56 SQL Problems (Easy to Hard Level)

Pranoy Chakraborty | pcslg1998@gmail.com |LinkedIn | GitHub

Problem 1 (Meta | Hard Level)

A table named “famous” has two columns called user id and follower id. It represents each user ID has a particular follower ID. These follower IDs are also users of #Facebook / Meta. Then, find the famous percentage of each user.

Famous Percentage = number of followers a user has / total number of users on the platform.

Explanation:

1. distinct_users CTE: Combines user_id and follower_id using UNION to get all unique users on the platform. This helps us determine the total number of users.
2. follower_count CTE: Counts the number of followers for each user_id by grouping the rows in the famous table. This gives a list of users with their follower counts.
3. Final SELECT Statement: Uses the data from follower_count and distinct_users to calculate the famous percentage for each user.

MySQL Solution:

```sql
WITH distinct_users AS(
    SELECT user_id AS users FROM famous
    UNION
    SELECT follower_id AS users FROM famous
),
follower_count AS(
    SELECT
        user_id, COUNT(follower_id) AS followers
    FROM
        famous
    GROUP BY user_id
)
SELECT
    f.user_id,
    ROUND((f.followers * 100.0) / (SELECT
                    COUNT(*)
                FROM
                    distinct_users),
            2) AS famous_percentage
FROM
    follower_count f;
```

MSSQL Solution:

```sql
WITH distinct_users AS(
    SELECT user_id AS users FROM famous
    UNION
    SELECT follower_id AS users FROM famous
),
follower_count AS(
    SELECT
        user_id, COUNT(follower_id) AS followers
    FROM
        famous
    GROUP BY user_id
)
SELECT
    f.user_id,
    CAST((f.followers * 100.0) / (
                SELECT
                    COUNT(*)
                FROM
                    distinct_users) AS DECIMAL(10,2)
            ) AS famous_percentage
FROM
    follower_count f;
```

Problem 2 (Amazon | Hard Level)

Given a table 'sf_transactions' of purchases by date, calculate the month-over-month percentage change in revenue. The output should include the year-month date (YYYY-MM) and percentage change, rounded to the 2nd decimal point, and sorted from the beginning of the year to the end of the year. The percentage change column will be populated from the 2nd month forward and calculated as

$((this\ month's\ revenue\ -\ last\ month's\ revenue)\ /\ last\ month's\ revenue) * 100.$

Explanation:

1. MonthlyRevenue CTE: Aggregates the total revenue for each month using FORMAT to convert the created_at date to the format YYYY-MM.
2. RevenueChange CTE: Adds a column previous_revenue using the LAG function, which fetches the total revenue of the previous month for each row.
3. Final SELECT: Calculates the percentage change as ((total_revenue - previous_revenue) / previous_revenue) * 100. The ROUND function ensures the percentage is rounded to two decimal places. The output is ordered by year_month to display the data chronologically.

MSSQL Server Solution:

```
WITH MonthlyRevenue AS (
    SELECT
        FORMAT(created_at, 'yyyy-MM') AS year_month,
        SUM(value) AS total_revenue
    FROM
        sf_transactions
    GROUP BY FORMAT(created_at, 'yyyy-MM')
),
RevenueChange AS (
    SELECT
        year_month,
        total_revenue,
        LAG(total_revenue) OVER (ORDER BY year_month) AS previous_revenue
    FROM
        MonthlyRevenue
)
SELECT
    year_month,
    total_revenue,
    ROUND(CASE
                WHEN previous_revenue IS NULL THEN NULL
                ELSE ((total_revenue - previous_revenue) / CAST(previous_revenue AS FLOAT)) * 100
            END,
            2) AS percentage_change
FROM
    RevenueChange
ORDER BY year_month;
```

MySQL Solution:

```sql
WITH MonthlyRevenue AS (
    SELECT
        DATE_FORMAT(created_at, '%Y-%m') AS yearmonth,
        SUM(value) AS total_revenue
    FROM
        sf_transactions
    GROUP BY yearmonth
),
RevenueChange AS (
    SELECT
        yearmonth,
        total_revenue,
        LAG(total_revenue) OVER (ORDER BY yearmonth) AS previous_revenue
    FROM
        MonthlyRevenue
)
SELECT
    yearmonth,
    total_revenue,
    ROUND(CASE
                WHEN previous_revenue IS NULL THEN NULL
                ELSE ((total_revenue - previous_revenue) / previous_revenue) * 100
            END,
            2) AS percentage_change
FROM
    RevenueChange
ORDER BY yearmonth;
```

Problem 3 (Google | Medium Level)

You are analyzing a social network dataset at Google. Your task is to find mutual friends between two users, Karl and Hans. There is only one user named Karl and one named Hans in the dataset.

The output should contain 'user_id' and 'user_name' columns.

Explanation:

1. The CTEs (karl_friends and hans_friends) efficiently find all friends for Karl and Hans, respectively.
2. The main query joins these CTEs with the users table to find the users who are present in both Karl's and Hans's friend lists (mutual friends).

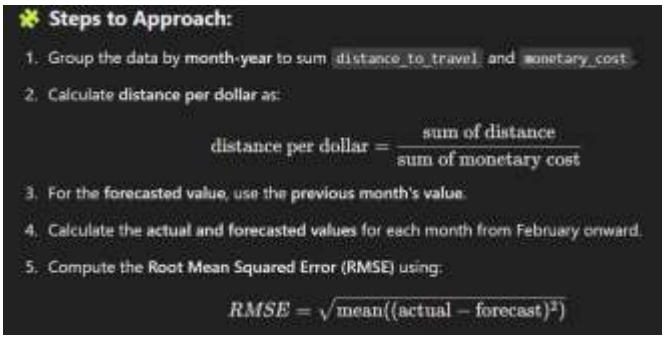
```sql
/*solution 1*/
WITH karl_friends AS(
    SELECT DISTINCT friend_id
    FROM friends
    WHERE user_id = (SELECT user_id FROM users WHERE user_name = 'Karl')
),
hans_friends AS(
    SELECT DISTINCT friend_id
    FROM friends
    WHERE user_id = (SELECT user_id FROM users WHERE user_name = 'Hans')
)
SELECT
    u.user_id, u.user_name
FROM
    users u
        JOIN
    karl_friends kf ON u.user_id = kf.friend_id
        JOIN
    hans_friends hf ON u.user_id = hf.friend_id;


/*solution 2*/
SELECT
    DISTINCT u.user_id, u.user_name
FROM
    users u
        JOIN
    friends f1 ON u.user_id = f1.friend_id
        JOIN
    friends f2 ON u.user_id = f2.friend_id
WHERE
    f1.user_id = (SELECT user_id FROM users WHERE user_name = 'Karl')
        AND f2.user_id = (SELECT user_id FROM users WHERE user_name = 'Hans');
```

Problem 4 (Uber | Hard Level)

Some forecasting methods are extremely simple and surprisingly effective. Naïve forecast is one of them. To create a naïve forecast for "distance per dollar" (defined as distance_to_travel / monetary_cost), first sum the "distance to travel" and "monetary cost" values monthly. This gives the actual value for the current month. For the forecasted value, use the previous month's value. After obtaining both actual and forecasted values, calculate the root mean squared error (RMSE) using the formula

$RMSE = sqrt(mean(square(actual - forecast)))$.

Report the RMSE rounded to two decimal places.



MySQL Solution:

```sql
WITH monthly_aggregates AS(
    SELECT
        DATE_FORMAT(request_date, '%Y-%m') AS yearmonth,
        SUM(distance_to_travel) AS total_distance,
        SUM(monetary_cost) AS total_cost
    FROM
        uber_request_logs
    GROUP BY yearmonth
),
-- Calculate actual distance per dollar
actual_distance AS(
    SELECT
        yearmonth,
        total_distance / total_cost AS actual_distance
    FROM
        monthly_aggregates
),
-- Generate Naïve Forecast
naive_forecast AS(
    SELECT
        yearmonth,
        actual_distance,
        LAG(actual_distance) OVER(ORDER BY yearmonth) AS forecasted_distance
    FROM
        actual_distance
)
-- Calculate RMSE
SELECT
    ROUND(SQRT(AVG(POWER(actual_distance - forecasted_distance, 2))) ,2) AS rmse
FROM
    naive_forecast
WHERE
    forecasted_distance IS NOT NULL;
```

MSSQL Solution:

```sql
-- Calculate monthly aggregates
WITH monthly_aggregates AS(
    SELECT
        FORMAT(request_date, 'yyyy-MM') AS yearmonth,
        SUM(distance_to_travel) AS total_distance,
        SUM(monetary_cost) AS total_cost
    FROM
        uber_request_logs
    GROUP BY FORMAT(request_date, 'yyyy-MM')
),
-- Calculate actual distance per dollar
actual_distance AS(
    SELECT
        yearmonth,
        total_distance / total_cost AS actual_distance
    FROM
        monthly_aggregates
),
-- Generate Naïve Forecast
naive_forecast AS(
    SELECT
        yearmonth,
        actual_distance,
        LAG(actual_distance) OVER(ORDER BY yearmonth) AS forecasted_distance
    FROM
        actual_distance
)
-- Calculate RMSE
SELECT
    ROUND(SQRT(AVG(POWER(actual_distance - forecasted_distance, 2))) ,2) AS rmse
FROM
    naive_forecast
WHERE
    forecasted_distance IS NOT NULL;
```

Problem 5 (Microsoft | Medium Level)

Given a list of projects and employees mapped to each project, calculate by the amount of project budget allocated to each employee. The output should include the project title and the project budget rounded to the closest integer. Order your list by projects with the highest budget per employee first.

Explanation:

1. Joining Tables: The initial step involves joining the ms_projects and ms_emp_projects tables on the project ID to combine project details (including titles and budgets) with employee assignments.
2. Grouping and Aggregating: The data is then grouped by project title and budget, allowing for the calculation of budget per employee by dividing the total budget of each project by the count of employees assigned to that project.
3. Rounding and Ordering: Finally, the computed budget per employee is rounded to the nearest integer, and the results are ordered in descending order to prioritize projects with the highest budget allocation per employee.

```sql
-- Solution
SELECT
    p.title AS project_title,
    ROUND(p.budget / COUNT(e.project_id), 0) AS budget_per_employee
FROM
    ms_projects p
        INNER JOIN
    ms_emp_projects e ON p.id = e.project_id
GROUP BY p.id , p.title , p.budget
ORDER BY budget_per_employee DESC;

-- in real case scenario there will be multiple employees tagged to one project
-- this one is correct solution
SELECT
    p.title AS project_title,
    ROUND(SUM(p.budget) / COUNT(e.project_id), 0) AS budget_per_employee
FROM
    ms_projects p
        INNER JOIN
    ms_emp_projects e ON p.id = e.project_id
GROUP BY p.id , p.title , p.budget
ORDER BY budget_per_employee DESC;
```

Problem 6 (Airbnb | Medium Level)

Find the total number of available beds per hosts' nationality. Output the nationality along with the corresponding total number of available beds. Sort records by the total available beds in descending order.

Explanation

1. Joining Tables: The first step involves joining the airbnb_apartments and airbnb_hosts tables on the host_id. This allows us to combine the apartment details (such as the number of beds) with the host's nationality information.
2. Grouping and Aggregating: Next, the data is grouped by the host's nationality, so that the total number of beds available for each nationality can be calculated. The SUM() function is used to add up the beds (n_beds) for all apartments hosted by individuals of the same nationality.
3. Sorting the Results: Finally, the results are ordered in descending order based on the total number of available beds.

```
35  -- Solution
36  SELECT
37      h.nationality,
38      SUM(a.n_beds) AS total_available_beds
39  FROM
40      airbnb_apartments a
41          INNER JOIN
42      airbnb_hosts h ON a.host_id = h.host_id
43  GROUP BY h.nationality;
```

Problem 7 (IBM | Hard Level)

IBM is working on a new feature to analyze user purchasing behavior for all Fridays in the first quarter of the year. For each Friday separately, calculate the average amount users have spent per order. The output should contain the week number of that Friday and average amount spent.

To solve this problem, we need to:

1. Identify the Fridays in the first quarter (Q1) of the year.
2. Calculate the week number for each of these Fridays.
3. Group the purchases by week number and calculate the average amount spent per order.

MSSQL Solution

```sql
-- solution with cte
WITH q1_friday AS(
    SELECT
        amount_spent, DATEPART(WEEK, date) AS week_number
    FROM
        user_purchases
    WHERE
        day_name = 'Friday'
)
SELECT
    week_number, ROUND(AVG(amount_spent), 2) AS avg_amount_spent
FROM
    q1_friday
GROUP BY week_number
ORDER BY week_number;

-- without cte
SELECT
    DATEPART(WEEK, date) AS week_number,
    ROUND(AVG(amount_spent), 2) AS avg_amount_spent
FROM
    user_purchases
WHERE
    day_name = 'Friday'
GROUP BY DATEPART(WEEK, date)
ORDER BY week_number;
```

MySQL Solution:

```sql
-- solution with cte
WITH q1_friday AS(
    SELECT
        amount_spent, WEEK(date) AS week_number
    FROM
        user_purchases
    WHERE
        day_name = 'Friday'
)
SELECT
    week_number, ROUND(AVG(amount_spent), 2) AS avg_amount_spent
FROM
    q1_friday
GROUP BY week_number
ORDER BY week_number;

-- without cte
SELECT
    WEEK(date) AS week_number,
    ROUND(AVG(amount_spent), 2) AS avg_amount_spent
FROM
    user_purchases
WHERE
    day_name = 'Friday'
GROUP BY week_number
ORDER BY week_number;
```

Problem 8 (Tesla | Medium Level)

You are given a table of product launches by company by year. Write a query to count the net difference between the number of products companies launched in 2020 with the number of products companies launched in the previous year. Output the name of the companies and a net difference of net products released for 2020 compared to the previous year.

Explanation:

1. Counting Products per Year: Using SUM with CASE statements, we count the number of products launched in 2020 and 2019 separately for each company.
2. Calculating Net Difference: We calculate the difference between 2020 and 2019 product counts to get the net change.
3. Ordering: The results are ordered by net_difference in descending order to show companies with the highest increase first.

```sql
-- solution
WITH products_counts AS(
    SELECT
        company_name,
        SUM(CASE WHEN year = 2020 THEN 1 ELSE 0 END) AS products_2020,
        SUM(CASE WHEN year = 2019 THEN 1 ELSE 0 END) AS products_2019
    FROM
        car_launches
    GROUP BY company_name
)
SELECT
    company_name,
    (products_2020 - products_2019) AS product_difference
FROM
    products_counts
ORDER BY product_difference DESC;
```

Problem 9 (Netflix | Hard Problem)

Find the genre of the person with the most number of oscar winnings.
If there are more than one person with the same number of oscar wins, return the first one in alphabetic order based on their name. Use the names as keys when joining the tables.

Explanation:

1. WinnerCount CTE: Calculates the total Oscar wins for each nominee by counting rows where winner = 1.
2. Final Selection: The TOP 1 clause fetches all rows with the highest total_wins, sorted alphabetically by name to handle ties. We join the WinnerCount CTE with nominee_information on the nominee's name to retrieve the top_genre for the top nominee(s) in terms of Oscar wins.

MSSQL Solution

```sql
-- solution
WITH winner_count AS(
SELECT
    nominee, COUNT(*) AS total_wins
FROM
    oscar_nominees
WHERE
    winner = 1
GROUP BY nominee
)
SELECT TOP 1
    ni.top_genre
FROM
    winner_count wc
        JOIN
    nominee_information ni ON wc.nominee = ni.name
ORDER BY wc.total_wins DESC , ni.name ASC;
```

MySQL Solution

```sql
-- solution
WITH winner_count AS(
SELECT
    nominee, COUNT(*) AS total_wins
FROM
    oscar_nominees
WHERE
    winner = 1
GROUP BY nominee
)
SELECT
    ni.top_genre
FROM
    winner_count wc
        JOIN
    nominee_information ni ON wc.nominee = ni.name
ORDER BY wc.total_wins DESC , ni.name ASC
LIMIT 1;
```

Problem 10 (Amazon | Medium Level)

Write a query that'll identify returning active users. A returning active user is a user that has made a second purchase within 7 days of any other of their purchases. Output a list of user_ids of these returning active users.

MSSQL Solution

```sql
-- solution 1
WITH cte_next_purchase AS(
    SELECT
        user_id,
        created_at,
        LAG(created_at) OVER(PARTITION BY user_id ORDER BY created_at DESC) AS next_purchase
    FROM
        amazon_transactions
)
SELECT
    user_id
FROM
    cte_next_purchase
WHERE
    DATEDIFF(DAY, created_at, next_purchase) <= 7;

-- solution 2
SELECT DISTINCT
    a.user_id
FROM
    amazon_transactions a
        JOIN
    amazon_transactions b ON a.user_id = b.user_id
        AND a.created_at < b.created_at
WHERE
    DATEDIFF(DAY, a.created_at, b.created_at) <= 7;
```

MySQL Solution

```sql
-- solution 1
WITH cte_next_purchase AS(
    SELECT
        user_id,
        created_at,
        LAG(created_at) OVER(PARTITION BY user_id ORDER BY created_at DESC) AS next_purchase
    FROM
        amazon_transactions
)
SELECT
    user_id
FROM
    cte_next_purchase
WHERE
    DATEDIFF(next_purchase, created_at) <= 7;

-- solution 2
SELECT DISTINCT
    a.user_id
FROM
    amazon_transactions a
        JOIN
    amazon_transactions b ON a.user_id = b.user_id
        AND a.created_at < b.created_at
WHERE
    DATEDIFF(b.created_at, a.created_at) <= 7;
```

Problem 11 (Nvidia, Microsoft | Medium Level)

Find the number of transactions that occurred for each product. Output the product name along with the corresponding number of transactions and order records by the product id in ascending order. You can ignore products without transactions.

Explanation:

1. Joining Tables: The INNER JOIN between excel_sql_inventory_data (aliased as inv) and excel_sql_transaction_data (aliased as trans) matches records by product_id. This way, only products with transactions are included.
2. Counting Transactions: Using COUNT(trans.transaction_id) counts the number of transactions for each product.
3. Grouping and Ordering: GROUP BY inv.product_id, inv.product_name groups by product_id and product_name to get the transaction count per product. ORDER BY inv.product_id ASC sorts the output by product_id in ascending order.

```sql
-- solution
SELECT
    i.product_name,
    COUNT(t.transaction_id) AS transcation_count
FROM
    excel_sql_inventory_data i
        JOIN
    excel_sql_transaction_data t ON i.product_id = t.product_id
GROUP BY t.product_id , i.product_name
ORDER BY t.product_id ASC;
```

Problem 12 (LinkedIn, Dropbox | Basic Level)

Write a query that calculates the difference between the highest salaries found in the marketing and engineering departments. Output just the absolute difference in salaries.

Explanation

1. CASE is used to selectively get the salary for the "marketing" and "engineering" departments.
2. MAX is applied to retrieve the highest salary in each department.
3. ABS calculates the absolute difference between the two maximum values.

```sql
-- solution
SELECT ABS(
    MAX(CASE WHEN d.department = 'marketing' THEN e.salary END)
        -
    MAX(CASE WHEN d.department = 'engineering' THEN e.salary END)
) AS salary_differnece
FROM
    db_employee e
        JOIN
    db_dept d ON e.department_id = d.id;
```

Problem 13 (Expedia, Airbnb | Basic Level)

Find the number of rows for each review score earned by 'Hotel Arena'. Output the hotel name (which should be 'Hotel Arena'), review score along with the corresponding number of rows with that score for the specified hotel.

```sql
-- solution
SELECT
    hotel_name,
    reviewer_score,
    COUNT(*) AS score_count
FROM
    hotel_reviews
WHERE
    hotel_name = 'Hotel Arena'
GROUP BY hotel_name , reviewer_score
ORDER BY reviewer_score DESC;
```

Problem 14 (Amazon, Salesforce | Basic Level)

What is the total sales revenue of Samantha and Lisa?

```sql
-- solution
SELECT
    SUM(sales_revenue) AS total_sales_revenue
FROM
    sales_performance
WHERE
    salesperson IN ('Samantha', 'Lisa');
```

Problem 15 (Google Medium | Level)

Find all records from days when the number of distinct users receiving emails was greater than the number of distinct users sending emails.

Explanation:

1. The distinct_counts CTE calculates the number of distinct to_user and from_user for each day.
2. The main query joins the original google_gmail_emails table with distinct_counts on the day field, selecting only records where distinct_receivers is greater than distinct_senders.

```sql
-- solution
WITH distinct_counts AS(
    SELECT
        day,
        COUNT(DISTINCT from_user) AS distinct_sender,
        COUNT(DISTINCT to_user) AS distinct_receiver
    FROM
        google_gmail_emails
    GROUP BY day
)
SELECT
    g.id, g.from_user, g.to_user, g.day
FROM
    google_gmail_emails g
        JOIN
    distinct_counts dc ON g.day = dc.day
WHERE
    dc.distinct_receiver > dc.distinct_sender;
```

Problem 16 (JP Morgan Chase, Bloomberg | Medium Level)

Bank of Ireland has requested that you detect invalid transactions in December 2022. An invalid transaction is one that occurs outside of the bank's normal business hours. The following are the hours of operation for all branches:
Monday - Friday 09:00 - 16:00
Saturday & Sunday Closed
Irish Public Holidays 25th and 26th December
Determine the transaction ids of all invalid transactions.

Explanation:

1. MONTH(time_stamp) = 12 AND YEAR(time_stamp) = 2022: This filters transactions to include only those in December 2022.
2. DATEPART(WEEKDAY, time_stamp) IN (1, 7): This checks if the transaction occurred on a Saturday (7) or Sunday (1).
3. CAST(time_stamp AS TIME) < '09:00:00': This checks if the transaction time is before the opening hours.
4. CAST(time_stamp AS TIME) > '16:00:00': This checks if the transaction time is after the closing hours.
5. (DATEPART(DAY, time_stamp) IN (25, 26) AND MONTH(time_stamp) = 12): This checks if the transaction occurred on the public holidays of December 25th or 26th.

MSSQL Solution:

```sql
-- solution
SELECT
    transaction_id
FROM
    boi_transactions
WHERE
    YEAR(time_stamp) = 2022
    AND MONTH(time_stamp) = 12
    AND (
        DATEPART(WEEKDAY, time_stamp) IN (1,7)
        OR CAST(time_stamp AS TIME) < '09:00:00'
        OR CAST(time_stamp AS TIME) > '16:00:00'
        OR (DATEPART(DAY, time_stamp) IN (25,26) AND MONTH(time_stamp) = 12)
    );
```

MySQL Solution

```sql
-- solution
SELECT
    transaction_id
FROM
    boi_transactions
WHERE
    YEAR(time_stamp) = 2022
    AND MONTH(time_stamp) = 12
    AND (
        DAYOFWEEK(time_stamp) IN (1 , 7)
        OR CAST(time_stamp AS TIME) < '09:00:00'
        OR CAST(time_stamp AS TIME) > '16:00:00'
        OR (DAY(time_stamp) IN (25 , 26) AND MONTH(time_stamp) = 12)
    );
```

Problem 17 (Uber | Medium Level)

You're given a table of Uber rides that contains the mileage and the purpose for the business expense. You're asked to find business purposes that generate the most miles driven for passengers that use Uber for their business transportation. Find the top 3 business purpose categories by total mileage.

```sql
-- solution
SELECT TOP 3
    purpose,
    SUM(miles) AS total_miles
FROM
    my_uber_drives
GROUP BY purpose
ORDER BY total_miles DESC;
```

Problem 18 (Amazon, Doordash | Medium Level)

You have been asked to find the job titles of the highest-paid employees.
Your output should include the highest-paid title or multiple titles with the same salary.

```sql
-- solution 1
-- using simple subquery
SELECT
    t.worker_title
FROM
    worker w
        JOIN
    title t ON w.worker_id = t.worker_ref_id
WHERE
    w.salary = (
        SELECT MAX(salary) FROM worker
    );

-- solution 2
-- using cte and window function
WITH salary_ranks AS(
    SELECT
        t.worker_title,
        w.salary,
        DENSE_RANK() OVER(ORDER BY w.salary DESC) AS worker_rank
    FROM
        worker w
            JOIN
        title t ON w.worker_id = t.worker_ref_id
)
SELECT
    worker_title
FROM
    salary_ranks
WHERE worker_rank = 1;
```

Problem 19 (Walmart | Hard Level)

Identify users who started a session and placed an order on the same day.
For these users, calculate the total number of orders and the total order value for that day.
Your output should include the user, the session date, the total number of orders, and the total order value for that day.

```sql
-- solution
SELECT
    s.user_id,
    s.session_date,
    COUNT(os.order_id) AS total_orders,
    SUM(os.order_value) AS total_order_value
FROM
    sessions s
        JOIN
    order_summary os ON s.user_id = os.user_id
        AND CAST(s.session_date AS DATE) = CAST(os.order_date AS DATE)
GROUP BY s.user_id , s.session_date
HAVING COUNT(os.order_id) > 0;
```

Problem 20 (Apple, Microsoft, Dell | Easy Level)

Write a query that returns the number of unique users per client per month

MSSQL Solution

```sql
-- solution
SELECT
    client_id,
    FORMAT(time_id, 'yyyy-MM') AS month_year,
    COUNT(DISTINCT user_id) AS unique_user_count
FROM
    fact_events
GROUP BY client_id, FORMAT(time_id, 'yyyy-MM')
ORDER BY client_id;
```

MySQL Solution

```sql
-- solution
SELECT
    client_id,
    DATE_FORMAT(time_id, '%Y-%m') AS month_year,
    COUNT(DISTINCT user_id) AS unique_user_count
FROM
    fact_events
GROUP BY client_id , month_year
ORDER BY client_id;
```

Problem 21 (Microsoft | Hard Level)

Find the total number of downloads for paying and non-paying users by date. Include only records where non-paying customers have more downloads than paying customers. The output should be sorted by earliest date first and contain 3 columns date, non-paying downloads, paying downloads.

MSSQL Solution:

```
-- solution
SELECT
    CONVERT (DATE, d.date) AS [date],
    SUM(CASE
        WHEN a.paying_customer = 'NO' THEN d.downloads
        ELSE 0
    END) AS non_paying_downloads,
    SUM(CASE
        WHEN a.paying_customer = 'YES' THEN d.downloads
        ELSE 0
    END) AS paying_downloads
FROM
    ms_user_dimension u
        JOIN
    ms_acc_dimension a ON u.acc_id = a.acc_id
        JOIN
    ms_download_facts d ON u.user_id = d.user_id
GROUP BY CONVERT (DATE, d.date)
HAVING
    SUM(CASE WHEN a.paying_customer = 'NO' THEN d.downloads ELSE 0 END) >
    SUM(CASE WHEN a.paying_customer = 'YES' THEN d.downloads ELSE 0 END)
ORDER BY [date];
```

MySQL Solution:

```sql
-- solution
SELECT
    DATE(d.date) AS date,
    SUM(CASE
        WHEN a.paying_customer = 'NO' THEN d.downloads
        ELSE 0
    END) AS non_paying_downloads,
    SUM(CASE
        WHEN a.paying_customer = 'YES' THEN d.downloads
        ELSE 0
    END) AS paying_downloads
FROM
    ms_user_dimension u
        JOIN
    ms_acc_dimension a ON u.acc_id = a.acc_id
        JOIN
    ms_download_facts d ON u.user_id = d.user_id
GROUP BY date
HAVING non_paying_downloads > paying_downloads
ORDER BY date;
```

Problem 22 (Walmart, Paypal | Medium Level)

Find managers with at least 7 direct reporting employees. In situations where user is reporting to himself/herself, count that also.
Output first names of managers.

```sql
-- solution
SELECT
    m.first_name
FROM
    employees e
        JOIN
    employees m ON e.manager_id = m.id
GROUP BY m.first_name
HAVING COUNT(e.id) >= 7;
```

Problem 23 (Oracle | Hard Level)

Write a query that compares each employee's salary to their manager's and the average department salary (excluding the manager's salary). Display the department, employee ID, employee's salary, manager's salary, and department average salary. Order by department, then by employee salary (highest to lowest).

```sql
-- solution
WITH department_avg AS(
    SELECT
        department,
        AVG(salary) AS avg_salary
    FROM
        employee_o
    GROUP BY department
)
SELECT
    e.department,
    e.id,
    e.salary AS employee_salary,
    m.salary AS manager_salary,
    d.avg_salary AS dept_avg_salary
FROM
    employee_o e
        LEFT JOIN
    employee_o m ON e.manager_id = m.id AND e.id != e.manager_id
        JOIN
    department_avg d ON e.department = d.department
ORDER BY e.department, e.salary DESC;
```

Problem 24 (Amazon | Hard Level)

Find products which are exclusive to only Amazon and therefore not sold at Top Shop and Macy's. Your output should include the product name, brand name, price, and rating.
Two products are considered equal if they have the same product name and same maximum retail price (mrp column).

```sql
-- solution
SELECT
    a.product_name,
    a.brand_name,
    a.price,
    a.rating
FROM
    innerwear_amazon_com a
        LEFT JOIN
    innerwear_macys_com m ON a.product_name = m.product_name AND a.mrp = m.mrp
        LEFT JOIN
    innerwear_topshop_com t ON a.product_name = t.product_name AND a.mrp = t.mrp
WHERE
    m.product_name IS NULL
        AND t.product_name IS NULL
ORDER BY a.product_name;
```

Problem 25 (American Express | Medium Level)

American Express is reviewing their customers' transactions, and you have been tasked with locating the customer who has the third highest total transaction amount. The output should include the customer's id, as well as their first name and last name. For ranking the customers, use type of ranking with no gaps between subsequent ranks.

```sql
-- solution
WITH customer_ranking AS(
    SELECT
        c.id,
        c.first_name,
        c.last_name,
        SUM(o.total_order_cost) AS total_cost,
        DENSE_RANK() OVER(ORDER BY SUM(o.total_order_cost) DESC) AS customer_rank
    FROM
        customers c
            JOIN
        card_orders o ON c.id = o.cust_id
    GROUP BY c.id, c.first_name, c.last_name
)
SELECT
    id, first_name, last_name
FROM
    customer_ranking
WHERE
    customer_rank = 3;
```

Problem 26 (LinkedIn | Hard Level)

Consider all LinkedIn users who, at some point, worked at Microsoft. For how many of them was Google their next employer right after Microsoft (no employers in between)?

```sql
-- solution
WITH employer_details AS(
    SELECT
        user_id,
        employer,
        position,
        start_date,
        end_date,
        LAG(employer) OVER(PARTITION BY user_id ORDER BY start_date) AS previous_employer,
        LEAD(employer) OVER(PARTITION BY user_id ORDER BY start_date) AS next_employer
    FROM
        linkedin_users
)
SELECT
    user_id,
    employer,
    position,
    start_date,
    end_date
FROM
    employer_details
WHERE
    employer = 'Microsoft' AND next_employer = 'Google';
```

Problem 27 (Goldman Sachs, Deloitte | Hard Level)

You are given a day worth of scheduled departure and arrival times of trains at one train station. One platform can only accommodate one train from the beginning of the minute it's scheduled to arrive until the end of the minute it's scheduled to depart. Find the minimum number of platforms necessary to accommodate the entire scheduled traffic.

Explanation:

1. TrainTimes CTE: We combine both the arrival and departure times into one unified dataset. For arrivals, we use 1 as the event_type to indicate the need for a platform. For departures, we use -1 as the event_type to indicate the freeing of a platform.
2. PlatformCount Subquery: We use a SUM with a window function (OVER clause) to maintain a running count of the platforms needed. The event_type for arrival adds one platform and the event_type for departure subtracts one.
3. Max(platforms_needed): Finally, we get the maximum value from the platforms_needed, which represents the maximum number of platforms required at any point in time.

```sql
-- solution
WITH TrainTimes AS(
    SELECT
        arrival_time AS event_time,
        1 AS event_type
    FROM
        train_arrivals
    UNION ALL
    SELECT
        departure_time AS event_time,
        - 1 AS event_type
    FROM
        train_departures
),
PlatformCount AS(
    SELECT
        event_time,
        SUM(event_type) OVER(ORDER BY event_time) AS platform_needed
    FROM
        TrainTimes
)
SELECT
    MAX(platform_needed) AS min_platform
FROM
    PlatformCount;
```

Problem 28 (Meta, Salesforce | Hard Level)

Find the highest salary among salaries that appears only once.

```sql
-- solution
WITH salarycount AS(
    SELECT
        salary,
        COUNT(salary) AS salary_count
    FROM employee_meta
    GROUP BY salary
)
SELECT
    MAX(salary) AS highest_unique_salary
FROM
    salarycount
WHERE salary_count = 1;
```

Problem 29 (Cisco | Hard Level)

Convert the first letter of each word found in content_text to uppercase, while keeping the rest of the letters lowercase. Your output should include the original text in one column and the modified text in another column.

Explanation:

1. STRING_SPLIT(content_text, ' ') splits content_text by spaces, breaking it down into words.
2. CROSS APPLY allows the function to be applied to each row, splitting content_text into individual words as rows.
3. UPPER(LEFT(value, 1)) converts the first letter of each word to uppercase.
4. LOWER(SUBSTRING(value, 2, LEN(value))) converts the rest of each word to lowercase.
5. STRING_AGG(..., ' ') aggregates the words back into a single string, with each word separated by a space.
6. The result is grouped by content_id and content_text, displaying both the original and modified text.

MSSQL Solution:

```sql
-- solution
SELECT
    content_id,
    content_text AS orginal_text,
    STRING_AGG(UPPER(LEFT(value, 1)) + LOWER(SUBSTRING(value, 2, LEN(value))), ' ') AS modified_text
FROM
    user_content
CROSS APPLY
    string_split(content_text, ' ')
GROUP BY
    content_id, content_text;
```

Problem 30 (Amazon | Hard Level)

Given the users' sessions logs on a particular day, calculate how many hours each user was active that day.
Note: The session starts when state=1 and ends when state=0.

Explanation:

1. SessionDurations CTE: Use the LAG() function to get the timestamp of the previous state for each cust_id. Filter rows where state=0 because this indicates the end of a session. The session_start is derived from the LAG() value, and session_end is the current timestamp.
2. ActiveHours CTE: Calculate the active duration for each session in minutes using DATEDIFF(MINUTE, session_start, session_end).
3. Final SELECT: Sum the active minutes for each cust_id. Divide the total minutes by 60 to convert them to hours.

```sql
-- solution
WITH SessionDuration AS(
    SELECT
        cust_id,
        CAST(LAG(timestamp) OVER (PARTITION BY cust_id ORDER BY timestamp) AS TIME) AS session_start,
        CAST(timestamp AS TIME) AS session_end
    FROM
        customer_state_log
    WHERE state = 0
),
ActiveHours AS(
    SELECT
        cust_id,
        DATEDIFF(HOUR, session_start, session_end) AS active_hours
    FROM
        SessionDuration
    WHERE session_start IS NOT NULL
)
SELECT
    cust_id,
    SUM(active_hours) AS total_active_hours
FROM
    ActiveHours
GROUP BY cust_id;
```

```sql
-- solution
WITH SessionDuration AS (
    SELECT
        cust_id,
        CAST(LAG(timestamp) OVER (PARTITION BY cust_id ORDER BY timestamp) AS TIME) AS session_start,
        CAST(timestamp AS TIME) AS session_end
    FROM
        customer_state_log
    WHERE state = 0
),
ActiveHours AS (
    SELECT
        cust_id,
        TIMESTAMPDIFF(HOUR, session_start, session_end) AS active_hours
    FROM
        SessionDuration
    WHERE session_start IS NOT NULL
)
SELECT
    cust_id,
    SUM(active_hours) AS total_active_hours
FROM
    ActiveHours
GROUP BY cust_id;
```

Problem 31 (Spotify | Hard Level)

Find the number of days a US track has stayed in the 1st position for both the US and worldwide rankings on the same day. Output the track name and the number of days in the 1st position. Order your output alphabetically by track name. If the region 'US' appears in dataset, it should be included in the worldwide ranking

```sql
-- solution
SELECT
    us.trackname,
    COUNT(*) AS days_in_first_position
FROM
    spotify_daily_rankings_2017_us us
        JOIN
    spotify_worldwide_daily_song_ranking w ON us.trackname = w.trackname
        AND us.date = w.date
WHERE
    us.position = 1
    AND w.position = 1
    AND w.region = 'US'
GROUP BY us.trackname
ORDER BY us.trackname;
```

Problem 32 (Accenture | Medium Level)

Following a recent advertising campaign, the marketing department wishes to classify its efforts based on the total number of units sold for each product.

You have been tasked with calculating the total number of units sold for each product and categorizing ad performance based on the following criteria for items sold:

- Outstanding: 30+
- Satisfactory: 20 - 29
- Unsatisfactory: 10 - 19
- Poor: 1 - 9

Your output should contain the product ID, total units sold in descending order, and its categorized ad performance.

```sql
-- solution
SELECT
    product_id,
    SUM(quantity) AS total_units_sold,
    CASE
        WHEN SUM(quantity) >= 30 THEN 'Outstanding'
        WHEN SUM(quantity) BETWEEN 20 AND 29 THEN 'Satisfactory'
        WHEN SUM(quantity) BETWEEN 10 AND 19 THEN 'Unsatisfactory'
        WHEN SUM(quantity) BETWEEN 1 AND 9 THEN 'Poor'
        ELSE 'No Sales'
    END AS categorized_ad_performance
FROM
    marketing_campaign
GROUP BY product_id
ORDER BY total_units_sold DESC;
```

Problem 33 (Google | Hard Level)

Calculate the average session distance traveled by Google Fit users using GPS data for two scenarios:
Considering Earth's curvature (Haversine formula).
Assuming a flat surface.
For each session, use the distance between the highest and lowest step IDs, and ignore sessions with only one step.
Calculate and output the average distance for both scenarios and the difference between them.
Formulas:
1. $Curved\ Earth: d = 6371 \times arccos(sin(\phi 1) \times sin(\phi 2) + cos(\phi 1) \times cos(\phi 2) \times cos(\lambda 2 - \lambda 1))$
2. $Flat\ Surface: d = 111 \times (lat2 - lat1)2 + (lon2 - lon1)2$

```sql
-- solution
WITH SessionDetails AS (
    SELECT
        session_id,
        MIN(step_id) AS min_step_id,
        MAX(step_id) AS max_step_id
    FROM
        google_fit_location
    GROUP BY
        session_id
    HAVING
        COUNT(step_id) > 1 -- Ignore sessions with only one step
),
SessionCoordinates AS (
    SELECT
        sd.session_id,
        gfl_min.latitude AS lat1,
        gfl_min.longitude AS lon1,
        gfl_max.latitude AS lat2,
        gfl_max.longitude AS lon2
    FROM
        SessionDetails sd
    JOIN google_fit_location gfl_min
        ON sd.session_id = gfl_min.session_id AND sd.min_step_id = gfl_min.step_id
    JOIN google_fit_location gfl_max
        ON sd.session_id = gfl_max.session_id AND sd.max_step_id = gfl_max.step_id
),
Distances AS (
    SELECT
        session_id,
        -- Haversine formula for curved Earth distance
        6371 * ACOS(
            COS(RADIANS(lat1)) * COS(RADIANS(lat2)) * COS(RADIANS(lon2) - RADIANS(lon1))
            + SIN(RADIANS(lat1)) * SIN(RADIANS(lat2))
        ) AS curved_distance,
        -- Flat surface formula
        111 * SQRT(POWER(lat2 - lat1, 2) + POWER(lon2 - lon1, 2)) AS flat_distance
    FROM
        SessionCoordinates
)
SELECT
    AVG(curved_distance) AS avg_curved_distance,
    AVG(flat_distance) AS avg_flat_distance,
    ABS(AVG(curved_distance) - AVG(flat_distance)) AS distance_difference
FROM
    Distances;
```

Problem 34 (Google, Airbnb, Expedia | Medium Level)

Find the three ten hotels with the highest ratings. Output the hotel name along with the corresponding average score.

Sort records based on the average score in descending order.

```sql
-- solution
SELECT TOP 3
    hotel_name,
    average_score
FROM
    hotel_address
ORDER BY average_score DESC;
```

```sql
-- solution
SELECT
    hotel_name,
    average_score
FROM
    hotel_address
ORDER BY average_score DESC
LIMIT 3 OFFSET 0;
```

Problem 35 (Twitter | Medium Level)

Find the top three distinct salaries for each department. Output the department name and the top 3 distinct salaries by each department.

Order your results alphabetically by department and then by highest salary to lowest.

```sql
-- solution
WITH DeptSalaryRank AS (
    SELECT
        department,
        salary,
        DENSE_RANK() OVER(PARTITION BY department ORDER BY salary DESC) AS salaryrank
    FROM
        employees_x
    WHERE salary IS NOT NULL
    GROUP BY department, salary
)
SELECT
    department,
    salary
FROM
    DeptSalaryRank
WHERE salaryrank <= 3
ORDER BY department ASC, salary DESC;
```

Problem 36 (Uber | Hard Level)

Find the most profitable location.

Write a query that calculates the average signup duration and average transaction amount for each location,

and then compare these two measures together by taking the ratio of the average transaction amount and average duration for each location.

```sql
-- solution
WITH SignupDuration AS (
    SELECT
        signup_id,
        location,
        DATEDIFF(MINUTE, signup_start_date, signup_stop_date) AS signup_duration_minute
    FROM
        signups
),
TransactionAmount AS(
    SELECT
        signup_id,
        AVG(amt) AS avg_transaction_amt
    FROM
        transactions
    GROUP BY signup_id
)
SELECT
    s.location,
    AVG(s.signup_duration_minute) AS avg_signup_duration,
    AVG(t.avg_transaction_amt) AS avg_transaction_amount,
    CASE
        WHEN AVG(s.signup_duration_minute) = 0 THEN 0
        ELSE AVG(t.avg_transaction_amt) / AVG(s.signup_duration_minute)
    END AS ratio
FROM
    SignupDuration s
        JOIN
    TransactionAmount t ON s.signup_id = t.signup_id
GROUP BY s.location
ORDER BY ratio DESC;
```

Problem 37 (Netflix | Hard Level)

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

MSSQL Solution:

```sql
-- solution
WITH activity_history AS(
    SELECT
        user_id,
        LAG(date) OVER(PARTITION BY user_id ORDER BY date ASC) AS previous_date,
        date AS present_date,
        LEAD(date) OVER(PARTITION BY user_id ORDER BY date ASC) AS next_date
    FROM
        sf_events
)
SELECT
    [user_id]
FROM
    activity_history
WHERE
    DATEDIFF(DAY, previous_date, present_date) = 1
        AND DATEDIFF(DAY, present_date, next_date) = 1;
```

MySQL Solution:

```sql
-- solution
WITH activity_history AS(
    SELECT
        user_id,
        LAG(date) OVER(PARTITION BY user_id ORDER BY date ASC) AS previous_date,
        date AS present_date,
        LEAD(date) OVER(PARTITION BY user_id ORDER BY date ASC) AS next_date
    FROM
        sf_events
)
SELECT
    user_id
FROM
    activity_history
WHERE
    DATEDIFF(present_date, previous_date) = 1
        AND DATEDIFF(next_date, present_date) = 1;
```

Problem 38 (Airbnb | Hard Level)

Estimate the growth of Airbnb each year using the number of hosts registered as the growth metric. The rate of growth is calculated by taking

$$((number\ of\ hosts\ registered\ in\ the\ current\ year - number\ of\ hosts\ registered\ in\ the\ previous\ year) / the\ number\ of\ hosts\ registered\ in\ the\ previous\ year) * 100$$

Output the year, number of hosts in the current year, number of hosts in the previous year, and the rate of growth. Round the rate of growth to the nearest percent and order the result in the ascending order based on the year.

```sql
-- Solution
WITH YearlyHostCount AS (
    SELECT
        YEAR(host_since) AS year,
        COUNT(DISTINCT id) AS hosts_in_current_year
    FROM
        airbnb_search_details
    WHERE
        host_since IS NOT NULL
    GROUP BY
        YEAR(host_since)
),
GrowthCalculation AS (
    SELECT
        yc1.year AS current_year,
        yc1.hosts_in_current_year,
        yc2.hosts_in_current_year AS hosts_in_previous_year,
        ROUND(
            CASE
                WHEN yc2.hosts_in_current_year = 0 THEN NULL
                ELSE ((yc1.hosts_in_current_year - yc2.hosts_in_current_year) * 100.0) / yc2.hosts_in_current_year
            END, 0
        ) AS growth_rate
    FROM
        YearlyHostCount yc1
            LEFT JOIN
        YearlyHostCount yc2 ON yc1.year = yc2.year + 1
)
SELECT
    current_year,
    hosts_in_current_year,
    hosts_in_previous_year,
    growth_rate
FROM
    GrowthCalculation
ORDER BY
    current_year;
```

Problem 39 (Walmart (Hard Level)

Identify users who started a session and placed an order on the same day. For these users, calculate the total number of orders and the total order value for that day. Your output should include the user, the session date, the total number of orders, and the total order value for that day.)

```sql
-- solution
SELECT
    s.user_id,
    CAST(s.session_date AS DATE) AS session_date,
    COUNT(o.order_id) AS total_orders,
    SUM(o.order_value) AS total_order_value
FROM
    sessions s
        JOIN
    order_summary o ON s.user_id = o.user_id
WHERE
    CAST(s.session_date AS DATE) = CAST(o.order_date AS DATE)
GROUP BY s.user_id , CAST(s.session_date AS DATE)
ORDER BY s.user_id , session_date;
```

Problem 40 (Walmart | Hard Level)

Identify users who started a session and placed an order on the same day. For these users, calculate the total number of orders and the total order value for that day. Your output should include the user, the session date, the total number of orders, and the total order value for that day.)

```sql
-- solution
SELECT
    s.user_id,
    CAST(s.session_date AS DATE) AS session_date,
    COUNT(o.order_id) AS total_orders,
    SUM(o.order_value) AS total_order_value
FROM
    sessions s
        JOIN
    order_summary o ON s.user_id = o.user_id
WHERE
    CAST(s.session_date AS DATE) = CAST(o.order_date AS DATE)
GROUP BY s.user_id , CAST(s.session_date AS DATE)
ORDER BY s.user_id , session_date;
```

Problem 41 (Meta | Easy Level)

Meta/Facebook (Easy Level)

Find all posts which were reacted to with a heart. For such posts output all columns from facebook_posts table.

```sql
-- solution
SELECT
    fp.post_id,
    fp.poster,
    fp.post_text,
    fp.post_keywords,
    fp.post_date
FROM
    facebook_reactions fr
        JOIN
    facebook_posts fp ON fr.post_id = fp.post_id
WHERE
    fr.reaction = 'heart';
```

Problem 42 (Microsoft | Medium Level)

Write a query that returns the company (customer id column) with highest number of users that use desktop only.

```sql
-- solution
WITH DesktopOnlyUser AS(
    SELECT
        customer_id,
        COUNT(CASE WHEN client_id = 'desktop' THEN user_id END) AS desktop_only_user_count,
        client_id,
        DENSE_RANK() OVER (ORDER BY COUNT(CASE WHEN client_id = 'desktop' THEN user_id END) DESC) AS rank
    FROM
        microsoft_fact_events
    GROUP BY customer_id, user_id, client_id
    )
SELECT
    customer_id,
    desktop_only_user_count
FROM
    DesktopOnlyUser
WHERE rank = 1;
```

Problem 43 (Apple | Hard Level)

Find the number of Apple product users and the number of total users with a device and group the counts by language. Assume Apple products are only MacBook-Pro, iPhone 5s, and iPad-air. Output the language along with the total number of Apple users and users with any device. Order your results based on the number of total users in descending order.

```sql
-- solution
WITH AppleUsers AS(
    SELECT
        pu.language,
        COUNT(DISTINCT pe.user_id) AS apple_user_count
    FROM
        playbook_users pu
            JOIN
        playbook_events pe ON pu.user_id = pe.user_id
    WHERE
        pe.device IN ('MacBook-Pro', 'iPhone 5s', 'iPad-air')
    GROUP BY pu.language
),
TotalUsers AS(
    SELECT
        pu.language,
        COUNT(DISTINCT pe.user_id) AS total_user_count
    FROM
        playbook_users pu
            JOIN
        playbook_events pe ON pu.user_id = pe.user_id
    WHERE
        pe.device IS NOT NULL
    GROUP BY pu.language
)
SELECT
    au.language,
    COALESCE(au.apple_user_count, 0) AS apple_user_count,
    tu.total_user_count
FROM
    AppleUsers au
        JOIN
    TotalUsers tu ON au.language = tu.language
ORDER BY total_user_count DESC;
```

Problem 44 (Amazon | Hard Level)

You are given the table with titles of recipes from a cookbook and their page numbers. You are asked to represent how the recipes will be distributed in the book. Produce a table consisting of three columns: left_page_number, left_title and right_title.

The k-th row (counting from 0), should contain the number and the title of the page with the number 2×k in the first and second columns respectively, and the title of the page with the number 2×k+1 in the third column.

```sql
-- solution
WITH LeftPage AS(
    SELECT
        page_number,
        title
    FROM
        cookbook_titles
    WHERE page_number % 2 = 0
),
RightPage AS(
    SELECT
        page_number,
        title
    FROM
        cookbook_titles
    WHERE page_number % 2 = 1
)
SELECT
    lp.page_number,
    lp.title AS left_page_title,
    rp.title AS right_page_title
FROM
    LeftPage lp
        LEFT JOIN
    RightPage rp ON lp.page_number + 1 = rp.page_number
ORDER BY lp.page_number;
```

Problem 45 (Visa | Medium Level)

Identify the top 3 areas with the highest customer density.

$$Customer\ density\ =\ (total\ number\ of\ unique\ customers\ in\ the\ area\ /\ area\ size)$$

Your output should include the area name and its calculated customer density.

```sql
SELECT TOP 3
    s.area_name,
    (COUNT(DISTINCT tr.customer_id) * 1.0 / s.area_size) AS customer_density
FROM
    transaction_records tr
        JOIN
    stores s ON tr.store_id = s.store_id
GROUP BY s.area_name, s.area_size
ORDER BY customer_density DESC;
```

Problem 46 (EY, TCS, Deloitte | Medium Level)

In a marathon, gun time is counted from the moment of the formal start of the race while net time is counted from the moment a runner crosses a starting line. Both variables are in seconds.
You are asked to check if the interval between the two times is different for male and female runners. First, calculate the average absolute difference between the gun time and net time. Group the results by available genders (male and female). Output the absolute difference between those two values.

```sql
-- solution
WITH AvgTimeDiff AS (
    SELECT
        'Male' AS gender,
        AVG(ABS(gun_time - net_time)) AS avg_time_diff
    FROM
        marathon_male
    UNION ALL
    SELECT
        'Female' AS gender,
        AVG(ABS(gun_time - net_time)) AS avg_time_diff
    FROM
        marathon_female
)
SELECT
    ABS(
        MAX(CASE WHEN gender = 'Male' THEN avg_time_diff END) -
        MAX(CASE WHEN gender = 'Female' THEN avg_time_diff END)
    ) AS absolute_diff
FROM
    AvgTimeDiff;
```

Problem 47 (Expedia, Airbnb, Tripadvisor | Medium Level)

Find the top two hotels with the most negative reviews.
Output the hotel name along with the corresponding number of negative reviews.
Negative reviews are all the reviews with text under negative review different than "No Negative".
Sort records based on the number of negative reviews in descending order.

```sql
-- solution
SELECT TOP 2
    hotel_name,
    COUNT(negative_review) AS count_of_negative_reviews
FROM
    airbnb_hotel_reviews
WHERE
    negative_review != 'No Negative'
GROUP BY hotel_name
ORDER BY count_of_negative_reviews DESC;
```

Problem 48 (Amazon, Doordash, | Bosch Medium Level)

Find all employees who have or had a job title that includes manager.
Output the first name along with the corresponding title.

```sql
-- solution
SELECT
    w.first_name,
    t.worker_title
FROM
    workers w
        JOIN
    titles t ON w.worker_id = t.worker_ref_id
WHERE
    LOWER(worker_title) LIKE '%manager%';
```

Problem 49 (Goldman Sachs | Medium Level)

You work for a multinational company that wants to calculate total sales across all their countries they do business in.

You have 2 tables, one is a record of sales for all countries and currencies the company deals with, and the other holds currency exchange rate information.

Calculate the total sales, per quarter, for the first 2 quarters in 2020, and report the sales in USD currency.

```sql
-- solution
SELECT
    DATEPART(QUARTER, sa.sales_date) AS sales_quarter,
    SUM(sa.sales_amount * er.exchange_rate) AS total_sales_usd
FROM
    sf_exchange_rate er
        JOIN
    sf_sales_amount sa ON er.source_currency = sa.currency
        AND CAST(sa.sales_date AS DATE) = CAST(er.date AS DATE)
WHERE
    sa.sales_date >= '2020-01-01' AND sa.sales_date <= '2020-07-01'
        AND er.target_currency = 'USD'
GROUP BY DATEPART(QUARTER, sa.sales_date);
```

Problem 50 (Meta | Hard Level)

Market penetration is an important metric for Spotify's growth in different regions. As part of the analytics team, calculate the active user penetration rate in specific countries. Active Users must meet these criteria:

Interacted with Spotify within the last 30 days (last_active_date >= 2024-01-01). At least 5 sessions. At least 10 listening hours.

$$Formula: Active\ User\ Penetration\ Rate = (Number\ of\ Active\ Spotify\ Users\ in\ the\ Country\ /\ Total\ Users\ in\ the\ Country)$$

Output: country, active_user_penetration_rate (rounded to 2 decimals).

```sql
-- solution
SELECT
    country,
    ROUND(
        CAST(
            SUM(
                CASE WHEN last_active_date >= '2024-01-01'
                      AND sessions >= 5
                      AND listening_hours >= 10
                    THEN 1
                    ELSE 0
                END
        ) AS FLOAT) / COUNT(*) * 100
    , 2) AS active_user_penetration_rate
FROM
    penetration_analysis
GROUP BY country;
```

Problem 51 (Amazon | Medium Level)

You have been asked to find the fifth highest salary without using TOP or LIMIT. Note: Duplicate salaries should not be removed.

```sql
-- solution
SELECT DISTINCT
    salary
FROM (
    SELECT
        salary,
        DENSE_RANK() OVER (ORDER BY salary DESC) AS rank
    FROM
        com_worker
) AS SalaryRanks
WHERE rank = 5;
```

Problem 52 (Tesla | Hard Level)

The company you are working for wants to anticipate their staffing needs by identifying their top two busiest times of the week. To find this, each day should be segmented into differents parts using following criteria:

- Morning: Before 12 p.m. (not inclusive)
- Early afternoon: 12 -15 p.m.
- Late afternoon: after 15 p.m. (not inclusive)

Your output should include the day and time of day combination for the two busiest times, i.e. the combinations with the most orders, along with the number of orders (e.g. top two results could be Friday Late afternoon with 12 orders and Sunday Morning with 10 orders).

The company has also requested that the day be displayed in text format (i.e. Monday).

```sql
--solution
WITH TimePeriodOrders AS (
    SELECT
        DATENAME(WEEKDAY, timestamp) AS day_of_week,  -- Get the day name (e.g., Monday, Tuesday)
        CASE
            WHEN DATEPART(HOUR, timestamp) < 12 THEN 'Morning'
            WHEN DATEPART(HOUR, timestamp) >= 12 AND DATEPART(HOUR, timestamp) < 15 THEN 'Early afternoon'
            ELSE 'Late afternoon'
        END AS time_of_day,
        COUNT(order_id) AS order_count
    FROM sales_log
    GROUP BY DATENAME(WEEKDAY, timestamp),
        CASE
            WHEN DATEPART(HOUR, timestamp) < 12 THEN 'Morning'
            WHEN DATEPART(HOUR, timestamp) >= 12 AND DATEPART(HOUR, timestamp) < 15 THEN 'Early afternoon'
            ELSE 'Late afternoon'
        END
),
RankedOrders AS (
    SELECT
        day_of_week,
        time_of_day,
        order_count,
        RANK() OVER (ORDER BY order_count DESC) AS rank
    FROM TimePeriodOrders
)
SELECT
    day_of_week,
    time_of_day,
    order_count
FROM RankedOrders
WHERE rank <= 2
ORDER BY rank, order_count DESC;
```

Problem 53 (DoorDash | Medium Level)

Calculate the average net earnings per order grouped by weekday (in text format, e.g., Monday) and hour from customer_placed_order_datetime.

The net earnings are computed as: order_total + tip_amount - discount_amount - refunded_amount. Round the result to 2 decimals.

```sql
-- solution
SELECT
    DATENAME(WEEKDAY, customer_placed_order_datetime) AS weekday,
    DATEPART(HOUR, customer_placed_order_datetime) AS order_hour,
    ROUND(AVG(order_total + tip_amount - discount_amount - refunded_amount), 2) AS average_earning
FROM
    doordash_delivery
GROUP BY
    DATENAME(WEEKDAY, customer_placed_order_datetime),
    DATEPART(HOUR, customer_placed_order_datetime)
ORDER BY
    weekday,
    order_hour;
```

Problem 54 (Meta | Hard level)

The sales department has given you the sales figures for the first two months of 2023. You've been tasked with determining the percentage of weekly sales on the first and last day of every week. Consider Sunday as last day of week and Monday as first day of week.

In your output, include the week number, percentage sales for the first day of the week, and percentage sales for the last day of the week. Both proportions should be rounded to the nearest whole number.

```sql
-- solution
WITH WeeklySales AS (
    SELECT
        DATEPART(WEEK, invoicedate) AS week_number,
        SUM(quantity * unitprice) AS total_weekly_sales
    FROM early_sales
    WHERE invoicedate BETWEEN '2023-01-01' AND '2023-02-28'
    GROUP BY DATEPART(WEEK, invoicedate)
),
SalesByDay AS (
    SELECT
        DATEPART(WEEK, invoicedate) AS week_number,
        DATEPART(WEEKDAY, invoicedate) AS day_of_week,
        SUM(quantity * unitprice) AS daily_sales
    FROM early_sales
    WHERE invoicedate BETWEEN '2023-01-01' AND '2023-02-28'
    GROUP BY DATEPART(WEEK, invoicedate), DATEPART(WEEKDAY, invoicedate)
),
FirstAndLastDaySales AS (
    SELECT
        s.week_number,
        COALESCE(SUM(CASE WHEN s.day_of_week = 2 THEN s.daily_sales END), 0) AS monday_sales,
        COALESCE(SUM(CASE WHEN s.day_of_week = 7 THEN s.daily_sales END), 0) AS sunday_sales
    FROM SalesByDay s
    GROUP BY s.week_number
)
SELECT
    ws.week_number,
    ROUND(100.0 * fl.monday_sales / ws.total_weekly_sales, 0) AS monday_sales_percentage,
    ROUND(100.0 * fl.sunday_sales / ws.total_weekly_sales, 0) AS sunday_sales_percentage
FROM WeeklySales ws
JOIN FirstAndLastDaySales fl ON ws.week_number = fl.week_number
ORDER BY ws.week_number;
```

Problem 55 (Amazon | Hard Level)

Find the 3-month rolling average of total revenue from purchases given a table with users, their purchase amount, and date purchased. Do not include returns which are represented by negative purchase values. Output the year-month (YYYY-MM) and 3-month rolling average of revenue, sorted from earliest month to latest month.

A 3-month rolling average is defined by calculating the average total revenue from all user purchases for the current month and previous two months.

The first two months will not be a true 3-month rolling average since we are not given data from last year. Assume each month has at least one purchase.

```sql
-- solution
WITH MonthlyRevenue AS (
    SELECT
        FORMAT(created_at, 'yyyy-MM') AS YearMonth,
        SUM(CASE WHEN purchase_amt > 0 THEN purchase_amt ELSE 0 END) AS TotalRevenue
    FROM amazon_purchases
    GROUP BY FORMAT(created_at, 'yyyy-MM')
),
RollingAverage AS (
    SELECT
        mr1.YearMonth,
        AVG(mr2.TotalRevenue * 1.0) AS RollingAvgRevenue
    FROM MonthlyRevenue mr1
    JOIN MonthlyRevenue mr2
        ON DATEDIFF(MONTH, mr2.YearMonth + '-01', mr1.YearMonth + '-01') BETWEEN 0 AND 2
    GROUP BY mr1.YearMonth
)
SELECT
    YearMonth,
    ROUND(RollingAvgRevenue, 2) AS RollingAvgRevenue
FROM RollingAverage
ORDER BY YearMonth;
```

Problem 56 (ESPN | Medium Level)

Find the quarterback who threw the longest throw in 2016. Output the quarterback name along with their corresponding longest throw.

The 'lg' column contains the longest completion by the quarterback.

```sql
-- solution
SELECT
    qb AS Quarterback,
    MAX(CAST(lg AS INT)) AS Longest_Throw
FROM
    qbstats_2015_2016
WHERE
    year = 2016
GROUP BY
    qb
ORDER BY
    Longest_Throw DESC;
```