Practice with Advanced-Intermediate SQL

Chapter 12

Michael Tsiang

Stats 167: Introduction to Databases

UCLA

Do not post, share, or distribute anywhere or with anyone without explicit permission.

Mid-Quarter Feedback Survey Results

Advanced-Intermediate SQL Exercises

Additional Topics

Mid-Quarter Feedback Survey Results

The Good

Things that are going well:

- Comprehensive lecture slides
- Interactive problem sessions during class
- Homework is good practice and connects well to the lecture
- Going beyond basic SQL (beyond Stats 147)

Getting Better

Things I intend, have changed, or will change (or consider) this quarter:

- Connecting to (real) databases rather than using files
- Improved clarity in some homework questions
- Inclusion of interview-type questions
- More time to think through problems in practice sessions
- DataCamp subscription
- Resources to learn more about advanced database topics
- SQL cheat sheet

For the Future

Things that I intend to change (or consider) for next time:

- Midterm/Exams
- Discussion section
- More practice problems (for no or minimal credit)
- Larger homework assignments
- Time spent on SQL in R and Python (three days total this time, can try to fit into two)
- Group projects?

Maybe Someday

Things I will likely keep the same despite a request (but open for future discussion):

- Not showcasing different IDEs (VS Code, Spyder)

I want to keep the focus of the class on supporting your learning of the main toolkit (including the standard R/Python interfaces) and the logic of solving problems in SQL, and showcasing the myriad of IDEs detracts from the intention of the class.

Advanced-Intermediate SQL Exercises

Ace the Data Science Interview

With window functions, CTEs, and views, we have now covered the core tools needed to tackle nearly any SQL question you might encounter in an entry-level data science interview.

At this stage, the essential skill is not just knowing individual commands but learning how to think in SQL: breaking down problems, structuring queries, and combining concepts together logically.

To develop this skill, we will practice with real interview questions from several major companies.

The first six of the following exercises are taken from *Ace the Data Science Interview* by Kevin Huo and Nick Singh, 2021.

Nick Singh now also runs DataLemur, an interactive platform with SQL tutorials and sample data science interview questions.

The rest are from Stratascratch, another data science platform with (over 1000!) real interview questions.

Exercise 1: Big Spenders

Assume you are given the table below on user transactions.

`user_transactions`

column name	type
transaction_id	integer
product_id	integer
user_id	integer
spend	float
transaction_date	datetime

Write a query to obtain the list of customers whose first transaction was valued at $50 or more.

Exercise 1: Big Spenders (Solution)

```sql
WITH purchase_num AS (
    SELECT user_id,
           spend,
           ROW_NUMBER() OVER (
             PARTITION BY user_id
             ORDER BY transaction_date
           ) AS rownum
    FROM user_transactions
)
SELECT user_id
FROM purchase_num
WHERE rownum = 1 AND spend >= 50
```

Exercise 2: 7-Day Rolling Average

Assume you are given the table below containing information on each user's tweets over a period of time.

`tweets`

column name	type
tweet_id	integer
msg	string
user_id	integer
tweet_date	datetime

Calculate the 7-day rolling average of tweets by each user for every date.

Exercise 2: 7-Day Rolling Average (Solution)

```sql
WITH daily_counts AS (
    SELECT user_id,
           CAST(tweet_date AS DATE) AS tweet_date,
           COUNT(*) AS num_tweets
    FROM tweets
    GROUP BY user_id, CAST(tweet_date AS DATE)
)
SELECT user_id,
       tweet_date,
       AVG(num_tweets) OVER (
         PARTITION BY user_id
         ORDER BY tweet_date
         ROWS BETWEEN 6 PRECEDING AND CURRENT ROW
       ) AS rolling_avg_7d
FROM tweet_counts
```

Note: The `CAST()` function casts values to a different datatype. The `CAST(tweet_date AS DATE)` command casts the datetime values (`'YYYY-MM-DD hh:mm:ss'`) into dates (`'YYYY-MM-DD'`). It might not be needed here, but it can ensure the tweets are grouped properly by date.

Exercise 3: Highest-Grossing Items

Assume you are given the table below containing information on customer spending on products belonging to various categories.

`product_spend`

column name	type
transaction_id	integer
category_id	integer
product_id	integer
user_id	integer
spend	float
transaction_date	datetime

Write a query to identify the top three highest-grossing items within each category in 2020.

Exercise 3: Highest-Grossing Items (Solution)

```sql
WITH product_category_spend AS (
    SELECT category_id,
           product_id,
           SUM(spend) AS total_product_spend
    FROM product_spend
    WHERE transaction_date BETWEEN '2020-01-01' AND '2020-12-31'
    GROUP BY category_id, product_id
),
top_spend AS (
    SELECT *,
           RANK() OVER (
             PARTITION BY category_id
             ORDER BY total_product_spend DESC
           ) AS rnk
    FROM product_category_spend
)
SELECT *
FROM top_spend
WHERE rnk <= 3
ORDER BY category_id, rnk
```

Exercise 4: Top Rated

Assume you are given the table below containing information on user reviews. Define a top-rated business as one whose reviews contain only 4 or 5 stars.

`reviews`

column name	type
business_id	integer
user_id	integer
review_text	string
review_stars	integer
review_date	datetime

Write a query to obtain the number and percentage of businesses that are top rated.

Exercise 4: Top Rated (Solution 1)

```sql
WITH min_review AS (
    SELECT business_id, MIN(review_stars) AS min_stars
    FROM reviews
    GROUP BY business_id
)
SELECT COUNT(*) AS num_top_rated,
       100.0 * (SUM(IIF(min_stars >= 4, 1, 0)) / COUNT(*))
         AS pct_top_rated
FROM min_review;
```

Note1 : Can add `ROUND(number, decimal_places)` around the calculation of `pct_top_rated` to make the output more readable.

Note 2: It is common to include the decimal in `100.0` to ensure a float is returned. Some SQL dialects will use *integer division* if dividing an integer by an integer. Another option is to use `CAST()` to cast values into a float, e.g., `CAST(min_stars AS FLOAT)`.

Exercise 4: Top Rated (Solution 2)

Using `CASE` instead of `IIF()`:

```sql
WITH min_review AS (
    SELECT business_id, MIN(review_stars) AS min_stars
    FROM reviews
    GROUP BY business_id
)
SELECT
    COUNT(*) AS num_top_rated,
    100.0 * AVG(CASE WHEN min_review >= 4 THEN 1 ELSE 0 END)
        AS pct_top_rated
FROM min_review;
```

Exercise 4: Top Rated (Solution 3)

Alternate solution with chained CTEs:

```
WITH low_reviews AS (
    SELECT business_id,
           COUNT(IIF(review_stars < 4, 1, 0)) AS num_low
    FROM reviews
    GROUP BY business_id
),
top_rated AS (
    SELECT business_id
    FROM low_reviews
    WHERE num_low = 0
)
SELECT COUNT(*) AS num_top_rated
       100.0 * COUNT(*) / (
         SELECT COUNT(DISTINCT business_id)
         FROM reviews)
       ) AS pct_top_rated
FROM top_rated;
```

Exercise 5: Total Session Duration

Assume you are given the table below containing information on user session activity for a certain social media app.

`sessions`

column name	type
session_id	integer
user_id	integer
session_type	string (“like”, “reply”, “retweet”)
duration	integer (in minutes)
start_time	datetime

Write a query that ranks users according to their total session durations for each session type between the start date (2021-01-01) and the end date (2021-02-01).

Exercise 5: Total Session Duration (Solution)

```sql
WITH user_duration AS (
   SELECT user_id,
          session_type,
          SUM(duration) AS tot_duration
   FROM sessions
   WHERE start_time BETWEEN '2021-01-01' AND '2021-02-01'
   GROUP BY user_id, session_type
 )
 SELECT user_id,
        session_type,
        RANK() OVER (
            PARTITION BY session_type
            ORDER BY tot_duration DESC
        ) AS rnk
FROM user_duration
ORDER BY session_type, rnk;
```

Exercise 6: Correlated Products

Assume you are given the following tables on customer transactions and products.

`transactions`

column name	type
transaction_id	integer
product_id	integer
user_id	integer
quantity	integer
transaction_time	datetime

`products`

column name	type
product_id	integer
product_name	string
price	float

Find the top 10 products that are most frequently bought together (i.e., purchased in the same transaction).

Exercise 6: Correlated Products (Solution)

```sql
WITH purchase_info AS (
    SELECT t.user_id,
           t.transaction_id,
           p.product_id,
           p.product_name
    FROM transactions AS t
    INNER JOIN products AS p
    ON t.product_id = p.product_id
)
SELECT p1.product_name AS product1,
       p2.product_name AS product2,
       COUNT(*) AS count
FROM purchase_info AS p1
INNER JOIN purchase_info AS p2
ON p1.transaction_id = p2.transaction_id
    AND p1.product_id < p2.product_id
GROUP BY product1, product2
ORDER BY count DESC
LIMIT 10;
```

Exercise 7: Second Highest Salary

Identify the second-highest salary in each department.

Your output should include the department, the second highest salary, and the employee ID. Do not remove duplicate salaries when ordering salaries.

For example, if multiple employees share the same highest salary, the second-highest salary will be the next salary that is lower than the highest salaries.

`employee_data`

column name	type
employee_id	integer
salary	integer
department	string
hire_date	date

Exercise 7: Second Highest Salary (Solution)

```sql
WITH salary_ranks AS (
    SELECT department,
           employee_id,
           salary,
           DENSE_RANK() OVER (
             PARTITION BY department
             ORDER BY salary DESC
           ) AS rnk
    FROM employee_data
)
SELECT department,
       employee_id,
       salary AS second_highest_salary
FROM salary_ranks
WHERE rnk = 2;
```

Exercise 8: Lowest Revenue Restaurants

Write a query that returns a list of the bottom 2% revenue generating restaurants. Return a list of restaurant IDs and their total revenue from when customers placed orders in May 2020.

You can calculate the total revenue by summing the order_total column. And you should calculate the bottom 2% by partitioning the total revenue into evenly distributed buckets.

`doordash_delivery`

column name	type
customer_placed_order_datetime	datetime
placed_order_with_restaurant_datetime	datetime
driver_at_restaurant_datetime	datetime
delivered_to_consumer_datetime	datetime
driver_id	integer
restaurant_id	integer
consumer_id	integer
delivery_region	string
order_total	float

Exercise 8: Lowest Revenue Restaurants (Solution)

```sql
WITH revenue_rnks AS (
    SELECT restaurant_id,
           SUM(order_total) AS total_revenue,
           PERCENT_RANK() OVER (
             ORDER BY SUM(order_total)
           ) AS perc_rank
    FROM doordash_delivery
    WHERE customer_placed_order_datetime
       BETWEEN '2020-05-01' AND '2020-05-31'
    GROUP BY restaurant_id
)
SELECT restaurant_id, total_revenue
FROM revenue_rnks
WHERE perc_rank < 0.02
```

Exercise 9: Consecutive Days

Find all the users who were active for 3 consecutive days or more.

`sf_events`

column name	type
record_date	date
account_id	string
user_id	string

Hint: The `DATEDIFF()` function can find the difference (i.e., number of days) between two dates.

Exercise 9: Consecutive Days (Solution 1)

```sql
WITH consecutive_days AS (
    SELECT user_id,
           record_date,
           LAG(record_date, 1) OVER (
             PARTITION BY user_id
             ORDER BY record_date
           ) AS prev_day
           LEAD(record_date, 1) OVER (
             PARTITION BY user_id
             ORDER BY record_date
           ) AS next_day
    FROM sf_events
)
SELECT DISTINCT user_id
FROM consecutive_days
WHERE DATEDIFF(record_date, prev_day) = 1
    AND DATEDIFF(next_day, record_date) = 1;
```

Exercise 9: Consecutive Days (Solution 2)

Using the `WINDOW` keyword:

```
WITH consecutive_days AS (
    SELECT user_id,
           record_date,
           LAG(record_date, 1) OVER (win) AS prev_day
           LEAD(record_date, 1) OVER (win) AS next_day
    FROM sf_events
)
SELECT DISTINCT user_id
FROM consecutive_days
WHERE DATEDIFF(record_date, prev_day) = 1
    AND DATEDIFF(next_day, record_date) = 1
WINDOW win AS (
    PARTITION BY user_id
    ORDER BY record_date
);
```

Exercise 10: Viewers Turned Streamers

Return the number of streamer sessions for each user whose very first session was as a viewer.

Include only those users whose earliest session (by `session_start`) was of type "viewer". Return the user ID and the number of streamer sessions they had, ordered by number of sessions descending, then user ID ascending.

`twitch_sessions`

column name	type
user_id	integer
session_start	datetime
session_end	datetime
session_id	integer
session_type	string ("streamer" or "viewer")

Exercise 10: Viewers Turned Streamers (Solution)

```sql
WITH rnks AS (
   SELECT user_id,
          session_type,
          ROW_NUMBER() OVER (
            PARTITION BY user_id
            ORDER BY session_start
          ) AS rnk
   FROM twitch_sessions
),
first_viewers AS (
   SELECT user_id,
   FROM rnks
   WHERE rnk = 1 AND session_type = 'viewer'
)
SELECT user_id, COUNT(*) AS streamer_cnt
FROM twitch_sessions
INNER JOIN first_viewers
ON twitch_sessions.user_id = first_viewers.user_id
WHERE session_type = 'streamer'
GROUP BY user_id
ORDER BY total_cnt DESC, user_id
```

Further Practice

For more interview-type practice questions:

- https://datalemur.com/sql-interview-questions
- https://www.stratascratch.com/
- https://leetcode.com/studyplan/top-sql-50/
- https://sqlguroo.com/

Additional Topics

Query Optimization

Indexes:

- https://www.geeksforgeeks.org/sql-indexes/
- https://www.geeksforgeeks.org/difference-between-clustered-and-non-clustered-index/
- https://www.sqlitetutorial.net/sqlite-index/
- https://www.tutorialspoint.com/sql/sql-indexes.htm

Query Optimization:

- https://www.datacamp.com/blog/sql-query-optimization
- https://www.geeksforgeeks.org/best-practices-for-sql-query-optimizations/

Strings and Dates

Functions for working with strings:

- https://datalemur.com/sql-tutorial/sql-string-text

Functions for working with dates and time:

- https://www.sql-easy.com/learn/sqlite-date-time/
- https://www.geeksforgeeks.org/sql-date-functions/
- https://www.dbvis.com/thetable/a-guide-to-the-sql-date-data-types/