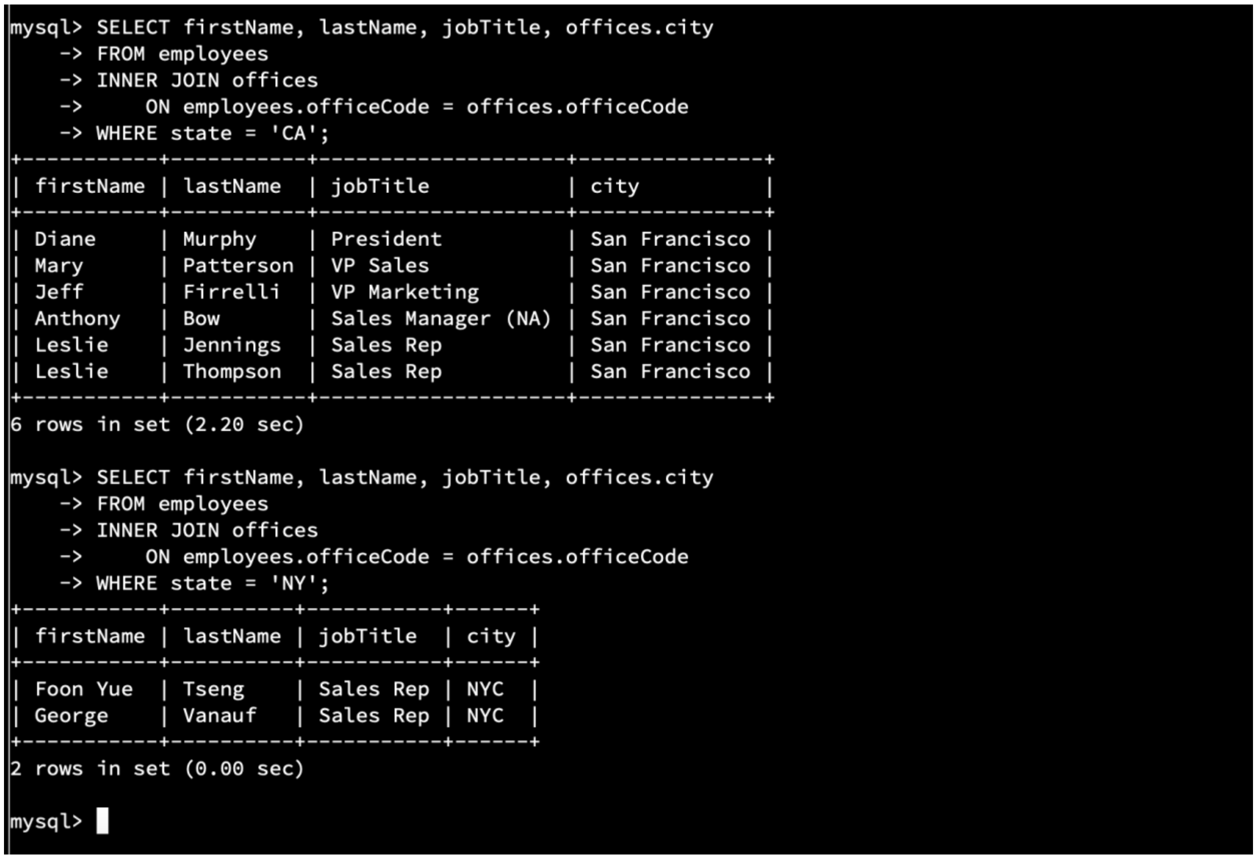
DAD 220 Cardinality and Targeted Data Template

Replace the bracketed text in this template with your screenshots and responses. Then submit it to the Module Four Lab for submission, grading, and feedback. Screenshots should be sized to approximately one quarter of a page. Written responses should be in complete sentences. Rename this document by adding your last name to the file name before you submit.

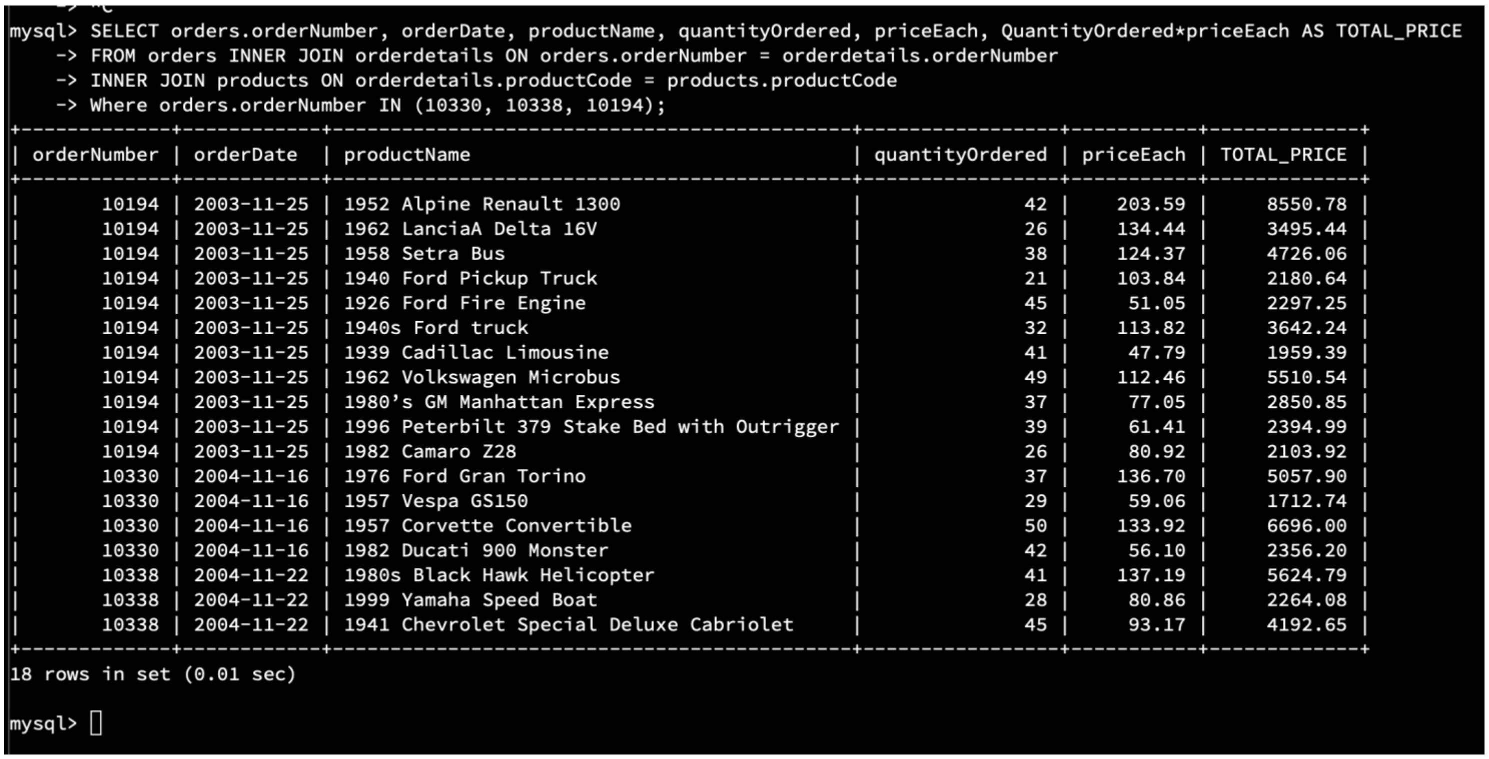
1. **Retrieve employee tuples and identify the number of employees** in San Francisco and New York.



```
mysql> SELECT firstName, lastName, jobTitle, offices.city
    -> FROM employees
    -> INNER JOIN offices
    ->     ON employees.officeCode = offices.officeCode
    -> WHERE state = 'CA';
+-----------+-----------+--------------------+---------------+
| firstName | lastName  | jobTitle           | city          |
+-----------+-----------+--------------------+---------------+
| Diane     | Murphy    | President          | San Francisco |
| Mary      | Patterson | VP Sales           | San Francisco |
| Jeff      | Firrelli  | VP Marketing       | San Francisco |
| Anthony   | Bow       | Sales Manager (NA) | San Francisco |
| Leslie    | Jennings  | Sales Rep          | San Francisco |
| Leslie    | Thompson  | Sales Rep          | San Francisco |
+-----------+-----------+--------------------+---------------+
6 rows in set (2.20 sec)

mysql> SELECT firstName, lastName, jobTitle, offices.city
    -> FROM employees
    -> INNER JOIN offices
    ->     ON employees.officeCode = offices.officeCode
    -> WHERE state = 'NY';
+-----------+----------+-----------+------+
| firstName | lastName | jobTitle  | city |
+-----------+----------+-----------+------+
| Foon Yue  | Tseng    | Sales Rep | NYC  |
| George    | Vanauf   | Sales Rep | NYC  |
+-----------+----------+-----------+------+
2 rows in set (0.00 sec)

mysql> 
```

Command used: select firstName, lastName, jobTitle, offices.city from employees inner join offices on employees.officeCode = offices.officeCode where state = 'CA'.
This command allowed me to Show there are 6 employees in San Francisco and 2 employees in New York.

2. **Retrieve order details** for orderNumber 10330, 10338, and 10194 and **identify** what **type of cardinality** this represents in the entity relationship model.

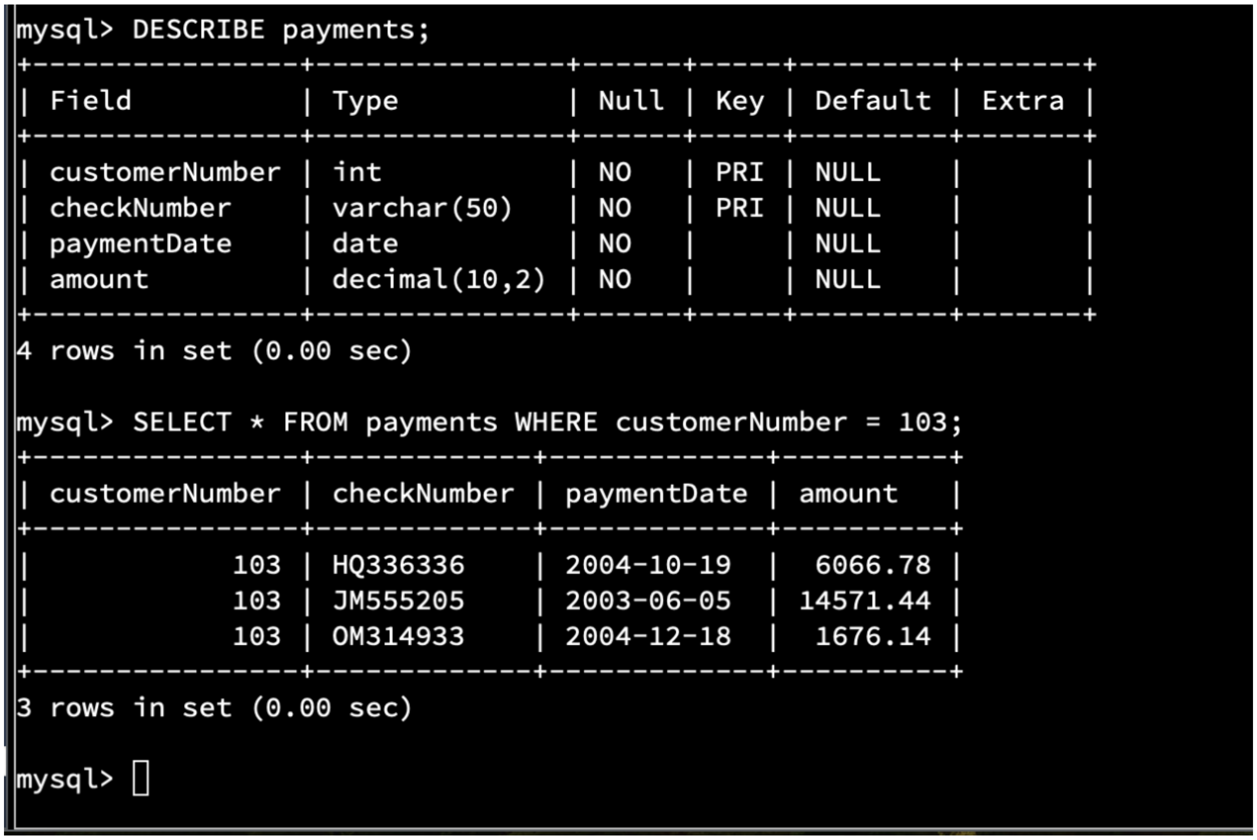


```
mysql> SELECT orders.orderNumber, orderDate, productName, quantityOrdered, priceEach, QuantityOrdered*priceEach AS TOTAL_PRICE
    -> FROM orders INNER JOIN orderdetails ON orders.orderNumber = orderdetails.orderNumber
    -> INNER JOIN products ON orderdetails.productCode = products.productCode
    -> Where orders.orderNumber IN (10330, 10338, 10194);
+-------------+------------+---------------------------------------------+-----------------+-----------+-------------+
| orderNumber | orderDate  | productName                                 | quantityOrdered | priceEach | TOTAL_PRICE |
+-------------+------------+---------------------------------------------+-----------------+-----------+-------------+
|       10194 | 2003-11-25 | 1952 Alpine Renault 1300                    |              42 |    203.59 |     8550.78 |
|       10194 | 2003-11-25 | 1962 LanciaA Delta 16V                      |              26 |    134.44 |     3495.44 |
|       10194 | 2003-11-25 | 1958 Setra Bus                              |              38 |    124.37 |     4726.06 |
|       10194 | 2003-11-25 | 1940 Ford Pickup Truck                      |              21 |    103.84 |     2180.64 |
|       10194 | 2003-11-25 | 1926 Ford Fire Engine                       |              45 |     51.05 |     2297.25 |
|       10194 | 2003-11-25 | 1940s Ford truck                            |              32 |    113.82 |     3642.24 |
|       10194 | 2003-11-25 | 1939 Cadillac Limousine                     |              41 |     47.79 |     1959.39 |
|       10194 | 2003-11-25 | 1962 Volkswagen Microbus                    |              49 |    112.46 |     5510.54 |
|       10194 | 2003-11-25 | 1980’s GM Manhattan Express                 |              37 |     77.05 |     2850.85 |
|       10194 | 2003-11-25 | 1996 Peterbilt 379 Stake Bed with Outrigger |              39 |     61.41 |     2394.99 |
|       10194 | 2003-11-25 | 1982 Camaro Z28                             |              26 |     80.92 |     2103.92 |
|       10330 | 2004-11-16 | 1976 Ford Gran Torino                       |              37 |    136.70 |     5057.90 |
|       10330 | 2004-11-16 | 1957 Vespa GS150                            |              29 |     59.06 |     1712.74 |
|       10330 | 2004-11-16 | 1957 Corvette Convertible                   |              50 |    133.92 |     6696.00 |
|       10330 | 2004-11-16 | 1982 Ducati 900 Monster                     |              42 |     56.10 |     2356.20 |
|       10338 | 2004-11-22 | 1980s Black Hawk Helicopter                 |              41 |    137.19 |     5624.79 |
|       10338 | 2004-11-22 | 1999 Yamaha Speed Boat                      |              28 |     80.86 |     2264.08 |
|       10338 | 2004-11-22 | 1941 Chevrolet Special Deluxe Cabriolet     |              45 |     93.17 |     4192.65 |
+-------------+------------+---------------------------------------------+-----------------+-----------+-------------+
18 rows in set (0.01 sec)

mysql> 
```

Command used: SELECT orders.orderNumber, orderDate, productName, quantityOrdered, priceEach, QuantityOrdered*priceEach AS TOTAL_PRICE
-> FROM orders INNER JOIN orderdetails ON orders.orderNumber = orderdetails.orderNumber
-> INNER JOIN products ON orderdetails.productCode = products.productCode
-> Where orders.orderNumber IN (10330, 10338, 10194);
The relationship is one of many. The order number can have many product names.

3. **Delete records** from the payments table where the customer number equals 103.

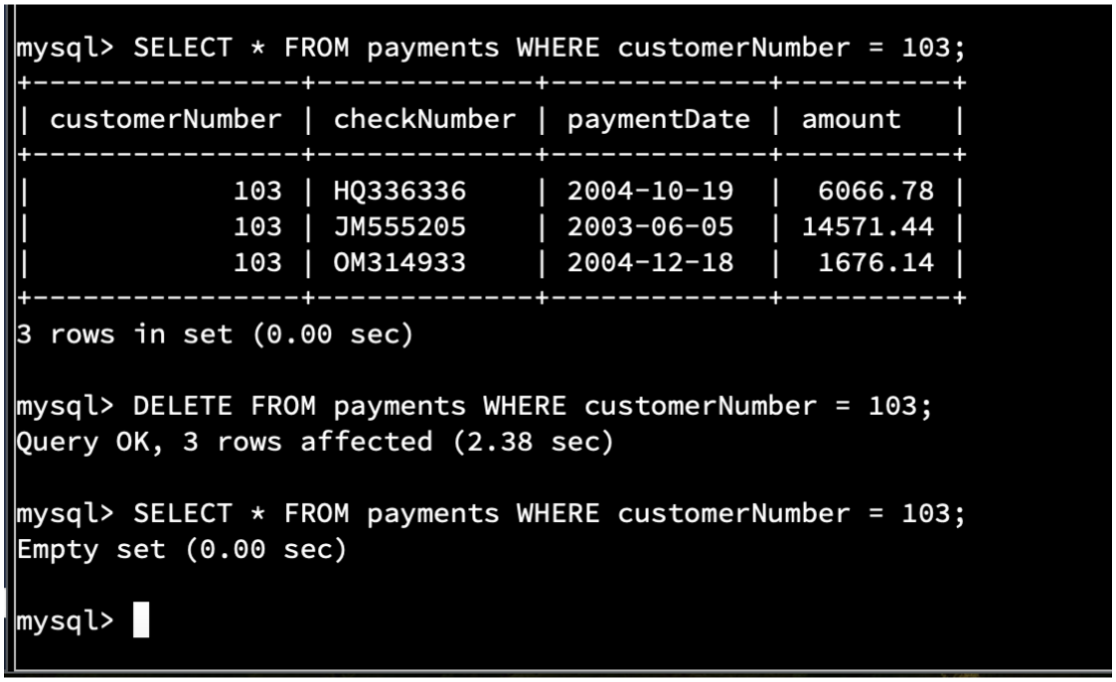
a.



```
mysql> DESCRIBE payments;
+----------------+---------------+------+-----+---------+-------+
| Field          | Type          | Null | Key | Default | Extra |
+----------------+---------------+------+-----+---------+-------+
| customerNumber | int           | NO   | PRI | NULL    |       |
| checkNumber    | varchar(50)   | NO   | PRI | NULL    |       |
| paymentDate    | date          | NO   |     | NULL    |       |
| amount         | decimal(10,2) | NO   |     | NULL    |       |
+----------------+---------------+------+-----+---------+-------+
4 rows in set (0.00 sec)

mysql> SELECT * FROM payments WHERE customerNumber = 103;
+----------------+-------------+-------------+----------+
| customerNumber | checkNumber | paymentDate | amount   |
+----------------+-------------+-------------+----------+
|            103 | HQ336336    | 2004-10-19  |  6066.78 |
|            103 | JM555205    | 2003-06-05  | 14571.44 |
|            103 | OM314933    | 2004-12-18  |  1676.14 |
+----------------+-------------+-------------+----------+
3 rows in set (0.00 sec)

mysql> 
```

Command used: DESCRIBE payments;
And, SELECT * FROM payments WHERE customerNumber = 103
This command is used to make sure the correct table is being updated.

b.



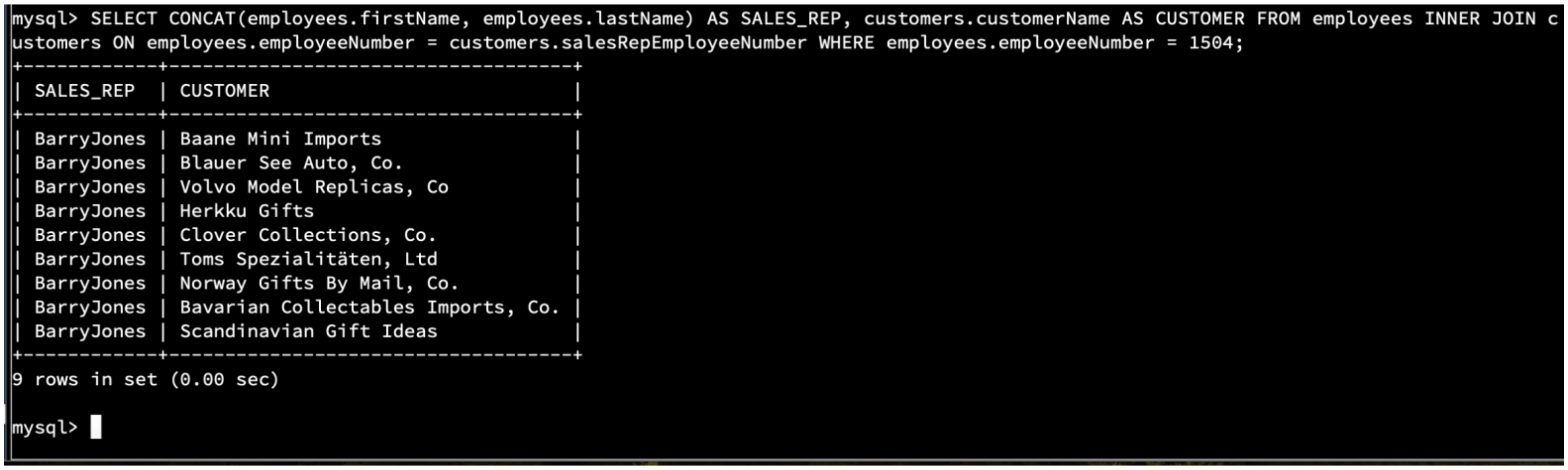
```
mysql> SELECT * FROM payments WHERE customerNumber = 103;
+----------------+-------------+-------------+----------+
| customerNumber | checkNumber | paymentDate | amount   |
+----------------+-------------+-------------+----------+
|            103 | HQ336336    | 2004-10-19  |  6066.78 |
|            103 | JM555205    | 2003-06-05  | 14571.44 |
|            103 | OM314933    | 2004-12-18  |  1676.14 |
+----------------+-------------+-------------+----------+
3 rows in set (0.00 sec)

mysql> DELETE FROM payments WHERE customerNumber = 103;
Query OK, 3 rows affected (2.38 sec)

mysql> SELECT * FROM payments WHERE customerNumber = 103;
Empty set (0.00 sec)

mysql> 
```

Command used: DELETE FROM payments WHERE customerNumber = 103;
And, SELECT * FROM payments WHERE customerNumber = 103;

The commands used were for deleting the customer account with account number 103 and showing the DELETE command worked.

4. **Retrieve customer records** for sales representative Barry Jones and **identify** if the **relationships** are one-to-one or one-to-many**.**

a.

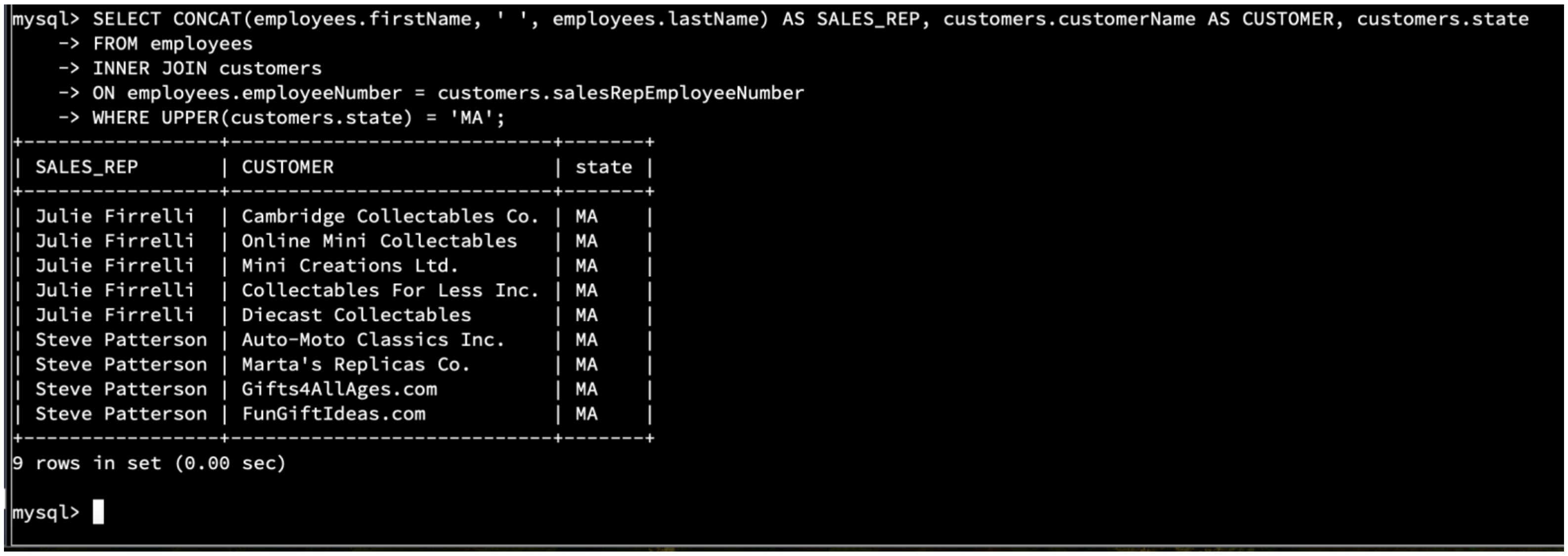


```
mysql> SELECT CONCAT(employees.firstName, employees.lastName) AS SALES_REP, customers.customerName AS CUSTOMER FROM employees INNER JOIN c
ustomers ON employees.employeeNumber = customers.salesRepEmployeeNumber WHERE employees.employeeNumber = 1504;
+------------+-----------------------------------+
| SALES_REP  | CUSTOMER                          |
+------------+-----------------------------------+
| BarryJones | Baane Mini Imports                |
| BarryJones | Blauer See Auto, Co.              |
| BarryJones | Volvo Model Replicas, Co          |
| BarryJones | Herkku Gifts                      |
| BarryJones | Clover Collections, Co.           |
| BarryJones | Toms Spezialitäten, Ltd           |
| BarryJones | Norway Gifts By Mail, Co.         |
| BarryJones | Bavarian Collectables Imports, Co. |
| BarryJones | Scandinavian Gift Ideas           |
+------------+-----------------------------------+
9 rows in set (0.00 sec)

mysql> 
```

Command used: SELECT CONCAT(employees.firstName, employees.lastName) AS SALES_REP, customers.customerName AS CUSTOMER FROM employees INNER JOIN customers ON employees.employeeNumber = customers.salesRepEmployeeNumber WHERE employees.employeeNumber = 1504;
This command allowed me to pull up a table of all of Barry Jones's customers. This shows a one-to-many relationship as he is one person with many customers.

5. **Retrieve records** for customers who reside in Massachusetts and **identify their sales rep and the relationship of entities**. Identify if these entities demonstrate one-to-one or many-to-many relationships.

a.



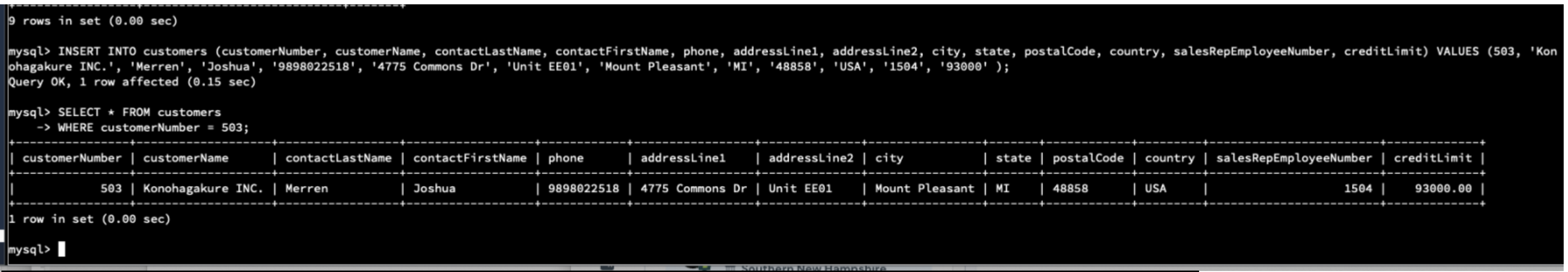
```
mysql> SELECT CONCAT(employees.firstName, ' ', employees.lastName) AS SALES_REP, customers.customerName AS CUSTOMER, customers.state
    -> FROM employees
    -> INNER JOIN customers
    -> ON employees.employeeNumber = customers.salesRepEmployeeNumber
    -> WHERE UPPER(customers.state) = 'MA';
+-----------------+----------------------------+-------+
| SALES_REP       | CUSTOMER                   | state |
+-----------------+----------------------------+-------+
| Julie Firrelli  | Cambridge Collectables Co. | MA    |
| Julie Firrelli  | Online Mini Collectables   | MA    |
| Julie Firrelli  | Mini Creations Ltd.        | MA    |
| Julie Firrelli  | Collectables For Less Inc. | MA    |
| Julie Firrelli  | Diecast Collectables       | MA    |
| Steve Patterson | Auto-Moto Classics Inc.    | MA    |
| Steve Patterson | Marta's Replicas Co.       | MA    |
| Steve Patterson | Gifts4AllAges.com          | MA    |
| Steve Patterson | FunGiftIdeas.com           | MA    |
+-----------------+----------------------------+-------+
9 rows in set (0.00 sec)

mysql> 
```

Command used: SELECT CONCAT(employees.firstName, ' ', employees.lastName) AS SALES_REP, customers.customerName AS CUSTOMER, customers.state
-> FROM employees

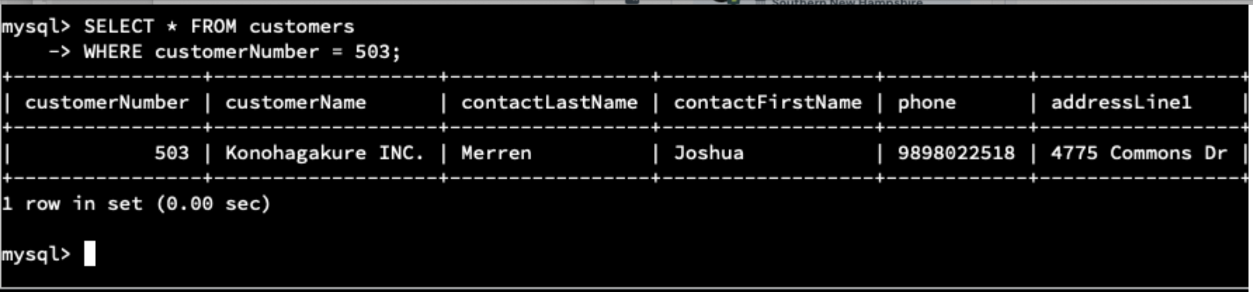
```
 -> INNER JOIN customers
 -> ON employees.employeeNumber = customers.salesRepEmployeeNumber
 -> WHERE UPPER(customers.state) = 'MA';
```

These commands allowed me to display all the customers who live in Massachusetts, and puts together the customer with their sales representative. These are a one-to-many relationship as both representatives are one person but have many customers.

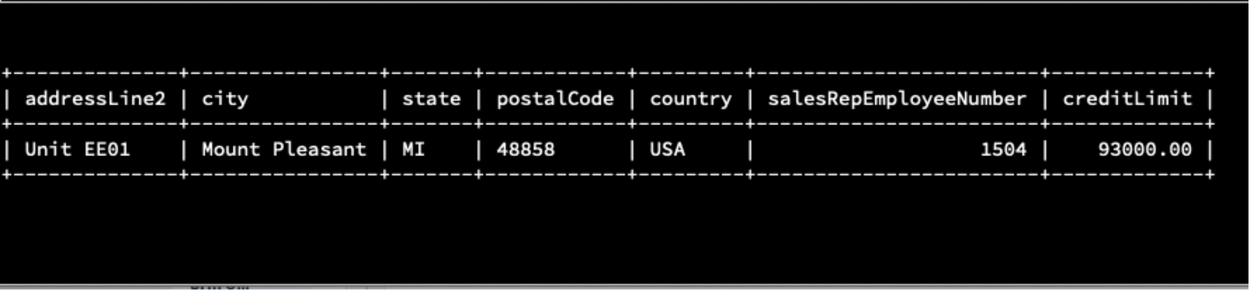
6. **Add one customer record** with your last name using an INSERT statement. You may use the name of a celebrity or fictional character if you don't use your own name.



```
9 rows in set (0.00 sec)

mysql> INSERT INTO customers (customerNumber, customerName, contactLastName, contactFirstName, phone, addressLine1, addressLine2, city, state, postalCode, country, salesRepEmployeeNumber, creditLimit) VALUES (503, 'Kon
ohagakure INC.', 'Merren', 'Joshua', '9898022518', '4775 Commons Dr', 'Unit EE01', 'Mount Pleasant', 'MI', '48858', 'USA', '1504', '93000' );
Query OK, 1 row affected (0.15 sec)

mysql> SELECT * FROM customers
    -> WHERE customerNumber = 503;
+----------------+-------------------+-----------------+------------------+------------+-----------------+--------------+----------------+-------+------------+---------+------------------------+-------------+
| customerNumber | customerName      | contactLastName | contactFirstName | phone      | addressLine1    | addressLine2 | city           | state | postalCode | country | salesRepEmployeeNumber | creditLimit |
+----------------+-------------------+-----------------+------------------+------------+-----------------+--------------+----------------+-------+------------+---------+------------------------+-------------+
|            503 | Konohagakure INC. | Merren          | Joshua           | 9898022518 | 4775 Commons Dr | Unit EE01    | Mount Pleasant | MI    | 48858      | USA     |                   1504 |    93000.00 |
+----------------+-------------------+-----------------+------------------+------------+-----------------+--------------+----------------+-------+------------+---------+------------------------+-------------+
1 row in set (0.00 sec)

mysql> 
```



```
mysql> SELECT * FROM customers
    -> WHERE customerNumber = 503;
+----------------+-------------------+-----------------+------------------+------------+-----------------+
| customerNumber | customerName      | contactLastName | contactFirstName | phone      | addressLine1    |
+----------------+-------------------+-----------------+------------------+------------+-----------------+
|            503 | Konohagakure INC. | Merren          | Joshua           | 9898022518 | 4775 Commons Dr |
+----------------+-------------------+-----------------+------------------+------------+-----------------+
1 row in set (0.00 sec)

mysql> 
```



```
+--------------+----------------+-------+------------+---------+------------------------+-------------+
| addressLine2 | city           | state | postalCode | country | salesRepEmployeeNumber | creditLimit |
+--------------+----------------+-------+------------+---------+------------------------+-------------+
| Unit EE01    | Mount Pleasant | MI    | 48858      | USA     |                   1504 |    93000.00 |
+--------------+----------------+-------+------------+---------+------------------------+-------------+
```

Command used: INSERT INTO customers (customerNumber, customerName, contactLastName, contactFirstName, phone, addressLine1, addressLine2, city, state, postalCode, country, salesRepEmployeeNumber, creditLimit) VALUES (503, 'Konohagakure INC.', 'Merren', 'Joshua', '9898022518', '4775 Commons Dr', 'Unit EE01', 'Mount Pleasant', 'MI', '48858', 'USA', '1504', '93000');
This command allowed me to enter my own information into the customers table. I then selected my customer Number by running the command SELECT * FROM customers WHERE customerNumber = 503;. This command allowed me to display only the new record of myself added to the table. It was difficult to read so I added the two other screenshots showing my information.

7. **Reflection**

a. **Define how cardinality is applied** to the databases you've been working with and why different numbers of records returned from the different offices.

Cardinality, in the context of our databases, refers to the relationships between tables, indicating how entities in one table relate to entities in another. This concept is crucial for designing effective database structures. In the office table, the primary key (officeCode) establishes a one-to-many (1:N) relationship with the employee's table through the foreign key officeCode. This means each office can have multiple employees, leading to different numbers of records returned for each office based on the number of associated employees.

Similarly, the customer's table has a foreign key (salesRepEmployeeNumber) connecting it to the employee's table, forming another one-to-many relationship. This relationship influences the number of records returned, as each sales representative can be associated with multiple customers. The orders table, with its foreign key (customerNumber), also contributes to varied record counts, as each customer can have multiple orders.

b. **Compare and contrast** the different **queries** you ran and how cardinality applies to them.

The queries conducted throughout this assignment offered insights into various cardinality scenarios, predominantly focusing on one-to-many relationships. The first query, involving an INNER JOIN of the orders and orderDetails tables, showed a one-to-many relationship. In this case, each order had multiple rows with different product names, reflecting the variety of products within an order. This cardinality scenario is crucial for understanding how products are associated with specific orders, providing a complete view of the items included in each transaction.

Another instance showing a one-to-many relationship was the query to retrieve customer records for Barry Jones. This query resulted in multiple rows, each representing a unique customer associated with Barry Jones. As a sales representative, Barry Jones is one entity connected to numerous customers, emphasizing the one-to-many cardinality. This scenario highlights the relationship between sales representatives and their clientele, illustrating the different amounts of customers managed by a single representative.

c. **Describe two** of the crucial **benefits of cardinality** in this type of database.

1. Data Integrity and Validation: Cardinality is pivotal in maintaining data integrity and validation. Accurately defining relationships between tables, such as one-to-many relationships between offices, employees, customers, and orders, ensures that each piece of information is correctly associated. For example, when an order is placed, it can be linked accurately to the specific customer, the employee assisting the customer, and the office's location. Avoiding data duplication and discrepancies enhances the correctness and dependability of the business's records.

2. Query Optimization: Understanding cardinality is essential for optimizing database queries. It guides the database management system in making efficient decisions on executing various queries. For instance, recognizing a one-to-many relationship informs the system to choose the most efficient join strategy, which enhances query performance. This optimization is crucial for improving the responsiveness of the database and ensuring efficient utilization of system resources.

In addition to these benefits, cardinality contributes to resource utilization efficiency, influencing decisions about indexing, caching, and partitioning strategies. By adequately defining cardinality, organizations can simplify application logic, ensure clarity and consistency in data, and extract valuable business insights to enhance their operations.