**What I realized after solving 100 leetcode SQL questions…**

There’re only 3 types of questions. They are…

[[Linda Chen](https://linnndachen.medium.com/?source=post_page-----df90bfe4c9c-----------------------------------)](https://linnndachen.medium.com/?source=post_page-----df90bfe4c9c-----------------------------------)

[Linda Chen](https://linnndachen.medium.com/?source=post_page-----df90bfe4c9c-----------------------------------)

[Jun 24, 2020·17 min read](https://towardsdatascience.com/sql-questions-summary-df90bfe4c9c?source=post_page-----df90bfe4c9c-----------------------------------)



A person preparing food on a table

Description automatically generated with low confidence

Photo by [Sebastian Coman Photography](https://unsplash.com/@sebastiancoman?utm_source=medium&utm_medium=referral) on [Unsplash](https://unsplash.com/?utm_source=medium&utm_medium=referral" \t "_blank)

Introduction:

There are lots of articles out there teaching you how to write queries. Knowing how to write a single query is not really the hard part. The hard part is that there are always multiple options of approaches and all of them require a combination of queries. To put questions into categories, it helps us to identify patterns and build better instincts on what kinds of queries we can use.

There are basically 3 types of SQL questions. The three types of questions are very simple in their original form. However, they can be leveled up by mix and match with 3 things. They are *Time Constraints*, *Calculation Requirments*, and *Comparison/Ranking Requirments*. I will show you what I meant with examples. Let’s clarify the most basic forms first.

All SQL questions can be boiled down to these 3 types:

*Type 1: Select all*

*Type 2: Select the group that did X*

*Type 3: Select the group that didn’t do X*

As you will see, Type-1 questions are basically testing our ability to use different kinds of joins. Both Type-2 and Type-3 questions are testing our ability to **precisely identify** the special group that satisfies certain constraints.



Diagram

Description automatically generated



Diagram, bubble chart

Description automatically generated

It is important to draw the distinction between Type 2 and Type 3. Type 2 requires only 1 step with the **where** statement. Type 3 requires a 2-steps approach. Firstly, we need to identify the special group with **where**statement. Secondly, we need to use **not in** to exclude that group in the **where**statement.

The reason is that if we directly select everything that is not green, then the samples that are both red and green will still be selected. I will provide examples below and it will become clearer. As you will see, Type 3 can become a lot more complicated than Type 2 questions.

Type 1: select all

* Easy Version: 1 type of join
* Hard Version: Multiple different types of join

Type 2 : Select the group that did X/met X description

* Easy Version: the "where" statement
* Harder Version: Find duplicates, Find uniques, Find consecutive

Several friends at a cinema ticket office would like to reserve consecutive available seats.Can you help to query all the consecutive available seats order by the seat\_id?| seat\_id | free |   
|---------|------|  
| 1 | 1 |  
| 2 | 0 |  
| 3 | 1 |  
| 4 | 1 |   
| 5 | 1 |

Type 3: Select the groups that did not do X/met X description

Write an SQL query that reports the **buyers** who have bought *S8* but not *iPhone*. Product table:  
+------------+--------------+------------+  
| product\_id | product\_name | unit\_price |  
+------------+--------------+------------+  
| 1 | S8 | 1000 |  
| 2 | G4 | 800 |  
| 3 | iPhone | 1400 |  
+------------+--------------+------------+  
  
Sales table:  
+------------+----------+------------+----------+-------+  
| product\_id | buyer\_id | sale\_date | quantity | price |  
+------------+----------+------------+----------+-------+  
| 1 | 1 | 2019-01-21 | 2 | 2000 |  
| 1 | 2 | 2019-02-17 | 1 | 800 |  
| 2 | 3 | 2019-06-02 | 1 | 800 |  
| 3 | 2 | 2019-05-13 | 2 | 2800 |  
+------------+----------+------------+----------+-------+

Notice buyer 2 bought both *S8* and *iPhone*and they are two separate samples*.*So this has to be solved with 2 steps. Identify buyers who bought an *iPhone* first, then select buyer\_id that is not in the buyers who bought an *iPhone*.

Level 1: Time Constraint

Dealing with the time format requires us to be familiar with the queries that deal with time. However, at this stage, the logic is very much the same. It gets hard when it combines with the other two things. It often gets combine with some calculations requirements.

* Easy Version: select a special time frame

Because we are selecting a special time frame, we are creating a special group. Therefore, these problems are either Type-2 or Type-3.

Write an SQL query that reports the **products** that were **only** sold in spring 2019. That is, between **2019-01-01** and **2019-03-31** inclusive.

* Harder Version: select the first time or the latest time of something.

When it comes to select the first or latest of something, I categorize them as Type-1 problems. The reason is we need to consider all the information without repetition and then select the best among them all.

The hard part is date information is often not the primary key. Therefore, it takes some effort to ensure that the date that we wanted is matched with the corresponded item. See the following example.

Write a SQL query that reports the **device** that is first logged in for each player.  
  
+-----------+-----------+------------+--------------+  
| player\_id | device\_id | event\_date | games\_played |  
+-----------+-----------+------------+--------------+  
| 1 | 2 | 2016-05-28 | 5 |  
| 1 | 5 | 2016-05-02 | 6 |  
| 2 | 3 | 2017-06-25 | 1 |  
| 3 | 1 | 2016-03-02 | 0 |  
| 3 | 4 | 2018-07-03 | 5 |  
+-----------+-----------+------------+--------------+

When we select **min(event\_date)**, the date is the earliest one but it will not match the corresponding device id because nor event\_date or device\_id is the primary key.

Level 2: Calculation Requirement

The most common calculations are count/sum/average/ctr. When Type-2 or Type-3 questions have calculation involves, we oftentimes need to use **case when** to label some samples. For calculations in Type-1 questions, it is mostly plugging in formulas.

The desired output can be separated into two categories. The first category has an output as one number. The second category has an output in the form of a table. The calculation is separated by users/dates/sessions.

1. The outcome as a number

* Easy Version:

Type 2: Write an SQL query to find the average for daily percentage of posts that got removed after being reported as spam, **rounded to 2 decimal places**.Actions table:  
+---------+---------+-------------+--------+--------+  
| user\_id | post\_id | action\_date | action | extra |  
+---------+---------+-------------+--------+--------+  
| 1 | 1 | 2019-07-01 | view | null |  
| 1 | 1 | 2019-07-01 | like | null |  
| 1 | 1 | 2019-07-01 | share | null |  
| 2 | 2 | 2019-07-04 | view | null |  
| 2 | 2 | 2019-07-04 | report | spam |  
| 3 | 4 | 2019-07-04 | view | null |  
| 3 | 4 | 2019-07-04 | report | spam |  
| 4 | 3 | 2019-07-02 | view | null |  
| 4 | 3 | 2019-07-02 | report | spam |  
| 5 | 2 | 2019-07-03 | view | null |  
| 5 | 2 | 2019-07-03 | report | racism |  
| 5 | 5 | 2019-07-03 | view | null |  
| 5 | 5 | 2019-07-03 | report | racism |  
+---------+---------+-------------+--------+--------+  
  
Removals table:  
+---------+-------------+  
| post\_id | remove\_date |  
+---------+-------------+  
| 2 | 2019-07-20 |  
| 3 | 2019-07-18 |  
+---------+-------------+  
  
Result table:  
+-----------------------+  
| average\_daily\_percent |  
+-----------------------+  
| 75.00 |  
+-----------------------+

Because it is a Type-2 question, we first need to identify the special group — removed posts. Then we label one group “1” and the other “0” to perform the calculation.

* Harder Version: combine with *Time Constraints.*

Type 1: Write an SQL query to find the average number of sessions per user for a period of 30 days ending **2019-07-27** inclusively, **rounded to 2 decimal places**. The sessions we want to count for a user are those with at least one activity in that time period.Activity table:  
+---------+------------+---------------+---------------+  
| user\_id | session\_id | activity\_date | activity\_type |  
+---------+------------+---------------+---------------+  
| 1 | 1 | 2019-07-20 | open\_session |  
| 1 | 1 | 2019-07-20 | scroll\_down |  
| 1 | 1 | 2019-07-20 | end\_session |  
| 2 | 4 | 2019-07-20 | open\_session |  
| 2 | 4 | 2019-07-21 | send\_message |  
| 2 | 4 | 2019-07-21 | end\_session |  
| 3 | 2 | 2019-07-21 | open\_session |  
| 3 | 2 | 2019-07-21 | send\_message |  
| 3 | 2 | 2019-07-21 | end\_session |  
| 3 | 5 | 2019-07-21 | open\_session |  
| 3 | 5 | 2019-07-21 | scroll\_down |  
| 3 | 5 | 2019-07-21 | end\_session |  
| 4 | 3 | 2019-06-25 | open\_session |  
| 4 | 3 | 2019-06-25 | end\_session |  
+---------+------------+---------------+---------------+  
  
Result table:  
+---------------------------+   
| average\_sessions\_per\_user |  
+---------------------------+   
| 1.33 |  
+---------------------------+

This is a Type-2 *Time Constraint*question*with Calculation Requirments*. We need to select the right time period and then perform the calculation

2. The output as a table

* Easier Version:

Type 2: Write an SQL query to find the ctr of each Ad.**Round** ctr to 2 decimal points. **Order** the result table by ctr in descending order and by ad\_id in ascending order in case of a tie.Ads table:  
+-------+---------+---------+  
| ad\_id | user\_id | action |  
+-------+---------+---------+  
| 1 | 1 | Clicked |  
| 2 | 2 | Clicked |  
| 3 | 3 | Viewed |  
| 5 | 5 | Ignored |  
| 1 | 7 | Ignored |  
| 2 | 7 | Viewed |  
| 3 | 5 | Clicked |  
| 1 | 4 | Viewed |  
| 2 | 11 | Viewed |  
| 1 | 2 | Clicked |  
+-------+---------+---------+  
Result table:  
+-------+-------+  
| ad\_id | ctr |  
+-------+-------+  
| 1 | 66.67 |  
| 3 | 50.00 |  
| 2 | 33.33 |  
| 5 | 0.00 |  
+-------+-------+

This is a Type-2 question because to calculate the CTR. CTR requires us to identify the “clicked” as one group and “viewed” as the other before performing any calculation. There is nothing new in the logic. We select these especial groups first and then apply the calculation.

However, for this particular question, it highlights the fact that we need to be careful of having 0 as the denominator in calculation. When this happens, we often need to treat that sample separately then **union** with the rest.

* Harder Version: multiple tables or with *Time Constraints*.

Type 1: Write an SQL query to find the number of times each student attended each exams. Order the result table by student\_id and subject\_name.Students table:  
+------------+--------------+  
| student\_id | student\_name |  
+------------+--------------+  
| 1 | Alice |  
| 2 | Bob |  
| 13 | John |  
| 6 | Alex |  
+------------+--------------+  
Subjects table:  
+--------------+  
| subject\_name |  
+--------------+  
| Math |  
| Physics |  
| Programming |  
+--------------+  
Examinations table:  
+------------+--------------+  
| student\_id | subject\_name |  
+------------+--------------+  
| 1 | Math |  
| 1 | Physics |  
| 1 | Programming |  
| 2 | Programming |  
| 1 | Physics |  
| 1 | Math |  
| 13 | Math |  
| 13 | Programming |  
| 13 | Physics |  
| 2 | Math |  
| 1 | Math |  
+------------+--------------+  
Result table:  
+------------+--------------+--------------+----------------+  
| student\_id | student\_name | subject\_name | attended\_exams |  
+------------+--------------+--------------+----------------+  
| 1 | Alice | Math | 3 |  
| 1 | Alice | Physics | 2 |  
| 1 | Alice | Programming | 1 |  
| 2 | Bob | Math | 1 |  
| 2 | Bob | Physics | 0 |  
| 2 | Bob | Programming | 1 |  
| 6 | Alex | Math | 0 |  
| 6 | Alex | Physics | 0 |  
| 6 | Alex | Programming | 0 |  
| 13 | John | Math | 1 |  
| 13 | John | Physics | 1 |  
| 13 | John | Programming | 1 |  
+------------+--------------+--------------+----------------+

This is a Type-1 question. Therefore, essentially, it is testing our understanding of different kinds of joins. For the above question, I used **cross join** and **left join**to solve it. This is a harder question because the information is scattered among 3 tables. As we join multiple tables together, it is tricky to avoid duplicates and includes all the information.

Type 2: Write an SQL query to compute moving average of how much customer paid in a 7 days window (current day + 6 days before).Customer table:  
+-------------+--------------+--------------+-------------+  
| customer\_id | name | visited\_on | amount |  
+-------------+--------------+--------------+-------------+  
| 1 | Jhon | 2019-01-01 | 100 |  
| 2 | Daniel | 2019-01-02 | 110 |  
| 3 | Jade | 2019-01-03 | 120 |  
| 4 | Khaled | 2019-01-04 | 130 |  
| 5 | Winston | 2019-01-05 | 110 |   
| 6 | Elvis | 2019-01-06 | 140 |   
| 7 | Anna | 2019-01-07 | 150 |  
| 8 | Maria | 2019-01-08 | 80 |  
| 9 | Jaze | 2019-01-09 | 110 |   
| 1 | Jhon | 2019-01-10 | 130 |   
| 3 | Jade | 2019-01-10 | 150 |   
+-------------+--------------+--------------+-------------+  
  
Result table:  
+--------------+--------------+----------------+  
| visited\_on | amount | average\_amount |  
+--------------+--------------+----------------+  
| 2019-01-07 | 860 | 122.86 |  
| 2019-01-08 | 840 | 120 |  
| 2019-01-09 | 840 | 120 |  
| 2019-01-10 | 1000 | 142.86 |  
+--------------+--------------+----------------+

This is a Type-2 question with *Time Constraints*. Thus, we first need to organize the table according to the time constraints. Then we apply the function to calculate. The hard part is we need to be familiar with the queries that handle time. For this particular question, I used **lag() over(order by)** and **sum(s\_amount) over(order by)**

Level 3: Comparison/Ranking Requirement

When it comes to comparison/ranking, it is often built on top of *Calculation Requirements*. For Type-1 questions, it is often to consider all the samples, rank them, and pick the best one. For Type-2 questions, it is often to categorize samples into groups and compare the two groups. For Type-3 questions, it is often to make a special group first, which requires ranking to do so. Then we select everything that has or does not have that group’s characteristics.

Text, letter

Description automatically generated

Text, letter

Description automatically generated

Type 1: Find the highest, nth highest, 2nd highest

Write a SQL query to get the *n*th highest salary from the Employee table. If there is no *n*th highest salary, then the query should return null.+----+--------+  
| Id | Salary |  
+----+--------+  
| 1 | 100 |  
| 2 | 200 |  
| 3 | 300 |  
+----+--------++------------------------+  
| getNthHighestSalary(2) |  
+------------------------+  
| 200 |  
+------------------------+

This is a tricky question because it is a function, which is uncommon in SQL. However, the underlying approach is the same. It is a Type-1 question, so we need to make sure no repetitive samples. Select **distinct(salary)** first, rank them using **order\_by**, then use **offset** and **fetch** to get the second row.

Type 2 questions: one group that has higher than/more than/at least…

Write a SQL query for a report that provides the pairs (actor\_id, director\_id) where the actor have cooperated with the director at least 3 times.ActorDirector table:  
+-------------+-------------+-------------+  
| actor\_id | director\_id | timestamp |  
+-------------+-------------+-------------+  
| 1 | 1 | 0 |  
| 1 | 1 | 1 |  
| 1 | 1 | 2 |  
| 1 | 2 | 3 |  
| 1 | 2 | 4 |  
| 2 | 1 | 5 |  
| 2 | 1 | 6 |  
+-------------+-------------+-------------+  
  
Result table:  
+-------------+-------------+  
| actor\_id | director\_id |  
+-------------+-------------+  
| 1 | 1 |  
+-------------+-------------+

We first precisely select the groups that we want using **where**or sometimes **having**first. Then we compare or put them into order.

Type 3: select the top 3 /Nth highest salary

* Easy Version:

Write a SQL query to find employees who have the highest salary in **each of the departments**. For the above tables, your SQL query should return the following rows (order of rows does not matter) +----+-------+--------+--------------+  
| Id | Name | Salary | DepartmentId |  
+----+-------+--------+--------------+  
| 1 | Joe | 70000 | 1 |  
| 2 | Jim | 90000 | 1 |  
| 3 | Henry | 80000 | 2 |  
| 4 | Sam | 60000 | 2 |  
| 5 | Max | 90000 | 1 |  
+----+-------+--------+--------------++----+----------+  
| Id | Name |  
+----+----------+  
| 1 | IT |  
| 2 | Sales |  
+----+----------++------------+----------+--------+  
| Department | Employee | Salary |  
+------------+----------+--------+  
| IT | Max | 90000 |  
| IT | Jim | 90000 |  
| Sales | Henry | 80000 |  
+------------+----------+--------+Write a SQL query to rank scores. If there is a tie between two scores, both should have the same ranking. Note that after a tie, the next ranking number should be the next consecutive integer value. In other words, there should be no "holes" between ranks.+----+-------+  
| Id | Score |  
+----+-------+  
| 1 | 3.50 |  
| 2 | 3.65 |  
| 3 | 4.00 |  
| 4 | 3.85 |  
| 5 | 4.00 |  
| 6 | 3.65 |  
+----+-------+Result:+-------+---------+  
| score | Rank |  
+-------+---------+  
| 4.00 | 1 |  
| 4.00 | 1 |  
| 3.85 | 2 |  
| 3.65 | 3 |  
| 3.65 | 3 |  
| 3.50 | 4 |  
+-------+---------+

Both questions have one sharing characteristic: you can have 2 or more samples sharing one rank. These two questions are easy because you can use **dense\_rank()**. It tolerates the same ranking and thus, saves our efforts from doing the two-steps approach.

* Harder version:

Write a SQL query to find employees who earn the top three salaries in each of the department. For the above tables, your SQL query should return the following rows (order of rows does not matter).+----+-------+--------+--------------+  
| Id | Name | Salary | DepartmentId |  
+----+-------+--------+--------------+  
| 1 | Joe | 85000 | 1 |  
| 2 | Henry | 80000 | 2 |  
| 3 | Sam | 60000 | 2 |  
| 4 | Max | 90000 | 1 |  
| 5 | Janet | 69000 | 1 |  
| 6 | Randy | 85000 | 1 |  
| 7 | Will | 70000 | 1 |  
+----+-------+--------+--------------++----+----------+  
| Id | Name |  
+----+----------+  
| 1 | IT |  
| 2 | Sales |  
+----+----------+Result:  
+------------+----------+--------+  
| Department | Employee | Salary |  
+------------+----------+--------+  
| IT | Max | 90000 |  
| IT | Randy | 85000 |  
| IT | Joe | 85000 |  
| IT | Will | 70000 |  
| Sales | Henry | 80000 |  
| Sales | Sam | 60000 |  
+------------+----------+--------+

This is a hard question because we have a nested logic. Here is what I meant. It is a Type-3 question, so we need to use the two-steps approach. Firstly, we make a group that consists of top-3 salaries from each department. They are 90000, 85000, 75000 for the IT department and 80000, 60000 for the Sales department. Secondly, we select all the samples that have one of these salaries.

To select a group with our desired salaries, we need to use a Type-2 logic: separate two groups and compare them. In this example, we make a copy of our table so that we have two identical tables side by side. From table 1, we pick a salary and compare it to all the salaries in table 2. If there are more than 3 salaries bigger than the salary we picked, then that salary must not be the top 3.

Level 4: Mix all requirements and constrains together

Here are two examples of questions where you have all three elements together: *Time Constrains*, *Calculation Requirments*, and *Comparison*. See if you can break down the components and solve it!

Average Calculation + Type-3 Comparision + Time: Find the movie name with the ***highest average*** rating in **February 2020**.In case of a tie, return lexicographically smaller movie name.Movies table:  
+-------------+--------------+  
| movie\_id | title |  
+-------------+--------------+  
| 1 | Avengers |  
| 2 | Frozen 2 |  
| 3 | Joker |  
+-------------+--------------+  
  
Users table:  
+-------------+--------------+  
| user\_id | name |  
+-------------+--------------+  
| 1 | Daniel |  
| 2 | Monica |  
| 3 | Maria |  
| 4 | James |  
+-------------+--------------+  
  
Movie\_Rating table:  
+-------------+--------------+--------------+-------------+  
| movie\_id | user\_id | rating | created\_at |  
+-------------+--------------+--------------+-------------+  
| 1 | 1 | 3 | 2020-01-12 |  
| 1 | 2 | 4 | 2020-02-11 |  
| 1 | 3 | 2 | 2020-02-12 |  
| 1 | 4 | 1 | 2020-01-01 |  
| 2 | 1 | 5 | 2020-02-17 |   
| 2 | 2 | 2 | 2020-02-01 |   
| 2 | 3 | 2 | 2020-03-01 |  
| 3 | 1 | 3 | 2020-02-22 |   
| 3 | 2 | 4 | 2020-02-25 |   
+-------------+--------------+--------------+-------------+  
  
Result table:  
+--------------+  
| results |  
+--------------+  
| Daniel |  
| Frozen 2 |  
+--------------+Sum Calculation + Type-2 Comparison + Time: Write an SQL query that reports the **books** that have sold **less than 10** copies in the last year, excluding books that have been available for less than 1 month from today. **Assume today is 2019-06-23**.Books table:  
+---------+--------------------+----------------+  
| book\_id | name | available\_from |  
+---------+--------------------+----------------+  
| 1 | "Kalila And Demna" | 2010-01-01 |  
| 2 | "28 Letters" | 2012-05-12 |  
| 3 | "The Hobbit" | 2019-06-10 |  
| 4 | "13 Reasons Why" | 2019-06-01 |  
| 5 | "The Hunger Games" | 2008-09-21 |  
+---------+--------------------+----------------+  
  
Orders table:  
+----------+---------+----------+---------------+  
| order\_id | book\_id | quantity | dispatch\_date |  
+----------+---------+----------+---------------+  
| 1 | 1 | 2 | 2018-07-26 |  
| 2 | 1 | 1 | 2018-11-05 |  
| 3 | 3 | 8 | 2019-06-11 |  
| 4 | 4 | 6 | 2019-06-05 |  
| 5 | 4 | 5 | 2019-06-20 |  
| 6 | 5 | 9 | 2009-02-02 |  
| 7 | 5 | 8 | 2010-04-13 |  
+----------+---------+----------+---------------+  
  
Result table:  
+-----------+--------------------+  
| book\_id | name |  
+-----------+--------------------+  
| 1 | "Kalila And Demna" |  
| 2 | "28 Letters" |  
| 5 | "The Hunger Games" |  
+-----------+--------------------+

So this how I break down complex questions into components. I hope my strategy helps!