**SQL\_Advanced Problems**

32. High-value customers

We want to send all of our high-value customers a special VIP gift. We're defining high-value customers as those who've made at least 1 order with a total value (not including the discount) equal to $10,000 or more. We only want to consider orders made in the year 2016.

select

c.customerid,

c.companyname,

o.orderid,

round(sum(od.unitprice \* od.quantity)::numeric, 2) as total\_amount

from

customers c

inner join

orders o on c.customerid = o.customerid

inner join

order\_details od on o.orderid = od.orderid

where

date\_part('year', o.orderdate) = 1998

group by

c.customerid, c.companyname, o.orderid

having

sum(od.unitprice \* od.quantity) >= 10000

Oorder by

total\_amount DESC;

33. High-value customers - total orders

The manager has changed his mind. Instead of requiring that customers have at least one individual orders totaling $10,000 or more, he wants to define high-value customers as those who have orders totaling $15,000 or more in 2016. How would you change the answer to the problem above?

select

c.customer\_id,

c.company\_name,

o.order\_id,

round(sum(od.unit\_price \* od.quantity)::numeric, 2) as total\_amount

from

customers c

inner join

orders o on c.customer\_id = o.customer\_id

inner join

order\_details od on o.orderid = od.orderid

where

date\_part('year', o.orderdate) = 1998

group by

c.customer\_id, c.company\_name, o.orderid

having

sum(od.unit\_price \* od.quantity) >= 15000

Oorder by

total\_amount DESC;

34. High-value customers - with discount

Change the above query to use the discount when calculating high-value customers. Order by the total amount which includes the discount.

SELECT

c.customer\_id,

c.company\_name,

round(sum(

(od.unit\_price \* od.quantity) \* (1 - od.discount)

):: numeric, 2) as total\_amount

FROM

customers c

INNER JOIN

orders o on c.customer\_id = o.customer\_id

INNER JOIN

order\_details od on o.order\_id = od.order\_id

WHERE

date\_part('year', o.order\_date) = 1998

GROUP BY

c.customer\_id, c.company\_name

HAVING

sum(od.unit\_price \* od.quantity \* (1 - od.discount)) >= 10000

ORDER BY

total\_amount DESC;

35. Month-end orders

At the end of the month, salespeople are likely to try much harder to get orders, to meet their month-end quotas. Show all orders made on the last day of the month. Order by Employee ID and Order ID.

SELECT

employee\_id,

order\_id,

order\_date

FROM

orders

WHERE

order\_date = ((date\_trunc('MONTH', order\_date) + INTERVAL '1 MONTH - 1 day') ::order\_date)

ORDER BY

employee\_id, order\_date; SELECT

employee\_id,

order\_id,

order\_date

FROM

orders

WHERE

order\_date = ((date\_trunc('MONTH', order\_date) + INTERVAL '1 MONTH - 1 day') ::order\_date)

ORDER BY

employee\_id, order\_date;

36. Orders with many line items

The North wind mobile app developers are testing an app that customers will use to show orders. In order to make sure that even the largest orders will show up correctly on the app, they'd like some samples of orders that have lots of individual line items. Show the 10 orders with the most line items, in order of total line items.

SELECT

order\_id,

count(order\_id) as total\_items

FROM

order\_details

GROUP BY order\_id

ORDER BY total\_items DESC

LIMIT 10;

37. Orders - random assortment

The North wind mobile app developers would now like to just get a random assortment of orders for beta testing on their app. Show a random set of 2% of all orders.

38. Orders - accidental double-entry

Janet Leverling, one of the salespeople, has come to you with a request.

She thinks that she accidentally double-entered a line item on an order,

with a different ProductID, but the same quantity. She remembers that

the quantity was 60 or more. Show all the OrderIDs with line items that

match this, in order of OrderID.

select

from

oder\_details od

inner join

orders o on od.order\_id = o.order\_id

where od.ouantity >= 60

group by

od.order\_id, od.ouantity

having count(od.ouantity) > 1

order by od.order\_id;

39. Orders - accidental double-entry details

Based on the previous question, we now want to show details of the order, for orders that match the above criteria.

select

od.order\_id

from

oder\_details od

inner join

orders o on od.order\_id = o.order\_id

where od.ouantity >= 60

group by

od.order\_id, od.ouantity

having count(od.ouantity) > 1

order by od.order\_id

SELECT

od.orderid,

od.productid,

od.unitprice,

od.quantity,

od.discount

FROM

order\_details od

WHERE

od.orderid in (SELECT orderid FROM repeated\_quantities);

40. Orders - accidental double-entry details, derived table

Here's another way of getting the same results as in the previous

problem, using a derived table instead of a CTE. However, there's a bug

in this SQL. It returns 20 rows instead of 16. Correct the SQL.

Problem SQL:

Select

OrderDetails.OrderID

,ProductID

,UnitPrice

,Quantity

,Discount

From OrderDetails

Join (

Select

OrderID

From OrderDetails

Where Quantity >= 60

Group By OrderID, Quantity

Having Count(\*) > 1

) PotentialProblemOrders

on PotentialProblemOrders.OrderID = OrderDetails.OrderID

Order by OrderID, ProductID

41. Late orders

Some customers are complaining about their orders arriving late. Which orders are late?

select

order\_id,

order\_date,

required\_date,

shipped\_date

from orders

where

required\_date < = shipped\_date;

42. Late orders - which employees?

Some salespeople have more orders arriving late than others. Maybe

they're not following up on the order process, and need more training.

Which salespeople have the most orders arriving late?

SELECT

o.employee\_id,

e.last\_name,

count(o.employee\_id) as total\_late\_orders

FROM

orders o

INNER JOIN

employees e on o.employee\_id = e.employee\_id

WHERE

o.required\_date <= o.shipped\_date

GROUP BY

o.employee\_id, e.last\_name

ORDER BY

total\_late\_orders DESC;

43. Late orders vs. total orders

Andrew, the VP of sales, has been doing some more thinking some more

about the problem of late orders. He realizes that just looking at the

number of orders arriving late for each salesperson isn't a good idea. It

needs to be compared against the *total* number of orders per

sales person. Return results like the following:

SELECT

o.employee\_id,

count(o.employee\_id) as total

FROM

orders o

WHERE

o.required\_date <= o.shipped\_date

GROUP BY

('o.employee\_id')

, orders\_summary as (

SELECT

o.employee\_id,

count(o.employee\_id) as total

FROM

orders o

GROUP BY

o.employee\_id

)

SELECT

os.employee\_id,

e.last\_name,

os.total,

lo.total as late\_orders

FROM

orders\_summary os

INNER JOIN

employees e on os.employee\_id = e.employee\_id

INNER JOIN

late\_orders lo on os.employee\_id = lo.employee\_id

ORDER BY

os.total DESC;