101
```
SELECT t.username, t.activity, t.startDate, t.endDate
FROM (
  SELECT username, activity, startDate, endDate,
        ROW_NUMBER() OVER (PARTITION BY username ORDER BY endDate DESC) AS rn
  FROM your_table_name
) t
WHERE t.rn = 2 OR (t.rn = 1 AND NOT EXISTS (
  SELECT 1 FROM your_table_name WHERE username = t.username AND rn = 2
));
```

102
```
SELECT username, activity, startDate, endDate
FROM (
  SELECT username, activity, startDate, endDate,
        ROW_NUMBER() OVER (PARTITION BY username ORDER BY endDate DESC) AS rn,
        COUNT(*) OVER (PARTITION BY username) AS activityCount
  FROM your_table_name
) t
WHERE rn = 2 OR (rn = 1 AND activityCount = 1)
ORDER BY username;
```

103
```
SELECT Name
FROM STUDENTS
WHERE Marks > 75
ORDER BY RIGHT(Name, 3), ID ASC;
```

104
```
SELECT name
FROM Employee
WHERE salary > 2000 AND months < 10
ORDER BY employee_id ASC;
```

105
```
SELECT
  CASE
    WHEN A = B AND B = C THEN 'Equilateral'
    WHEN A = B OR B = C OR A = C THEN 'Isosceles'
    WHEN A + B <= C OR A + C <= B OR B + C <= A THEN 'Not A Triangle'
    ELSE 'Scalene'
  END AS triangle_type
FROM TRIANGLES;
```

106
```
SELECT CEIL(AVG(salary) - AVG(REPLACE(salary, '0', ''))) AS error
FROM EMPLOYEES;
```

107
```
SELECT MAX(months * salary) AS max_earnings, COUNT(*) AS employee_count
FROM Employee
WHERE months * salary = (
  SELECT MAX(months * salary)
  FROM Employee
);
```

108
```
SELECT CONCAT('There are a total of ', COUNT(*), ' ', LOWER(occupation), 's.')
FROM OCCUPATIONS
GROUP BY occupation
ORDER BY COUNT(*), occupation;
```

109
```
SELECT
  MAX(CASE WHEN Occupation = 'Doctor' THEN Name END) AS Doctor,
  MAX(CASE WHEN Occupation = 'Professor' THEN Name END) AS Professor,
  MAX(CASE WHEN Occupation = 'Singer' THEN Name END) AS Singer,
  MAX(CASE WHEN Occupation = 'Actor' THEN Name END) AS Actor
FROM OCCUPATIONS
GROUP BY Name
ORDER BY Name;
```

110
```
SELECT N,
  CASE
    WHEN P IS NULL THEN 'Root'
    WHEN N NOT IN (SELECT P FROM BST) THEN 'Leaf'
    ELSE 'Inner'
  END AS NodeType
FROM BST
```

```
ORDER BY N;
```

111

```
SELECT
  c.company_code,
  c.founder,
  COUNT(DISTINCT lm.lead_manager_code) AS total_lead_managers,
  COUNT(DISTINCT sm.senior_manager_code) AS total_senior_managers,
  COUNT(DISTINCT m.manager_code) AS total_managers,
  COUNT(DISTINCT e.employee_code) AS total_employees
FROM Company c
LEFT JOIN Lead_Manager lm ON c.company_code = lm.company_code
LEFT JOIN Senior_Manager sm ON lm.lead_manager_code = sm.lead_manager_code
AND c.company_code = sm.company_code
LEFT JOIN Manager m ON sm.senior_manager_code = m.senior_manager_code AND
lm.lead_manager_code = m.lead_manager_code AND c.company_code =
m.company_code
LEFT JOIN Employee e ON m.manager_code = e.manager_code AND
sm.senior_manager_code = e.senior_manager_code AND lm.lead_manager_code =
e.lead_manager_code AND c.company_code = e.company_code
GROUP BY c.company_code, c.founder
ORDER BY c.company_code ASC;
```

112

```
SELECT LISTAGG(L, '&') WITHIN GROUP (ORDER BY L) AS prime_numbers
FROM (
  SELECT LEVEL AS L
  FROM DUAL
  CONNECT BY LEVEL <= 1000
)
WHERE L <= (
  SELECT LEVEL
  FROM DUAL
  CONNECT BY LEVEL <= L
  GROUP BY LEVEL
  HAVING COUNT(CASE WHEN L / LEVEL = TRUNC(L / LEVEL) THEN 'Y' END) = 2
)
GROUP BY L;
```

113
```
SELECT REPLACE(SYS_CONNECT_BY_PATH(NULL, '* '), '/')
FROM DUAL
CONNECT BY LEVEL <= 20
START WITH LEVEL = 1;
```

114
```
SELECT REPLACE(SYS_CONNECT_BY_PATH('*', ' '), '/')
FROM DUAL
CONNECT BY LEVEL <= 20
START WITH LEVEL = 20;
```

115
```
SELECT Name
FROM STUDENTS
WHERE Marks > 75
ORDER BY RIGHT(Name, 3), ID ASC;
```

116
```
SELECT name
FROM Employee
ORDER BY name ASC;
```

117
```
SELECT name
FROM Employee
WHERE salary > 2000 AND months < 10
ORDER BY employee_id ASC;
```

118
```
SELECT
  CASE
    WHEN A + B <= C OR A + C <= B OR B + C <= A THEN 'Not A Triangle'
    WHEN A = B AND B = C THEN 'Equilateral'
    WHEN A = B OR B = C OR A = C THEN 'Isosceles'
    ELSE 'Scalene'
  END AS triangle_type
FROM TRIANGLES;
```

119

```
WITH yearly_spend AS (
  SELECT
    product_id,
    EXTRACT(YEAR FROM transaction_date) AS year,
    SUM(spend) AS total_spend
  FROM user_transactions
  GROUP BY product_id, EXTRACT(YEAR FROM transaction_date)
)
SELECT
  t1.year,
  t1.product_id,
  t1.total_spend AS curr_year_spend,
  t2.total_spend AS prev_year_spend,
  ROUND(((t1.total_spend - t2.total_spend) / t2.total_spend) * 100, 2) AS yoy_rate
FROM yearly_spend t1
LEFT JOIN yearly_spend t2 ON t1.product_id = t2.product_id AND t1.year = t2.year + 1
ORDER BY t1.product_id, t1.year;
```

120

```
SELECT
  item_type,
  SUM(square_footage) AS total_square_footage,
  COUNT(*) AS item_count
FROM inventory
GROUP BY item_type
ORDER BY item_type;
```

121

```
SELECT
  MONTH(event_date) AS month,
  COUNT(DISTINCT user_id) AS monthly_active_users
FROM user_actions ua
WHERE EXISTS (
  SELECT 1
  FROM user_actions
  WHERE user_id = ua.user_id
    AND MONTH(event_date) = MONTH(ua.event_date) - 1
    AND YEAR(event_date) = YEAR(ua.event_date)
    AND event_type IN ('sign-in', 'like', 'comment')
)
AND MONTH(event_date) = 7
AND YEAR(event_date) = 2022
GROUP BY month;
```

122

```
UPDATE advertiser
SET status = (
  SELECT CASE
    WHEN advertiser.status = 'NEW' AND daily_pay.user_id IS NULL THEN 'CHURN'
    WHEN advertiser.status = 'NEW' AND daily_pay.user_id IS NOT NULL THEN
'EXISTING'
    WHEN advertiser.status = 'EXISTING' AND daily_pay.user_id IS NULL THEN
'CHURN'
    WHEN advertiser.status = 'EXISTING' AND daily_pay.user_id IS NOT NULL THEN
'EXISTING'
    WHEN advertiser.status = 'CHURN' AND daily_pay.user_id IS NULL THEN 'CHURN'
    WHEN advertiser.status = 'CHURN' AND daily_pay.user_id IS NOT NULL THEN
'RESURRECT'
    WHEN advertiser.status = 'RESURRECT' AND daily_pay.user_id IS NULL THEN
'CHURN'
    WHEN advertiser.status = 'RESURRECT' AND daily_pay.user_id IS NOT NULL
THEN 'EXISTING'
  END
  FROM daily_pay
  WHERE advertiser.user_id = daily_pay.user_id
)
WHERE EXISTS (
  SELECT 1
  FROM daily_pay
  WHERE advertiser.user_id = daily_pay.user_id
);
```

124

```
SELECT FLOOR(SUM(DATEDIFF(stop_time, start_time))) AS total_uptime_days
FROM (
  SELECT server_id, MIN(status_time) AS start_time, MAX(status_time) AS stop_time
  FROM (
    SELECT server_id, status_time,
      ROW_NUMBER() OVER (PARTITION BY server_id ORDER BY status_time) AS rn,
      LAG(session_status) OVER (PARTITION BY server_id ORDER BY status_time) AS
prev_status
    FROM server_utilization
  ) t
  WHERE prev_status = 'start' AND session_status = 'stop'
  GROUP BY server_id
) t2;
```

125

```
SELECT COUNT(*) AS payment_count
FROM transactions t1
WHERE EXISTS (
    SELECT 1
    FROM transactions t2
    WHERE t2.merchant_id = t1.merchant_id
        AND t2.credit_card_id = t1.credit_card_id
        AND t2.amount = t1.amount
        AND t2.transaction_timestamp > t1.transaction_timestamp
        AND t2.transaction_timestamp <= DATE_ADD(t1.transaction_timestamp,
INTERVAL 10 MINUTE)
)
```

126

```
SELECT ROUND((COUNT(DISTINCT o.order_id) / COUNT(DISTINCT c.customer_id))
* 100, 2) AS bad_experience_pct
FROM orders o
JOIN customers c ON o.customer_id = c.customer_id
JOIN trips t ON o.trip_id = t.trip_id
WHERE c.signup_timestamp >= '2022-06-01' AND c.signup_timestamp < '2022-07-01'
  AND o.order_timestamp >= c.signup_timestamp AND o.order_timestamp <
DATE_ADD(c.signup_timestamp, INTERVAL 14 DAY)
  AND (
    o.status = 'completed incorrectly'
    OR o.status = 'never received'
    OR t.actual_delivery_timestamp > DATE_ADD(t.estimated_delivery_timestamp,
INTERVAL 30 MINUTE)
  )
```

127

```
SELECT gender, day, SUM(score_points) AS total
FROM Scores
GROUP BY gender, day
ORDER BY gender, day ASC;
```

128

```
WITH global_avg AS (
  SELECT AVG(duration) AS avg_duration
  FROM Calls
)
SELECT c.name AS country
FROM Country c
JOIN Person p ON c.country_code = SUBSTRING(p.phone_number, 1, 3)
JOIN Calls ca ON p.id = ca.caller_id OR p.id = ca.callee_id
GROUP BY c.name
HAVING AVG(ca.duration) > (SELECT avg_duration FROM global_avg);
```

129

```
WITH decompressed AS (
 SELECT num
 FROM Numbers
 WHERE frequency > 0
 CONNECT BY LEVEL <= frequency
)
SELECT ROUND(AVG(num), 1) AS median
```

```
FROM (
  SELECT num, ROW_NUMBER() OVER (ORDER BY num) AS row_num, COUNT(*)
OVER () AS total_count
  FROM decompressed
)
WHERE row_num = (total_count + 1) / 2 OR row_num = (total_count + 2) / 2;
```

130

```
WITH department_avg AS (
  SELECT department_id, AVG(amount) AS avg_salary
  FROM Salary
  GROUP BY department_id
), company_avg AS (
  SELECT AVG(amount) AS avg_salary
  FROM Salary
)
SELECT to_char(pay_date, 'YYYY-MM') AS pay_month, e.department_id,
  CASE
    WHEN department_avg.avg_salary > company_avg.avg_salary THEN 'higher'
    WHEN department_avg.avg_salary < company_avg.avg_salary THEN 'lower'
    ELSE 'same'
  END AS comparison
FROM Salary s
JOIN Employee e ON s.employee_id = e.employee_id
JOIN department_avg ON e.department_id = department_avg.department_id
CROSS JOIN company_avg
GROUP BY pay_month, e.department_id, department_avg.avg_salary,
company_avg.avg_salary;
```

131

```
WITH installs AS (
  SELECT event_date AS install_dt, COUNT(DISTINCT player_id) AS installs
  FROM Activity
  GROUP BY event_date
), day1_retention AS (
  SELECT install_dt, COUNT(DISTINCT player_id) AS day1_players
  FROM Activity a1
  WHERE EXISTS (
    SELECT 1
    FROM Activity a2
    WHERE a2.player_id = a1.player_id
      AND a2.event_date = DATE_ADD(a1.event_date, INTERVAL 1 DAY)
  )
  GROUP BY install_dt
)
SELECT i.install_dt, i.installs,
  ROUND((d.day1_players / i.installs), 2) AS Day1_retention
FROM installs i
LEFT JOIN day1_retention d ON i.install_dt = d.install_dt;
```

132

```
WITH player_scores AS (
  SELECT p.group_id, m.first_player AS player_id, SUM(m.first_score) AS total_score
  FROM Players p
  JOIN Matches m ON p.player_id = m.first_player
  GROUP BY p.group_id, m.first_player
  UNION ALL
  SELECT p.group_id, m.second_player AS player_id, SUM(m.second_score) AS
total_score
  FROM Players p
  JOIN Matches m ON p.player_id = m.second_player
  GROUP BY p.group_id, m.second_player
), group_ranks AS (
  SELECT group_id, player_id, total_score,
        ROW_NUMBER() OVER (PARTITION BY group_id ORDER BY total_score
DESC, player_id ASC) AS rank
  FROM player_scores
)
SELECT group_id, player_id
FROM group_ranks
WHERE rank = 1;
```

137

```
SELECT CEIL(AVG(salary)) - AVG(REPLACE(salary, '0', '')::integer) AS error
FROM EMPLOYEES;
```

138
```
SELECT MAX(months * salary) AS max_earnings, COUNT(*) AS num_employees
FROM Employee
WHERE months * salary = (SELECT MAX(months * salary) FROM Employee);
```

139
```
SELECT CONCAT(name, '(', LEFT(occupation, 1), ')') AS full_name
FROM OCCUPATIONS
ORDER BY name;
```

140
```
SELECT
  GROUP_CONCAT(DISTINCT CASE WHEN Occupation = 'Doctor' THEN Name ELSE
NULL END ORDER BY Name) AS Doctor,
  GROUP_CONCAT(DISTINCT CASE WHEN Occupation = 'Professor' THEN Name
ELSE NULL END ORDER BY Name) AS Professor,
  GROUP_CONCAT(DISTINCT CASE WHEN Occupation = 'Singer' THEN Name ELSE
NULL END ORDER BY Name) AS Singer,
  GROUP_CONCAT(DISTINCT CASE WHEN Occupation = 'Actor' THEN Name ELSE
NULL END ORDER BY Name) AS Actor
FROM OCCUPATIONS;
```

141
```
SELECT
  N,
  CASE
    WHEN P IS NULL THEN 'Root'
    WHEN N NOT IN (SELECT DISTINCT P FROM BST) THEN 'Leaf'
    ELSE 'Inner'
  END AS NodeType
FROM BST
ORDER BY N;
```

142
```
SELECT
```

```
  C.company_code,
  C.founder,
  COUNT(DISTINCT LM.lead_manager_code) AS total_lead_managers,
  COUNT(DISTINCT SM.senior_manager_code) AS total_senior_managers,
  COUNT(DISTINCT M.manager_code) AS total_managers,
  COUNT(DISTINCT E.employee_code) AS total_employees
FROM
  Company AS C
  LEFT JOIN Lead_Manager AS LM ON C.company_code = LM.company_code
  LEFT JOIN Senior_Manager AS SM ON LM.lead_manager_code =
SM.lead_manager_code
  LEFT JOIN Manager AS M ON SM.senior_manager_code = M.senior_manager_code
  LEFT JOIN Employee AS E ON M.manager_code = E.manager_code
GROUP BY
  C.company_code,
  C.founder
ORDER BY
  C.company_code ASC;
```

143

```
SELECT X, Y
FROM Functions
GROUP BY X, Y
HAVING COUNT(*) > 1 AND X <= Y
ORDER BY X ASC;
```

144

```
SELECT s.Name
FROM Students s
INNER JOIN Friends f ON s.ID = f.ID
INNER JOIN Packages p1 ON s.ID = p1.ID
INNER JOIN Packages p2 ON f.Friend_ID = p2.ID
WHERE p2.Salary > p1.Salary
ORDER BY p2.Salary;
```

145

```
SELECT h.hacker_id, h.name
```

```
FROM Hackers h
JOIN Submissions s ON h.hacker_id = s.hacker_id
JOIN Challenges c ON s.challenge_id = c.challenge_id AND s.score = c.difficulty_level
GROUP BY h.hacker_id, h.name
HAVING COUNT(DISTINCT c.challenge_id) > 1
ORDER BY COUNT(DISTINCT c.challenge_id) DESC, h.hacker_id ASC;
```

148

```
SELECT COUNT(*) AS unique_relationships
FROM (
   SELECT payer_id, recipient_id
   FROM payments
   GROUP BY payer_id, recipient_id
   HAVING COUNT(*) = 2
) AS subquery;
```

149

```
SELECT COUNT(*) AS users
FROM (
   SELECT user_id
   FROM (
      SELECT user_id, spend, transaction_date,
           ROW_NUMBER() OVER (PARTITION BY user_id ORDER BY
transaction_date) AS rn
      FROM user_transactions
   ) AS subquery
   WHERE rn = 1 AND spend >= 50
) AS result;
```

150

```
SELECT
   DATE(measurement_time) AS measurement_day,
   SUM(CASE WHEN measurement_id % 2 = 1 THEN measurement_value ELSE 0
END) AS odd_sum,
   SUM(CASE WHEN measurement_id % 2 = 0 THEN measurement_value ELSE 0
END) AS even_sum
FROM measurements
```

```
GROUP BY measurement_day;
```