Title Page

DB Assignment 3

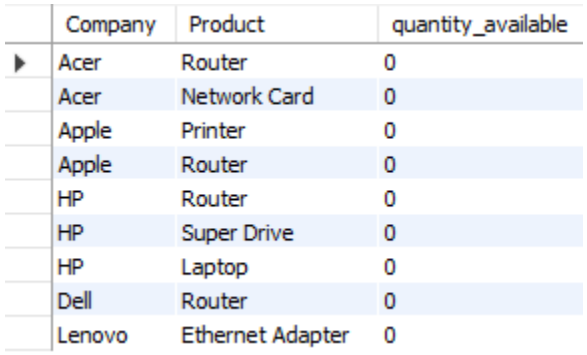
Oscar Nama

10/17/25

SQL Section

Query #1: List names and sellers of products that are no longer available (quantity=0)

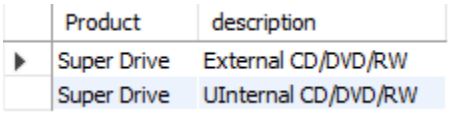
```sql
SELECT merchants.name AS Company, products.name AS Product,
sell.quantity_available
FROM merchants JOIN sell ON merchants.mid = sell.mid
JOIN products ON sell.pid = products.pid
WHERE quantity_available = 0;
```

Result	Explanation
	Here I filter for merchant names, product names, and quantity_available, joining merchants to sell and sell to products, and filtering for rows where quantity_available = 0.

Company	Product	quantity_available
Acer	Router	0
Acer	Network Card	0
Apple	Printer	0
Apple	Router	0
HP	Router	0
HP	Super Drive	0
HP	Laptop	0
Dell	Router	0
Lenovo	Ethernet Adapter	0

Query #2: List names and descriptions of products that are not sold.

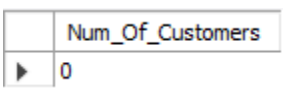
```sql
SELECT p.name AS Product, p.description
FROM products p
WHERE p.pid NOT IN ( -- WHERE clause filters against products in contain
      -- Subquery selects products that are in contain, and therefore in
customer orders
    SELECT DISTINCT c.pid
    FROM contain c
);
```

Result	Explanation
	This query utilizes a subquery that joins products and contain to filter against products that are not part of orders.

Product	description
Super Drive	External CD/DVD/RW
Super Drive	UInternal CD/DVD/RW

Query #3: How many customers bought SATA drives but not any routers?

```sql
SELECT count(customers.cid) AS Num_Of_Customers
FROM customers
JOIN place ON customers.cid = place.cid
JOIN orders ON place.oid = orders.oid
JOIN contain ON orders.oid = contain.oid
JOIN products ON contain.pid = products.pid
WHERE products.description LIKE '%SATA%' -- Checks if product description
includes SATA
       AND customers.cid NOT IN ( -- Filters against subquery results
    SELECT DISTINCT customers.cid -- Queries for customers placing orders on
routers
    FROM customers
    JOIN place ON customers.cid = place.cid
    JOIN orders ON place.oid = orders.oid
    JOIN contain ON orders.oid = contain.oid
    JOIN products ON contain.pid = products.pid
    WHERE products.name LIKE '%Router%'
    );
```

Result	Explanation
	This query is a long one because it requires joining across almost every table, first filtering for products with a description that mention SATA. Then, using a subquery, any rows where the product is a router are filtered against.

Num_Of_Customers
0

Query #4: HP has a 20% sale on all its Networking products.

```sql
SELECT ROUND(s.price * 0.8) AS "Price (20% Off)", s.price AS "Full Price",
m.name AS Company, p.category AS Category
FROM merchants m
JOIN sell s ON m.mid = s.mid
JOIN products p ON s.pid = p.pid
WHERE m.name = "HP" AND Category = "Networking";
```

Result	Explanation

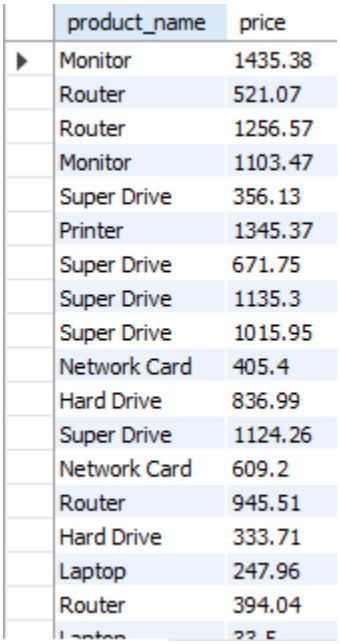
Price (20% Off)	Full Price	Company	Category
828	1034.46	HP	Networking
924	1154.68	HP	Networking
276	345.01	HP	Networking
210	262.2	HP	Networking
1008	1260.45	HP	Networking
164	205.56	HP	Networking
1180	1474.87	HP	Networking
442	552.02	HP	Networking
81	100.95	HP	Networking
943	1179.01	HP	Networking

I interpreted this query to be asking for a comparison of HP's networking product prices, showing the difference between full prices and eighty percent off. I did this by selecting price twice but manipulating the one column, and joining the necessary tables while filtering for HP and Networking.

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

```sql
SELECT DISTINCT p.name AS product_name,
       s.price AS price
FROM customers c
JOIN place pl ON c.cid = pl.cid
JOIN contain co ON pl.oid = co.oid
JOIN products p ON co.pid = p.pid
```

```
JOIN sell s ON p.pid = s.pid
JOIN merchants m ON s.mid = m.mid
WHERE c.fullname = 'Uriel Whitney'
  AND m.name = 'Acer';
```

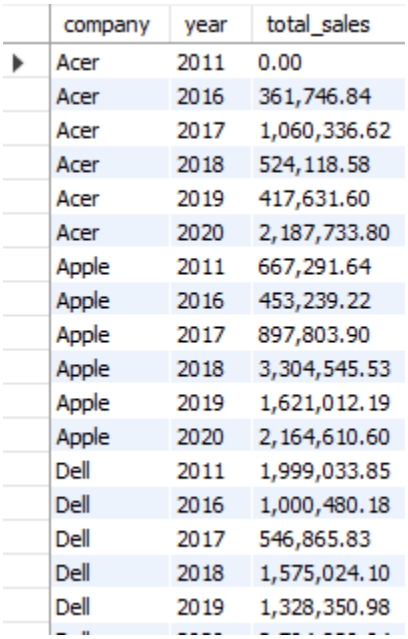
Result	Explanation
	This query joins the necessary tables to find what Uriel Whitney ordered from Acer by filtering for those specific values after the join.

product_name	price
Monitor	1435.38
Router	521.07
Router	1256.57
Monitor	1103.47
Super Drive	356.13
Printer	1345.37
Super Drive	671.75
Super Drive	1135.3
Super Drive	1015.95
Network Card	405.4
Hard Drive	836.99
Super Drive	1124.26
Network Card	609.2
Router	945.51
Hard Drive	333.71
Laptop	247.96
Router	394.04

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

```
SQL
-- Assumption: Total Sales equal to the price of products multiplied by the
quantity available
SELECT m.name AS company,
       YEAR(pl.order_date) AS year, -- Gets only the year from order_date
       FORMAT(SUM(s.price) * s.quantity_available, 2) AS total_sales -- string
formatting for cleaner values
FROM merchants m
JOIN sell s ON m.mid = s.mid
```

```
JOIN products p ON s.pid = p.pid
JOIN contain c ON p.pid = c.pid
JOIN orders o ON c.oid = o.oid
JOIN place pl ON o.oid = pl.oid
GROUP BY m.name, YEAR(pl.order_date)
ORDER BY m.name, YEAR(pl.order_date);
```

Result	Explanation
	In this query, I used the YEAR() function to select for specific years and the FORMAT() function to clean up my total sales values, which I assumed were found by multiplying product price sums by the number of available products. Then, I performed the necessary joins and made sure to group by merchant name and the order year.

company	year	total_sales
Acer	2011	0.00
Acer	2016	361,746.84
Acer	2017	1,060,336.62
Acer	2018	524,118.58
Acer	2019	417,631.60
Acer	2020	2,187,733.80
Apple	2011	667,291.64
Apple	2016	453,239.22
Apple	2017	897,803.90
Apple	2018	3,304,545.53
Apple	2019	1,621,012.19
Apple	2020	2,164,610.60
Dell	2011	1,999,033.85
Dell	2016	1,000,480.18
Dell	2017	546,865.83
Dell	2018	1,575,024.10
Dell	2019	1,328,350.98

Query #7: Which company had the highest annual revenue and in what year?

SQL

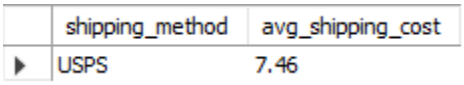
```
-- Assumption: Total Sales/Revenue equal to the price of products multiplied by
the quantity available
SELECT m.name AS company,
             YEAR (pl.order_date) AS year, -- Gets only the year from
order_date
```

```
        ROUND(SUM(s.price), 2) * s.quantity_available AS total_sales -- cleaner
values, but no string formatting to prevent number ordering errors
FROM merchants m
JOIN sell s ON m.mid = s.mid
JOIN products p ON s.pid = p.pid
JOIN contain c ON p.pid = c.pid
JOIN orders o ON c.oid = o.oid
JOIN place pl ON o.oid = pl.oid
GROUP BY m.name, YEAR(pl.order_date)
ORDER BY total_sales desc
LIMIT 1; -- Limited instead of subqueried because it is unlikely that there
would be ties
```

Result	Explanation
	In this query I used similar functions to the last query, except I used ROUND() instead of FORMAT() since I still needed to use the integer value of total_sales. This query I interpreted as being the same as the previous, but I needed to find only the highest revenue. I used ORDER and LIMIT instead of a subquery since I assumed the value would be too specific for any companies to tie.

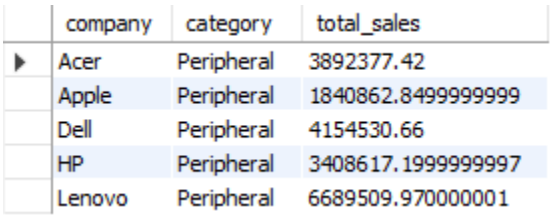
Query #8: On average, what was the cheapest shipping method used ever?

```
SQL
SELECT o.shipping_method, ROUND(AVG(o.shipping_cost), 2) AS avg_shipping_cost
FROM orders o
GROUP BY o.shipping_method
HAVING AVG(o.shipping_cost) = ( -- Filters for average shipping cost equaling
subquery value
    SELECT MIN(avg_cost) -- subquery finds minimum average shipping cost for
each shipping method
    FROM (
        SELECT AVG(o2.shipping_cost) AS avg_cost
        FROM orders o2
        GROUP BY o2.shipping_method
    ) AS sub
);
```

Result	Explanation
	The subquery finds the average cost for each shipping method. The outer query does the same, but uses HAVING and SELECT MIN to keep only the shipping method whose average shipping cost is equal to the minimum from the subquery.

Query #9: What is the best sold ($) category for each company?

```SQL
WITH totals AS ( -- CTE represents each company's categories and their sale values
  SELECT
      m.mid,
      m.name AS company,
      p.category,
      ROUND(SUM(s.price), 2) * 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
  JOIN contain c ON p.pid = c.pid
  JOIN orders o ON c.oid = o.oid
  JOIN place pl ON o.oid = pl.oid
  GROUP BY m.mid, m.name, p.category
)
SELECT t1.company, t1.category, t1.total_sales
FROM totals t1
WHERE t1.total_sales = ( -- Filters for value equaling a subquery value
    SELECT MAX(t2.total_sales) -- Queries for max sales and checks for matching merchant when doing so
    FROM totals t2
    WHERE t2.mid = t1.mid
)
ORDER BY t1.company;
```

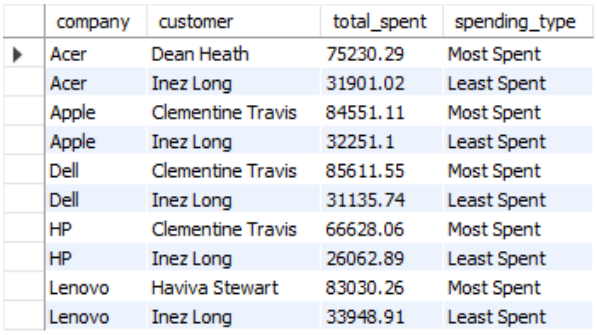
Result	Explanation
	In this query I used a CTE to create a table representing each category at each company and their total sales. Then, in my query itself, I used a subquery that found the max sales for each category. I ensured that the merchant IDs matched across queries.

company	category	total_sales
Acer	Peripheral	3892377.42
Apple	Peripheral	1840862.8499999999
Dell	Peripheral	4154530.66
HP	Peripheral	3408617.1999999997
Lenovo	Peripheral	6689509.970000001

Query #10: For each company, find out which customers have spent the most and the least amounts.

```sql
WITH totals AS ( -- CTE represents each customer and how much they have spent at each merchant
    SELECT
        m.mid,
        m.name AS company,
        c.fullname AS customer,
        ROUND(SUM(s.price), 2) AS total_spent
    FROM merchants m
    JOIN sell s ON m.mid = s.mid
    JOIN products p ON s.pid = p.pid
    JOIN contain co ON p.pid = co.pid
    JOIN orders o ON co.oid = o.oid
    JOIN place pl ON o.oid = pl.oid
    JOIN customers c ON pl.cid = c.cid
    GROUP BY m.mid, m.name, c.fullname
)

SELECT -- 1st select statement, finds customer who has spent the most at each company
    t1.company,
    t1.customer,
    t1.total_spent,
    'Most Spent' AS spending_type
FROM totals t1
WHERE t1.total_spent = ( -- Filters for value equaling subquery value
    SELECT MAX(t2.total_spent) -- Queries for max total spent and matches by merchant id
```

```
    FROM totals t2
    WHERE t2.mid = t1.mid
)

UNION -- UNION merges the two queries for a singular result

SELECT -- 2nd select statement, finds customer who has spent the least at each
company
    t1.company,
    t1.customer,
    t1.total_spent,
    'Least Spent' AS spending_type
FROM totals t1
WHERE t1.total_spent = ( -- Filters for value equaling subquery value
    SELECT MIN(t2.total_spent) -- Queries for min total spent and matches by
merchant id
    FROM totals t2
    WHERE t2.mid = t1.mid
)

ORDER BY company, spending_type DESC;
```

Result	Explanation
	In this query, I used a similar CTE to find the total amount of money each customer has spent at each company. Then, I used UNION to join two similar queries. The first query finds out who has spent the most at each company by using a subquery, matching one row from the outer query to the max spent from the subquery. In a similar fashion, the second query in the UNION uses a subquery that filters for the minimum value. Using UNION, I get the customer who has spent the most at each company and the one who has spent the least.

company	customer	total_spent	spending_type
Acer	Dean Heath	75230.29	Most Spent
Acer	Inez Long	31901.02	Least Spent
Apple	Clementine Travis	84551.11	Most Spent
Apple	Inez Long	32251.1	Least Spent
Dell	Clementine Travis	85611.55	Most Spent
Dell	Inez Long	31135.74	Least Spent
HP	Clementine Travis	66628.06	Most Spent
HP	Inez Long	26062.89	Least Spent
Lenovo	Haviva Stewart	83030.26	Most Spent
Lenovo	Inez Long	33948.91	Least Spent