exemplifies a data loading procedure intended for the "Customers, Orders, and RMA" tables. Within this operation, the "LOAD DATA INFILE" statement assumes a pivotal role, orchestrating the seamless importation of data sourced from a trio of CSV files: "customer.csv," "orders.csv," and "rma.csva." These files are securely located at the directory path '/home/codio/workspace/' as stipulated by the directive. The "INTO TABLE" clause is strategically employed to delineate the precise tables into which the incoming data shall be meticulously incorporated. Furthermore, the "FIELDS TERMINATED BY ','" clause serves as a pivotal indicator, elucidating that the CSV files employ commas as the designated field separators. Concurrently, the "LINES TERMINATED BY '\r\n'" clause meticulously defines the formatting protocol governing line endings within the CSV file. Upon execution, this SQL command instigates a harmonious process within the database server. Herein, the server adeptly assimilates the content encapsulated within the CSV files, judiciously mapping the comma-delimited values to their respective columns across the designated tables. This meticulous orchestration culminates in the seamless insertion of data, thereby enriching the tables with the pertinent information.

1. **Write basic queries against imported tables to organize and analyze targeted data.** For each query, replace the bracketed text with a screenshot of the query and its output. You should also include a 1- to 3-sentence description of the output.
   * Write an SQL query that returns the **count** of orders for customers located only in the city of Framingham, Massachusetts.
     1. How many records were returned? **505 records.**

A screenshot of a computer

Description automatically generated

SELECT COUNT(\*)

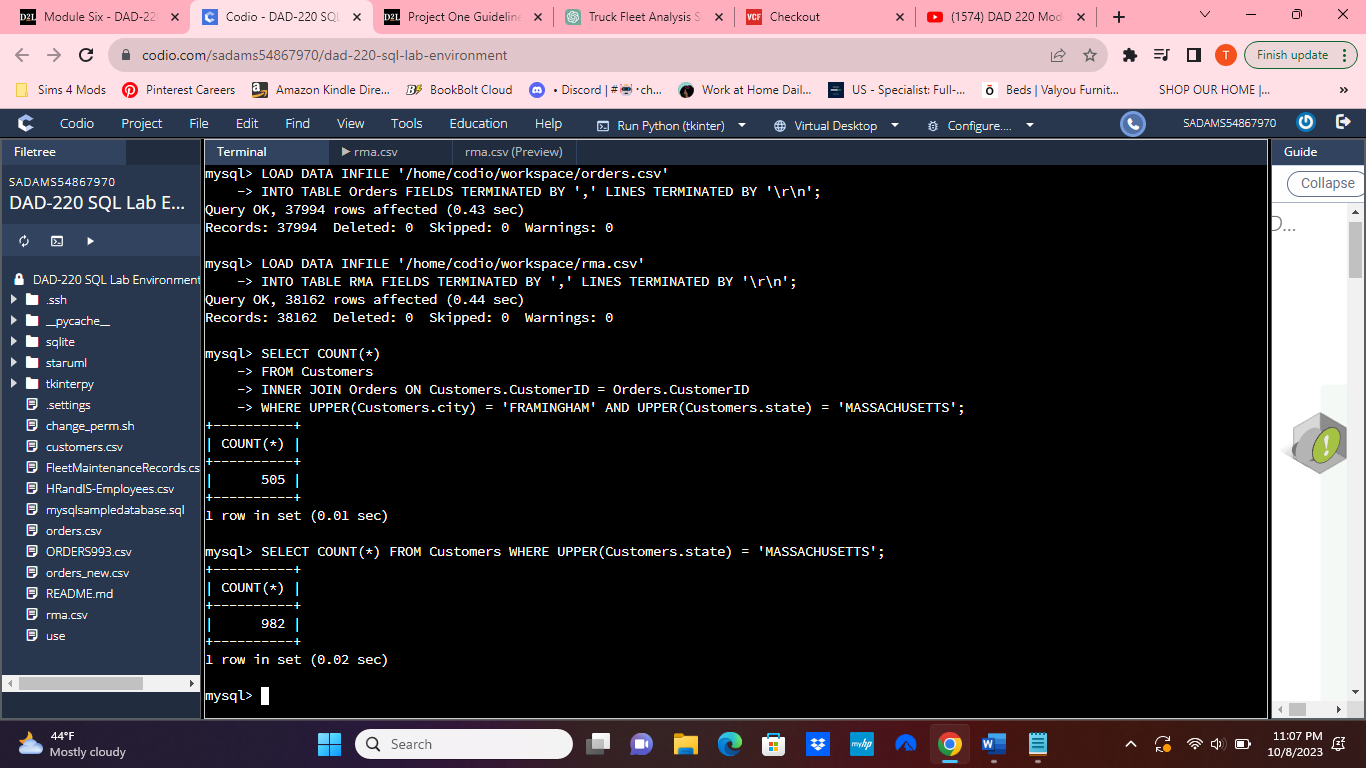
FROM Customers

INNER JOIN Orders ON Customers.CustomerID = Orders.CustomerID

WHERE UPPER(Customers.city) = 'FRAMINGHAM' AND UPPER(Customers.state) = 'MASSACHUSETTS';

This SQL query proficiently tallied the volume of orders initiated by residents of 'FRAMINGHAM', 'MASSACHUSETTS'. Effecting a synergy between the "Customers" and "Orders" tables, an "INNER JOIN" operation is orchestrated, predicated on the congruence of the "CustomerID" column. This meticulous linkage ensures the inclusion solely of customers with corresponding orders. The "WHERE" clause acts as an additional sieve, winnowing the dataset to encompass customers hailing from the specified city and state. A judicious application of the "UPPER()" function facilitates a case-insensitive comparison. The crux of this query lies in the "COUNT(\*)" function, which adeptly consolidates the data and furnishes the tally of orders. The resultant output unequivocally delineates that 505 orders emanate from customers situated in Framingham, Massachusetts.

* + Write an SQL query to **select all** of the Customers located in the state of Massachusetts.
    1. Use a WHERE clause to limit the number of records in the Customers table to only those who are located in Massachusetts.
    2. Record an answer to the following question: How many records were returned? **982 records.**



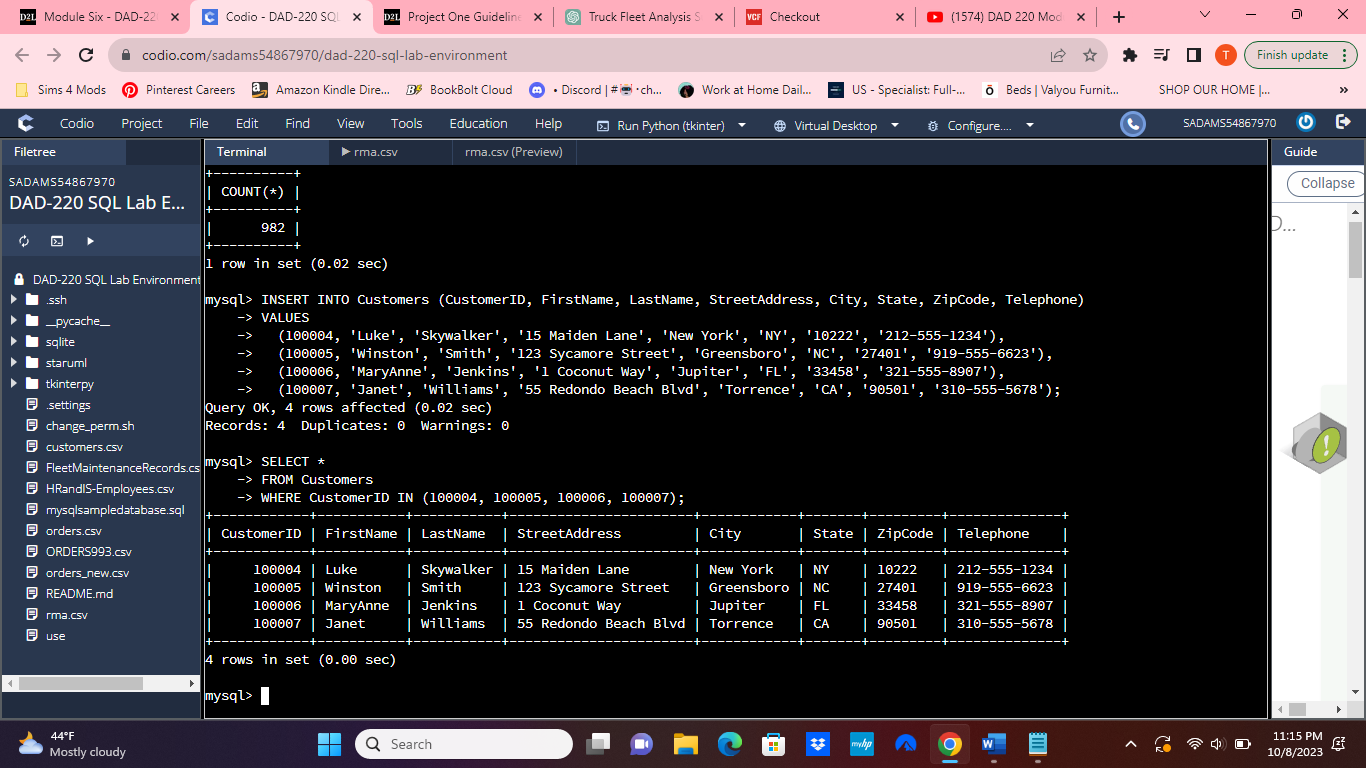
SELECT COUNT(\*) FROM Customers WHERE UPPER(Customers.state) = 'MASSACHUSETTS';

This SQL query is used to retrieve and count customer records that are located in the state of Massachusetts. It accomplishes this by first selecting the "Customers" table from which the data will be extracted. The crucial part of the query is the WHERE clause, which filters the records based on a specific condition. In this case, it checks if the "state" column, after converting its contents to uppercase, matches the string 'MASSACHUSETTS.' This conversion to uppercase ensures a case-insensitive comparison, so it will capture any variations in capitalization. The COUNT(\*) function is then applied to count the number of rows that meet this condition. The result of 982 records indicates that there are a total of 982 customers whose state is recorded as 'MASSACHUSETTS' in the dataset.

* + Write a SQL query to insert four new records into the Orders and Customers tables using the following data:

**Customers Table**

| **CustomerID** | **FirstName** | **LastName** | **StreetAddress** | **City** | **State** | **ZipCode** | **Telephone** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 100004 | Luke | Skywalker | 15 Maiden Lane | New York | NY | 10222 | 212-555-1234 |
| 100005 | Winston | Smith | 123 Sycamore Street | Greensboro | NC | 27401 | 919-555-6623 |
| 100006 | MaryAnne | Jenkins | 1 Coconut Way | Jupiter | FL | 33458 | 321-555-8907 |
| 100007 | Janet | Williams | 55 Redondo Beach Blvd | Torrence | CA | 90501 | 310-555-5678 |



INSERT INTO Customers (CustomerID, FirstName, LastName, StreetAddress, City, State, ZipCode, Telephone)

VALUES

(100004, 'Luke', 'Skywalker', '15 Maiden Lane', 'New York', 'NY', '10222', '212-555-1234'),

(100005, 'Winston', 'Smith', '123 Sycamore Street', 'Greensboro', 'NC', '27401', '919-555-6623'),

(100006, 'MaryAnne', 'Jenkins', '1 Coconut Way', 'Jupiter', 'FL', '33458', '321-555-8907'),

(100007, 'Janet', 'Williams', '55 Redondo Beach Blvd', 'Torrence', 'CA', '90501', '310-555-5678');

SELECT \*

FROM Customers

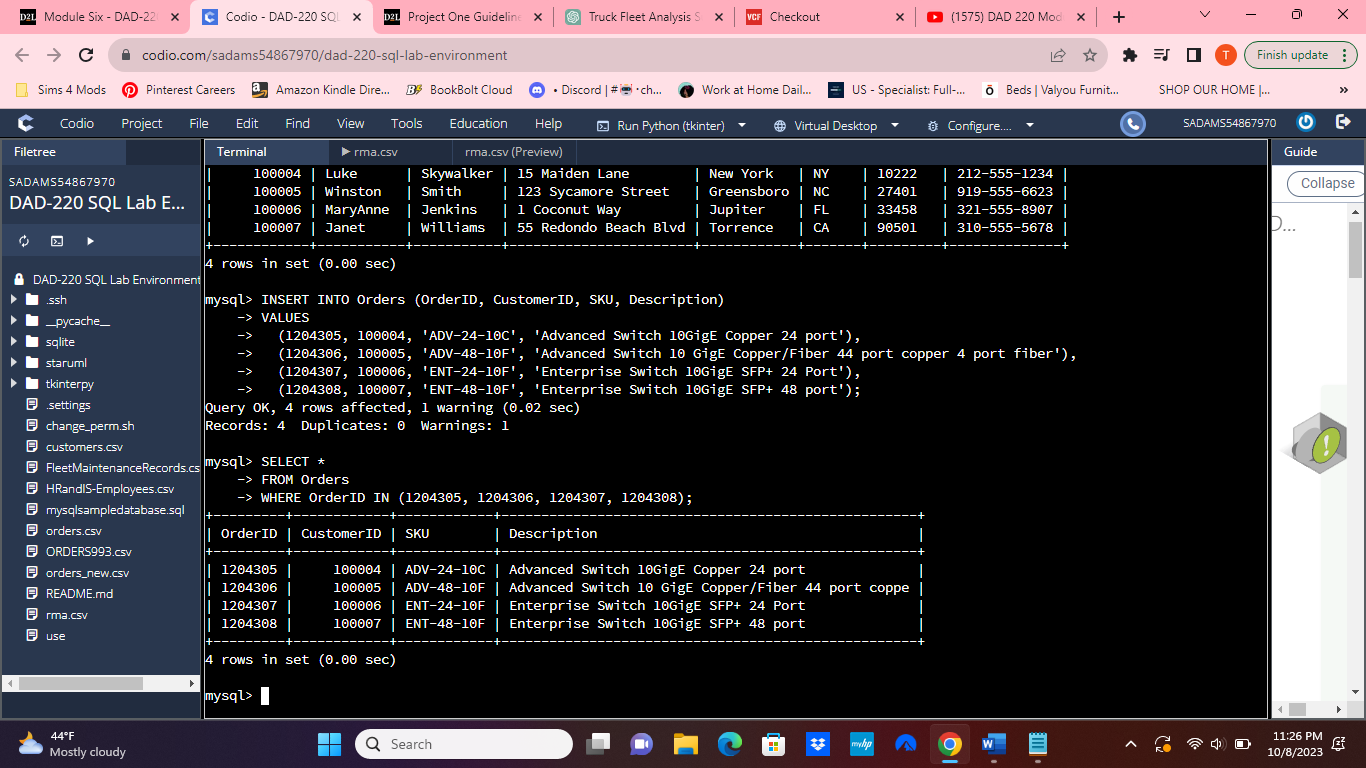
WHERE CustomerID IN (100004, 100005, 100006, 100007);

This SQL command is an INSERT INTO statement designed to insert new records into the "Customers" table. In this command, it specifies both the target table, which is "Customers," and the columns where the data will be added. The command includes four sets of values, each enclosed in parentheses, representing the details of individual customers. For instance, the first set includes a CustomerID of 100004, along with the customer's first name (Luke), last name (Skywalker), street address (15 Maiden Lane), city (New York), state (NY), ZIP code (10222), and telephone number (212-555-1234). The subsequent sets follow the same pattern, providing unique CustomerIDs and specific information for each customer. As a result, this command effectively inserts four new customer records into the "Customers" table, augmenting the existing dataset with these additional customer profiles.

This SQL command, SELECT \* FROM Customers WHERE CustomerID IN (100004, 100005, 100006, 100007);, is designed to retrieve specific customer records from the "Customers" table based on their unique customer IDs. The command begins with SELECT \*, indicating that we want to retrieve all columns ("\*") of data. It specifies the source table, "Customers," using FROM Customers. The WHERE clause serves as a filter, allowing us to narrow down the results. In this case, it filters records where the CustomerID column matches any of the four values listed within the parentheses: 100004, 100005, 100006, and 100007. As a result, the command will return detailed information for customers whose IDs match these values, providing a concise selection of customer data from the "Customers" table.

**Orders Table**

| **OrderID** | **CustomerID** | **SKU** | **Description** |
| --- | --- | --- | --- |
| 1204305 | 100004 | ADV-24-10C | Advanced Switch 10GigE Copper 24 port |
| 1204306 | 100005 | ADV-48-10F | Advanced Switch 10 GigE Copper/Fiber 44 port copper 4 port fiber |
| 1204307 | 100006 | ENT-24-10F | Enterprise Switch 10GigE SFP+ 24 Port |
| 1204308 | 100007 | ENT-48-10F | Enterprise Switch 10GigE SFP+ 48 port |



INSERT INTO Orders (OrderID, CustomerID, SKU, Description)

VALUES

(1204305, 100004, 'ADV-24-10C', 'Advanced Switch 10GigE Copper 24 port'),

(1204306, 100005, 'ADV-48-10F', 'Advanced Switch 10 GigE Copper/Fiber 44 port copper 4 port fiber'),

(1204307, 100006, 'ENT-24-10F', 'Enterprise Switch 10GigE SFP+ 24 Port'),

(1204308, 100007, 'ENT-48-10F', 'Enterprise Switch 10GigE SFP+ 48 port');

SELECT \*

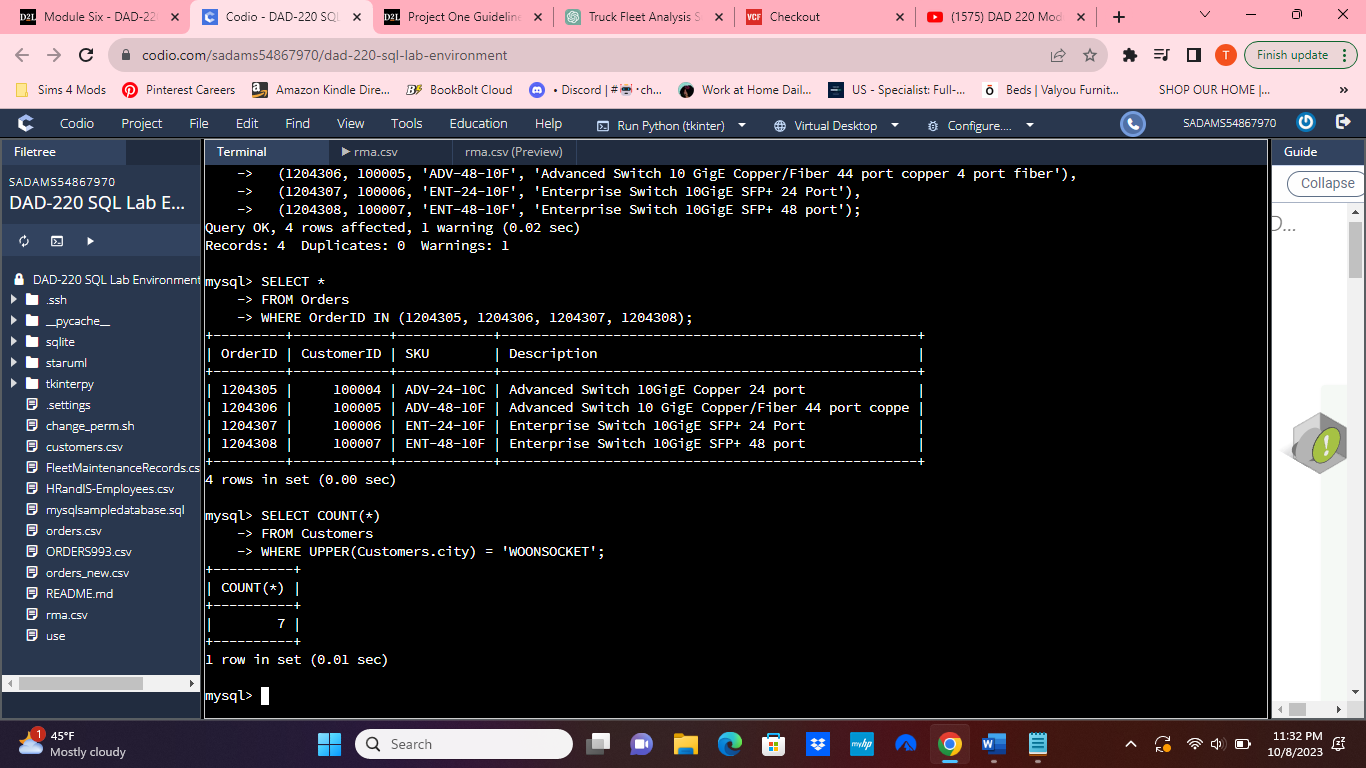
FROM Orders

WHERE OrderID IN (1204305, 1204306, 1204307, 1204308);

This SQL command is an "INSERT INTO" statement designed to insert multiple records into the "Orders" table of a database. This command is particularly useful when you need to add multiple rows of data simultaneously. It begins by specifying the target columns into which the data will be inserted, namely "OrderID," "CustomerID," "SKU" (Stock Keeping Unit), and "Description." Following the column list, the "VALUES" clause is used to define the values for each record to be inserted. In this case, four sets of values are enclosed in parentheses, each representing a distinct order. For instance, the first set (1204305, 100004, 'ADV-24-10C', 'Advanced Switch 10GigE Copper 24 port') includes the order's unique identifier (OrderID), the corresponding customer's identifier (CustomerID), the product's SKU code, and a descriptive text. This command efficiently inserts all four records into the "Orders" table at once, streamlining the process of populating the database with detailed order information.

This SQL command is a "SELECT" statement used to retrieve data from the "Orders" table within a database. In this query, the asterisk (\*) signifies that all columns in the "Orders" table should be included in the output. The "FROM" clause specifies the source table, which, in this case, is the "Orders" table. The "WHERE" clause is used to apply a condition to the query. Specifically, it employs the "IN" operator to filter rows where the "OrderID" matches one of the specified values enclosed in parentheses. In this instance, the query is configured to retrieve rows with "OrderID" values equal to 1204305, 1204306, 1204307, or 1204308. This command allows for the selection of specific orders from the table based on their unique identifiers, enabling precise retrieval of order data.

* + In the Customers table, perform a query to count all records where the city is Woonsocket, Rhode Island.
    1. How many records are in the Customers table where the field “city” equals “Woonsocket”? **7 records.**



SELECT COUNT(\*)

FROM Customers

WHERE UPPER(Customers.city) = 'WOONSOCKET';

This SQL command is designed to count the number of records in the "Customers" table where the city is specified as "Woonsocket." It utilizes the "COUNT(\*)" function, which calculates the total number of rows that meet the specified criteria. In this case, the "FROM" clause indicates that the source table for this operation is the "Customers" table. The "WHERE" clause is crucial in filtering the data and ensuring that only records matching the criteria are counted. Specifically, it employs the "UPPER()" function to convert the values in the "City" column to uppercase for case-insensitive comparison. It then checks whether the resulting city names are equal to 'WOONSOCKET.' Consequently, when this query is executed, it returns the count of customer records where the city is indeed 'Woonsocket.' This count represents the total number of customers residing in Woonsocket, regardless of the letter case used in the city names in the database.

* + In the RMA database, update a customer’s records.
    1. Write an SQL statement to select the current fields of **status** and **step** for the record in the **RMA** table with an **orderid** value of “5175.”
       1. What are the current status and step? **Awaiting customer Documentation; pending.**

A computer screen shot of a black screen

Description automatically generated

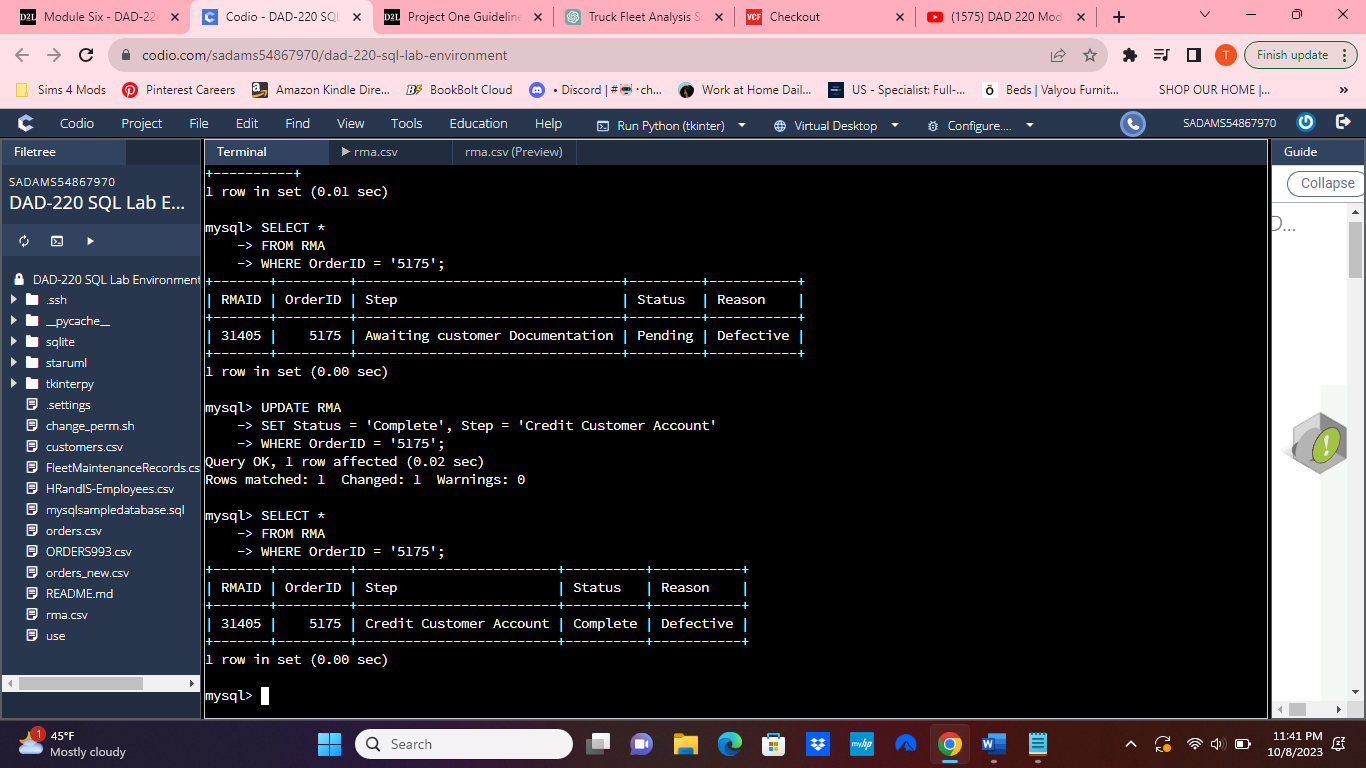
SELECT \*

FROM RMA

WHERE OrderID = '5175';

The SQL command "SELECT \* FROM RMA WHERE OrderID = '5175';" is used to query the "RMA" table in a database. It seeks to retrieve all the columns (indicated by the asterisk "\*") for the records where the "OrderID" field matches the value '5175.' When this command is executed, it scans the "RMA" table and returns all the information associated with the specific OrderID '5175.' This query is useful for retrieving detailed records related to a particular order or transaction, allowing you to view and work with the data associated with that specific order within the RMA table.

* + 1. Write an SQL statement to update the **status** and **step** for the **OrderID**, 5175 to **status** = “Complete” and **step** = “Credit Customer Account.”
       1. What are the updated **status** and **step** values for this record? **Credit Customer Account; Complete**



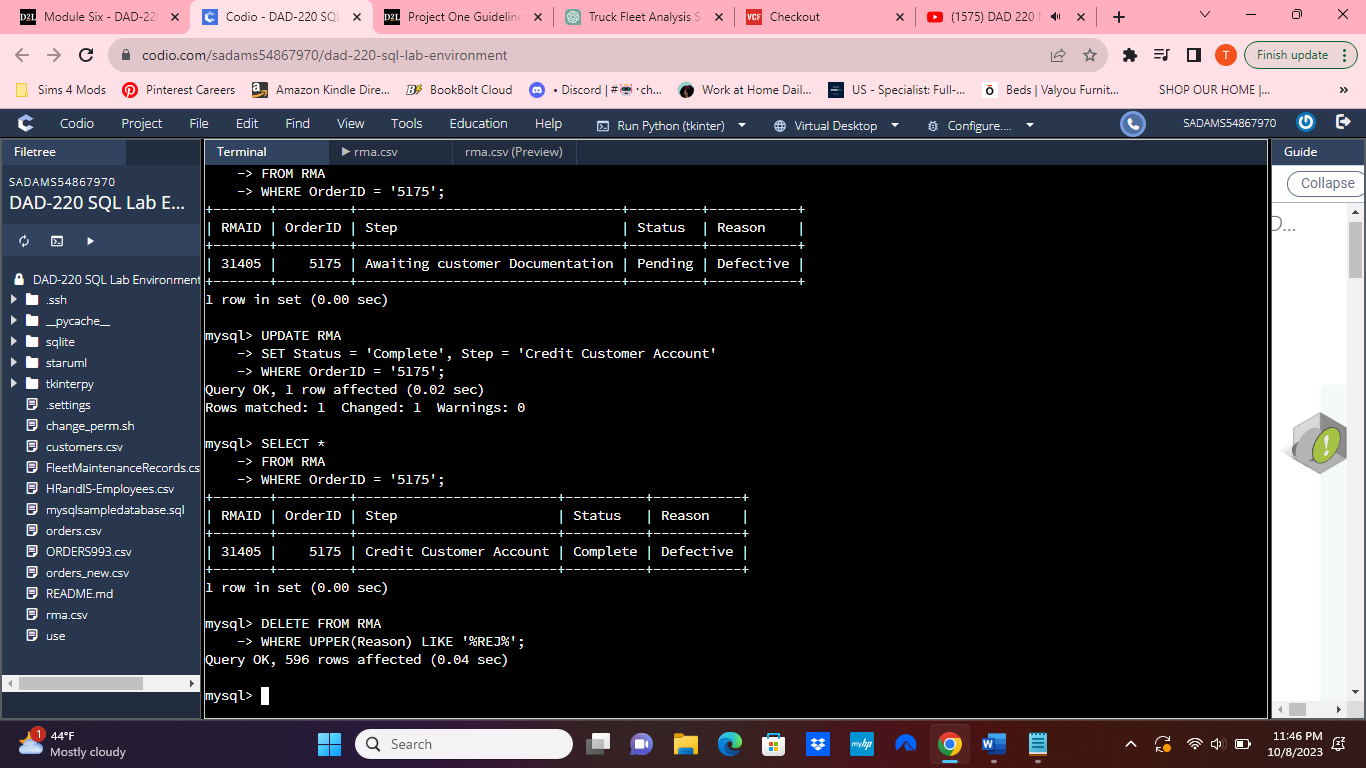
UPDATE RMA

SET Status = 'Complete', Step = 'Credit Customer Account'

WHERE OrderID = '5175';

This SQL command specifies the table to be updated, which is the "RMA" table. It then uses the "SET" clause to define the new values for the "Status" and "Step" columns. The "WHERE" clause filters the update operation to apply only to records where the "OrderID" matches '5175.' When executed, this statement will change the status to "Complete" and the step to "Credit Customer Account" for the specified order in the RMA table.

* + Delete RMA records.
    1. Write an SQL statement to delete all records with a reason of “Rejected.”
       1. How many records were deleted? **596 records.**

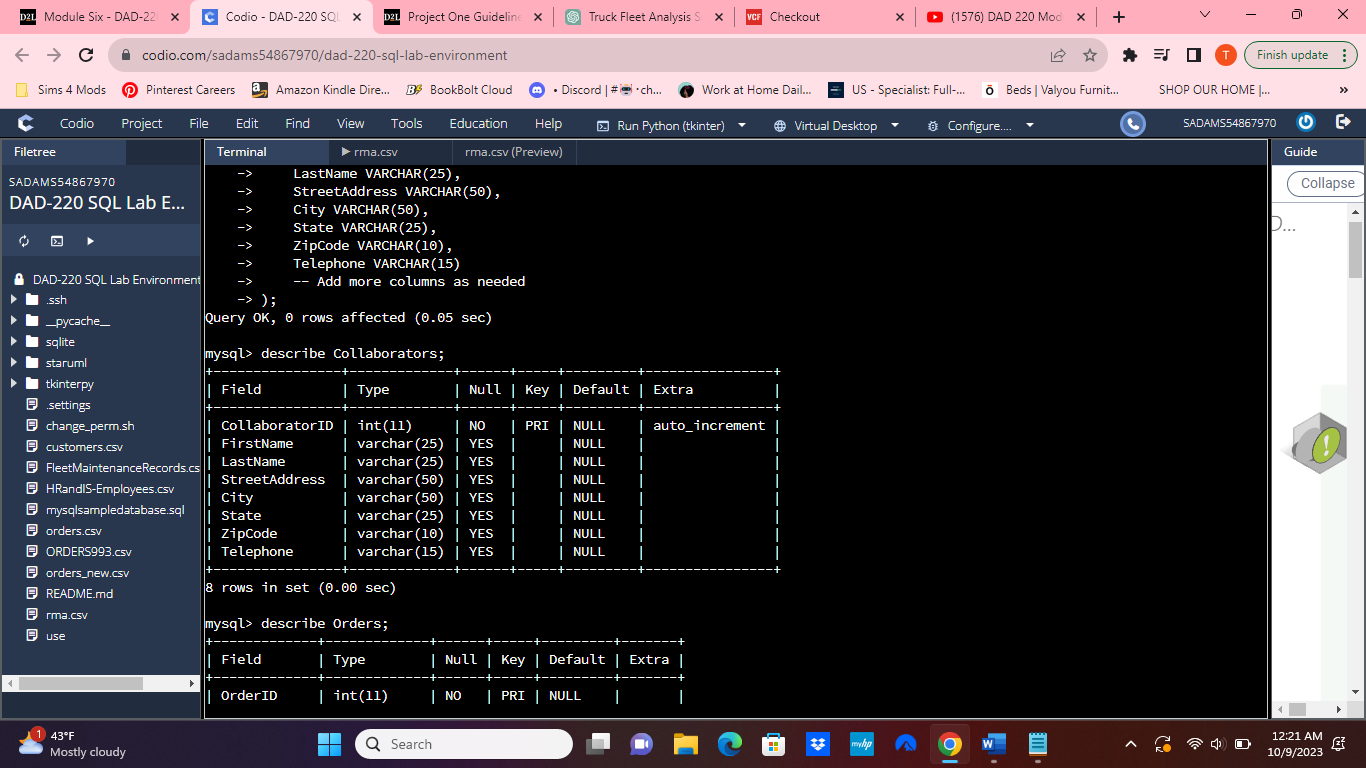


DELETE FROM RMA

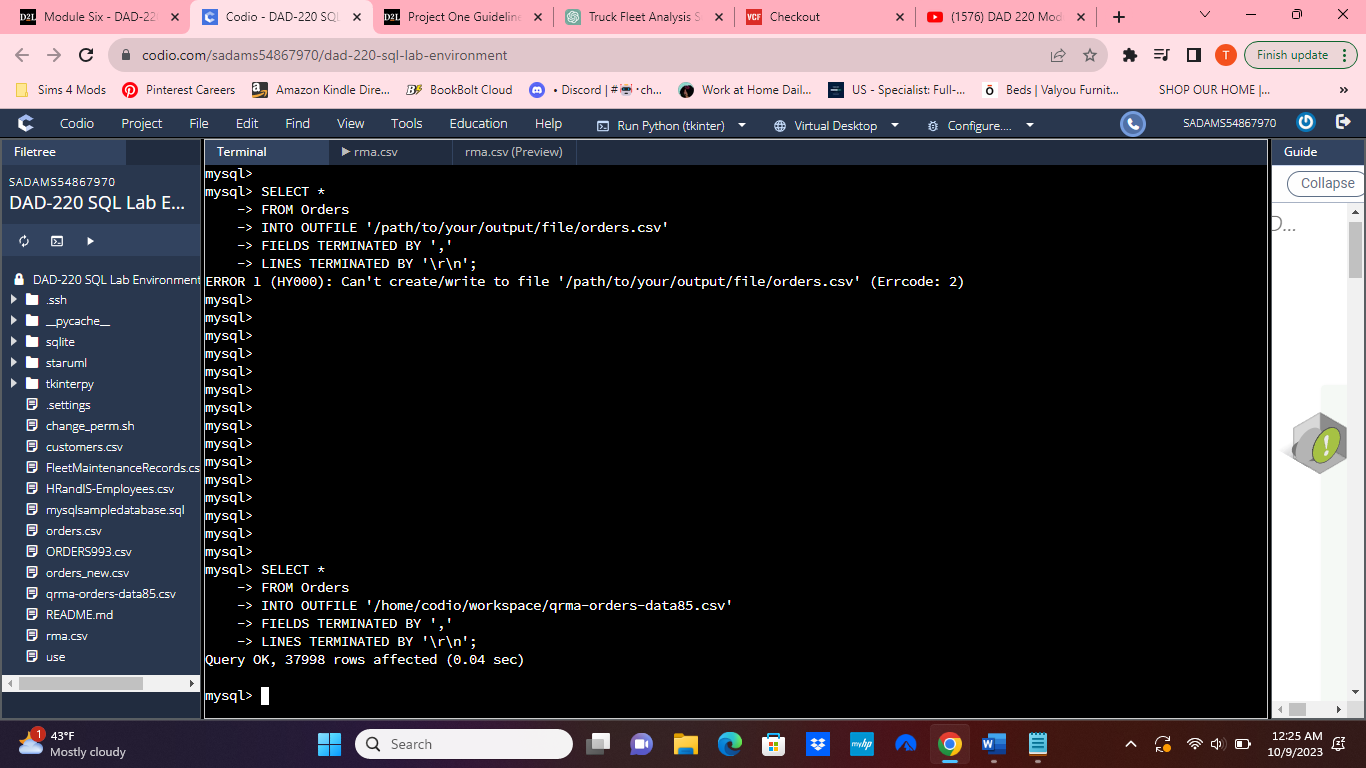
WHERE UPPER(Reason) LIKE '%REJ%';

This SQL command provided aims to delete records from the "RMA" table within the database that match a specific condition. In this case, the condition is specified in the "WHERE" clause. The command starts with the "DELETE FROM" statement, which indicates the intention to delete records from the specified table, which is "RMA" in this context. The "WHERE" clause further refines the operation by filtering records based on a specific condition. In this instance, the condition involves the "Reason" column, which is subject to case-insensitive comparison using the "UPPER()" function. The "LIKE" operator is employed to search for records where the "Reason" contains the substring "REJ" in any position. This means that records with reasons like "Rejected," "REJECTED," or "Rejection" will all be targeted for deletion. The command essentially deletes all RMA records where the reason partially matches "REJ," providing a flexible way to remove various types of rejected records from the table.

1. **Update your existing tables** from “Customer” to “Collaborator” using SQL based on this change in requirements. Provide the SQL commands you ran against MySQL to complete this successfully in your answer:
   1. Rename all instances of “Customer” to “Collaborator.”



1. **Create an output file of the required query results.** Write an SQL statement to list the contents of the **Orders** table and send the output to a file that has a .csv extension.



SELECT \*

FROM Orders

INTO OUTFILE '/home/codio/workspace/qrma-orders-data85.csv'

FIELDS TERMINATED BY ','

LINES TERMINATED BY '\r\n';

This SQL command is used to export the entire contents of the `Orders` table into a CSV (Comma-Separated Values) file named "qrma-orders-data85.csv." The `SELECT \*` statement retrieves all rows and columns from the `Orders` table. The `INTO OUTFILE` clause specifies the path and name of the output file, which in this case is set to '/home/codio/workspace/qrma-orders-data85.csv'. The `FIELDS TERMINATED BY ','` clause indicates that the fields in the CSV file should be separated by commas, a common convention in CSV files. Additionally, the `LINES TERMINATED BY '\r\n'` clause specifies that lines in the output file should be terminated with a carriage return followed by a newline character, which is a standard line-ending format. This command, when executed, will create the CSV file in the specified workspace directory, containing the data from the `Orders` table, formatted in CSV format for further analysis or storage.