# The Best Medium-Hard Data Analyst SQL Interview Questions

By Zachary Thomas ([zthomas.nc@gmail.com](mailto:zthomas.nc@gmail.com), [Twitter](https://twitter.com/zach_i_thomas), [LinkedIn](https://www.linkedin.com/in/thomaszi/))

**Tip:** See the Table of Contents (document outline) by hovering over the vertical line on the right side of the page

**Update:** Thanks everyone for the support and feedback! See this discussion on this post on [Hacker News](https://news.ycombinator.com/item?id=23053981), [Linkedin](https://www.linkedin.com/posts/thomaszi_the-best-medium-hard-data-analyst-sql-interview-activity-6662828341382516736-L5If), Eric Weber’s [Linkedin post](https://www.linkedin.com/posts/eric-weber-060397b7_datascience-analytics-sql-activity-6663082042990952449-wuOK)

## Background & Motivation

The first 70% of SQL is pretty straightforward but the remaining 30% can be pretty tricky.

Between the fall of 2015 and the summer of 2019 I interviewed for data analyst and data scientists positions four separate times, getting to onsite interviews at over a dozen companies. After an interview in 2017 went poorly — mostly due to me floundering at the more difficult SQL questions they asked me — I started putting together a study guide of medium and hard SQL questions to better prepare and found it particularly useful during my 2019 interview cycle. Over the past year I have shared that guide with a couple of friends, and with the extra time on my hands due to the coronavirus pandemic, I have polished it up into this doc.

There are plenty of great beginner SQL guides out there. My favorites are Codecademy’s [interactive SQL courses](https://www.codecademy.com/learn/learn-sql) and Zi Chong Kao’s [Select Star SQL](https://selectstarsql.com/). However, like I told a friend, while the first 70% of SQL is pretty straightforward, the remaining 30% can be pretty tricky. Data analyst and data scientist interview questions at technology companies often pull from that 30%.

Strangely, I have never really found a comprehensive source online for those medium-hard SQL questions, which is why I put together this guide.

Working through this guide should improve your performance on data analyst interviews. It should also make you better at your current and future job positions. Personally, I find some of the SQL patterns found in this doc useful for ETLs powering reporting tools featuring trends over time.

To be clear, data analyst and data scientist interviews consist of more than SQL questions. Other common topics include explaining past projects, A/B testing (I like [Udacity’s course](https://www.udacity.com/course/ab-testing--ud257) on the subject), metric development and open-ended analytical problems. This [Quora answer](https://qr.ae/pNrdGV) has Facebook’s product analyst interview guide circa 2017, which discusses this topic in more depth. That said, if improving your SQL skills can make your interviews less stressful than they already are, it could very well be worth your time.

In the future, I may transition this doc to a website like [Select Star SQL](https://selectstarsql.com/) with an embedded SQL editor so that readers can write SQL statements to questions and get real-time feedback on their code. Another option could be adding these questions as problems on Leetcode. For the time being though I just wanted to publish this doc so that people could find it useful now.

**I would love to get your feedback on this doc. Please drop a note if you find this useful, have improvements/corrections, or encounter other good resources for medium/hard difficulty SQL questions.**

## Assumptions & How to use this guide

**Assumptions about SQL proficiency:**This guide assumes you have a working knowledge of SQL. You probably use it frequently at work already but want to sharpen your skills on topics like self-joins and window functions.

**How to use this guide:** Since interviews usually utilize a whiteboard or a virtual (non-compiling) notepad, my recommendation is to get out a pencil and paper and write out your solutions to each part of the problem, and once complete compare your answers to the answer key. Or, complete these with a friend who can act as the interviewer!

* Small SQL syntax errors aren’t a big deal during whiteboard/notepad interviews. However, they can be distracting to the interviewer, so ideally practice reducing these so your logic shines through in the interview.
* The answers I provide may not be the only way to successfully solve the question. Feel free to message with additional solutions and I can add them to this guide!

## Tips on solving difficult SQL interview questions

This advice mirrors typical code interview advice ...

1. Listen carefully to problem description, repeat back the crux of the problem to the interviewer
2. Spell out an edge case to demonstrate you actually understand problem (i.e. a row that *wouldn’t* be included in the output of the SQL you are about to sketch out)
3. (If the problem involves a self-join) For your own benefit sketch out what the self-join will look like — this will typically be at least three columns: a column of interest from the main table, the column to join from the main table, and the column to join from the secondary table
   1. Or, as you get more used to self-join problems, you can explain this step verbally
4. Start writing SQL — err towards writing SQL versus trying to perfectly understand the problem. Verbalize your assumptions as you go so your interviewer can correct you if you go astray.

## Acknowledgments and Additional Resources

Some of the problems listed here are adapted from old Periscope blog posts (mostly written around 2014 by [Sean Cook](https://www.linkedin.com/in/seangcook/), although his authorship seems to have been removed from the posts following SiSense's [merger with](https://www.sisense.com/blog/sisense-and-periscope-data-merge-2/) Periscope) or discussions from Stack Overflow; I've noted them at the start of questions as appropriate.

[Select Star SQL](https://selectstarsql.com) has good [challenge questions](https://selectstarsql.com/questions.html#challenge_questions) that are complementary to the questions in this doc.

Please note that these questions are not literal copies of SQL interview questions I have encountered while interviewing nor were they interview questions used at a company I have worked at or work at.

# Self-Join Practice Problems

## #1: MoM Percent Change

**Context:** Oftentimes it's useful to know how much a key metric, such as monthly active users, changes between months. Say we have a table logins in the form:

| user\_id | date       |  
|---------|------------|  
| 1       | 2018-07-01 |  
| 234     | 2018-07-02 |  
| 3       | 2018-07-02 |  
| 1       | 2018-07-02 |  
| ...     | ...        |  
| 234     | 2018-10-04 |

**Task**: Find the month-over-month percentage change for monthly active users (MAU).

WITH CTE AS MAU\_table

(SELECT COUNT(user\_id) AS MAU, CONCAT(YEAR(date), ‘-‘, MONTH(date)) AS dt

FROM logins

GROUP BY YEAR(date), MONTH(date))

SELECT MAU - LAG(MAU, 1) OVER (ORDER BY dt)

FROM MAU\_table

***Solution:***

*(This solution, like other solution code blocks you will see in this doc, contains comments about SQL syntax that may differ between flavors of SQL or other comments about the solutions as listed)*

WITH mau AS   
(  
  SELECT   
   /\*   
    \* Typically, interviewers allow you to write psuedocode for date functions   
    \* i.e. will NOT be checking if you have memorized date functions.   
    \* Just explain what your function does as you whiteboard   
    \*  
    \* DATE\_TRUNC() is available in Postgres, but other SQL date functions or   
    \* combinations of date functions can give you a identical results     
    \* See <https://www.postgresql.org/docs/9.0/functions-datetime.html#FUNCTIONS-DATETIME-TRUNC>  
    \*/   
    DATE\_TRUNC('month', date) month\_timestamp,  
    COUNT(DISTINCT user\_id) mau  
  FROM   
    logins   
  GROUP BY   
    DATE\_TRUNC('month', date)  
  )  
   
 SELECT   
    /\*  
    \* You don't literally need to include the previous month in this SELECT statement.   
    \*   
    \* However, as mentioned in the "Tips" section of this guide, it can be helpful   
    \* to at least sketch out self-joins to avoid getting confused which table   
    \* represents the prior month vs current month, etc.   
    \*/   
    a.month\_timestamp previous\_month,   
    a.mau previous\_mau,   
    b.month\_timestamp current\_month,   
    b.mau current\_mau,   
    ROUND(100.0\*(b.mau - a.mau)/a.mau,2) AS percent\_change   
 FROM  
    mau a   
 JOIN   
    /\*  
    \* Could also have done `ON b.month\_timestamp = a.month\_timestamp + interval '1 month'`   
    \*/  
    mau b ON a.month\_timestamp = b.month\_timestamp - interval '1 month' 

## #2: Tree Structure Labeling

**Context:** Say you have a table tree with a column of nodes and a column corresponding parent nodes

node   parent  
1       2  
2       5  
3       5  
4       3  
5       NULL

**Task:** Write SQL such that we label each node as a “leaf”, “inner” or “Root” node, such that for the nodes above we get:

node    label    
1       Leaf  
2       Inner  
3       Inner  
4       Leaf  
5       Root

A solution which works for the above example will receive full credit, although you can receive extra credit for providing a solution that is generalizable to a tree of any depth (not just depth = 2, as is the case in the example above).

(Side note: [this link](http://ceadserv1.nku.edu/longa//classes/mat385_resources/docs/trees.html) has more details on Tree data structure terminology. Not needed to solve the problem though!)

***Solution:***

**Note:** This solution works for the example above with tree depth = 2, but is not generalizable beyond that.

WITH join\_table AS   
(  
SELECT   
    a.node a\_node,  
    a.parent a\_parent,  
    b.node b\_node,   
    b.parent b\_parent  
 FROM  
    tree a   
 LEFT JOIN   
    tree b ON a.parent = b.node   
 )  
   
 SELECT   
    a\_node node,   
    CASE   
        WHEN b\_node IS NULL and b\_parent IS NULL THEN 'Root'  
        WHEN b\_node IS NOT NULL and b\_parent IS NOT NULL THEN 'Leaf'          
        ELSE 'Inner'   
    END AS label   
 FROM   
    join\_table 

An alternate solution, that is generalizable to any tree depth:

**Acknowledgement:** this more generalizable solution was contributed by Fabian Hofmann on 5/2/20. Thank, FH!

WITH join\_table AS  
(  
    SELECT   
        cur.node,   
        cur.parent,   
        COUNT(next.node) AS num\_children  
    FROM   
        tree cur  
    LEFT JOIN   
        tree next ON (next.parent = cur.node)  
    GROUP BY   
        cur.node,   
        cur.parent  
)  
  
SELECT  
    node,  
    CASE  
        WHEN parent IS NULL THEN "Root"  
        WHEN num\_children = 0 THEN "Leaf"  
        ELSE "Inner"  
    END AS label  
FROM   
    join\_table

An alternate solution, without explicit joins:

**Acknowledgement:** William Chargin on 5/2/20 noted that WHERE parent IS NOT NULL  is needed to make this solution return Leaf instead of NULL. Thanks, WC!

SELECT   
    node,  
    CASE   
        WHEN parent IS NULL THEN 'Root'  
        WHEN node NOT IN   
            (SELECT parent FROM tree WHERE parent IS NOT NULL) THEN 'Leaf'  
        WHEN node IN (SELECT parent FROM tree) AND parent IS NOT NULL THEN 'Inner'  
    END AS label   
 from   
    tree

## #3: Retained Users Per Month (multi-part)

**Acknowledgement:**this problem is adapted from SiSense’s [“Using Self Joins to Calculate Your Retention, Churn, and Reactivation Metrics”](https://www.sisense.com/blog/use-self-joins-to-calculate-your-retention-churn-and-reactivation-metrics/) blog post

### Part 1:

**Context:** Say we have login data in the table logins:

| user\_id | date       |  
|---------|------------|  
| 1       | 2018-07-01 |  
| 234     | 2018-07-02 |  
| 3       | 2018-07-02 |  
| 1       | 2018-07-02 |  
| ...     | ...        |  
| 234     | 2018-10-04 |

**Task:** Write a query that gets the number of retained users per month. In this case, retention for a given month is defined as the number of users who logged in that month who also logged in the immediately previous month.

***Solution:***

SELECT   
    DATE\_TRUNC('month', a.date) month\_timestamp,   
    COUNT(DISTINCT a.user\_id) retained\_users   
 FROM   
    logins a   
 JOIN   
    logins b ON a.user\_id = b.user\_id   
        AND DATE\_TRUNC('month', a.date) = DATE\_TRUNC('month', b.date) +   
                                             interval '1 month'  
 GROUP BY   
    date\_trunc('month', a.date)

**Acknowledgement:** Tom Moertel pointed out de-duping user-login pairs before the self-join would make the solution more efficient and contributed the alternate solution below. Thanks, TM!

**Note:** De-duping logins would also make the given solutions to Parts 2 and 3 of this problem more efficient as well.

Alternate solution:

WITH DistinctMonthlyUsers AS (  
  /\*  
  \* For each month, compute the \*set\* of users having logins.  
  \*/  
    SELECT DISTINCT  
      DATE\_TRUNC('MONTH', date) AS month\_timestamp,  
      user\_id  
    FROM logins  
  )  
  
SELECT  
  CurrentMonth.month\_timestamp month\_timestamp,  
  COUNT(PriorMonth.user\_id) AS retained\_user\_count  
FROM   
    DistinctMonthlyUsers AS CurrentMonth  
LEFT JOIN   
    DistinctMonthlyUsers AS PriorMonth  
  ON  
    CurrentMonth.month\_timestamp = PriorMonth.month\_timestamp + INTERVAL '1 MONTH'  
    AND   
    CurrentMonth.user\_id = PriorMonth.user\_id

### Part 2:

**Task:** Now we’ll take retention and turn it on its head: Write a query to find many users last month *did not* come back this month. i.e. the number of churned users.

***Solution:***

SELECT   
    DATE\_TRUNC('month', a.date) month\_timestamp,   
    COUNT(DISTINCT b.user\_id) churned\_users   
FROM   
    logins a   
FULL OUTER JOIN   
    logins b ON a.user\_id = b.user\_id   
        AND DATE\_TRUNC('month', a.date) = DATE\_TRUNC('month', b.date) +   
                                         interval '1 month'  
WHERE   
    a.user\_id IS NULL   
GROUP BY   
    DATE\_TRUNC('month', a.date)

Note that there are solutions to this problem that can use LEFT or RIGHT joins.

### Part 3:

**Context**: You now want to see the number of active users this month *who have been reactivated* — in other words, users who have churned but this month they became active again. Keep in mind a user can reactivate after churning *before* the previous month. An example of this could be a user active in February (appears in logins), no activity in March and April, but then active again in May (appears in logins), so they count as a reactivated user for May .

**Task:** Create a table that contains the number of reactivated users per month.

***Solution:***

     SELECT   
        DATE\_TRUNC('month', a.date) month\_timestamp,  
        COUNT(DISTINCT a.user\_id) reactivated\_users,  
        /\*   
        \* At least in the flavors of SQL I have used, you don't need to   
        \* include the columns used in HAVING in the SELECT statement.  
        \* I have written them out for clarity here.    
        \*/   
        MAX(DATE\_TRUNC('month', b.date)) most\_recent\_active\_previously   
     FROM   
        logins a  
     JOIN  
        logins b ON a.user\_id = b.user\_id   
                AND   
                DATE\_TRUNC('month', a.date) > DATE\_TRUNC('month', b.date)  
     GROUP BY   
        DATE\_TRUNC('month', a.date)  
     HAVING   
        month\_timestamp > most\_recent\_active\_previously + interval '1 month'

## #4: Cumulative Sums

**Acknowledgement:** This problem was inspired by Sisense’s [“Cash Flow modeling in SQL”](https://www.sisense.com/blog/cash-flow-modeling-in-sql/) blog post

**Context:** Say we have a table transactions in the form:

| date       | cash\_flow |  
|------------|-----------|  
| 2018-01-01 | -1000     |  
| 2018-01-02 | -100      |  
| 2018-01-03 | 50        |  
| ...        | ...       |

Where cash\_flow is the revenues minus costs for each day.

**Task:** Write a query to get *cumulative* cash flow for each day such that we end up with a table in the form below:

| date       | cumulative\_cf |  
|------------|---------------|  
| 2018-01-01 | -1000         |  
| 2018-01-02 | -1100         |  
| 2018-01-03 | -1050         |  
| ...        | ...           |

***Solution:***

SELECT   
    a.date date,   
    SUM(b.cash\_flow) as cumulative\_cf   
FROM  
    transactions a  
JOIN b   
    transactions b ON a.date >= b.date   
GROUP BY   
    a.date   
ORDER BY   
    date ASC

Alternate solution using a window function (more efficient!):

SELECT   
    date,   
    SUM(cash\_flow) OVER (ORDER BY date ASC) as cumulative\_cf   
FROM  
    transactions   
ORDER BY   
    date ASC

## #5: Rolling Averages

**Acknowledgement:** This problem is adapted from Sisense’s [“Rolling Averages in MySQL and SQL Server”](https://www.sisense.com/blog/rolling-average/) blog post

**Note:** there are different ways to compute rolling/moving averages. Here we'll use a preceding average which means that the metric for the 7th day of the month would be the average of the preceding 6 days and that day itself.

**Context**: Say we have table signups in the form:

| date       | sign\_ups |  
|------------|----------|  
| 2018-01-01 | 10       |  
| 2018-01-02 | 20       |  
| 2018-01-03 | 50       |  
| ...        | ...      |  
| 2018-10-01 | 35       |

**Task**: Write a query to get 7-day rolling (preceding) average of daily sign ups.

***Solution:***

SELECT   
  a.date,   
  AVG(b.sign\_ups) average\_sign\_ups   
FROM   
  signups a   
JOIN   
  signups b ON a.date <= b.date + interval '6 days' AND a.date >= b.date  
GROUP BY   
  a.date

## #6: Multiple Join Conditions

**Acknowledgement:** This problem was inspired by Sisense’s [“Analyzing Your Email with SQL”](https://www.sisense.com/blog/analyzing-your-email-with-sql/) blog post

**Context:** Say we have a table emails that includes emails sent to and from [zach@g.com](mailto:zach@g.com):

| id | subject  | from         | to           | timestamp           |  
|----|----------|--------------|--------------|---------------------|  
| 1  | Yosemite | zach@g.com   | thomas@g.com | 2018-01-02 12:45:03 |  
| 2  | Big Sur  | sarah@g.com  | thomas@g.com | 2018-01-02 16:30:01 |  
| 3  | Yosemite | thomas@g.com | zach@g.com   | 2018-01-02 16:35:04 |  
| 4  | Running  | jill@g.com   | zach@g.com   | 2018-01-03 08:12:45 |  
| 5  | Yosemite | zach@g.com   | thomas@g.com | 2018-01-03 14:02:01 |  
| 6  | Yosemite | thomas@g.com | zach@g.com   | 2018-01-03 15:01:05 |  
| .. | ..       | ..           | ..           | ..                  |

**Task:**Write a query to get the response time per email (id) sent to zach@g.com . Do not include ids that did not receive a response from [zach@g.com](mailto:zach@g.com). Assume each email thread has a unique subject. Keep in mind a thread may have multiple responses back-and-forth between [zach@g.com](mailto:zach@g.com) and another email address.

***Solution:***

SELECT   
    a.id,   
    MIN(b.timestamp) - a.timestamp as time\_to\_respond   
FROM   
    emails a   
JOIN  
    emails b   
        ON   
            b.subject = a.subject   
        AND   
            a.to = b.from  
        AND   
            a.from = b.to   
        AND   
            a.timestamp < b.timestamp   
 WHERE   
    a.to = 'zach@g.com'   
 GROUP BY   
    a.id

# Window Function Practice Problems

## #1: Get the ID with the highest value

**Context:** Say we have a table salaries with data on employee salary and department in the following format:

  depname  | empno | salary |       
-----------+-------+--------+  
 develop   |    11 |   5200 |   
 develop   |     7 |   4200 |   
 develop   |     9 |   4500 |   
 develop   |     8 |   6000 |   
 develop   |    10 |   5200 |   
 personnel |     5 |   3500 |   
 personnel |     2 |   3900 |   
 sales     |     3 |   4800 |   
 sales     |     1 |   5000 |   
 sales     |     4 |   4800 |

**Task**: Write a query to get the empno with the highest salary. Make sure your solution can handle ties!

***Solution:***

WITH max\_salary AS (  
    SELECT   
        MAX(salary) max\_salary  
    FROM   
        salaries  
    )  
SELECT   
    s.empno  
FROM   
    salaries s  
JOIN   
    max\_salary ms ON s.salary = ms.max\_salary

Alternate solution using RANK():

WITH sal\_rank AS   
  (SELECT   
    empno,   
    RANK() OVER(ORDER BY salary DESC) rnk  
  FROM   
    salaries)  
SELECT   
  empno  
FROM  
  sal\_rank  
WHERE   
  rnk = 1;

## #2: Average and rank with a window function (multi-part)

### Part 1:

**Context**: Say we have a table salaries in the format:

  depname  | empno | salary |       
-----------+-------+--------+  
 develop   |    11 |   5200 |   
 develop   |     7 |   4200 |   
 develop   |     9 |   4500 |   
 develop   |     8 |   6000 |   
 develop   |    10 |   5200 |   
 personnel |     5 |   3500 |   
 personnel |     2 |   3900 |   
 sales     |     3 |   4800 |   
 sales     |     1 |   5000 |   
 sales     |     4 |   4800 |

**Task:** Write a query that returns the same table, but with a new column that has average salary per depname. We would expect a table in the form:

  depname  | empno | salary | avg\_salary |       
-----------+-------+--------+------------+  
 develop   |    11 |   5200 |       5020 |  
 develop   |     7 |   4200 |       5020 |   
 develop   |     9 |   4500 |       5020 |  
 develop   |     8 |   6000 |       5020 |   
 develop   |    10 |   5200 |       5020 |   
 personnel |     5 |   3500 |       3700 |  
 personnel |     2 |   3900 |       3700 |  
 sales     |     3 |   4800 |       4867 |   
 sales     |     1 |   5000 |       4867 |   
 sales     |     4 |   4800 |       4867 |

***Solution:***

SELECT   
    \*,   
    /\*  
    \* AVG() is a Postgres command, but other SQL flavors like BigQuery use   
    \* AVERAGE()  
    \*/   
    ROUND(AVG(salary),0) OVER (PARTITION BY depname) avg\_salary  
FROM  
    salaries

### Part 2:

**Task:** Write a query that adds a column with the rank of each employee based on their salary within their department, where the employee with the highest salary gets the rank of 1. We would expect a table in the form:

  depname  | empno | salary | salary\_rank |       
-----------+-------+--------+-------------+  
 develop   |    11 |   5200 |           2 |  
 develop   |     7 |   4200 |           5 |   
 develop   |     9 |   4500 |           4 |  
 develop   |     8 |   6000 |           1 |   
 develop   |    10 |   5200 |           2 |   
 personnel |     5 |   3500 |           2 |  
 personnel |     2 |   3900 |           1 |  
 sales     |     3 |   4800 |           2 |   
 sales     |     1 |   5000 |           1 |   
 sales     |     4 |   4800 |           2 |

***Solution:***

SELECT   
    \*,   
    RANK() OVER(PARTITION BY depname ORDER BY salary DESC) salary\_rank  
 FROM    
    salaries

# Other Medium/Hard SQL Practice Problems

## #1: Histograms

**Context:** Say we have a table sessions where each row is a video streaming session with length in seconds:

| session\_id | length\_seconds |  
|------------|----------------|  
| 1          | 23             |  
| 2          | 453            |  
| 3          | 27             |  
| ..         | ..             |

**Task:** Write a query to count the number of sessions that fall into bands of size 5, i.e. for the above snippet, produce something akin to:

| bucket  | count |  
|---------|-------|  
| 20-25   | 2     |  
| 450-455 | 1     |

Get complete credit for the proper string labels (“5-10”, etc.) but near complete credit for something that is communicable as the bin.

***Solution:***

WITH bin\_label AS   
(SELECT   
    session\_id,   
    FLOOR(length\_seconds/5) as bin\_label   
 FROM  
    sessions   
 )  
 SELECT   
    CONCATENTATE(STR(bin\_label\*5), '-', STR(bin\_label\*5+5)) bucket,   
    COUNT(DISTINCT session\_id) count   
 GROUP BY   
    bin\_label  
 ORDER BY   
    bin\_label ASC

## #2: CROSS JOIN (multi-part)

### Part 1:

**Context:** Say we have a table state\_streams where each row is a state and the total number of hours of streaming from a video hosting service:

| state | total\_streams |  
|-------|---------------|  
| NC    | 34569         |  
| SC    | 33999         |  
| CA    | 98324         |  
| MA    | 19345         |  
| ..    | ..            |

(In reality these kinds of aggregate tables would normally have a date column, but we’ll exclude that component in this problem)

**Task:** Write a query to get the pairs of states with total streaming amounts within 1000 of each other. For the snippet above, we would want to see something like:

| state\_a | state\_b |  
|---------|---------|  
| NC      | SC      |  
| SC      | NC      |

***Solution:***

SELECT  
    a.state as state\_a,   
    b.state as state\_b   
 FROM     
    state\_streams a  
 CROSS JOIN   
    state\_streams b   
 WHERE   
    ABS(a.total\_streams - b.total\_streams) < 1000  
    AND   
    a.state <> b.state

FYI, CROSS JOIN s can also be written without explicitly specifying a join:

SELECT  
    a.state as state\_a,   
    b.state as state\_b   
 FROM     
    state\_streams a, state\_streams b   
 WHERE   
    ABS(a.total\_streams - b.total\_streams) < 1000  
    AND   
    a.state <> b.state

### Part 2:

**Note:** This question is considered more of a bonus problem than an actual SQL pattern. Feel free to skip it!

**Task:** How could you modify the SQL from the solution to Part 1 of this question so that duplicates are removed? For example, if we used the sample table from Part 1, the pair NC and SC should only appear in one row instead of two.

***Solution:***

SELECT  
    a.state as state\_a,   
    b.state as state\_b   
 FROM     
    state\_streams a, state\_streams b   
 WHERE   
    ABS(a.total\_streams - b.total\_streams) < 1000  
    AND   
    a.state > b.state

## #3: Advancing Counting

**Acknowledgement:** This question is adapted from [this Stack Overflow question](https://stackoverflow.com/questions/54488894/using-case-to-properly-count-items-with-if-else-logic-in-sql) by me (zthomas.nc)

**Note:** this question is probably more complex than the kind you would encounter in an interview. Consider it a challenge problem, or feel free to skip it!

**Context:** Say I have a table table in the following form, where a user can be mapped to multiple values of class:

| user | class |  
|------|-------|  
| 1    | a     |  
| 1    | b     |  
| 1    | b     |  
| 2    | b     |  
| 3    | a     |

**Task:** Assume there are only two possible values for class. Write a query to count the number of users in each class such that any user who has label a and b gets sorted into b, any user with just a gets sorted into a and any user with just b gets into b.

For table that would result in the following table:

| class | count |  
|-------|-------|  
| a     | 1     |  
 | b     | 2     |

***Solution:***

WITH usr\_b\_sum AS   
(  
    SELECT   
        user,   
        SUM(CASE WHEN class = 'b' THEN 1 ELSE 0 END) num\_b  
    FROM   
        table  
    GROUP BY   
        user  
),   
  
usr\_class\_label AS   
(  
    SELECT   
        user,   
        CASE WHEN num\_b > 0 THEN 'b' ELSE 'a' END class   
    FROM   
        usr\_b\_sum  
)  
  
SELECT   
    class,   
    COUNT(DISTINCT user) count   
FROM  
    usr\_class\_label  
GROUP BY   
    class   
ORDER BY   
    class ASC

Alternate solution: Using SELECTs in the SELECT statement and UNION:

SELECT   
    "a" class,  
    COUNT(DISTINCT user\_id) -   
        (SELECT COUNT(DISTINCT user\_id) FROM table WHERE class = 'b') count   
UNION  
SELECT   
    "b" class,  
    (SELECT COUNT(DISTINCT user\_id) FROM table WHERE class = 'b') count

Alternate solution: Since the problem as stated didn’t ask for generalizable solution, you can leverage the fact that b > a to produce this straightforward solution:

**Acknowledgement**: Thanks to Karan Gadiya for contributing this solution. Thanks, KG!

WITH max\_class AS (  
    SELECT  
        user,   
        MAX(class) as class   
    FROM   
        table   
    GROUP BY   
        user  
)  
  
SELECT   
    class,   
    COUNT(user)  
FROM  
    max\_class  
GROUP BY   
    class