Tufts University
CS 115: Database Systems
Problem Set 2
Spring 2022

Complete the following exercises to the best of your ability and submit your answers on Gradescope by 11:59 PM on Wednesday, March 2, 2022. There will be a 10% deduction for submissions made by Thursday, March 3 at 11:59 PM, and a 20% deduction for submissions made by Friday, March 4 at 11:59 PM.

You can access Gradescope here: https://www.gradescope.com/. Click Log In and then School Credentials and scroll down to Tufts to log in using your Tufts credentials.

Format your answers as a PDF file. I recommend making a copy of this file as a Google Doc (File > Make a copy), filling in your answers, and then saving it as a PDF (File > Download > PDF document). You can also download this Google Doc as a Word document and save that as a PDF, or use other software entirely.

If you choose to, you can work with 1-2 partners on this assignment. If you worked with partners, mark in Gradescope the partners that you worked with. Make only one submission per group.

Direct any questions about the assignment to Piazza.

Problem 1 (10 points, 2 points each)

Consider the following tables:

Package

package_id	ship_date	weight
110934	2020-02-29	13 oz.
930955	2020-03-03	5 oz.
123456	2020-02-20	1.1 lbs
298475	2020-02-29	null

ShippedPackage

package_id	shipping_id
110934	100
930955	102
298475	100

ShippingType

shipping_id	name	price
100	priority	15.50
101	two day	18.50
102	next day	25.99
103	tracking	10.99

Where ShippedPackage(package_id) is a foreign key to Package(package_id), and ShippedPackage(shipping_id) is a foreign key to ShippingType(shipping_id)

Which of these insertions would be allowed?

If an insertion would *not* be allowed, explain why.

If an insertion *would* be allowed, assume that it does not actually get executed, i.e., assume for each question that the database is as it appears above. No explanation is necessary for insertions that you think are successful.

a. Insert ('345342', '2020-02-29', null) into Package

Would be allowed.

b. Insert ('298475', '100') into ShippedPackage

Would not be allowed because the primary key of ShippedPackage is (package_id, shipping_id) and must be unique. ('298475', '100') already is an entry in ShippedPackage so it cannot be inserted.

c. Insert ('298475', '103') into ShippedPackage

Would be allowed.

d. Insert ('345344', '103') into ShippedPackage

Would not be allowed because ShippedPackage(package_id) is a foreign key to Package(package_id) and there is no entry in Package where Package(package_id) = ‘345344’.

e. Insert ('103', 'international', 55.99) into ShippingType

Would not be allowed because the primary key of ShippingType is (shipping_id) and must be unique. There is already an entry in ShippingType with a shipping_id of ‘103’, so ('103', 'international', 55.99) cannot be inserted.

Problem 2 (10 points)

Consider the following SQL query:

```
SELECT result.officeCode, num1, COUNT(*) AS num2
FROM employees E, customers C, (SELECT officeCode, COUNT(*) AS num1
                                FROM employees
                                GROUP BY officeCode) AS result
WHERE E.employeeNumber = C.salesRepEmployeeNumber
  AND E.officeCode = result.officeCode
GROUP BY result.officeCode, num1;
```

We want to figure out what this query does, but unfortunately (for you), the writer of this query did not descriptively name the aliases used in the query (`num1`, `num2`, and `result`).

Describe what this query does in common terms (i.e., not using technical terminology). What result does it find? Included in your description, you must specifically mention what `num1`, `num2`, and `result` represent.

Note: we discussed in class that only columns that appear in a `GROUP BY` clause can appear alongside an aggregate function in the `SELECT` clause. Therefore, you should read **`GROUP BY result.officeCode, num1`** to mean: form groups using `result.officecode`, and include `num1` just so that it can also appear in the `SELECT` clause.

This query gets the office code of each office, the total number of employees that work in each office, and the total number of customers who have a sales representative that works in each office.

`num1:` represents the total number of employees working in each office.
`num2:` represents the total number of customers that have a sales representative in a respective office.
`result:` represents a table that includes each office code (one column) and the total number of employees working in each office (one column).

Problem 3 (15 points)

a. (10 points) Write a `CREATE TABLE` statement that creates a `Professor` table with:

- Professor ID, a fixed-length string of 7 characters, which is the primary key.
- Email, a variable-length string of up to 8 characters, which should be unique.
- Name, a variable-length string of up to 32 characters, which should be not null.
- Salary, a real number that is non-negative
- Course taught, a fixed-length string of 7 characters, which refers to a primary key column `course_code` in a table named `Course` (not given in this exercise, but pretend it exists). In this schema, each professor teaches exactly one course, so this field should not be allowed to be null.

```
CREATE TABLE Professor (
      id CHAR(7) PRIMARY KEY,
      email VARCHAR(8),
      name VARCHAR(32) NOT NULL,
      salary REAL CHECK (salary >= 0),
      course_taught CHAR(7) NOT NULL,
      UNIQUE (email),
      FOREIGN KEY (course_taught) REFERENCES Course(course_code)
);
```

b. (5 points) Write an `INSERT` statement that inserts the following row into the `Professor` table:

'Cody Doucette', with professor ID 'P123456', whose email handle is 'cody.doucette'. He teaches the course 'CS115-1'. There is no information available about his salary.

```
INSERT INTO Professor(id, email, name, salary, course_taught) VALUES
('P123456', 'cody.doucette', 'Cody Doucette', null, 'CS115-1');
```

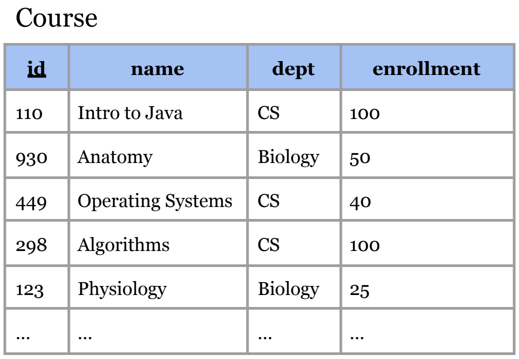
Problem 4 (15 points)

In lecture, we did not cover the `LIMIT` clause. `LIMIT` can be used (sometimes with an `ORDER BY` clause) to only include a certain number of results. For example, if we only wanted to show the top three R-rated movies with the highest earnings rank from the `Movie` table, we could write:

```
SELECT name
FROM Movie
WHERE rating = 'R'
ORDER BY earnings_rank
LIMIT 3;
```

This query selects only the movies whose rating is R, orders them by their `earnings_rank` (with earnings rank 1 being the highest), and then *limits* the results to only the first 3 rows.

Imagine a scenario in which we want to find the biology course with the largest enrollment from the following (only partially shown) table:



Course

id	name	dept	enrollment
110	Intro to Java	CS	100
930	Anatomy	Biology	50
449	Operating Systems	CS	40
298	Algorithms	CS	100
123	Physiology	Biology	25
...	...	...	...

Consider this query, which uses `LIMIT 1` to attempt to do so:

```
SELECT name
FROM Course
WHERE dept = 'Biology'
ORDER BY enrollment DESC
LIMIT 1;
```

a. (5 points) Describe a situation in which this query will not produce the desired results. Hint: think about what this query would show if it was instead run with `dept = 'CS'` given the partial table above.

This query will not produce the desired results when more than one class has the largest enrollment (i.e. multiple classes are tied for largest enrollment) because it will only return one of them.

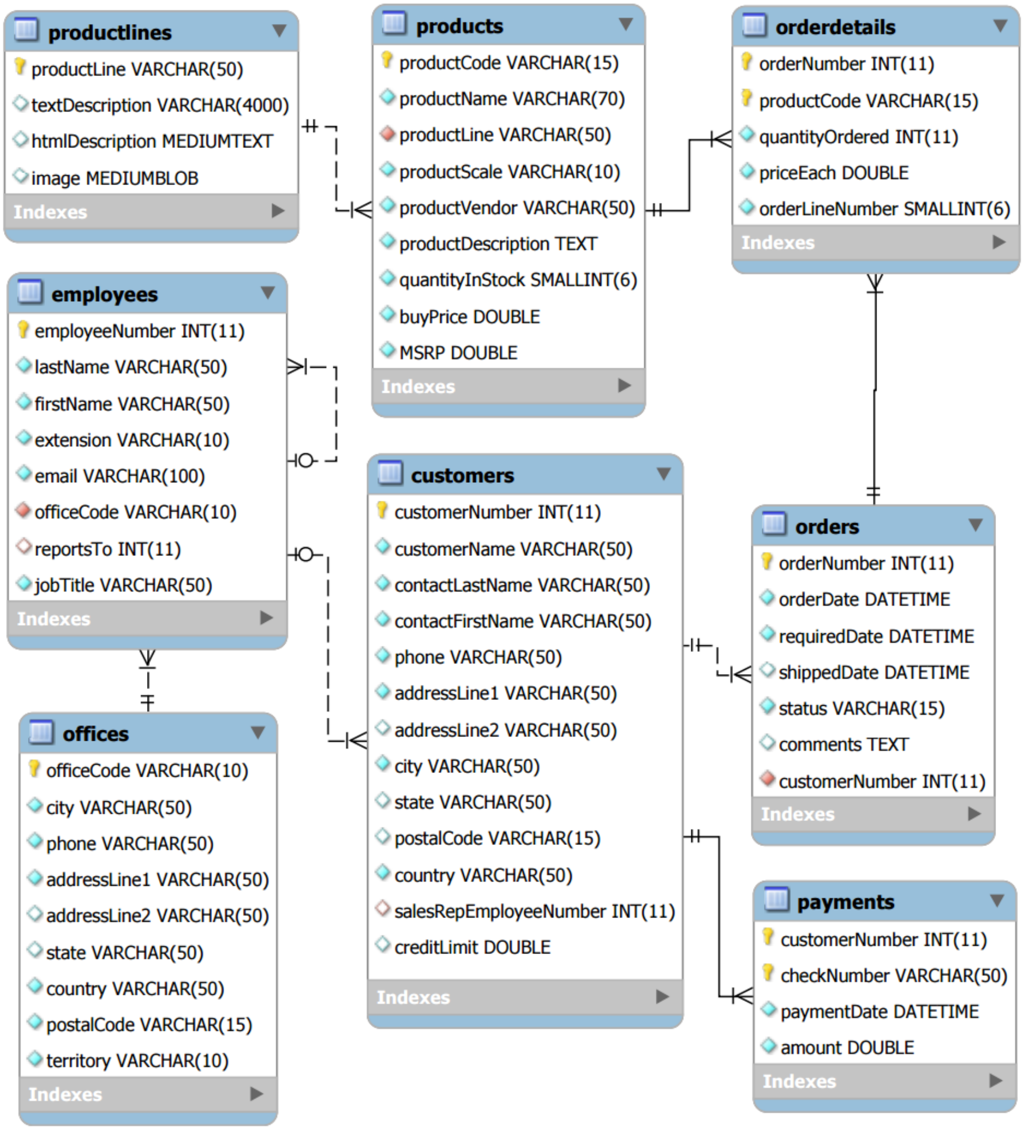
b. (10 points) Rewrite this query without using `LIMIT` so that it corrects the issue you identified in (a). You should not need any tables or columns other than those used in the query above.

```
SELECT name
FROM Course
WHERE dept = 'Biology' AND enrollment = (SELECT MAX(enrollment) FROM
Course);
```

Problem 5 (50 points, 5 points for each query)

Database Description

The following SQL queries will use the practice database that we've been using in lecture. Here's an ER diagram of the database:



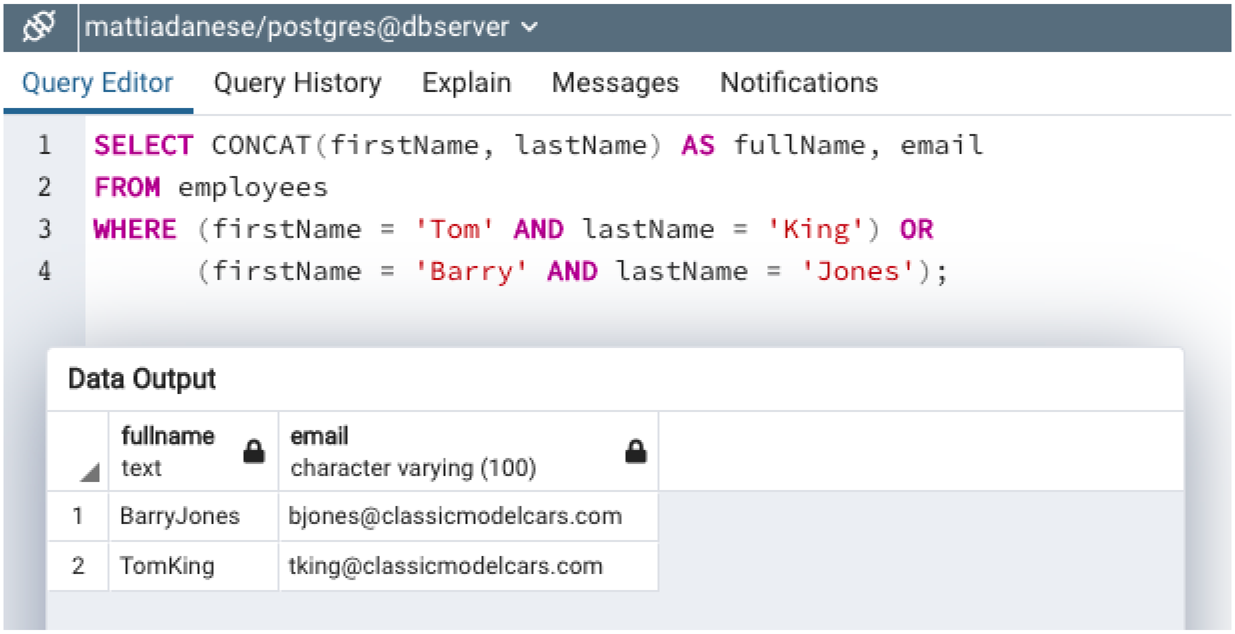
Instructions

Write SQL queries to answer the prompts below. Your answer for each problem must be a **single** query. Except where noted, for each answer you should include both (1) the query itself (2) a screenshot of the query results. If you are missing either one, you will not receive any credit for that problem.

Query Problems

1. Find the email addresses for employees Tom King and Barry Jones. The resulting tuples should be of the form (full name, email address). You should use their names in the query itself, i.e., don't find out what their IDs are and use the IDs in the query.

```
SELECT CONCAT(firstName, lastName) AS fullName, email
FROM employees
WHERE (firstName = ‘Tom’ AND lastName = ‘King’) OR
      (firstName = ‘Barry’ AND lastName = ‘Jones’);
```



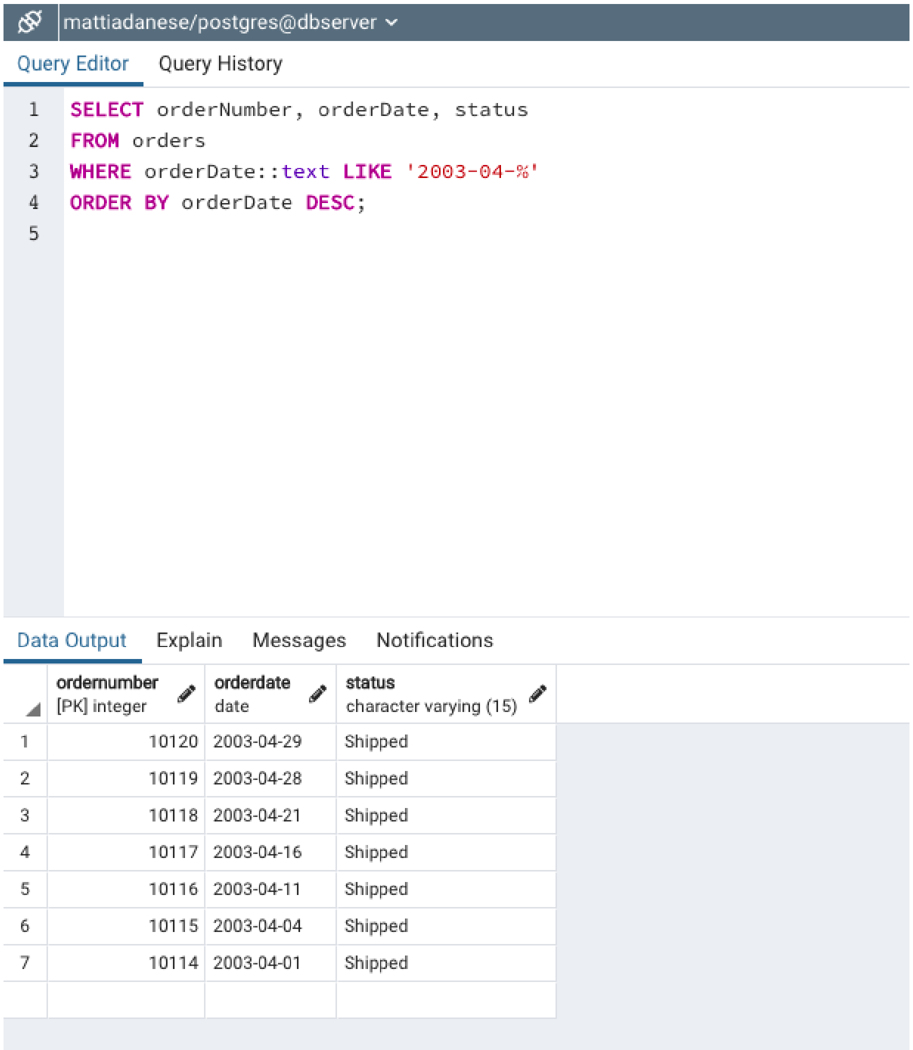
```
SELECT CONCAT(firstName, lastName) AS fullName, email
FROM employees
WHERE (firstName = 'Tom' AND lastName = 'King') OR
      (firstName = 'Barry' AND lastName = 'Jones');
```

Data Output

	fullname text	email character varying (100)
1	BarryJones	bjones@classicmodelcars.com
2	TomKing	tking@classicmodelcars.com

2. Find all orders from April, 2003. The resulting tuples should be of the form (order number, order date, status). Sort the tuples from most recent to least recent.

```
SELECT orderNumber, orderDate, status
FROM orders
WHERE orderDate::text LIKE ‘2003-04-%’
ORDER BY orderDate DESC;
```

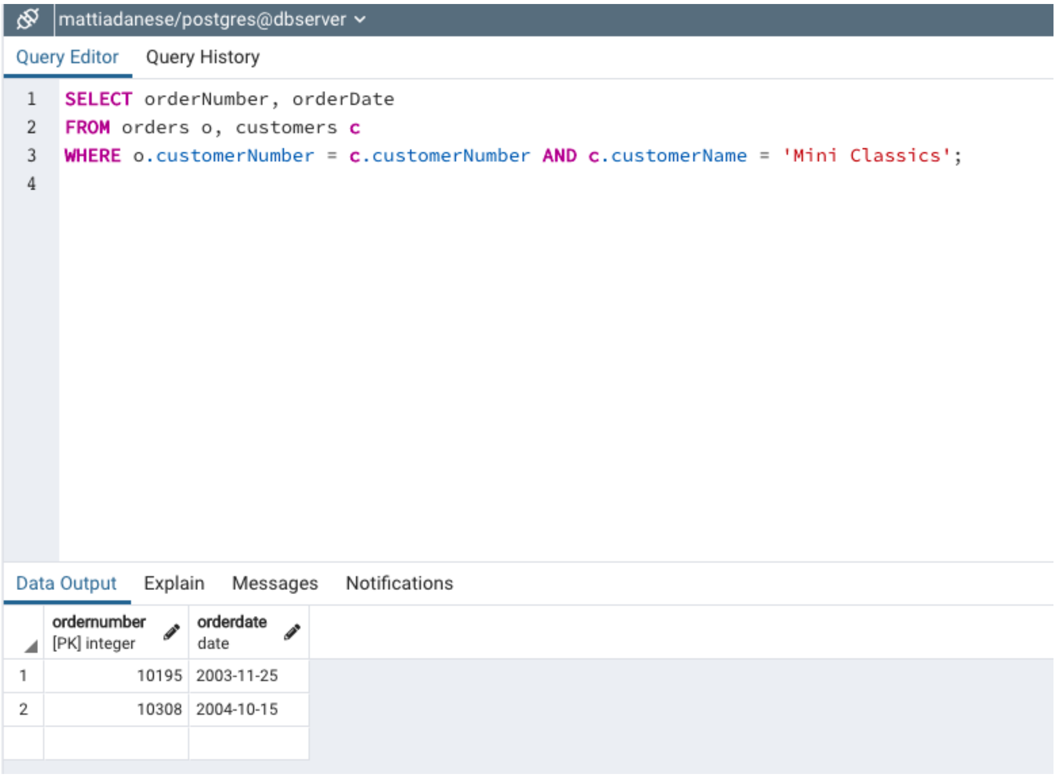


```
SELECT orderNumber, orderDate, status
FROM orders
WHERE orderDate::text LIKE '2003-04-%'
ORDER BY orderDate DESC;
```

	ordernumber [PK] integer	orderdate date	status character varying (15)
1	10120	2003-04-29	Shipped
2	10119	2003-04-28	Shipped
3	10118	2003-04-21	Shipped
4	10117	2003-04-16	Shipped
5	10116	2003-04-11	Shipped
6	10115	2003-04-04	Shipped
7	10114	2003-04-01	Shipped

3. Find all orders by customer Mini Classics. The resulting tuples should be of the form (order number, order date).

```
SELECT orderNumber, orderDate
FROM orders o, customers c
WHERE o.customerNumber = c.customberNumber AND c.customerName = ‘Mini
Classics’;
```

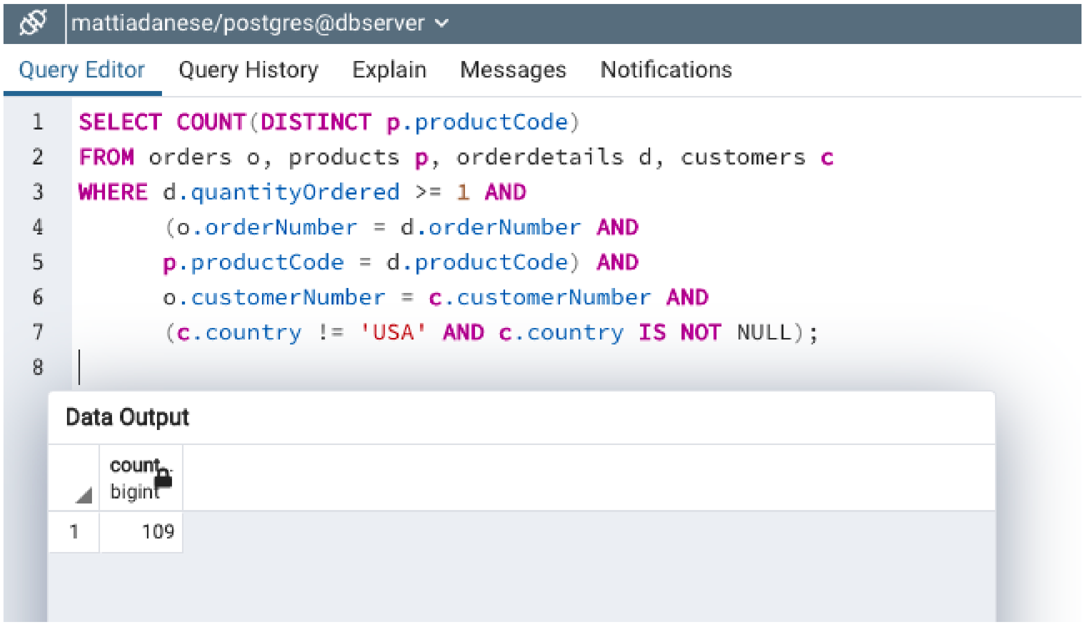


```
SELECT orderNumber, orderDate
FROM orders o, customers c
WHERE o.customerNumber = c.customerNumber AND c.customerName = 'Mini Classics';
```

	ordernumber [PK] integer	orderdate date
1	10195	2003-11-25
2	10308	2004-10-15

4. Find the number of products in the database that have been bought by at least one foreign customer. Foreign customers have a non-null country field whose value is not `'USA'`. The result of your query should be a single number.

```
SELECT COUNT(DISTINCT p.productCode)
FROM orders o, products p, orderdetails d, customers c
WHERE d.quantityOrdered >= 1 AND
      (o.orderNumber = d.orderNumber AND
      p.productCode = d.productCode) AND
      o.customerNumber = c.customerNumber AND
      (c.country != 'USA' AND c.country IS NOT NULL);
```

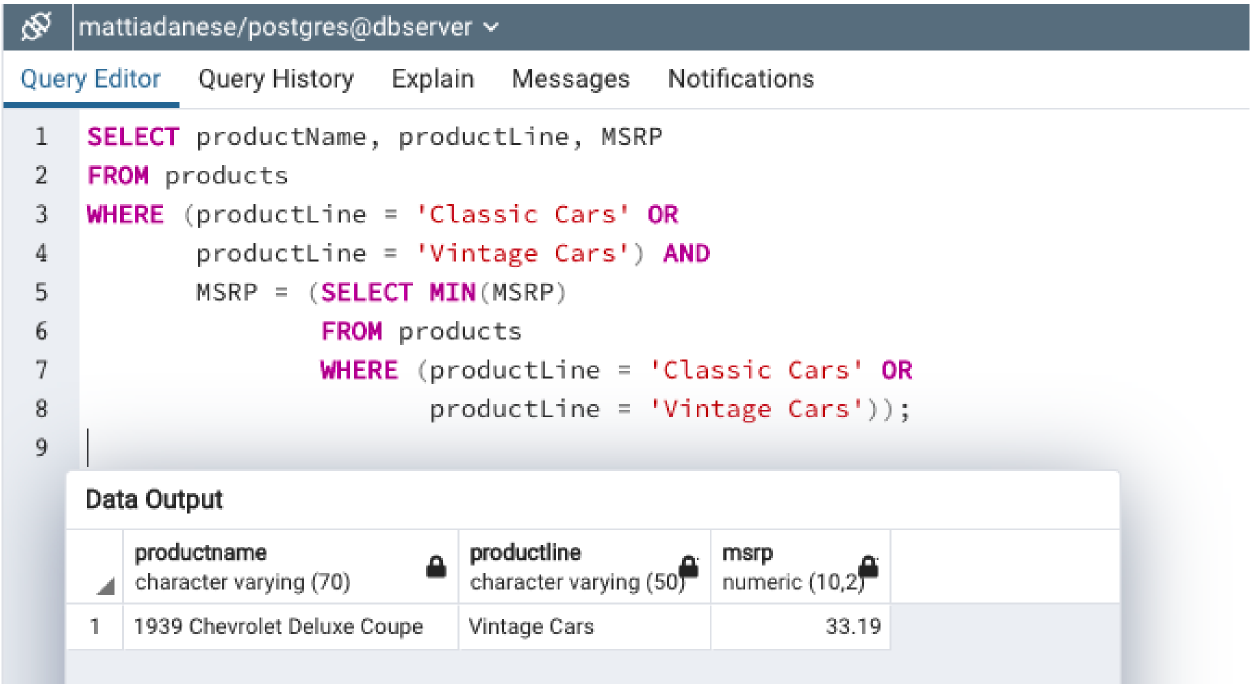


```
SELECT COUNT(DISTINCT p.productCode)
FROM orders o, products p, orderdetails d, customers c
WHERE d.quantityOrdered >= 1 AND
      (o.orderNumber = d.orderNumber AND
      p.productCode = d.productCode) AND
      o.customerNumber = c.customerNumber AND
      (c.country != 'USA' AND c.country IS NOT NULL);
```

```
   count
   bigint
1     109
```

5. Find the car product(s) in the database with the lowest MSRP. All car products in the database have a product line that is either Classic Cars or Vintage Cars. The result(s) of your query should be of the form (product name, product line, MSRP).

```
SELECT productName, productLine, MSRP
FROM products
WHERE (productLine = 'Classic Cars' OR
         productLine = 'Vintage Cars') AND
         MSRP = (SELECT MIN(MSRP)
                     FROM products
                     WHERE (productLine = 'Classic Cars' OR
                               productLine = 'Vintage Cars'));
```

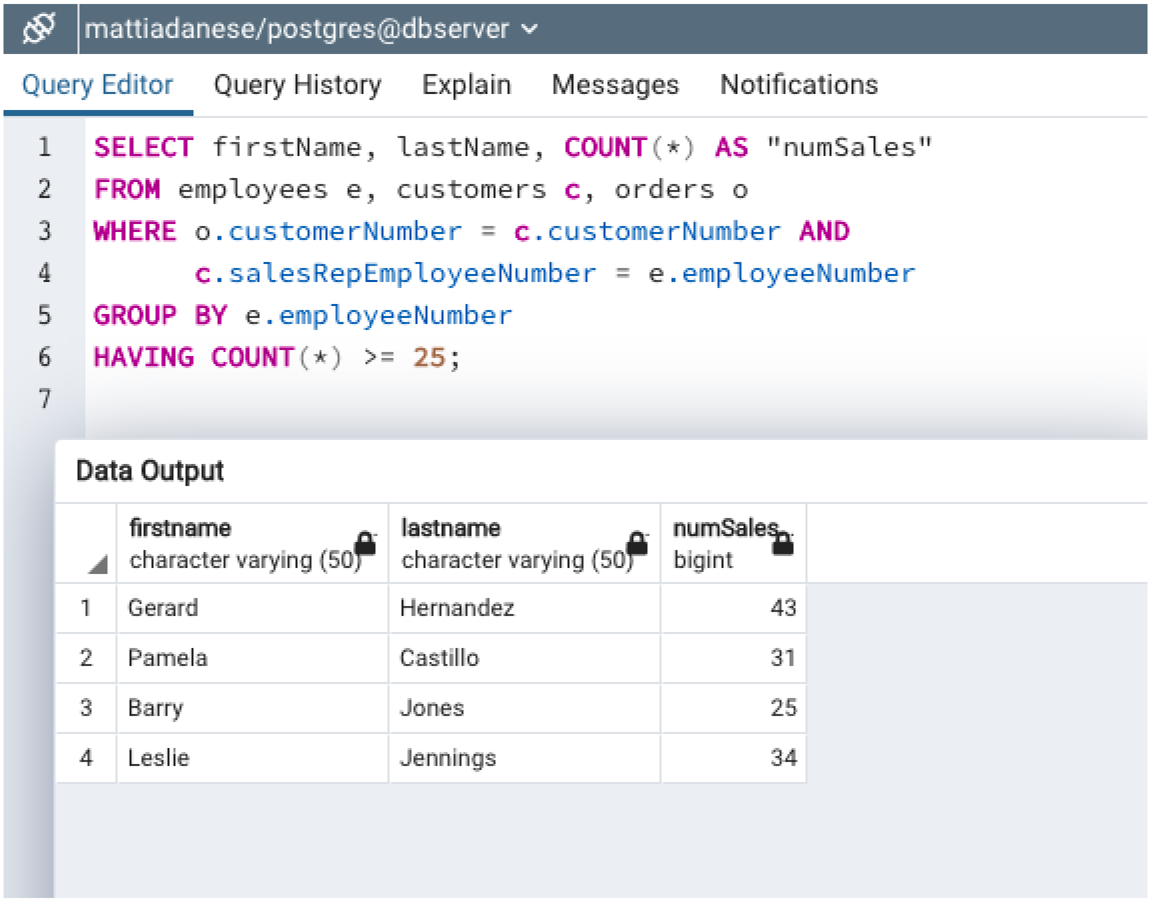


```
SELECT productName, productLine, MSRP
FROM products
WHERE (productLine = 'Classic Cars' OR
       productLine = 'Vintage Cars') AND
       MSRP = (SELECT MIN(MSRP)
               FROM products
               WHERE (productLine = 'Classic Cars' OR
                      productLine = 'Vintage Cars'));
```

	productname character varying (70)	productline character varying (50)	msrp numeric (10,2)
1	1939 Chevrolet Deluxe Coupe	Vintage Cars	33.19

6. Find all employees in the database who have made at least 25 sales (i.e. have been the salesperson for at least 25 customer orders). The results of your query should be of the form (employee first name, employee last name, number of sales). Name the number of sales column `numSales`.

```
SELECT firstName, lastName, COUNT(*) AS “numSales”
FROM employees e, customers c, orders o
WHERE o.customerNumber = c.customerNumber AND
         c.salesRepEmployeeNumber = e.employeeNumber
GROUP BY e.employeeNumber
HAVING COUNT(*) >= 25;
```



```
SELECT firstName, lastName, COUNT(*) AS "numSales"
FROM employees e, customers c, orders o
WHERE o.customerNumber = c.customerNumber AND
      c.salesRepEmployeeNumber = e.employeeNumber
GROUP BY e.employeeNumber
HAVING COUNT(*) >= 25;
```

	firstname character varying (50)	lastname character varying (50)	numSales bigint
1	Gerard	Hernandez	43
2	Pamela	Castillo	31
3	Barry	Jones	25
4	Leslie	Jennings	34

7. For each customer, find the number of products from each product line that they have purchased. The results of your tuple should be of the form (customer name, product line name, number of products from this product line).

You don't need to include the query results for this one. Your answer should just be the query itself.

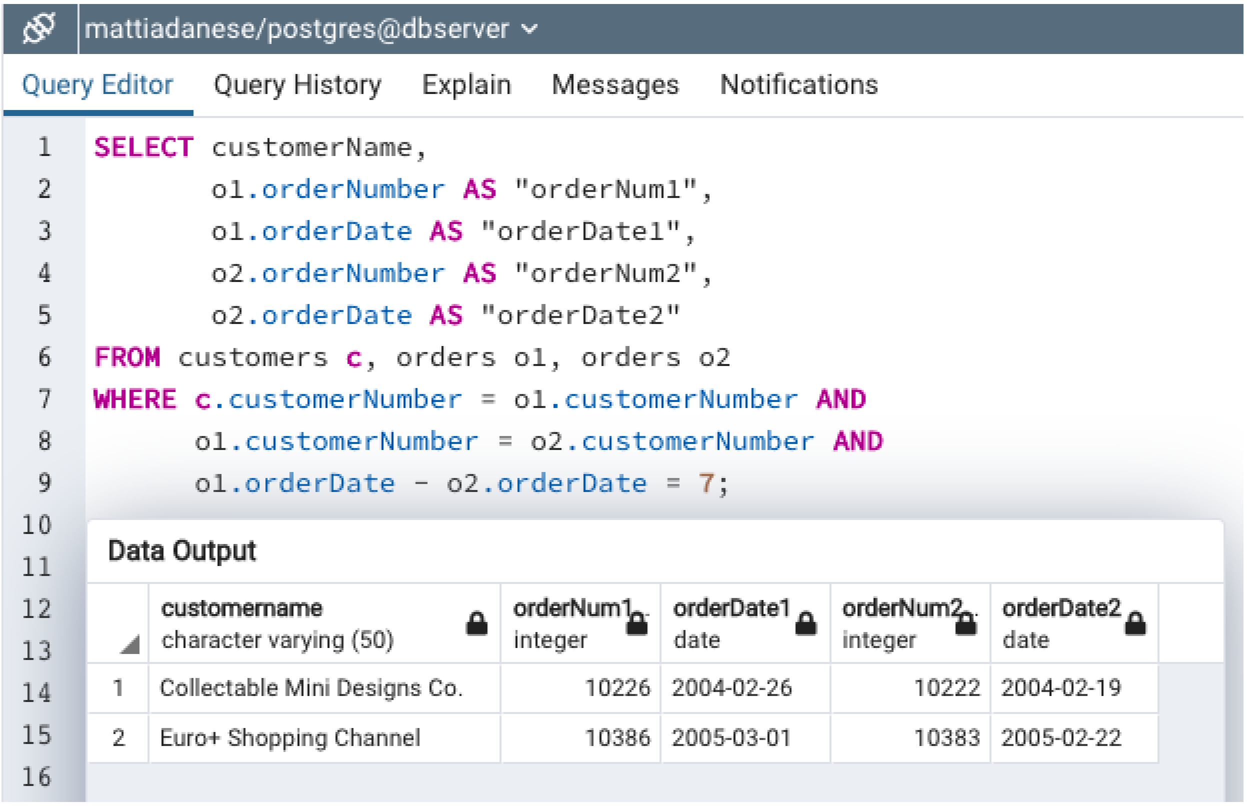
Hint: the results are for *each* customer and for *each* product line, meaning you will need a `GROUP BY` clause that groups by two columns.

```
SELECT c.customerName, p.productLine, COUNT(*) AS "numProducts"
FROM customers c, orders o, orderdetails d, products p
WHERE p.productCode = d.productCode AND
      d.orderNumber = o.orderNumber AND
      o.customerNumber = c.customerNumber
GROUP BY c.customerNumber, p.productLine
ORDER BY c.customerName;
```

8. Find all instances where the same customer placed different orders exactly 7 days apart. The results of your query should be of the form (customer name, order number 1, order date 1, order number 2, order date 2).

Hint: In order to get the difference between two dates, you can subtract them, i.e., for date columns d1 and d2, the number of days between the two dates is d1 - d2.

```
SELECT customerName,
       o1.orderNumber AS "orderNum1",
       o1.orderDate AS "orderDate1",
       o2.orderNumber AS "orderNum2",
       o2.orderDate AS "orderDate2"
FROM customers c, orders o1, orders o2
WHERE c.customerNumber = o1.customerNumber AND
         o1.customerNumber = o2.customerNumber AND
         o1.orderDate - o2.orderDate = 7;
```



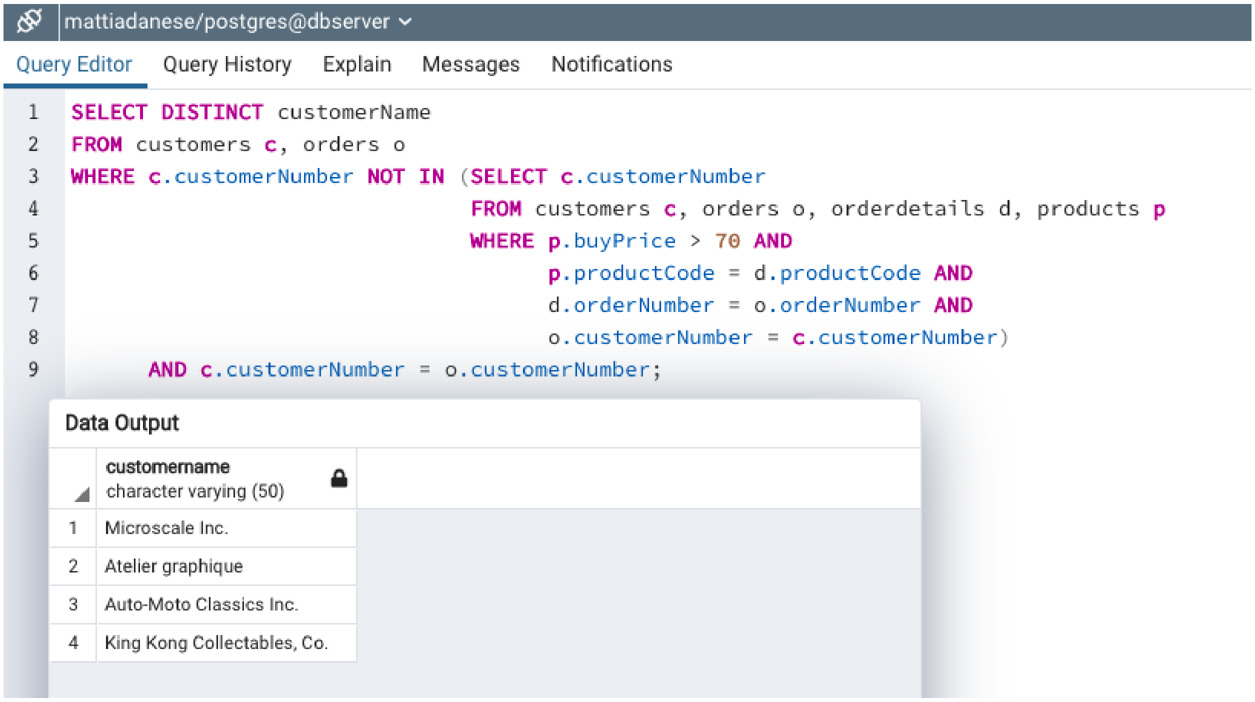
```
SELECT customerName,
       o1.orderNumber AS "orderNum1",
       o1.orderDate AS "orderDate1",
       o2.orderNumber AS "orderNum2",
       o2.orderDate AS "orderDate2"
FROM customers c, orders o1, orders o2
WHERE c.customerNumber = o1.customerNumber AND
      o1.customerNumber = o2.customerNumber AND
      o1.orderDate - o2.orderDate = 7;
```

Data Output

	customername character varying (50)	orderNum1 integer	orderDate1 date	orderNum2 integer	orderDate2 date
1	Collectable Mini Designs Co.	10226	2004-02-26	10222	2004-02-19
2	Euro+ Shopping Channel	10386	2005-03-01	10383	2005-02-22

9. In this database, there are some customers who have not made any orders. Among the customers who *have* made orders, find those who have *never* bought a product priced more than $70. The results should be a list of customer names.

```
SELECT DISTINCT customerName
FROM customers c, orders o
WHERE c.customerNumber NOT IN
            (SELECT c.customerNumber
             FROM customers c, orders o, orderdetails d, products p
             WHERE p.buyPrice > 70 AND
                   p.productCode = d.productCode AND
                   d.orderNumber = o.orderNumber AND
                   o.customerNumber = c.customerNumber)
      AND c.customerNumber = o.customerNumber;
```



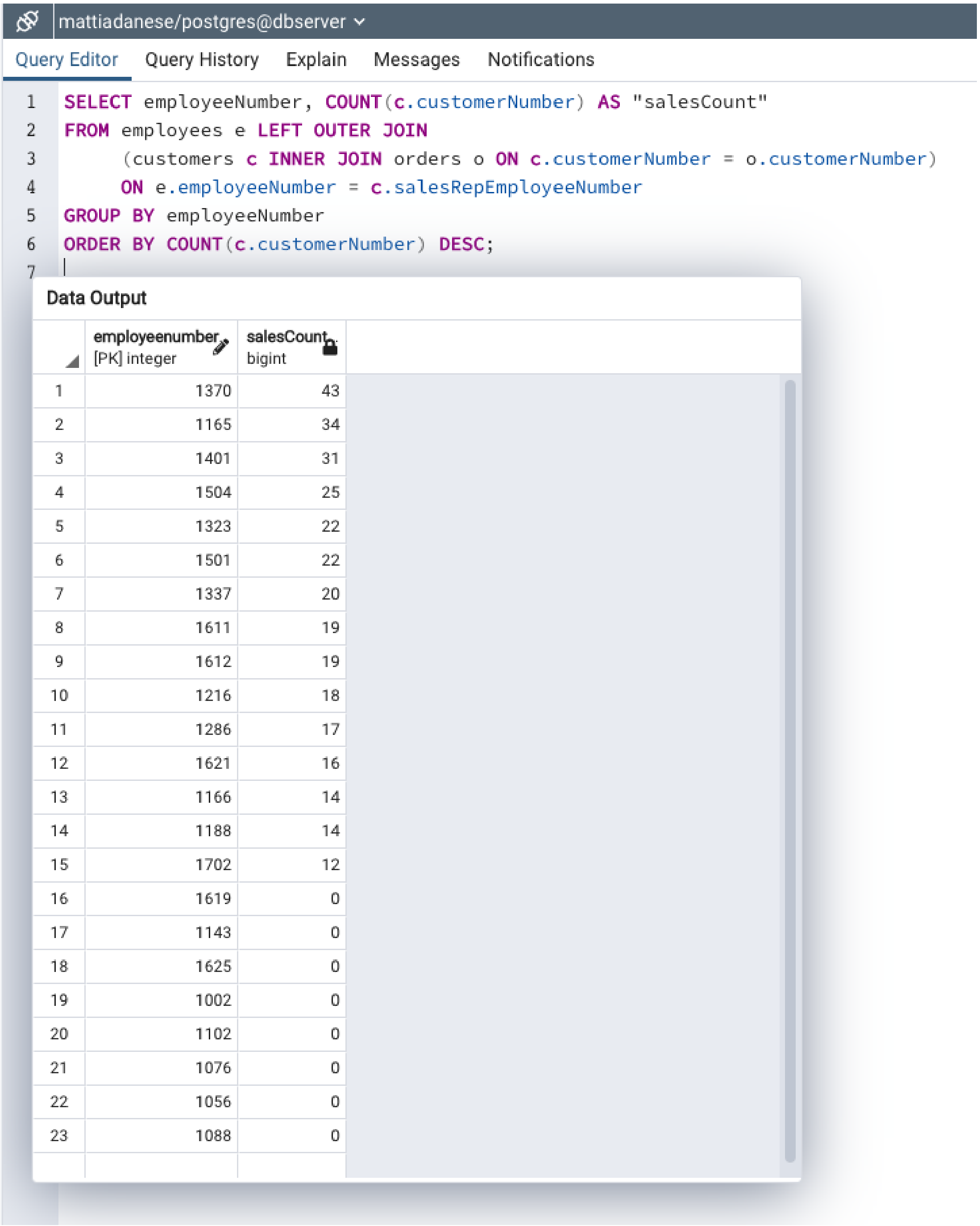
```
SELECT DISTINCT customerName
FROM customers c, orders o
WHERE c.customerNumber NOT IN (SELECT c.customerNumber
                               FROM customers c, orders o, orderdetails d, products p
                               WHERE p.buyPrice > 70 AND
                                     p.productCode = d.productCode AND
                                     d.orderNumber = o.orderNumber AND
                                     o.customerNumber = c.customerNumber)
      AND c.customerNumber = o.customerNumber;
```

	customername character varying (50)
1	Microscale Inc.
2	Atelier graphique
3	Auto-Moto Classics Inc.
4	King Kong Collectables, Co.

10. Find the number of sales made by each employee, including those who have not made any sales (e.g. the managers, the president, etc.). The results of your query should be of the form (employee ID, count of sales made). Employees who have never made a sale should have an associated count of zero.

```
SELECT employeeNumber, COUNT(c.customerNumber) AS "salesCount"
FROM employees e LEFT OUTER JOIN
                    (customers c INNER JOIN orders o ON
                    c.customerNumber = o.customerNumber)
                  ON e.employeeNumber = c.salesRepEmployeeNumber
GROUP BY employeeNumber
ORDER BY COUNT(c.customerNumber) DESC;
```

Ï



```
SELECT employeeNumber, COUNT(c.customerNumber) AS "salesCount"
FROM employees e LEFT OUTER JOIN
     (customers c INNER JOIN orders o ON c.customerNumber = o.customerNumber)
     ON e.employeeNumber = c.salesRepEmployeeNumber
GROUP BY employeeNumber
ORDER BY COUNT(c.customerNumber) DESC;
```

Data Output

	employeenumber [PK] integer	salesCount bigint
1	1370	43
2	1165	34
3	1401	31
4	1504	25
5	1323	22
6	1501	22
7	1337	20
8	1611	19
9	1612	19
10	1216	18
11	1286	17
12	1621	16
13	1166	14
14	1188	14
15	1702	12
16	1619	0
17	1143	0
18	1625	0
19	1002	0
20	1102	0
21	1076	0
22	1056	0
23	1088	0