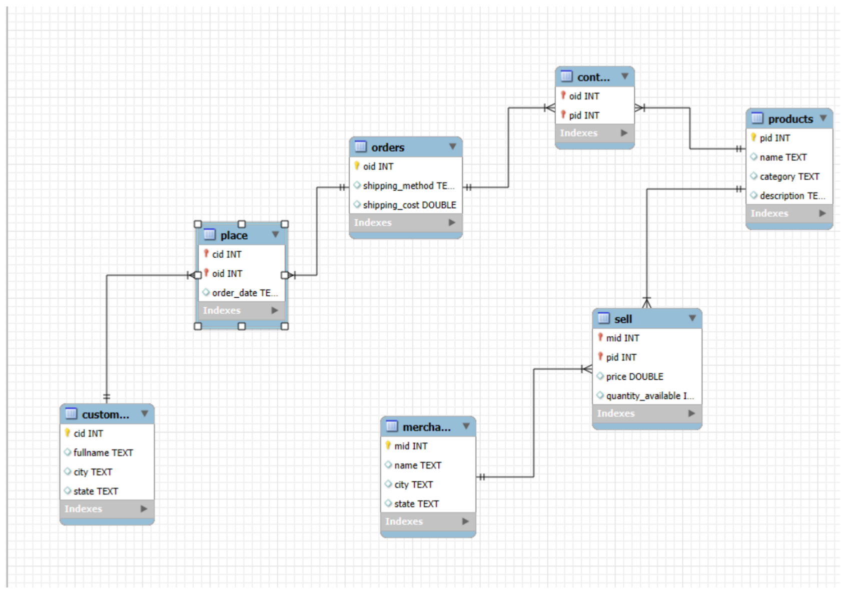
DB Assignment 3

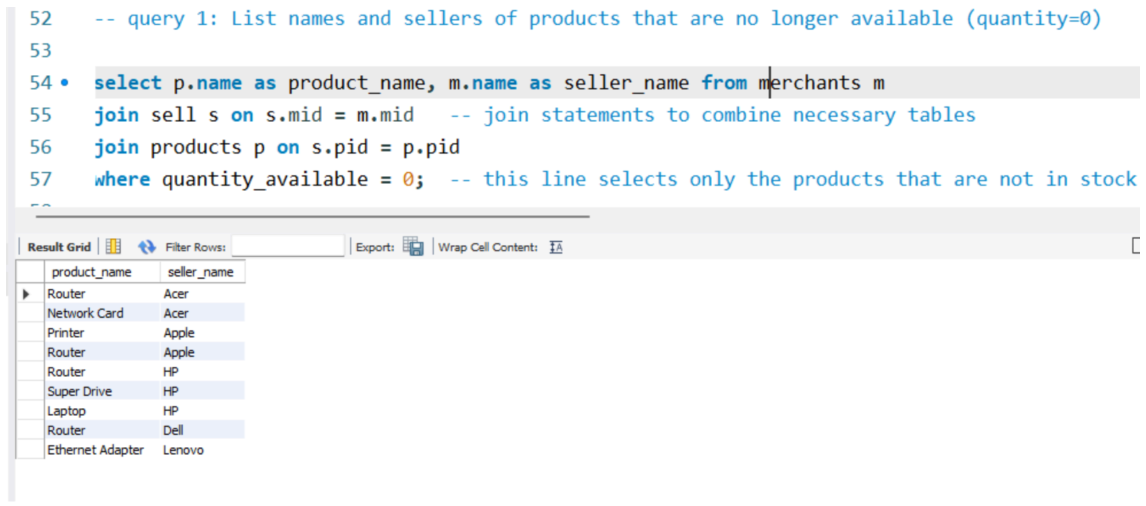
Sean Gor

11 October 2024

ER Diagram:



Query 1:

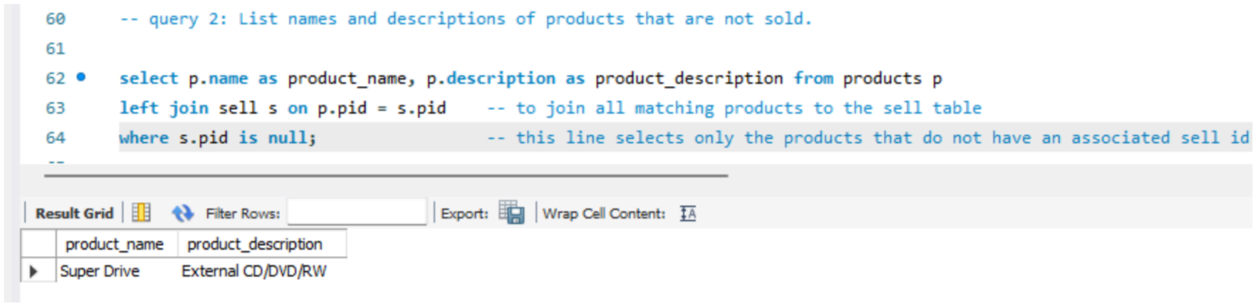


```sql
-- query 1: List names and sellers of products that are no longer available (quantity=0)

select p.name as product_name, m.name as seller_name from merchants m
join sell s on s.mid = m.mid    -- join statements to combine necessary tables
join products p on s.pid = p.pid
where quantity_available = 0;   -- this line selects only the products that are not in stock
```

product_name	seller_name
Router	Acer
Network Card	Acer
Printer	Apple
Router	Apple
Router	HP
Super Drive	HP
Laptop	HP
Router	Dell
Ethernet Adapter	Lenovo

This query selects all products and their sellers for which they are no longer being sold. It uses join statements to gain the necessary information, and a where clause to select only products which satisfy that condition.

Query 2:



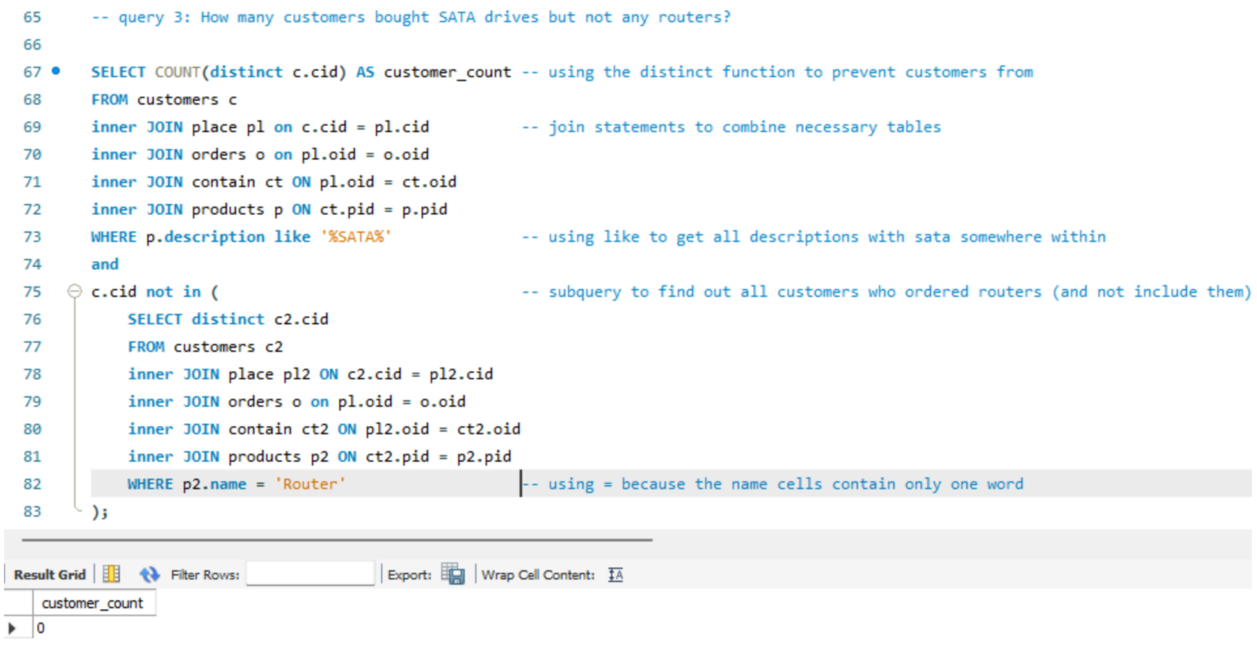
```
-- query 2: List names and descriptions of products that are not sold.

select p.name as product_name, p.description as product_description from products p
left join sell s on p.pid = s.pid    -- to join all matching products to the sell table
where s.pid is null;                 -- this line selects only the products that do not have an associated sell id
```

```
Result Grid | Filter Rows: | Export: | Wrap Cell Content:
```

product_name	product_description
Super Drive	External CD/DVD/RW

This query gets all products and their descriptions so long as they are being sold. It uses a left join statement to get products whose product under the sell table is null. The output displays those products only.

Query 3:



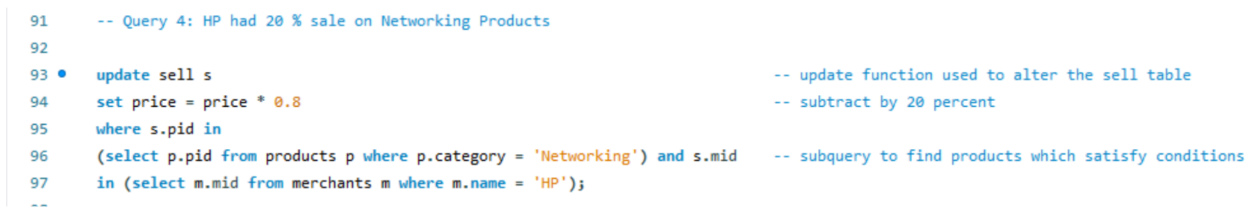
```
-- query 3: How many customers bought SATA drives but not any routers?

SELECT COUNT(distinct c.cid) AS customer_count -- using the distinct function to prevent customers from
FROM customers c
inner JOIN place pl on c.cid = pl.cid          -- join statements to combine necessary tables
inner JOIN orders o on pl.oid = o.oid
inner JOIN contain ct ON pl.oid = ct.oid
inner JOIN products p ON ct.pid = p.pid
WHERE p.description like '%SATA%'              -- using like to get all descriptions with sata somewhere within
and
c.cid not in (                                 -- subquery to find out all customers who ordered routers (and not include them)
    SELECT distinct c2.cid
    FROM customers c2
    inner JOIN place pl2 ON c2.cid = pl2.cid
    inner JOIN orders o on pl.oid = o.oid
    inner JOIN contain ct2 ON pl2.oid = ct2.oid
    inner JOIN products p2 ON ct2.pid = p2.pid
    WHERE p2.name = 'Router'                   -- using = because the name cells contain only one word
);
```

```
Result Grid | Filter Rows: | Export: | Wrap Cell Content:
```

customer_count
0

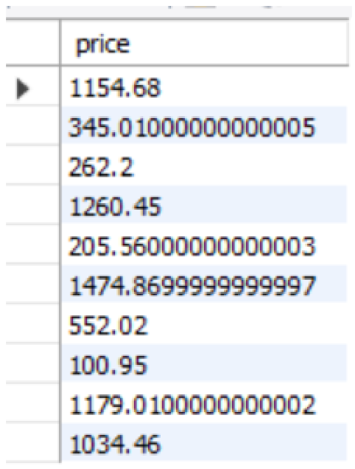
This query first selects all distinct customers who bought a product with SATA somewhere under its description. It then uses a subquery to select every customer who purchased a router. Finally, this query includes all customers who satisfy the first subquery, but not the second. In this case, no customers met these criteria.

Query 4:



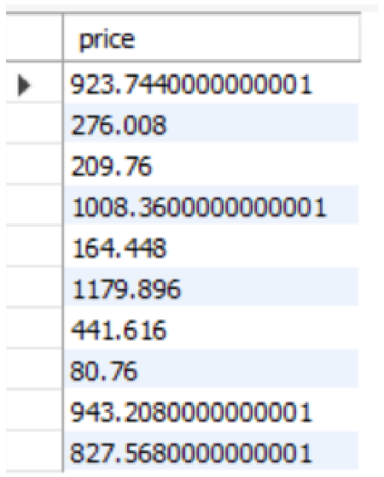
```
-- Query 4: HP had 20 % sale on Networking Products

update sell s                                                                    -- update function used to alter the sell table
set price = price * 0.8                                                          -- subtract by 20 percent
where s.pid in
(select p.pid from products p where p.category = 'Networking') and s.mid       -- subquery to find products which satisfy conditions
in (select m.mid from merchants m where m.name = 'HP');
```

Output Before:



price
1154.68
345.01000000000005
262.2
1260.45
205.56000000000003
1474.8699999999997
552.02
100.95
1179.0100000000002
1034.46

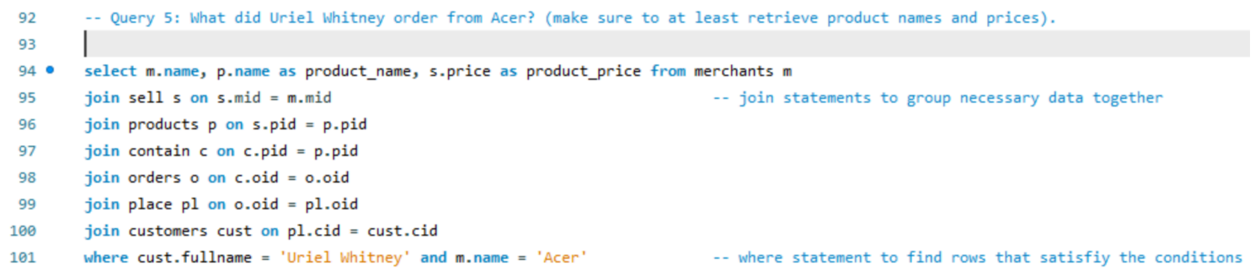
Output After:



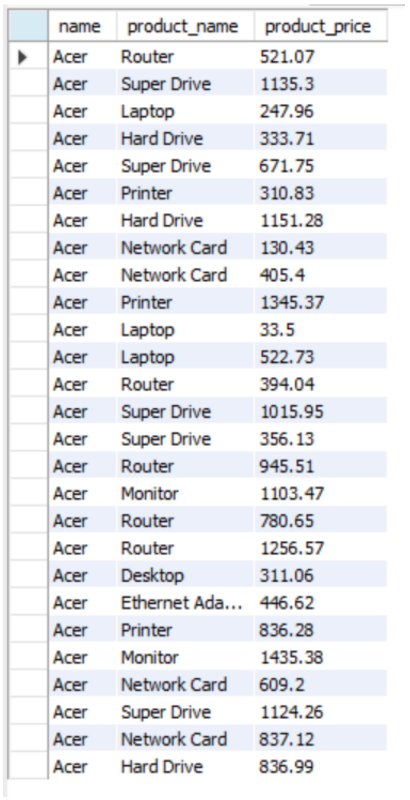
price
923.7440000000001
276.008
209.76
1008.3600000000001
164.448
1179.896
441.616
80.76
943.2080000000001
827.5680000000001

This query updates the sell table using the update function, and multiplies each price by 0.8 to yield the average sales price. It uses a where function to make sure only the products which satisfy the necessary conditions are met. Above are screenshots of the prices before and after the update.

Query 5:



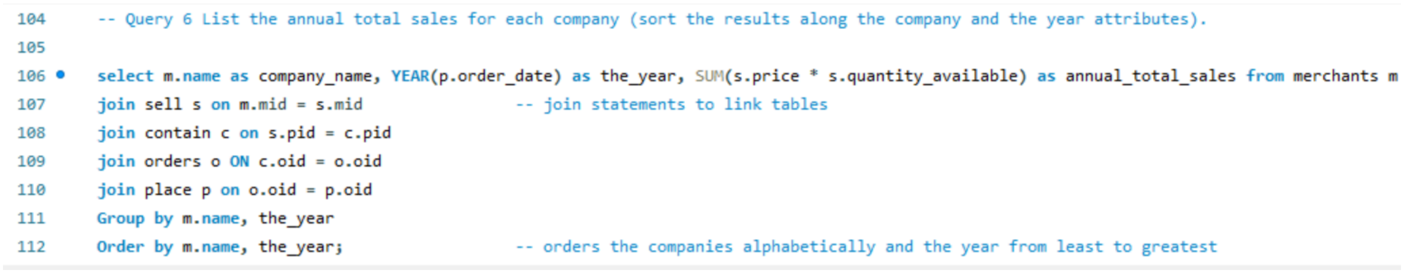
```
-- Query 5: What did Uriel Whitney order from Acer? (make sure to at least retrieve product names and prices).

select m.name, p.name as product_name, s.price as product_price from merchants m
join sell s on s.mid = m.mid                                        -- join statements to group necessary data together
join products p on s.pid = p.pid
join contain c on c.pid = p.pid
join orders o on c.oid = o.oid
join place pl on o.oid = pl.oid
join customers cust on pl.cid = cust.cid
where cust.fullname = 'Uriel Whitney' and m.name = 'Acer'           -- where statement to find rows that satisfiy the conditions
```



name	product_name	product_price
Acer	Router	521.07
Acer	Super Drive	1135.3
Acer	Laptop	247.96
Acer	Hard Drive	333.71
Acer	Super Drive	671.75
Acer	Printer	310.83
Acer	Hard Drive	1151.28
Acer	Network Card	130.43
Acer	Network Card	405.4
Acer	Printer	1345.37
Acer	Laptop	33.5
Acer	Laptop	522.73
Acer	Router	394.04
Acer	Super Drive	1015.95
Acer	Super Drive	356.13
Acer	Router	945.51
Acer	Monitor	1103.47
Acer	Router	780.65
Acer	Router	1256.57
Acer	Desktop	311.06
Acer	Ethernet Ada...	446.62
Acer	Printer	836.28
Acer	Monitor	1435.38
Acer	Network Card	609.2
Acer	Super Drive	1124.26
Acer	Network Card	837.12
Acer	Hard Drive	836.99

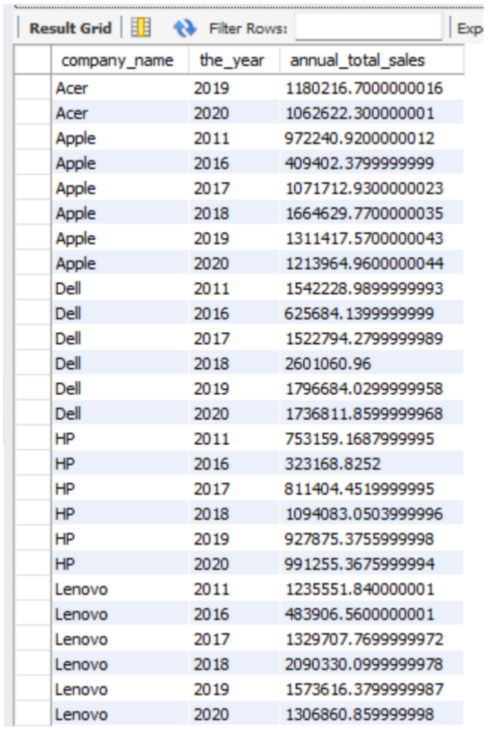
This query lists all product names and prices that the customer Uriel Whitney ordered specifically from Acer. It uses join statements to get all the product names and prices, and a where statement to filter rows that only satisfy the query requirements.

Query 6:



```
-- Query 6 List the annual total sales for each company (sort the results along the company and the year attributes).

select m.name as company_name, YEAR(p.order_date) as the_year, SUM(s.price * s.quantity_available) as annual_total_sales from merchants m
join sell s on m.mid = s.mid                -- join statements to link tables
join contain c on s.pid = c.pid
join orders o ON c.oid = o.oid
join place p on o.oid = p.oid
Group by m.name, the_year
Order by m.name, the_year;                  -- orders the companies alphabetically and the year from least to greatest
```

Output:

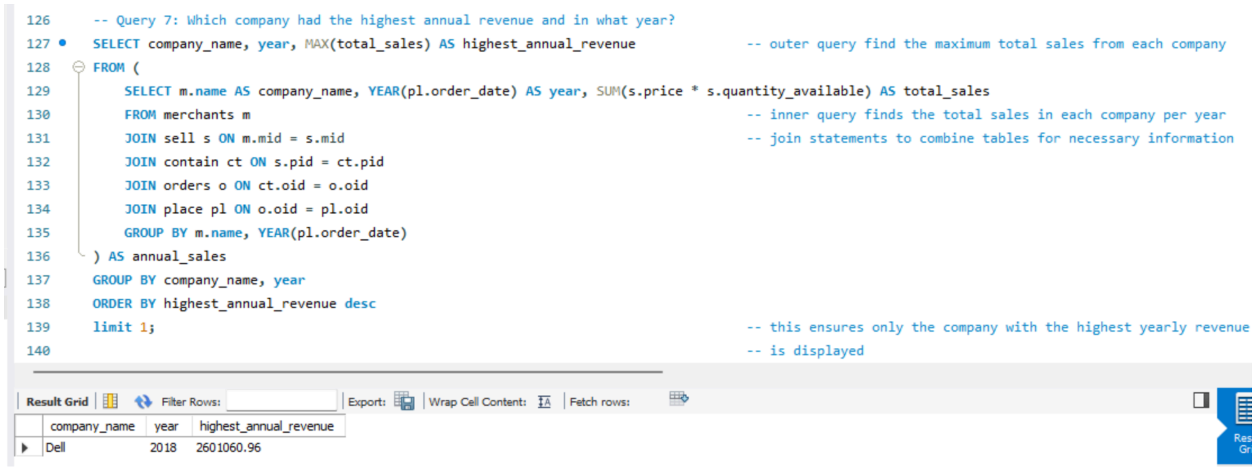


Result Grid | Filter Rows: | Exp

company_name	the_year	annual_total_sales
Acer	2019	1180216.7000000016
Acer	2020	1062622.300000001
Apple	2011	972240.9200000012
Apple	2016	409402.3799999999
Apple	2017	1071712.9300000023
Apple	2018	1664629.7700000035
Apple	2019	1311417.5700000043
Apple	2020	1213964.9600000044
Dell	2011	1542228.9899999993
Dell	2016	625684.1399999999
Dell	2017	1522794.2799999989
Dell	2018	2601060.96
Dell	2019	1796684.0299999958
Dell	2020	1736811.8599999968
HP	2011	753159.1687999995
HP	2016	323168.8252
HP	2017	811404.4519999995
HP	2018	1094083.0503999996
HP	2019	927875.3755999998
HP	2020	991255.3675999994
Lenovo	2011	1235551.840000001
Lenovo	2016	483906.5600000001
Lenovo	2017	1329707.7699999972
Lenovo	2018	2090330.0999999978
Lenovo	2019	1573616.3799999987
Lenovo	2020	1306860.859999998

This query lists the sales of each company per year that they sold items. It uses the year function to retrieve the year of each order date, the sum function to get the total sales, and join statements to access data from other tables. It then uses an order by function to order the results alphabetically by company, and ascending by year.

Query 7:

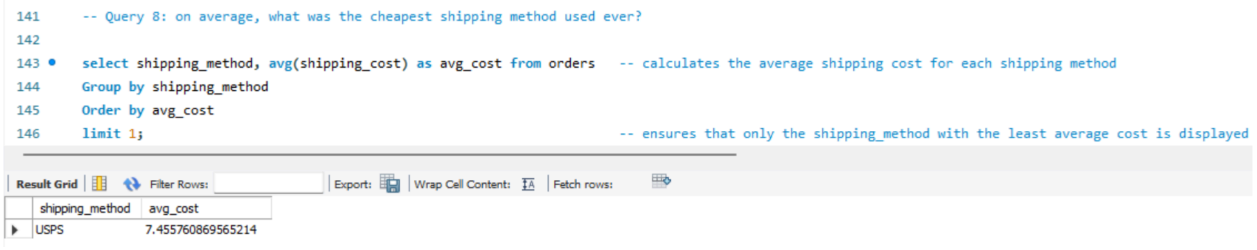


```sql
-- Query 7: Which company had the highest annual revenue and in what year?
SELECT company_name, year, MAX(total_sales) AS highest_annual_revenue          -- outer query find the maximum total sales from each company
FROM (
    SELECT m.name AS company_name, YEAR(pl.order_date) AS year, SUM(s.price * s.quantity_available) AS total_sales
    FROM merchants m                                                            -- inner query finds the total sales in each company per year
    JOIN sell s ON m.mid = s.mid                                                -- join statements to combine tables for necessary information
    JOIN contain ct ON s.pid = ct.pid
    JOIN orders o ON ct.oid = o.oid
    JOIN place pl ON o.oid = pl.oid
    GROUP BY m.name, YEAR(pl.order_date)
) AS annual_sales
GROUP BY company_name, year
ORDER BY highest_annual_revenue desc
limit 1;                                                                        -- this ensures only the company with the highest yearly revenue
                                                                                -- is displayed
```

company_name	year	highest_annual_revenue
Dell	2018	2601060.96

This query finds the company and year with the highest yearly revenue. Once again, it uses the year function to retrieve all the years, a sum function to calculate the total sales, and a subquery to find the total sales for a company before finding the maximum one. It then orders the revenues from highest to lowest, and uses the limit function to print only the highest one.

Query 8:

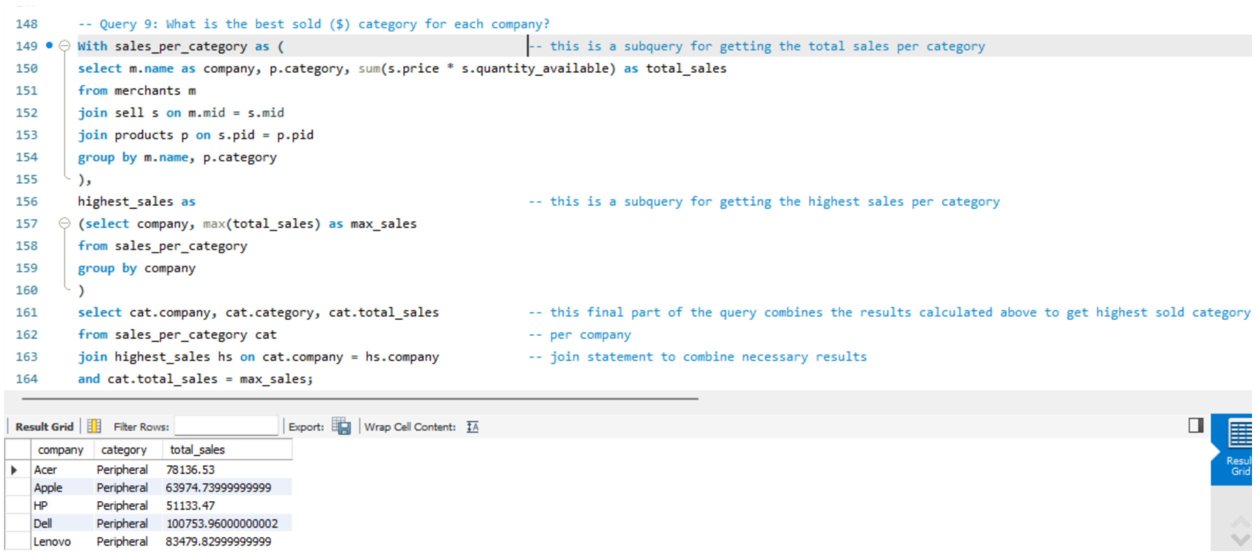


```sql
-- Query 8: on average, what was the cheapest shipping method used ever?

select shipping_method, avg(shipping_cost) as avg_cost from orders   -- calculates the average shipping cost for each shipping method
Group by shipping_method
Order by avg_cost
limit 1;                                                             -- ensures that only the shipping_method with the least average cost is displayed
```

shipping_method	avg_cost
USPS	7.455760869565214

This query finds the average shipping costs for all shipping methods. It uses the avg function to find this average, orders ascendingly by average cost, and limits to only 1 displayed result to yield only the shipping method with the least average cost.

Query 9:



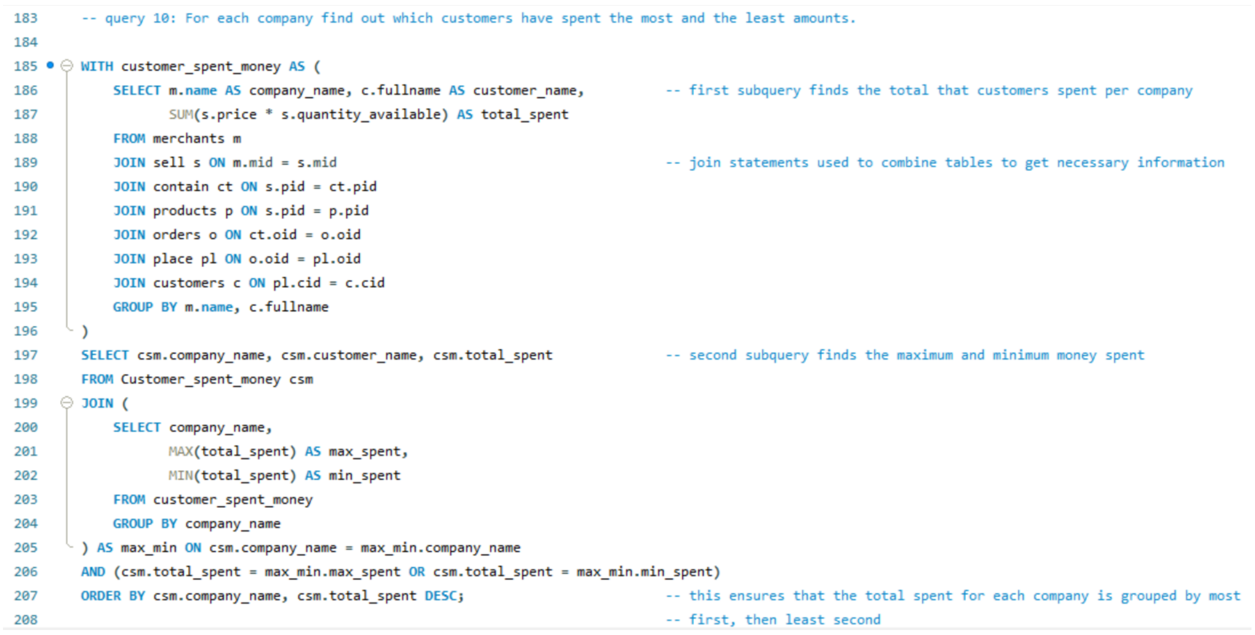
```sql
-- Query 9: What is the best sold ($) category for each company?
With sales_per_category as (                                    -- this is a subquery for getting the total sales per category
select m.name as company, p.category, sum(s.price * s.quantity_available) as total_sales
from merchants m
join sell s on m.mid = s.mid
join products p on s.pid = p.pid
group by m.name, p.category
),
highest_sales as                                                -- this is a subquery for getting the highest sales per category
(select company, max(total_sales) as max_sales
from sales_per_category
group by company
)
select cat.company, cat.category, cat.total_sales               -- this final part of the query combines the results calculated above to get highest sold category
from sales_per_category cat                                     -- per company
join highest_sales hs on cat.company = hs.company               -- join statement to combine necessary results
and cat.total_sales = max_sales;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

company	category	total_sales
Acer	Peripheral	78136.53
Apple	Peripheral	63974.73999999999
HP	Peripheral	51133.47
Dell	Peripheral	100753.96000000002
Lenovo	Peripheral	83479.82999999999

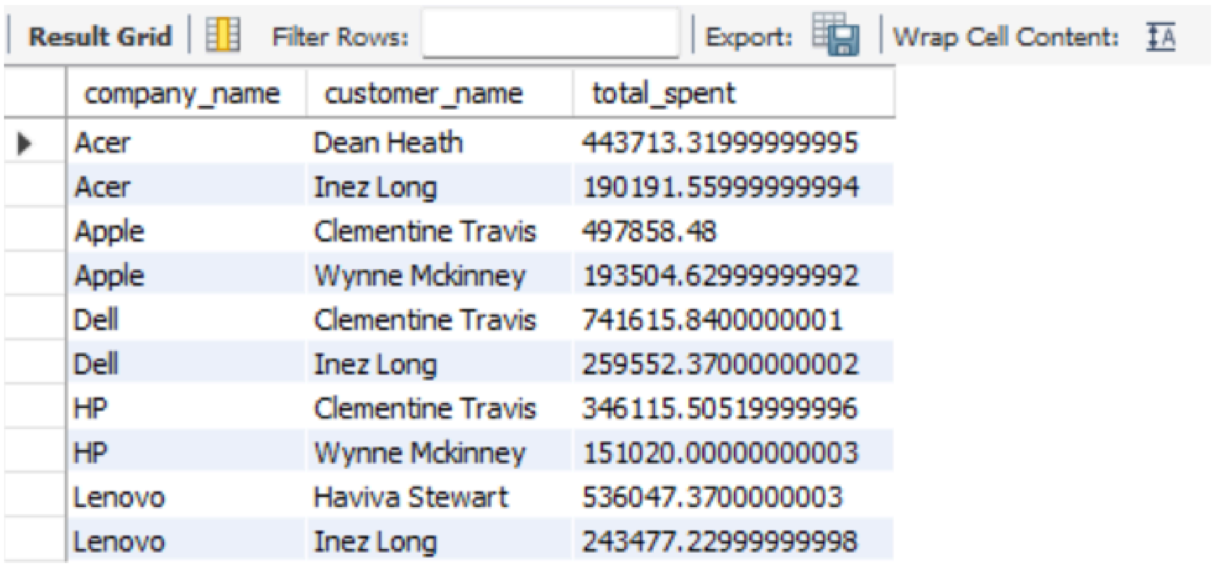
This query uses a with clause to save the results of each category's total sales per category under a temporary entity. The second subquery uses the max() function to find the highest amount of sales of each company based on the first subquery, and saves this result under the category highest_sales. Finally, the last part of this query (after the with clause) joins the two calculated temporary entities to yield the highest total sales per category for each company.

Query 10:



```sql
-- query 10: For each company find out which customers have spent the most and the least amounts.

WITH customer_spent_money AS (
    SELECT m.name AS company_name, c.fullname AS customer_name,          -- first subquery finds the total that customers spent per company
           SUM(s.price * s.quantity_available) AS total_spent
    FROM merchants m
    JOIN sell s ON m.mid = s.mid                                      -- join statements used to combine tables to get necessary information
    JOIN contain ct ON s.pid = ct.pid
    JOIN products p ON s.pid = p.pid
    JOIN orders o ON ct.oid = o.oid
    JOIN place pl ON o.oid = pl.oid
    JOIN customers c ON pl.cid = c.cid
    GROUP BY m.name, c.fullname
)
SELECT csm.company_name, csm.customer_name, csm.total_spent           -- second subquery finds the maximum and minimum money spent
FROM Customer_spent_money csm
JOIN (
    SELECT company_name,
           MAX(total_spent) AS max_spent,
           MIN(total_spent) AS min_spent
    FROM customer_spent_money
    GROUP BY company_name
) AS max_min ON csm.company_name = max_min.company_name
AND (csm.total_spent = max_min.max_spent OR csm.total_spent = max_min.min_spent)
ORDER BY csm.company_name, csm.total_spent DESC;                      -- this ensures that the total spent for each company is grouped by most
                                                                      -- first, then least second
```

Output:



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

company_name	customer_name	total_spent
Acer	Dean Heath	443713.31999999995
Acer	Inez Long	190191.55999999994
Apple	Clementine Travis	497858.48
Apple	Wynne Mckinney	193504.62999999992
Dell	Clementine Travis	741615.8400000001
Dell	Inez Long	259552.37000000002
HP	Clementine Travis	346115.50519999996
HP	Wynne Mckinney	151020.00000000003
Lenovo	Haviva Stewart	536047.3700000003
Lenovo	Inez Long	243477.22999999998

This query uses a with clause to first compute the money that the customers spent for each company. Then, it uses another subquery to find the maximum and minimum amounts of money that each customer spent using the max() and min() aggregate functions. Furthermore, it uses an "and" statement to make sure only customers with the highest and lowest amounts in each company are being displayed as the output. Finally, it orders the customer's money by each company such that the highest goes first, followed by the lowest (and so on for each company).