DSO 552: Practice Problems

Final Exam

1. **Northwind has a policy where after the 1st late order for a customer, it gives a 20% refund for all subsequent late orders per customer. The 20% refund is applied to the total order value (`quantity x unitprice`). Calculate amount of refunds Northwind has paid per productname in total as a result of late orders. Disregard discounts.**

```sql
WITH late_orders AS (
    SELECT od.orderid,
           DENSE_RANK() OVER (
               PARTITION BY customerid ORDER BY orderdate, od.orderid) late_order_number,
           od.*,
           p.productname
    FROM orders o
             JOIN order_details od ON od.orderid = o.orderid
             JOIN products p ON p.productid = od.productid
    WHERE requireddate < shippeddate)
SELECT productname,
       (SUM(unitprice * quantity))::INT       AS total_value_of_late_orders,
       (SUM(unitprice * quantity) * .20)::INT AS total_refunded_value
FROM late_orders
WHERE late_order_number > 1
GROUP BY 1
ORDER BY 2 DESC;
```

Expected Result:

productname	total_value_of_late_orders	total_refunded_value
Mishi Kobe Niku	6402	1280
Schoggi Schokolade	5268	1054
Camembert Pierrot	3359	672
Northwoods Cranberry Sauce	1600	320
Manjimup Dried Apples	1484	297
Rössle Sauerkraut	1368	274
Thüringer Rostbratwurst	1238	248
Ikura	1085	217
Vegie-spread	1054	211
Pavlova	873	175
Perth Pasties	524	105
Chartreuse verte	378	76
Nord-Ost Matjeshering	373	75
Outback Lager	360	72
Singaporean Hokkien Fried Mee	280	56
Rhönbräu Klosterbier	279	56
Jack's New England Clam Chowder	232	46
Gnocchi di nonna Alice	190	38
Inlagd Sill	114	23
Guaraná Fantástica	45	9
Geitost	40	8

2. **Northwind's HR team is performing an analysis of managers at Northwind, to see if there are wide disparities between the responsibilities of different managers. List each manager at Northwind, along with the number of employees they manage, the number of regions and territories they oversee, the number of orders their reports have processed, and the number of customers associated with these orders.**

```sql
SELECT re.firstname || ' ' || re.lastname AS manager_name,
       COUNT(DISTINCT t.regionid)         AS regions,
       COUNT(DISTINCT e.employeeid)       AS employees,
       COUNT(DISTINCT t.territoryid)      AS territories,
       COUNT(DISTINCT o.orderid)          AS orders,
       COUNT(DISTINCT o.customerid)       AS customers
FROM employees e
         JOIN employees re ON re.employeeid = e.reportsto
         JOIN employeeterritories et ON et.employeeid = e.employeeid
         JOIN territories t ON t.territoryid = et.territoryid
         JOIN region r ON r.regionid = t.regionid
         JOIN orders o ON o.employeeid = e.employeeid
GROUP BY 1;
```

Expected Result:

manager_name	regions	employees	territories	orders	customers
Andrew Fuller	3	5	20	552	89
Steven Buchanan	2	3	22	182	74

3. **For orders by German customers, list in chronoogical order their order IDs, order dates, order totals (quantity x unitprice with discount applied), running order total, and average order total.**

```sql
WITH order_totals AS (SELECT o.orderid,
                             o.orderdate,
                             SUM((1 - od.discount) * (od.unitprice * od.quantity)) AS order_total
                      FROM orders o
                               JOIN order_details od ON od.orderid = o.orderid
                               JOIN customers c ON c.customerid = o.customerid
                      WHERE c.country = 'Germany'
                      GROUP BY 1, 2)
SELECT *,
       SUM(order_total) OVER (ORDER BY orderdate, orderid) AS running_total,
       AVG(order_total) OVER (ORDER BY orderdate, orderid) AS average_order_total
FROM order_totals
LIMIT 10
```

Expected Result:

orderid	orderdate	order_total	running_total	average_order_total
10249	1996-07-05	1863.400	1863.40	1863.400
10260	1996-07-19	1504.650	3368.05	1684.025
10267	1996-07-29	3536.600	6904.65	2301.550
10273	1996-08-05	2037.280	8941.93	2235.483
10277	1996-08-09	1200.800	10142.73	2028.546
10279	1996-08-13	351.000	10493.73	1748.955
10284	1996-08-19	1170.375	11664.11	1666.301
10285	1996-08-20	1743.360	13407.47	1675.933
10286	1996-08-21	3016.000	16423.47	1824.829
10301	1996-09-09	755.000	17178.47	1717.847

4. **We need to cut back on unpopular product lines. List all products that have had a total markdown value of over $3,000. The markdown value is the difference between the unitprice of the product and the unit price of the order x quantity. Do not list Meat/Poultry category products.**

```sql
SELECT productid, p.productname, SUM((p.unitprice - od.unitprice) * od.quantity) AS total_markdown_orde
FROM products p
         JOIN order_details od USING (productid)
         JOIN categories c USING (categoryid)
WHERE c.categoryname <> 'Meat/Poultry'
GROUP BY 1, 2
HAVING SUM((p.unitprice - od.unitprice) * od.quantity) > 3000
ORDER BY 3 DESC
```

Expected Result:

productid	productname	total_markdown_order_value
38	Côte de Blaye	14176.299
59	Raclette Courdavault	5984.000
62	Tarte au sucre	3563.999
60	Camembert Pierrot	3332.000

5. **List out each employee, the number of orders they have processed, the percentage of total order volume that employee has contributed to, and also the difference between their order number and the average orders per employee. Categorize employees with under 50**

orders as Associates, 51-100 orders as Senior Associates, and 101+ as Principals. Order by the number of orders processed per employee.

```
WITH order_counts_per_employee AS (
    SELECT e.employeeid, e.firstname || ' ' || e.lastname fullname, COUNT(DISTINCT o.orderid) AS orders
    FROM employees e
             JOIN orders o USING (employeeid)
    GROUP BY 1, 2)
SELECT *,
       ROUND(orders / (SELECT SUM(orders) FROM order_counts_per_employee)::NUMERIC, 2) AS pct_of_order,
       ROUND(orders - (SELECT AVG(orders) FROM order_counts_per_employee)::NUMERIC, 2) AS order_differer
       CASE
           WHEN orders >= 101 THEN 'Principal'
           WHEN orders >= 51 THEN 'Senior Associate'
           ELSE 'Associate' END                                                        AS title
FROM order_counts_per_employee
ORDER BY 3 DESC
```

Expected Result:

employeeid	fullname	orders	pct_of_order	order_differential	title
4	Margaret Peacock	156	0.19	63.78	Principal
3	Janet Leverling	127	0.15	34.78	Principal
1	Nancy Davolio	123	0.15	30.78	Principal
8	Laura Callahan	104	0.13	11.78	Principal
2	Andrew Fuller	96	0.12	3.78	Senior Associate
7	Robert King	72	0.09	-20.22	Senior Associate
6	Michael Suyama	67	0.08	-25.22	Senior Associate
9	Anne Dodsworth	43	0.05	-49.22	Associate
5	Steven Buchanan	42	0.05	-50.22	Associate

6. **Produce the query for a dashboard that will display in the C-Suite monitors, that shows the number of employees, customers, orders, and territories.**

```
SELECT *
FROM (
         SELECT 'Number of Employees' AS category, (SELECT COUNT(*) FROM employees)
         UNION
         SELECT 'Number of Customers' AS category, (SELECT COUNT(*) FROM customers)
         UNION
         SELECT 'Number of Orders' AS category, (SELECT COUNT(*) FROM orders)
         UNION
         SELECT 'Number of Territories' AS category, (SELECT COUNT(*) FROM territories)) t1
ORDER BY 2 DESC
```

Expected Result:

category	count
Number of Orders	830
Number of Customers	91
Number of Territories	53
Number of Employees	9

7. **Challenge: Generate a report of the order IDs, order totals (`quantity` x `unitprice`- disregard `discount`), and also the running last 3 order average order value (the average of the**

order values of the current order and the previous 2 orders)

```
WITH order_totals AS (SELECT o.orderid, o.orderdate, SUM(od.quantity * od.unitprice) AS order_total
                      FROM order_details od
                               JOIN orders o USING (orderid)
                      GROUP BY 1, 2)
SELECT orderdate,
       order_total,
       orderid,
       AVG(order_total)
           -- generates a sliding window of only the current row and the previous 2 rows
       OVER (ORDER BY orderdate, orderid ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS last_3_avg_order_
FROM order_totals
LIMIT 10;
```

Expected Result:

orderdate	order_total	orderid	last_3_avg_order_total
1996-07-04	440.0	10248	440.000
1996-07-05	1863.4	10249	1151.700
1996-07-08	1813.0	10250	1372.133
1996-07-08	670.8	10251	1449.067
1996-07-09	3730.0	10252	2071.267
1996-07-10	1444.8	10253	1948.533
1996-07-11	625.2	10254	1933.333
1996-07-12	2490.5	10255	1520.167
1996-07-15	517.8	10256	1211.167
1996-07-16	1119.9	10257	1376.067

You can also accomplish this without the new syntax `ROWS BETWEEN ... PRECEDING AND CURRENT ROW`:

```
WITH order_totals AS (SELECT o.orderid, o.orderdate, SUM(od.quantity * od.unitprice) AS order_total
                      FROM order_details od
                               JOIN orders o USING (orderid)
                      GROUP BY 1, 2),
     order_total_rows AS (
         SELECT ROW_NUMBER() OVER (ORDER BY orderdate, orderid), * FROM order_totals)

SELECT row_number,
       orderdate,
       order_total,
       orderid,
       (SELECT AVG(order_total)
        FROM order_total_rows
        WHERE row_number BETWEEN otr.row_number - 2 AND otr.row_number) AS last_3_avg_order_total
FROM order_total_rows otr
LIMIT 10;
```

Expected Result:

row_number	orderdate	order_total	orderid	last_3_avg_order_total
1	1996-07-04	440.0	10248	440.000
2	1996-07-05	1863.4	10249	1151.700
3	1996-07-08	1813.0	10250	1372.133
4	1996-07-08	670.8	10251	1449.067
5	1996-07-09	3730.0	10252	2071.267
6	1996-07-10	1444.8	10253	1948.533
7	1996-07-11	625.2	10254	1933.333
8	1996-07-12	2490.5	10255	1520.167
9	1996-07-15	517.8	10256	1211.167
10	1996-07-16	1119.9	10257	1376.067