Stats 167 Final Project

Khang Thai

2025-05-28

Option 1: Practice SQL Interview Questions

Question 1)

```sql
SELECT product_id, sale_data, amount,
  SUM(amount) OVER (PARTITION BY product_id ORDER BY sale_date) AS sales_overtime
FROM sales;
```

Logic

- We want to use all three columns because we are looking for the total amount of sales from each product id over the sale date.
- SUM(amount) is taking the total amount in the amount column
- OVER makes it a window function meaning that we creating a group of rows to perform the SUM.
- PARTITION BY essentially just divides the rows into multiple groups and in this case was want to partition by the product_id and then order the rows in each partition.

Question 2)

```sql
WITH revenue_per_product AS(
  SELECT products.id,
    products.name,
    products.category,
    SUM(order_items.quantity * order_items.price) AS revenue
  FROM order_items
  JOIN products ON order_items.product_id = products.id
  GROUP BY products.id, products.name, products.category
),
ranked_products AS (
  SELECT name,
    category,
    revenue,
  RANK() OVER (PARTITION BY category ORDER BY revenue DESC)  AS revenue_per_product
  FROM revenue_per_product
)
SELECT name, category, revenue
FROM ranked_products
WHERE revenue_per_prouct <= 2
```

Logic

- We join the products table with the order_items table to have access to both tables
- We want to group by products to calculate total revenue

- Using a window function will help rank the products within each category.
- Filter to get only the two 2 products per category

Question 3)

```sql
SELECT orders.customer_id, COUNT(DISTINCT order_items.product_id)
FROM order_items
JOIN orders ON order_items.order_id = orders.id
GROUP BY orders.customer_id
HAVING COUNT(DISTINCT order_items.product_id) = 1
```

Logic

- We join the order_items table to link each customer to the products they ordered.
- We the group by the customer_id to have each custmer's order history.
- COUNT will get the total number of distinct products each customer orered.
- Filtering the total number of distinct products to one will help identify all the unique products.

Question 4)

```sql
WITH previous_race AS (
  SELECT runner_id, race_number, finish_time,
  LAG(finish_time) OVER (PARTITION BY runner_id ORDER BY race_number) AS previous_finish_time
  FROM marathon_times
)
SELECT runner_id
FROM marathon_times
WHERE runner_id NOT IN (
  SELECT runner_id
  FROM previous_race
  WHERE previous_finish_time IS NOT NULL
    AND finish_time >= previous_finish_time
)
```

Logic

- We want to find the runners whose finish times decreases in every race they ran
- Use LAG() as the window function to compare each race's finish time to the previous one for the same funner.
- We want to then identify runners who ever had a non-decreasing time and exclude those from the final result.
- Return the remaining runner_id who only improved or only ran once.

Question 5)

```sql
WITH day_one AS (
  SELECT user_id, MIN(login_date) AS first_login
  FROM user_logins
  GROUP BY user_id
) SELECT COUNT(*) * 1.0 / (SELECT COUNT(*) FROM day_one) AS retention_rate
FROM day_one d
JOIN user_logins u ON d.user_id = u.user_id
  AND u.login_date = d.first_login + 1
```

Logic

- We want to find the proportion of users who logged in exactly one day after the first-ever login.
- Create a CTE to hold the the temporary table of each users earliest login date.
- Join the original login data to the CTE and keep only the rows exactly one day after the user's first login
- Then count the total number of users who logged in on day one and divide the retained user by the total users

Question 6)

```sql
SELECT
  LEAST(s.seat_number, s.seat_right) AS seat1,
  GREATEST(s.seat_number, s.seat_right) AS seat2
FROM seatmap s
JOIN availability a1 ON s.seat_number = a1.seat_number
JOIN availability a2 ON s.seat_right = a2.seat_number
WHERE a1.is_available = 1 AND a2.is_available = 1

UNION

SELECT
  LEAST(s.seat_number, s.seat_left) AS seat1,
  GREATEST(s.seat_number, s.seat_left) AS seat2
FROM seatmap s
JOIN availability a1 ON s.seat_number = a1.seat_number
JOIN availability a2 ON s.seat_left = a2.seat_number
WHERE a1.is_available = 1 AND a2.is_available = 1
```

Logic

- We want to find both the neighbor for the left and right seat.
- Join the seat and the right neighbor to the availability table to find if they are both available.
- Do the same to the left neighbor seats and the use union to merge the pairs.

Questio 7)

```sql
WITH artist_total AS (
  SELECT s.artist_name, a.genre,
    SUM(stream_count) AS total_stream
  FROM spotify_streams s
  JOIN artist_info a ON s.artist_name = a.name
  GROUP BY s.artist_name, a.genre
), max_stream AS (
  SELECT MAX(total_stream) AS top_stream
  FROM artist_total
)
SELECT artist_name, genre
FROM artist_total, max_stream
WHERE total_stream = top_stream
ORDER BY genre
```

Logic

- We want to find the total stream of each artist first, so we create a CTE to put the total stream in there

- Join both tables to get the genres from the artist_info table and the stream count from the spotify_streams table
- We only want the top streams from each artist so we create another CTE to put the max stream for each artist.
- Lastly we want to filter those top-streamed artist and sort it by the genre in alphabetical order.

Question 8)

```sql
SELECT p.user_id,
  COUNT(CASE WHEN i.promotion_type = 'Full Price' THEN 1 END) AS full_price_count,
  COUNT(CASE WHEN i.promotion_type = 'Sale' THEN 1 END) AS sale_count,
  COUNT(CASE WHEN i.promotion_type = 'Clearance' THEN 1 END) AS clearance_count
FROM nordstrom_purchase_items p
JOIN nordstrom_items_labels i ON p.product_id = i.id
GROUP BY user_id;
```

Logic

- We want to find the total sales for each promotion type, so we first have to individually find the total for each type but using the case function.
- Join both tables to get the promotion type in the nordstrom_items_labels and the total sales and user id from the nordstrom_purchase_items table.
- Lastly we want to return the user ID and the count for each item type so we sort it by the user ID.

Question 9)

```sql
WITH review_count AS (
  SELECT business_id, COUNT(DISTINCT(user_id)) AS total_count
  FROM yelp_reviews
  WHERE EXTRACT(YEAR FROM review_date) = 2025
  GROUP BY business_id
), max_count AS (
  SELECT MAX(total_count) AS max_count
  FROM review_count
)
SELECT r.business_id
FROM review_count r
JOIN max_count m ON r.total_count = m.max_count;
```

Logic

- We create a CTE to place the number of distinct users per business_id in 2025
- We then create another CTE to find the maximum number of unique reviews based on those business_id in 2025
- Lastly we filter the businesses with the maximum count.

Question 10)

```sql
  SELECT p.name AS partner_name,
    AVG(o.amount) AS avg_order_amount
  FROM ue_orders o
  JOIN ue_cities c ON o.city_id = c.id
  JOIN ue_partners p ON o.seller_id = p.id
  WHERE c.name = 'Los Angeles'
```

```
    AND LOWER(p.name) LIKE '%fresh%'
    AND o.order_timestamp >= CURRENT_DATE - 90
    AND o.order_timestamp < CURRENT_DATE
  GROUP BY p.name
  ORDER BY avg_order_amount DESC;
```

Logic

- We want to find names that contains the word fresh that are located in Los Angeles.
- We also want to find orders that have been received in the past 90 days.
- Within those orders, we want to find the average amount for those orders and list the partner's names along with their average order amount.
- Lastly we want to sort that list from highest or lower average order amount.

Question 11)

```
WITH month_sales AS (
  SELECT d.business_id,
    TO_CHAR(d.actual_delivery_time, 'YYYY-MM') AS month,
    SUM(o.sales_amount) AS total_sales
  FROM ue_delivery_orders d
  JOIN ue_orders_value o ON d.delivery_id = o.delivery_id
  WHERE d.actual_delivery_time IS NOT NULL
    AND EXTRACT(YEAR FROM actual_delivery_time) = 2024
  GROUP BY d.business_id, TO_CHAR(d.actual_delivery_time, 'YYYY-MM')
), total AS (
  SELECT month, COUNT(*) AS total_business,
    COUNT(CASE WHEN total_sales >= 150 THEN 1 END) AS sum_150
  FROM month_sales
  GROUP BY month
)
SELECT month, ROUND(100.0 * sum_150 / total_business, 2) AS percent_sum_150
FROM total
ORDER BY month;
```

Logic

- First we want to create a CTE to show the monthly sales and join the two tables
- We want to filter out orders that were canceled and for the year 2024 only.
- Create another CTE for the total for each business per month find the total of how many have at least $150 in sales.
- Lastly, find the percentage of businesses that reached 150 and order it by the month.

Question 12)

```
SELECT id
FROM delta_customers
WHERE actual_arrival_time IS NOT NULL
  AND EXTRACT(YEAR FROM flight_date) IN (2023, 2024)
GROUP BY id
HAVING COUNT(*) >= 6;
```

Logic

- We want to find flights that are not canceled and are in the years 2023 and 2024.

- Sort by the customer's unique ID.
- Keep only customers who had at least 7 flights.

Question 13)

```sql
WITH customer_cart AS (
  SELECT t.customer_id, s.store_brand,
    COUNT(DISTINCT t.transaction_id) AS transaction_count,
    SUM(t.sales) AS total_sales,
    SUM(t.sales) / COUNT(DISTINCT t.transaction_id) AS avg_cart
  FROM amzn_transactions t
  JOIN amzn_stores s ON t.store_id = s.store_id
  WHERE EXTRACT(YEAR FROM t.transaction_date) = 2025
  GROUP BY t.customer_id, s.store_brand
), customer_status AS (
  SELECT *,
    CASE
      WHEN avg_cart > 50 THEN 'High'
      WHEN avg_cart >= 25 THEN 'Medium'
      ELSE 'Low'
    END AS status
  FROM customer_cart
)
SELECT store_brand, status,
  COUNT(DISTINCT customer_id) AS num_customers,
  SUM(transaction_count) AS total_transactions,
  SUM(total_sales) AS total_sales,
  ROUND(SUM(total_sales) / SUM(transaction_count), 2) AS avg_cart
  FROM customer_status
  GROUP BY store_brand, status
  ORDER BY store_brand, status;
```

Logic

- Create a CTE that filters the amazon transaction for 2025 and group is by customer and store brand
- Get the total number of transactions and total sales and divide them from each other to get the average cart size.
- Create another CTE to assign the different status for each customer based on their average cart size
- Lastly we want to count the unique customers and find the sum of transactions and sales and calculate the total cart size for each group.