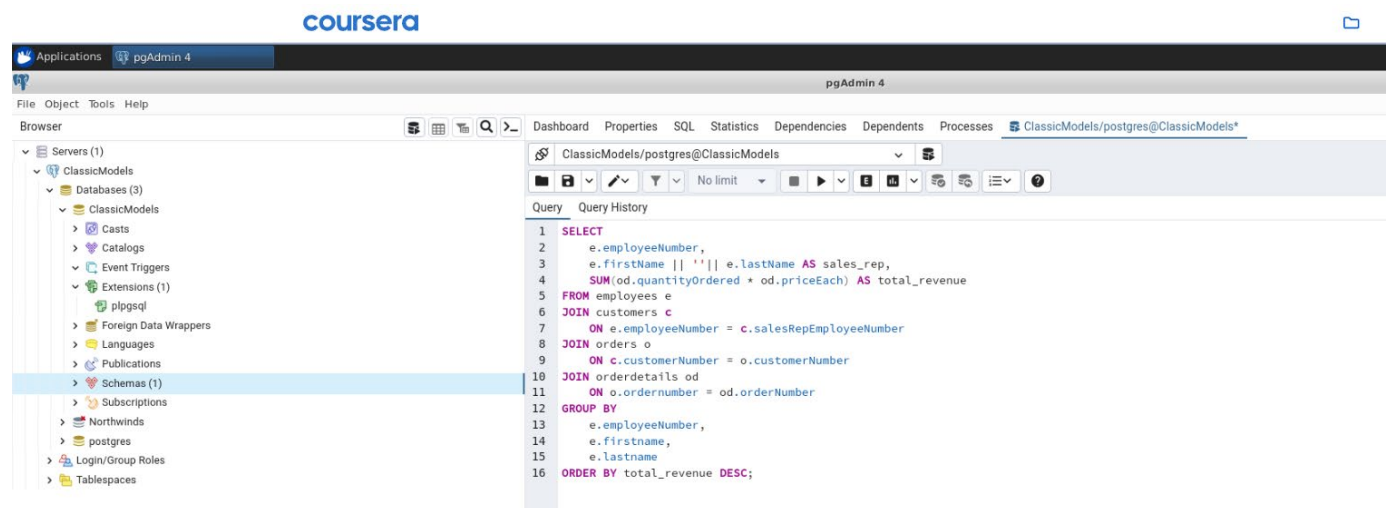


# Problem 1: Identifying High-Value Customers

## Analysis Challenge

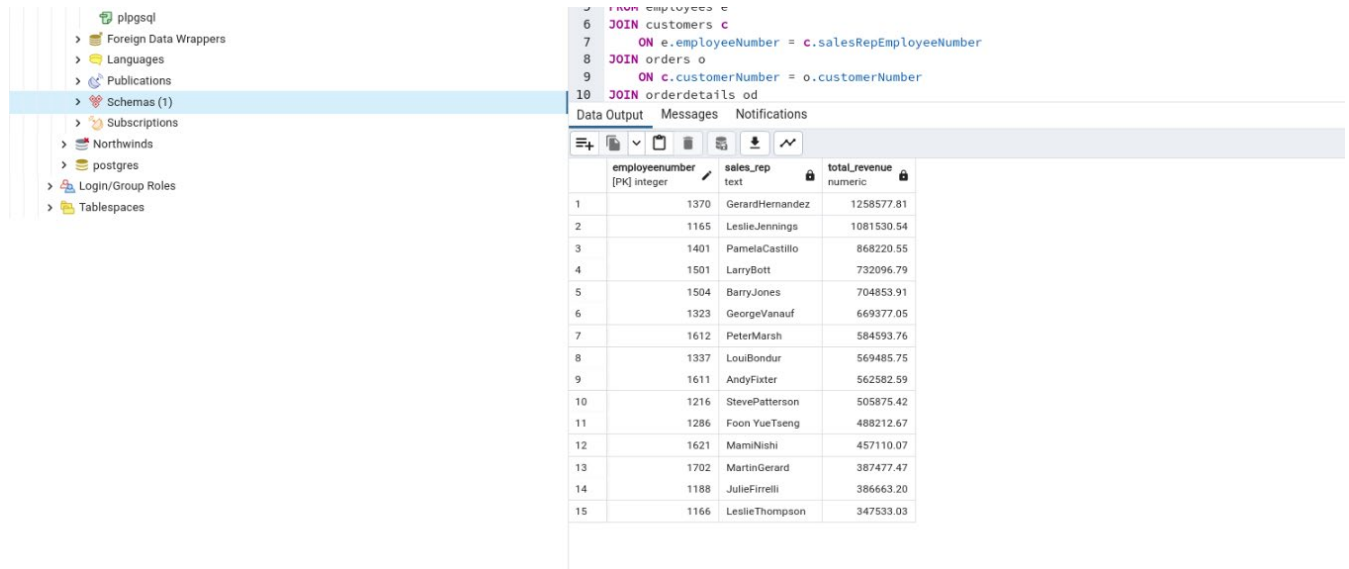
The sales team wants to focus retention efforts on customers who generate the most revenue. I need to identify customers whose total order value exceeds the average total order value across all customers. This analysis helps prioritize high-impact customer relationships.

## Code



```
SELECT
  e.employeeNumber,
  e.firstName || ' ' || e.lastName AS sales_rep,
  SUM(od.quantityOrdered * od.priceEach) AS total_revenue
FROM employees e
JOIN customers c
  ON e.employeeNumber = c.salesRepEmployeeNumber
JOIN orders o
  ON c.customerNumber = o.customerNumber
JOIN orderdetails od
  ON o.orderNumber = od.orderNumber
GROUP BY
  e.employeeNumber,
  e.firstName,
  e.lastName
ORDER BY total_revenue DESC;
```

## Output



The screenshot shows a database management tool interface. On the left is a sidebar with a tree view containing items like 'Foreign Data Wrappers', 'Languages', 'Publications', 'Schemas (1)', 'Subscriptions', 'Northwinds', 'postgres', 'Login/Group Roles', and 'Tablespaces'. The 'Schemas (1)' item is selected. The main area displays a SQL query in a text editor:

```
FROM employees e
6 JOIN customers c
7 ON e.employeeNumber = c.salesRepEmployeeNumber
8 JOIN orders o
9 ON c.customerNumber = o.customerNumber
10 JOIN orderdetails od
```

Below the query editor, there are tabs for 'Data Output', 'Messages', and 'Notifications'. The 'Data Output' tab is active, showing a table with the following data:

	employeenumber [PK] integer	sales_rep text	totalRevenue numeric
1	1370	GerardHernandez	1258577.81
2	1165	LeslieJennings	1081530.54
3	1401	PamelaCastillo	868220.55
4	1501	LarryBott	732096.79
5	1504	BarryJones	704853.91
6	1323	GeorgeVanauf	669377.05
7	1612	PeterMarsh	584593.76
8	1337	LouiBondur	569485.75
9	1611	AndyFixter	562582.59
10	1216	StevePatterson	505875.42
11	1286	FoonYueTseng	488212.67
12	1621	MamiNishi	457110.07
13	1702	MartinGerard	387477.47
14	1188	JulieFirrelli	386663.20
15	1166	LeslieThompson	347533.03

## Sales Representative Revenue Ranking

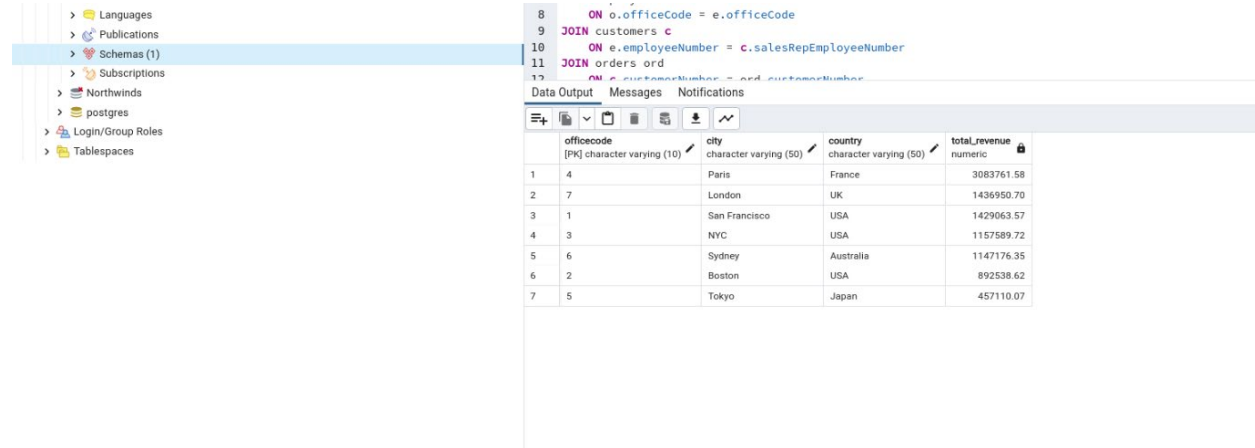
- Gerard Hernandez — \$1,258,577.81
- Leslie Jennings — \$1,081,530.54
- Pamela Castillo — \$868,220.55
- Larry Bott — \$732,096.79
- Barry Jones — \$704,853.91
- George Vanauf — \$669,377.05
- Peter Marsh — \$584,593.76
- Loui Bondur — \$569,485.75
- Andy Fixter — \$562,582.59
- Steve Patterson — \$505,875.42
- Foon Yue Tseng — \$488,212.67
- Mami Nishi — \$457,110.07
- Martin Gerard — \$387,477.47
- Julie Firrelli — \$386,663.20
- Leslie Thompson — \$347,533.03

## Problem 2: Office-Level Sales Performance

### Analysis Challenge

Executive leadership wants to compare regional office performance to guide future expansion and staffing decisions. I need to determine the total revenue generated by each office, based on the customers and employees assigned to that office.

### Code



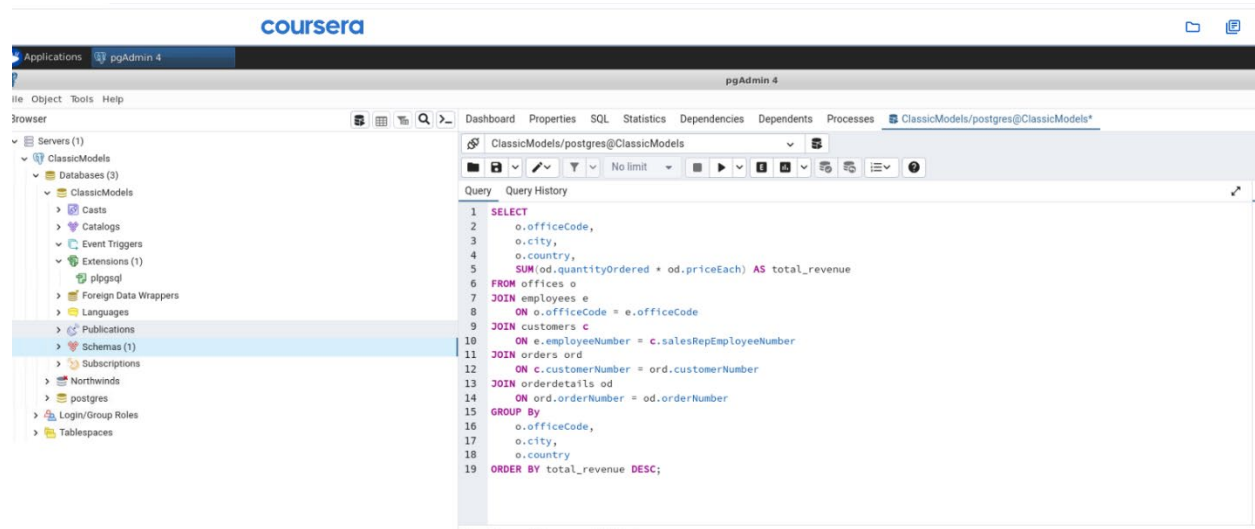
The screenshot shows a database IDE with a left sidebar containing a tree view of database objects: Languages, Publications, Schemas (1), Subscriptions, Northwinds, postgres, Login/Group Roles, and Tablespaces. The main area displays a SQL query with line numbers 8 through 17. The query is a SELECT statement that joins offices, employees, customers, orders, and orderdetails to calculate total revenue by office. Below the query, there are tabs for 'Data Output', 'Messages', and 'Notifications'. The 'Data Output' tab is active, showing a table with 7 rows and 4 columns: officecode, city, country, and total\_revenue. The data is sorted by total\_revenue in descending order.

```
8      ON o.officeCode = e.officeCode
9  JOIN customers c
10     ON e.employeeNumber = c.salesRepEmployeeNumber
11  JOIN orders ord
12     ON c.customerNumber = ord.customerNumber
```

	officecode [PK] character varying (10)	city character varying (50)	country character varying (50)	total_revenue numeric
1	4	Paris	France	3083761.58
2	7	London	UK	1436950.70
3	1	San Francisco	USA	1429063.57
4	3	NYC	USA	1157589.72
5	6	Sydney	Australia	1147176.35
6	2	Boston	USA	892538.62
7	5	Tokyo	Japan	457110.07

```
SELECT
    o.officeCode,
    o.city,
    o.country,
    SUM(od.quantityOrdered * od.priceEach) AS total_revenue
FROM offices o
JOIN employees e
    ON o.officeCode = e.officeCode
JOIN customers c
    ON e.employeeNumber = c.salesRepEmployeeNumber
JOIN orders ord
    ON c.customerNumber = ord.customerNumber
JOIN orderdetails od
    ON ord.orderNumber = od.orderNumber
GROUP BY
    o.officeCode,
    o.city,
    o.country
ORDER BY total_revenue DESC;
```

## Output



### Office Revenue by Location

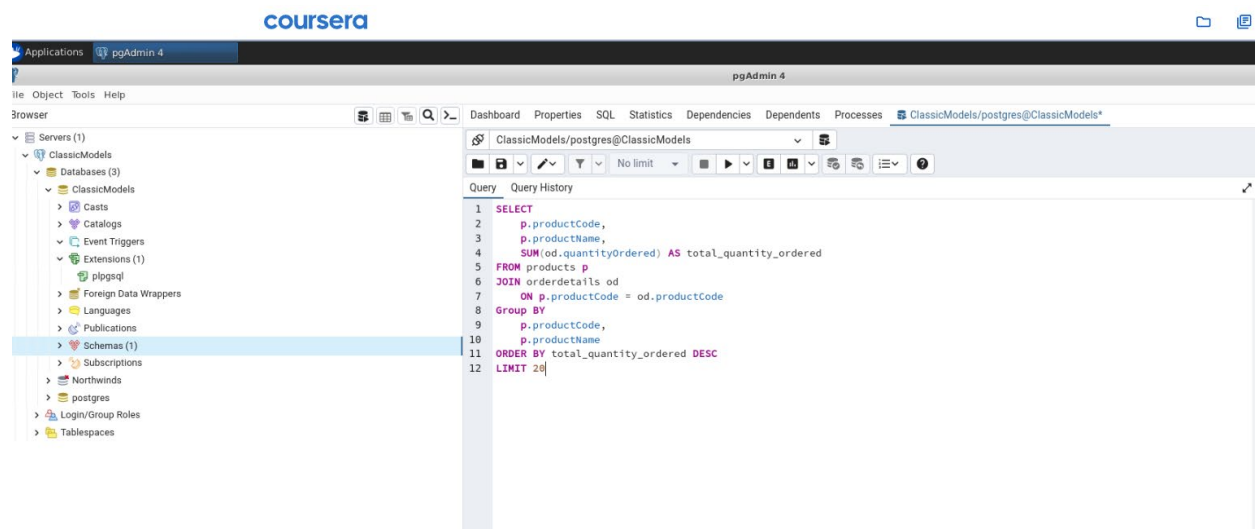
- Paris, France — \$3,083,761.58
- London, UK — \$1,436,950.70
- San Francisco, USA — \$1,429,063.57
- New York City, USA — \$1,157,589.72
- Sydney, Australia — \$1,147,176.35
- Boston, USA — \$892,538.62
- Tokyo, Japan — \$457,110.07

# Problem 3: Product Demand Analysis

## Analysis Challenge

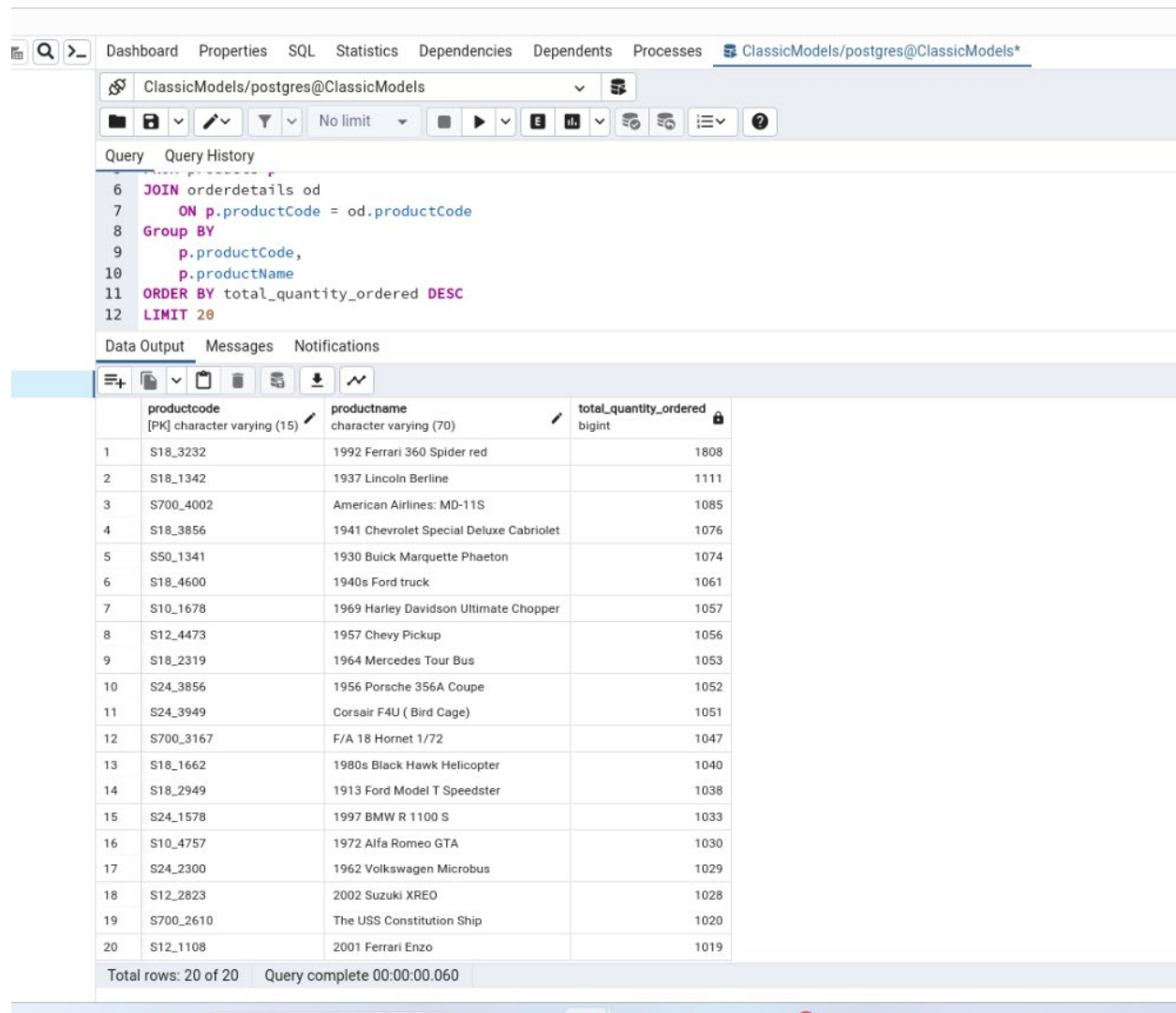
The inventory team needs to understand which products are most frequently ordered in order to optimize stock levels. I need to identify the top 20 products by total quantity ordered, sorted from highest to lowest demand.

## Code



```
SELECT
  p.productCode,
  p.productName,
  SUM(od.quantityOrdered) AS total_quantity_ordered
FROM products p
JOIN orderdetails od
  ON p.productCode = od.productCode
GROUP BY
  p.productCode,
  p.productName
ORDER BY total_quantity_ordered DESC
LIMIT 20;
```

## Output



The screenshot shows a database management interface with a menu bar (Dashboard, Properties, SQL, Statistics, Dependencies, Dependents, Processes) and a toolbar. The active tab is 'ClassicModels/postgres@ClassicModels\*'. Below the toolbar, there's a 'Query' tab with a SQL query: 

```
6 JOIN orderdetails od
7 ON p.productCode = od.productCode
8 Group BY
9 p.productCode,
10 p.productName
11 ORDER BY total_quantity_ordered DESC
12 LIMIT 20
```

 The 'Data Output' tab shows a table with 20 rows. The columns are 'productcode' (PK, character varying (15)), 'productname' (character varying (70)), and 'total\_quantity\_ordered' (bigint). The data is sorted by 'total\_quantity\_ordered' in descending order. The bottom status bar indicates 'Total rows: 20 of 20' and 'Query complete 00:00:00.060'.

	productcode [PK] character varying (15)	productname character varying (70)	total_quantity_ordered bigint
1	S18_3232	1992 Ferrari 360 Spider red	1808
2	S18_1342	1937 Lincoln Berline	1111
3	S700_4002	American Airlines: MD-11S	1085
4	S18_3856	1941 Chevrolet Special Deluxe Cabriolet	1076
5	S50_1341	1930 Buick Marquette Phaeton	1074
6	S18_4600	1940s Ford truck	1061
7	S10_1678	1969 Harley Davidson Ultimate Chopper	1057
8	S12_4473	1957 Chevy Pickup	1056
9	S18_2319	1964 Mercedes Tour Bus	1053
10	S24_3856	1956 Porsche 356A Coupe	1052
11	S24_3949	Corsair F4U ( Bird Cage)	1051
12	S700_3167	F/A 18 Hornet 1/72	1047
13	S18_1662	1980s Black Hawk Helicopter	1040
14	S18_2949	1913 Ford Model T Speedster	1038
15	S24_1578	1997 BMW R 1100 S	1033
16	S10_4757	1972 Alfa Romeo GTA	1030
17	S24_2300	1962 Volkswagen Microbus	1029
18	S12_2823	2002 Suzuki XREO	1028
19	S700_2610	The USS Constitution Ship	1020
20	S12_1108	2001 Ferrari Enzo	1019

Total rows: 20 of 20    Query complete 00:00:00.060

### Top 20 Products by Quantity Ordered

- 1992 Ferrari 360 Spider Red — 1,808
- 1937 Lincoln Berline — 1,111
- American Airlines: MD-11S — 1,085
- 1941 Chevrolet Special Deluxe Cabriolet — 1,076
- 1930 Buick Marquette Phaeton — 1,074
- 1940s Ford Truck — 1,061
- 1969 Harley Davidson Ultimate Chopper — 1,057
- 1957 Chevy Pickup — 1,056
- 1964 Mercedes Tour Bus — 1,053
- 1956 Porsche 356A Coupe — 1,052
- Corsair F4U (Bird Cage) — 1,051

- F/A 18 Hornet 1/72 — 1,047
- 1980s Black Hawk Helicopter — 1,040
- 1913 Ford Model T Speedster — 1,038
- 1997 BMW R 1100 S — 1,033
- 1972 Alfa Romeo GTA — 1,030
- 1962 Volkswagen Microbus — 1,029
- 2002 Suzuki XREO — 1,028
- The USS Constitution Ship — 1,020
- 2001 Ferrari Enzo — 1,019