
# SQL Cheat Sheet – Technical Concepts for the Job Interview

*This SQL cheat sheet gives you a concise and practical overview of the SQL concepts you'll need for the job interview.*

We intended for this SQL cheat sheet to help you whenever you need to skim through some SQL concepts and remember how they work. To have a cheat sheet to refer to is especially useful if you're still new to some concepts or don't use them very often.

In this SQL cheat sheet, we'll keep the explanations short. Along with them, every concept will be explained by the graphical representation and the interview question from the StrataScratch platform.

And the concepts we will cover in this SQL cheat sheet are:

1. JOINs
    - LEFT JOIN
    - INNER JOIN
    - Multiple Joins
    - Self Joins
2. Time and Date Functions
    - DATE_PART()
    - EXTRACT()
    - INTERVAL
3. Aggregate Functions
    - COUNT()
    - SUM()
    - AVG()
    - MIN() & MAX()
4. Window Functions
    - PARTITION BY
    - ORDER BY
    - RANGE or ROW
    - Ranking Window Functions
        i. RANK()
        ii. DENSE_RANK()
        iii. ROW_NUMBER()
5. Set Operators
    - UNION
    - UNION ALL

# SQL Cheat Sheet: JOINs

The JOIN clause in SQL allows you to use data from two or more tables by joining them via the same column found in both tables.

The JOINs most often appearing in the job interview are:

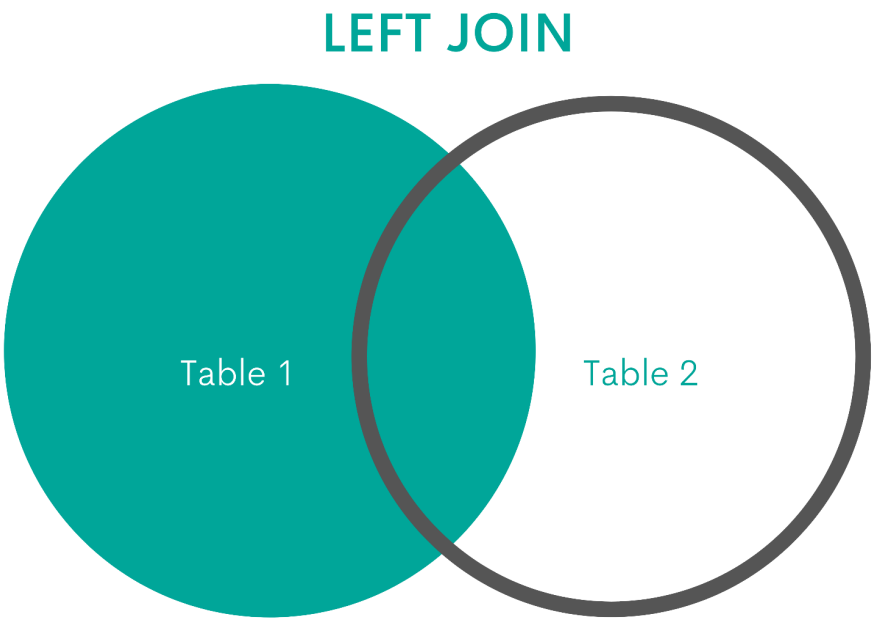
- LEFT JOIN
- INNER JOIN
- Multiple Joins
- Self Joins

## LEFT JOIN

### Explanation

The LEFT JOIN or a LEFT OUTER JOIN returns all data from the first or left table, and only the matching rows from the second or right table. If there are non-matching rows, it will return NULL values.

Graphical Representation




Interview Question

## SMS Confirmations From Users

Interview Question Date: November 2020

Meta/Facebook Medium Interview Questions ID 10291 1 0

Meta/Facebook sends SMS texts when users attempt to 2FA (2-factor authenticate) into the platform to log in. In order to successfully 2FA they must confirm they received the SMS text message. Confirmation texts are only valid on the date they were sent. Unfortunately, there was an ETL problem with the database where friend requests and invalid confirmation records were inserted into the logs, which are stored in the 'fb_sms_sends' table. These message types should not be in the table. Fortunately, the 'fb_confirmers' table contains valid confirmation records so you can use this table to identify SMS text messages that were confirmed by the user.

Calculate the percentage of confirmed SMS texts for August 4, 2020.

Tables: fb_sms_sends, fb_confirmers

Link to the question: https://platform.stratascratch.com/coding/10291-sms-confirmations-from-users

**Solution:**

```
SELECT COUNT(b.phone_number)::float  / COUNT(a.phone_number) * 100 AS perc
FROM fb_sms_sends a
LEFT JOIN fb_confirmers b ON a.ds = b.date
AND a.phone_number = b.phone_number
WHERE a.ds = '08-04-2020'
   AND a.type = 'message';
```

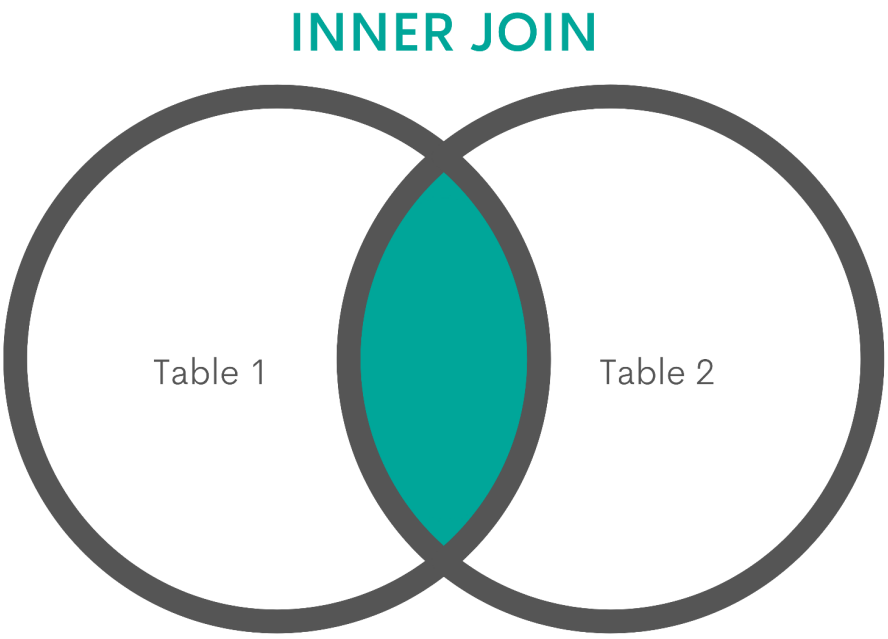
The LEFT JOIN connects the table fb_sms_sends (the left table) with the table fb_confirmers. It does that on the columns ds and date from both tables. Using this type of JOIN will get you all the rows from the table fb_sms_sends and only the matching rows from the second table.

## INNER JOIN

### Explanation

The JOIN or INNER JOIN outputs only the matching rows or the rows found in both tables.

### Graphical Representation




Interview Question

**Find the total number of available beds per hosts' nationality**

Interview Question Date: January 1970

Airbnb Medium General Practice ID 10187 0 0

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.

Tables: airbnb_apartments, airbnb_hosts

Link to the question:
https://platform.stratascratch.com/coding/10187-find-the-total-number-of-available-beds-per-hosts-nationality

**Solution:**

```
SELECT nationality,
       SUM(n_beds) AS total_beds_available
```

```
FROM airbnb_hosts h
INNER JOIN airbnb_apartments a ON h.host_id = a.host_id
GROUP BY nationality
ORDER BY total_beds_available DESC;
```

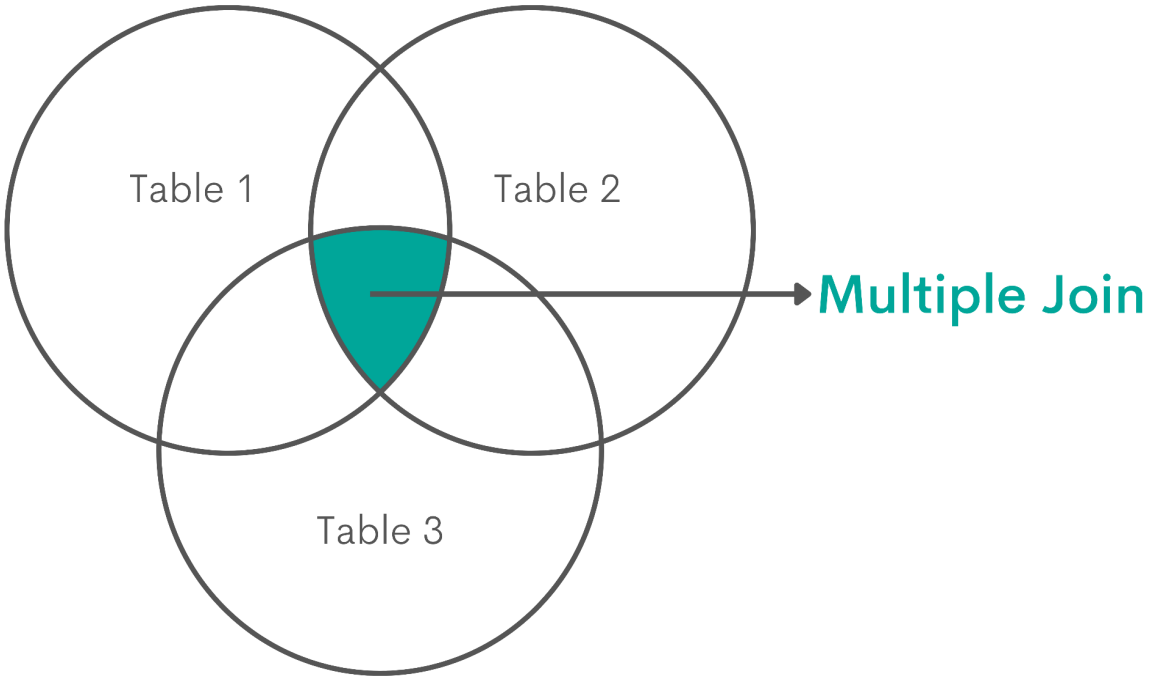
The INNER JOIN will join table airbnb_hosts with the table airbnb_apartments. The column on which this is done is host_id. This join will return only hosts that appear in both tables.

## Multiple Joins

### Explanation

The multiple joins are not a special type of JOIN. It simply means joining more than two tables while using any type of SQL JOIN. It works the same way as joining two tables, but you add a three, four, five, or any number of tables into the chain of JOINs.

### Graphical Representation



Interview Question

# Premium vs Freemium

Interview Question Date: November 2020

Microsoft Hard Interview Questions ID 10300 0 0

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.

Tables: ms_user_dimension, ms_acc_dimension, ms_download_facts

Link to the question: https://platform.stratascratch.com/coding/10300-premium-vs-freemium

**Solution:**

```
SELECT date,
       non_paying,
       paying
FROM
  (SELECT date, SUM(CASE
                        WHEN paying_customer = 'yes' THEN downloads
                    END) AS paying,
                SUM(CASE
                        WHEN paying_customer = 'no' THEN downloads
                    END) AS non_paying
   FROM ms_user_dimension a
   INNER JOIN ms_acc_dimension b ON a.acc_id = b.acc_id
   INNER JOIN ms_download_facts c ON a.user_id=c.user_id
   GROUP BY date
   ORDER BY date) t
WHERE (non_paying - paying) > 0
ORDER BY t.date ASC;
```

In this solution, three tables are joined in the subquery using the INNER JOIN. First, the tables ms_user_dimension and ms_acc_dimension are joined on the column acc_id. Then the third table,

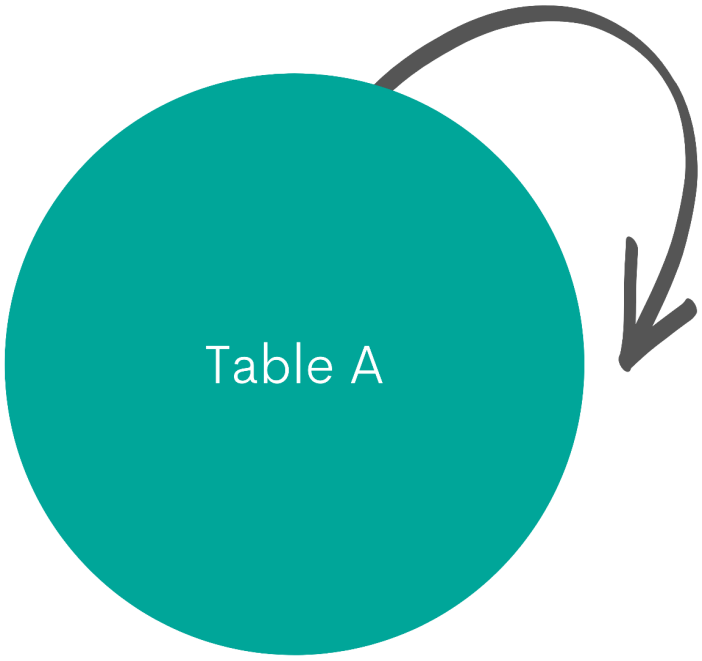
ms_download_facts, is added using the second INNER JOIN. This table is joined with the first table on the column user_id.

## Self-JOIN

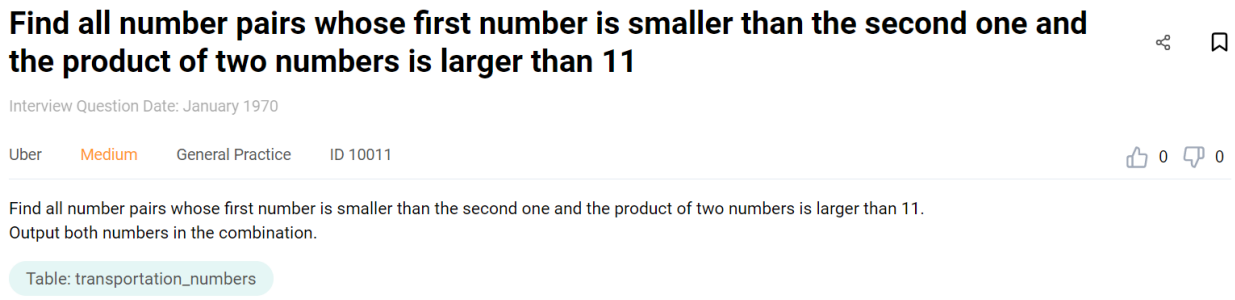
### Explanation

Again, the self-join isn't a specific JOIN clause in SQL. Any JOIN can be used to self-join. It means you're joining the table with itself.

### Graphical Representation




### Interview Question




Link to the question:
https://platform.stratascratch.com/coding/10011-find-all-number-pairs-whose-first-number-is-smaller-than-the-second-one-and-the-product-of-two-numbers-is-larger-than-11

**Solution:**

```
SELECT DISTINCT n1.number AS num1,
                n2.number AS num2
FROM transportation_numbers n1
INNER JOIN transportation_numbers n2 ON n1.index <> n2.index
WHERE n1.number < n2.number
  AND n1.number * n2.number > 11;
```

This solution uses the INNER JOIN to join the table transportation_numbers with itself. It's done by having the table appear twice in the INNER JOIN: once as a left table and the second time as a right table. Both tables are given aliases so you can make a distinction between them. The table is self-joined on the column index.

# SQL Cheat Sheet: Time and Date Functions

The time and date functions are the SQL functions used to calculate and manipulate the time and date data types. They can differ between the SQL dialects, but the three most common in PostgreSQL are:

- DATE_PART()
- EXTRACT()
- INTERVAL

# DATE_PART()

## Explanation

This function is used to get a part of date or time from data. For example, it can return a year or month from the date or minutes from the time data.

Graphical Representation

Syntax:

DATE_PART(text, source)

Example:

| id | date |
|---|---|
| 1 | 2022-04-28 |
| 2 | 2022-04-13 |
| 3 | 2022-04-12 |

```
SELECT id,
  date,
     DATE_PART('year', date) AS year
FROM example_data;
```

| id | date | year |
|---|---|---|
| 1 | 2022-04-28 | 2022 |
| 2 | 2022-04-13 | 2022 |
| 3 | 2022-04-12 | 2022 |

Interview Question

# Find the day of the week that most people check-in

Interview Question Date: January 1970

Airbnb Hard General Practice ID 9762 0 0

Find the day of the week that most people want to check-in.
Output the day of the week alongside the corresponding check-incount.
Order records based on the check-in count in descending order.

Table: airbnb_contacts

Link to the question:
https://platform.stratascratch.com/coding/9762-find-the-day-of-the-week-that-most-people-check-in

**Solution:**

```
SELECT DATE_PART('dow', ds_checkin :: DATE) AS day_of_week,
       COUNT(*) AS checkin_count
FROM airbnb_contacts
GROUP BY day_of_week
ORDER BY checkin_count DESC
LIMIT 1;
```

The DATE_PART function is applied to the column ds_checkin from the table airbnb_contacts. To get the day of the week, 'dow' has to be written in the function.

## EXTRACT()

### Explanation

The EXTRACT() function does the same thing as the DATE_PART() function: it returns the part of the date or time. It is a function in compliance with the SQL standard, while the DATE_PART() is the PostgreSQL function.

Graphical Representation

Syntax:

EXTRACT(field FROM source)

Example:

| id | date |
|---|---|
| 1 | 2022-04-28 |
| 2 | 2022-04-13 |
| 3 | 2022-04-12 |

```
SELECT id,
  date,
      EXTRACT(year FROM date) AS year
FROM example_data;
```

| id | date | year |
|---|---|---|
| 1 | 2022-04-28 | 2022 |
| 2 | 2022-04-13 | 2022 |
| 3 | 2022-04-12 | 2022 |

Interview Question

# Products Report Summary

Interview Question Date: June 2021

Whole Foods Market Easy Active Interview ID 2039 0 0

Find the number of transactions and total sales for each of the product categories in 2017. Output the product categories, number of transactions, and total sales in descending order. The sales column represents the total cost the customer paid for the product so no additional calculations need to be done on the column.
Only include product categories that have products sold.

Tables: wfm_transactions, wfm_products

Link to the question: https://platform.stratascratch.com/coding/2039-products-report-summary

**Solution:**

```
SELECT product_category,
       COUNT(transaction_id) AS transactions,
       SUM(sales) AS sales
FROM wfm_transactions AS tr
INNER JOIN wfm_products AS pr ON pr.product_id = tr.product_id
WHERE EXTRACT(YEAR
              FROM transaction_date) = 2017
GROUP BY product_category
ORDER BY sum(sales) DESC;
```

The EXTRACT() function is used on the column transaction_date to extract year from it. Used in the WHERE clause like above means it will show only transactions from 2017.

## INTERVAL

### Explanation

The INTERVAL function is used to add or subtract a time interval to the already existing date or time.

Graphical Representation

## Syntax:

INTERVAL text

## Example:

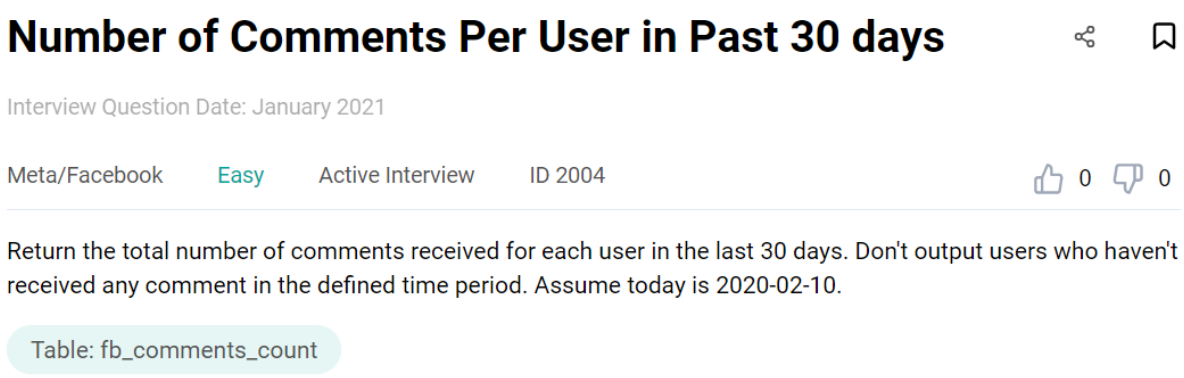
| id | date |
|---|---|
| 1 | 2022-04-28 |
| 2 | 2022-04-13 |
| 3 | 2022-04-12 |

+30 days

```
SELECT id, date, date + INTERVAL '30 days' AS added_days
FROM example_data;
```

| id | date | added_days |
|---|---|---|
| 1 | 2022-04-28 | 2022-05-28 0:00:00 |
| 2 | 2022-04-13 | 2022-05-13 0:00:00 |
| 3 | 2022-04-12 | 2022-05-12 0:00:00 |

Interview Question




Link to the question:
https://platform.stratascratch.com/coding/2004-number-of-comments-per-user-in-past-30-days

**Solution:**

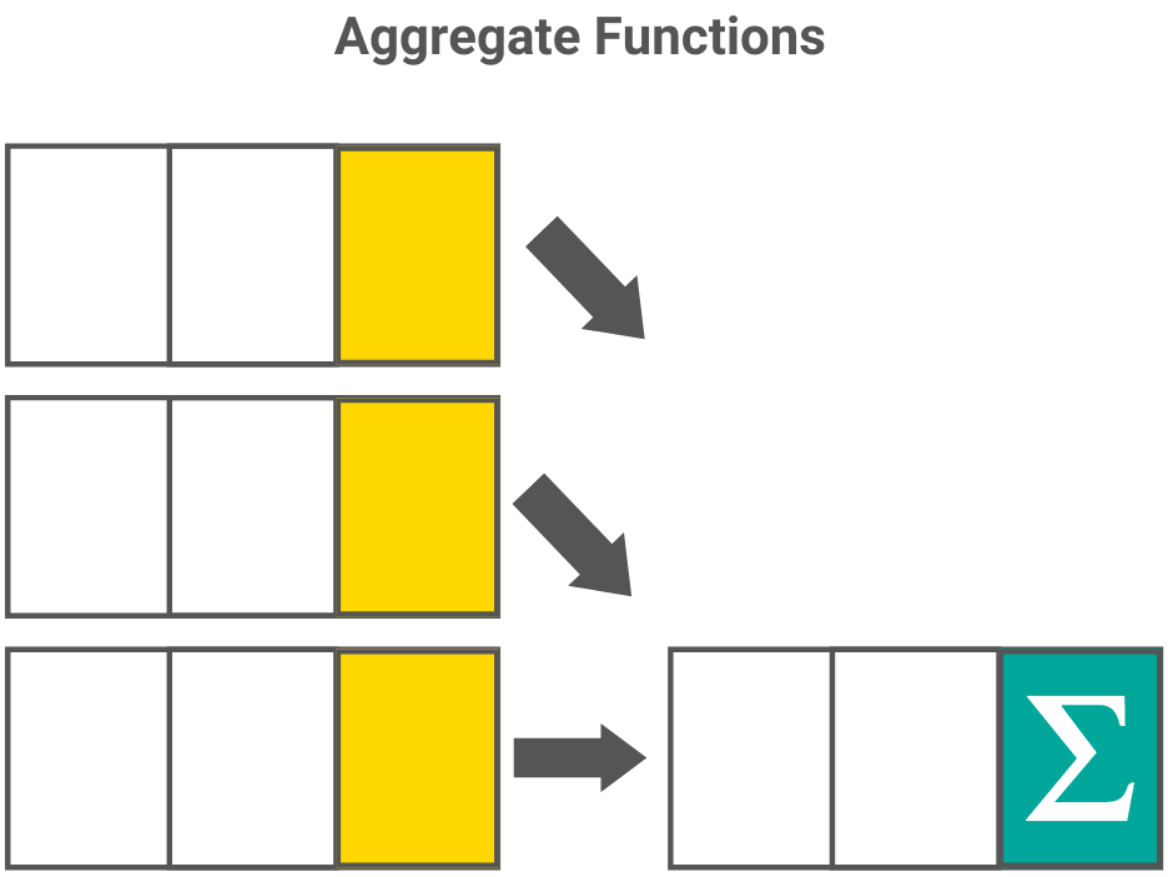
```
SELECT user_id,
       SUM(number_of_comments) AS number_of_comments
FROM fb_comments_count
WHERE created_at BETWEEN '2020-02-10'::date - 30 * INTERVAL '1 day' AND
'2020-02-10'::date
GROUP BY user_id;
```

To answer the question, finding the date 30 days prior to the current date is required. The assumption is that the current date is 2020-02-10. Subtracting INTERVAL '1 day' from it means it will deduct one day from 2020-02-10, which is 2020-02-09. If the 30 * INTERVAL '1 day' is subtracted, this means 30 days are subtracted from the current date. This results in getting the comments from the last 30 days. To make sure there are no comments after today, there is the second condition in the WHERE clause.

## SQL Cheat Sheet: Aggregate Functions

The SQL aggregate functions are the functions that perform a calculation on the set of rows and return a single value.

They can be represented in the following manner.




The most often used functions are:

- COUNT()
- SUM()
- AVG()
- MIN()
- MAX()

The aggregate functions can also be used as window functions. What they are will be explained in the window functions section.

# COUNT()

## Explanation

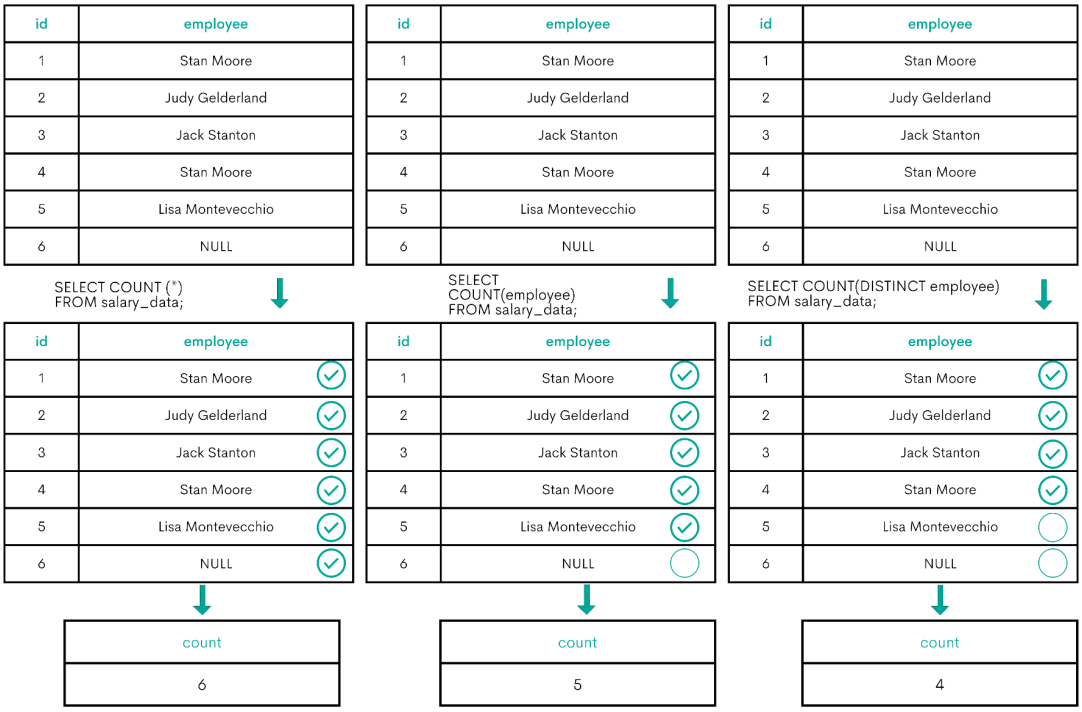
The COUNT() function counts rows. There are three ways of using this function.

COUNT(*) returns all the rows from the table, including duplicates and NULL values.

The second usage is COUNT(expression). This one will return all the rows from the table, including duplicates, but will ignore the NULL values.

The COUNT (DISTINCT expression) result is the number of distinct (unique) rows without the NULL values.

## Graphical Representation




## Interview Question

# Businesses Open On Sunday

Interview Question Date: January 1970

Yelp Medium General Practice ID 10178 0 0

Find the number of businesses that are open on Sundays. Output the slot of operating hours along with the corresponding number of businesses open during those time slots. Order records by total number of businesses opened during those hours in descending order.

Tables: yelp_business_hours, yelp_business

Link to the question: https://platform.stratascratch.com/coding/10178-businesses-open-on-sunday

**Solution:**

```
SELECT sunday,
       COUNT(*) AS total_business
FROM yelp_business_hours business_hours
LEFT JOIN yelp_business business ON business_hours.business_id =
business.business_id
WHERE sunday IS NOT NULL
  AND is_open = 1
GROUP BY sunday
ORDER BY total_business DESC;
```

Once the tables are joined and data filtered, using the COUNT(*) function will count the number of business per slot of operating hours. In this case, using the COUNT(business.business_id) or COUNT(DISTINCT business.business_id) would get you the same result.

## SUM()

### Explanation

The SUM() function is an aggregate function used for getting the sum of the rows' values.

## Graphical Representation

| id | employee | salary |
|---|---|---|
| 1 | Stan Moore | 14,874.44 |
| 2 | Judy Gelderland | 23,148.16 |
| 3 | Jack Stanton | 8,747.52 |
| 4 | Stan Moore | 21,874.65 |
| 5 | Lisa Montevecchio | 13,325.17 |

```
SELECT SUM(salary)
FROM salaries;
```

| sum |
|---|
| 81,969.94 |

## Interview Question

**Calculate Samantha's and Lisa's total sales revenue**

Interview Question Date: January 1970

Salesforce Easy General Practice ID 10127 0 0

What is the total sales revenue of Samantha and Lisa?

Table: sales_performance

Link to the question:
https://platform.stratascratch.com/coding/10127-calculate-samanthas-and-lisas-total-sales-revenue

**Solution:**

```
SELECT SUM(sales_revenue) AS total_revenue
FROM sales_performance
WHERE salesperson = 'Samantha'
```

```
  OR salesperson = 'Lisa';
```

The solution uses the SUM() function on the sales_revenue column to calculate Samantha's and Lisa's sales.

# AVG()

## Explanation

The AVG() function calculates the average value of the rows. In other words, it sums the values of the rows and divides the result by the number of rows. If there are NULL values, the AVG() will ignore it, i.e. it divides the sum only by the number of rows with the non-NULL values.

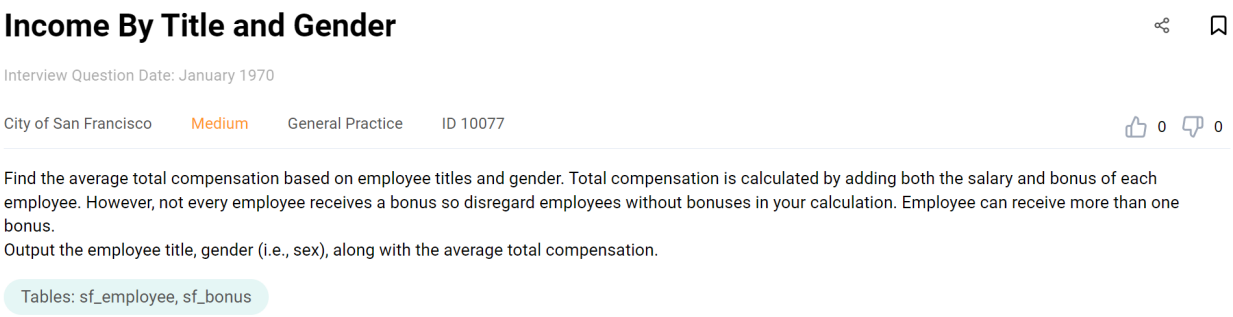
## Graphical Representation

| id | employee | salary |
|---|---|---|
| 1 | Stan Moore | 14,874.44 |
| 2 | Judy Gelderland | 23,148.16 |
| 3 | Jack Stanton | 8,747.52 |
| 4 | Stan Moore | 21,874.65 |
| 5 | Lisa Montevecchio | 13,325.17 |

```
SELECT AVG(salary)
FROM salaries;
```

| avg |
|---|
| 16393.9880 |

## Interview Question




Link to the question:
https://platform.stratascratch.com/coding/10077-income-by-title-and-gender

**Solution:**

```
SELECT e.employee_title,
       e.sex,
       AVG(e.salary + b.ttl_bonus) AS avg_compensation
FROM sf_employee e
INNER JOIN
  (SELECT worker_ref_id,
          SUM(bonus) AS ttl_bonus
   FROM sf_bonus
   GROUP BY worker_ref_id) b ON e.id = b.worker_ref_id
GROUP BY employee_title,
         sex;
```

The AVG() function is applied on the sum of the columns salary and ttl_bonus, because these two columns constitute the total compensation. Therefore, the query will show the average salary by the employee title and gender.

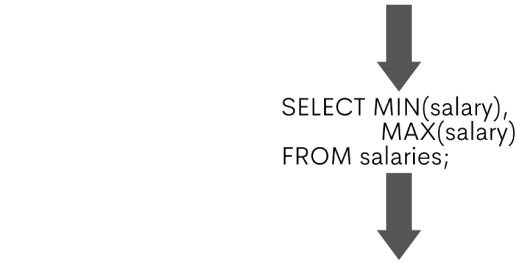
## MIN() & MAX()

### Explanation

The MIN() and MAX() aggregate functions are two sides of the same coin. The MIN() returns the minimum or the lowest value, while the MAX() will show the maximum or the highest value.

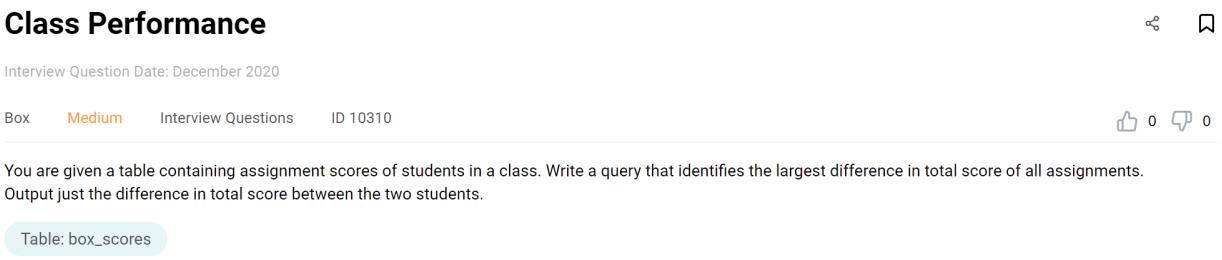
## Graphical Representation

| id | employee | salary |
|---|---|---|
| 1 | Stan Moore | 14,874.44 |
| 2 | Judy Gelderland | 23,148.16 |
| 3 | Jack Stanton | 8,747.52 |
| 4 | Stan Moore | 21,874.65 |
| 5 | Lisa Montevecchio | 13,325.17 |




| min | max |
|---|---|
| 8,747.52 | 23,148.16 |

## Interview Question




Link to the question: https://platform.stratascratch.com/coding/10310-class-performance

**Solution:**

```
SELECT MAX(score)-MIN(score) AS difference_in_scores
FROM
  (SELECT student,
          SUM(assignment1+assignment2+assignment3) AS score
   FROM box_scores
   GROUP BY student) a;
```

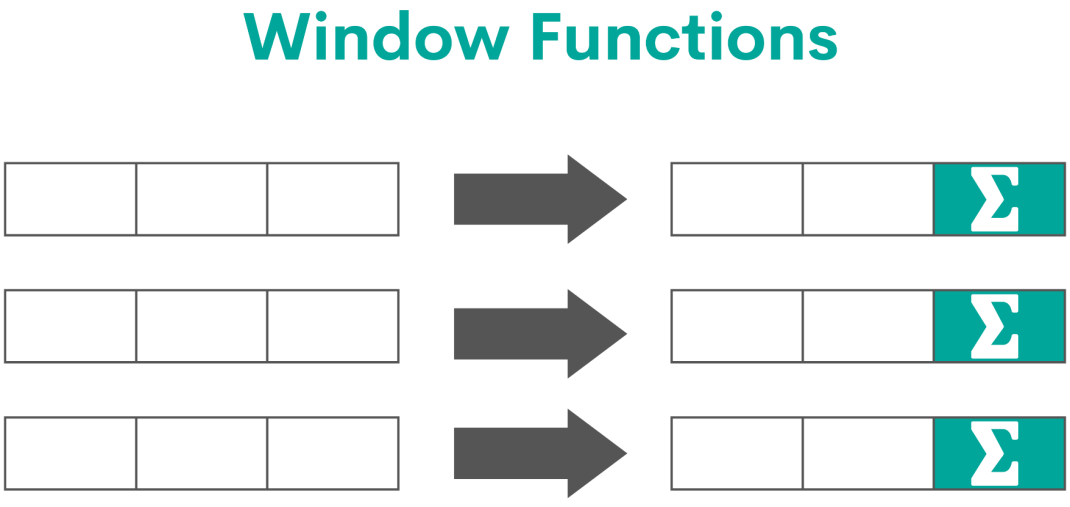
To get the largest difference in total score, the maximum and minimum score has to be calculated. The MAX() and MIN() functions will do that when applied to the column score. Once these values are calculated, they have to be subtracted to get the difference between the highest and lowest score.

## SQL Cheat Sheet: Window Functions

The SQL window functions are functions executing the calculation over the rows that are related to the current row.

They are similar to the aggregate functions used with the GROUP BY clause. The main difference is the window functions do not collapse the individual rows while showing the aggregated values.

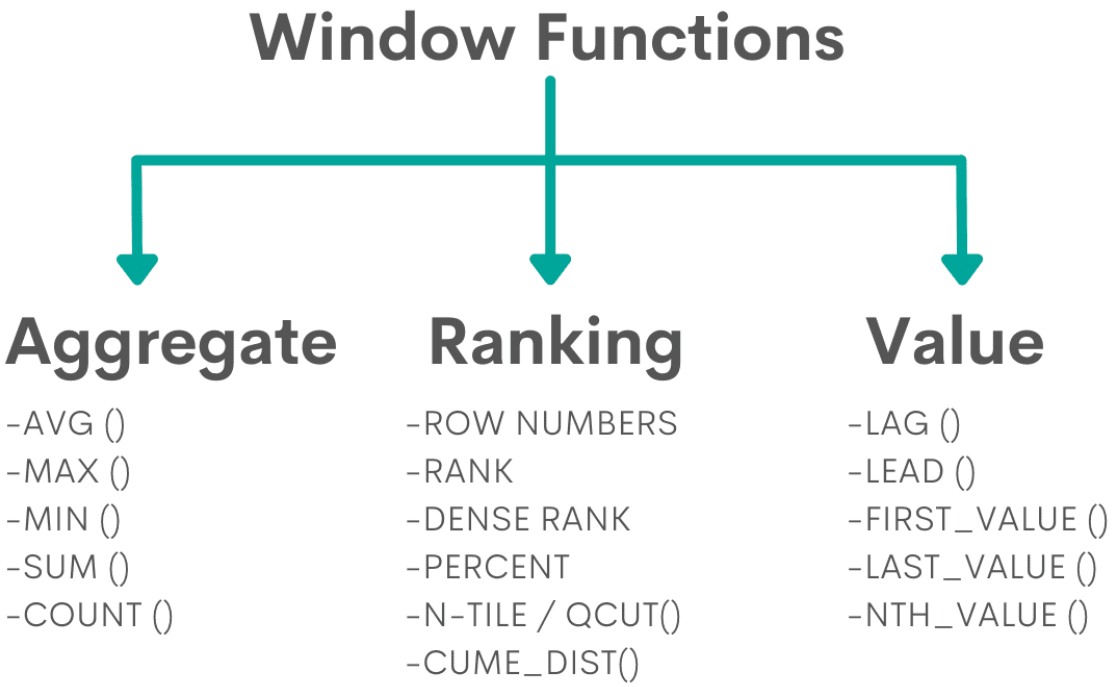
This is how they work:



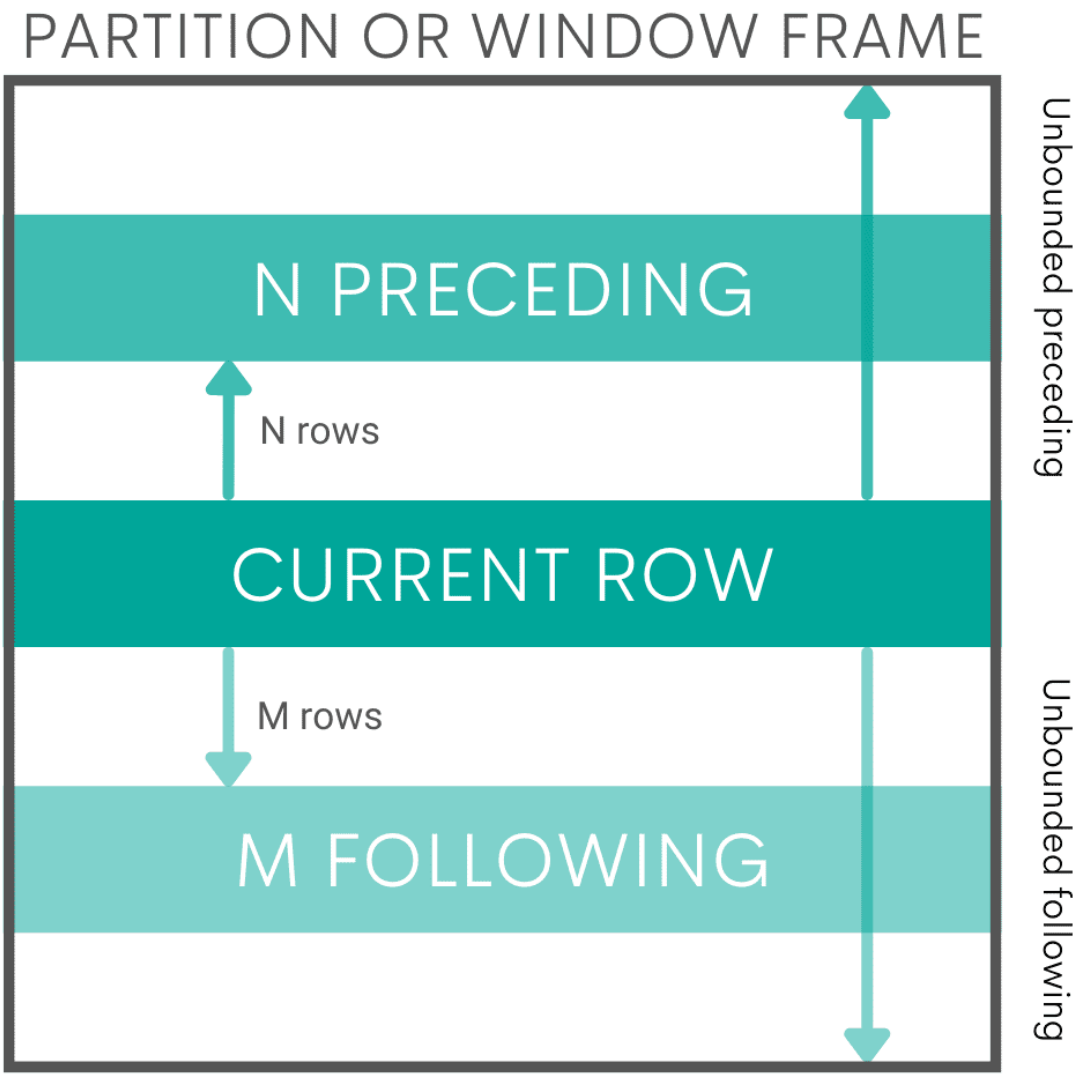
They can broadly be divided into three categories:

- Aggregate Window Functions
- Ranking Window Functions
- Value Window Functions

The interviewers are usually most interested in the Ranking Window Functions.




The window functions' name comes from the window frame. It is the way of dividing the set of data into blocks. The window frame can be all the rows with the same data in the specified column(s). Or it can be a certain number of rows preceding and following the current row.




The window functions syntax is:

```
window_function ( expression ) OVER ([PARTITION BY list_of_columns] [ORDER BY list_of_columns] [ROW or RANGE clause])
```

***PARTITION BY***

The PARTITION BY clause is used to define the window frame. In other words, the column according to which the data will be partitioned is defined using the PARTITION BY. The window function will be executed on each partition separately when this is done.

***ORDER BY***

The ORDER BY is another sub-clause used in the OVER() clause. Unlike the 'regular' ORDER BY, it is not used to order data in the query output. In the window functions, it is used to specify the order in which the window function will be executed. This can be in ascending or descending order.

***RANGE or ROW***

The third important part of the OVER() clause is the RANGE or ROW clause. They are also used to define a window frame. They don't do it according to the table column, like PARTITION BY. The RANGE and ROW clauses define the start and end rows of the window frame.

The RANGE clause does that logically by specifying the rows in relation to the current row's *value*.

The ROW clause doesn't look at the current row's value. It simply defines a window frame by the *number* of rows before and/or after the current row.

The options for using these clauses are:

- UNBOUNDED PRECEDING – all the rows before the current row and including the current row
- UNBOUNDED FOLLOWING – all the rows after the current row and including the current row
- N PRECEDING – defined number of rows before the current row and including the current row
- M FOLLOWING – defined number of rows after the current row and including the current row
- CURRENT ROW – only the current row

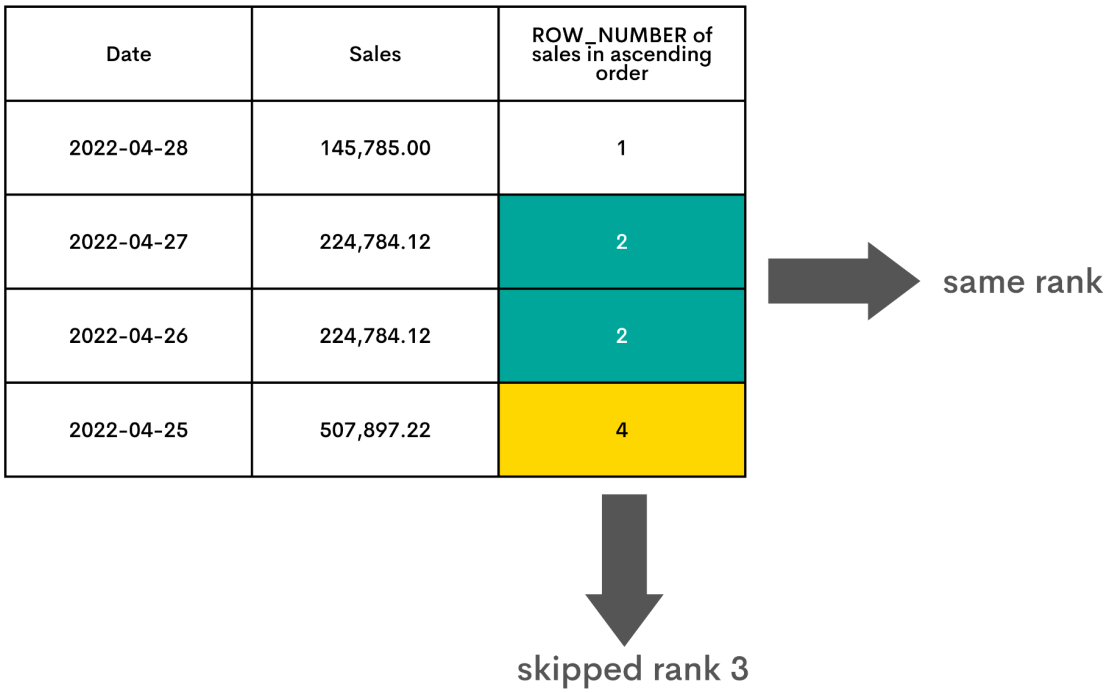
These options can be combined freely according to the needs.

# RANK()

## Explanation

The RANK() is the function that ranks data in a table and skips the ties. This means it will allocate the same rank where the values are the same, but the next non-tie value will be non-sequentially ranked. It will skip the number of ranks in line with the number of ties.

## Graphical Representation




## Interview Question

**Update Call Duration**

Interview Question Date: January 2021

Redfin Medium Active Interview ID 2022

0 0

Redfin helps clients to find agents. Each client will have a unique request_id and each request_id has several calls. For each request_id, the first call is an "initial call" and all the following calls are "update calls". What's the average call duration for all update calls?

Table: redfin_call_tracking

Link to the question: https://platform.stratascratch.com/coding/2022-update-call-duration

**Solution:**

```
SELECT AVG(call_duration)
FROM redfin_call_tracking
```

```
WHERE id in
    (SELECT id
     FROM
       (SELECT *,
               RANK() OVER (PARTITION BY request_id
                            ORDER BY created_on) AS rk
        FROM redfin_call_tracking) sq
     WHERE rk > 1);
```

The above solution uses the RANK() window function in the subquery. After the RANK() function, the OVER() clause is written, which signals the window function.

The data is partitioned by the request ID, which means every request ID will be ranked separately. The ORDER BY means the order in which the ranking will be performed. In this case, the ranking will be executed according to the creation date in the ascending order.

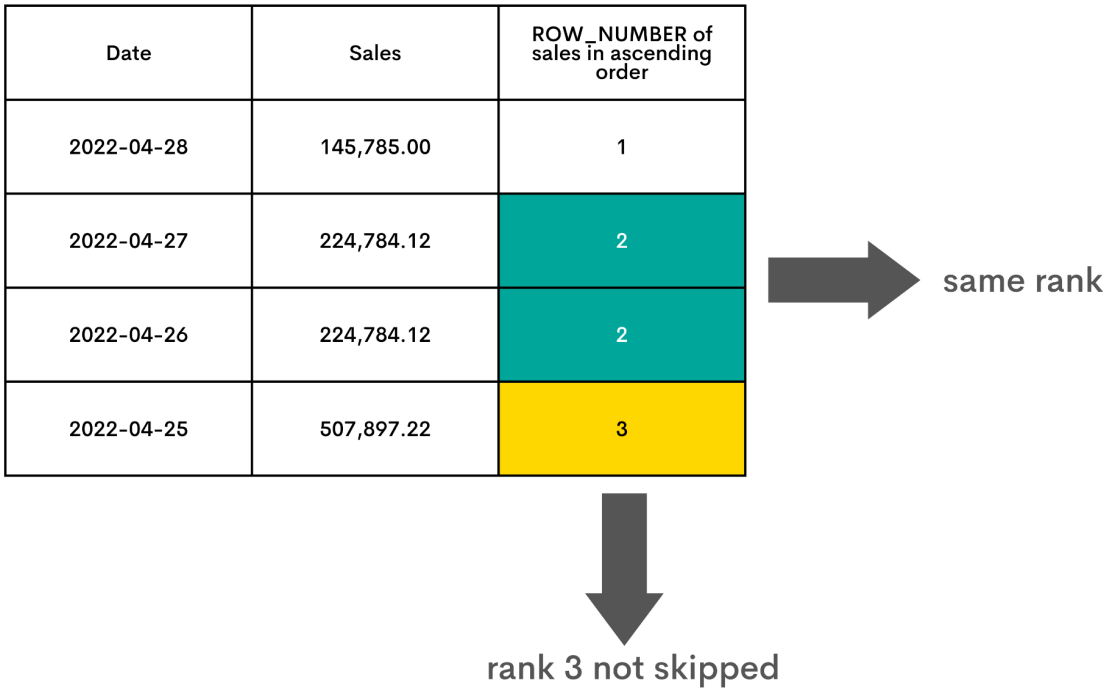
In practice, this means the RANK() function will rank the request ID from the oldest to the newest date. Once all dates within one request ID are ranked, the window functions will go to the next request ID and start ranking from 1 again.

## DENSE_RANK()

### Explanation

The DENSE_RANK() is also the Ranking Window Function. Unlike RANK(), it doesn't skip ties. This means all the non-tied rows will be ranked sequentially, while the tied values will have the same rank.

## Graphical Representation




## Interview Question

**Ranking Hosts By Beds**

Interview Question Date: July 2020

Airbnb Medium Interview Questions ID 10161 0 0

Rank each host based on the number of beds they have listed. The host with the most beds should be ranked 1 and the host with the least number of beds should be ranked last. Hosts that have the same number of beds should have the same rank. A host can also own multiple properties.
Output the host ID, number of beds, and rank from highest rank to lowest.

Table: airbnb_apartments

Link to the question: https://platform.stratascratch.com/coding/10161-ranking-hosts-by-beds

**Solution:**

```
SELECT host_id,
       SUM(n_beds) AS number_of_beds,
       DENSE_RANK() OVER(
```

```
                        ORDER BY SUM(n_beds) DESC) AS rank
FROM airbnb_apartments
GROUP BY host_id
ORDER BY number_of_beds DESC;
```

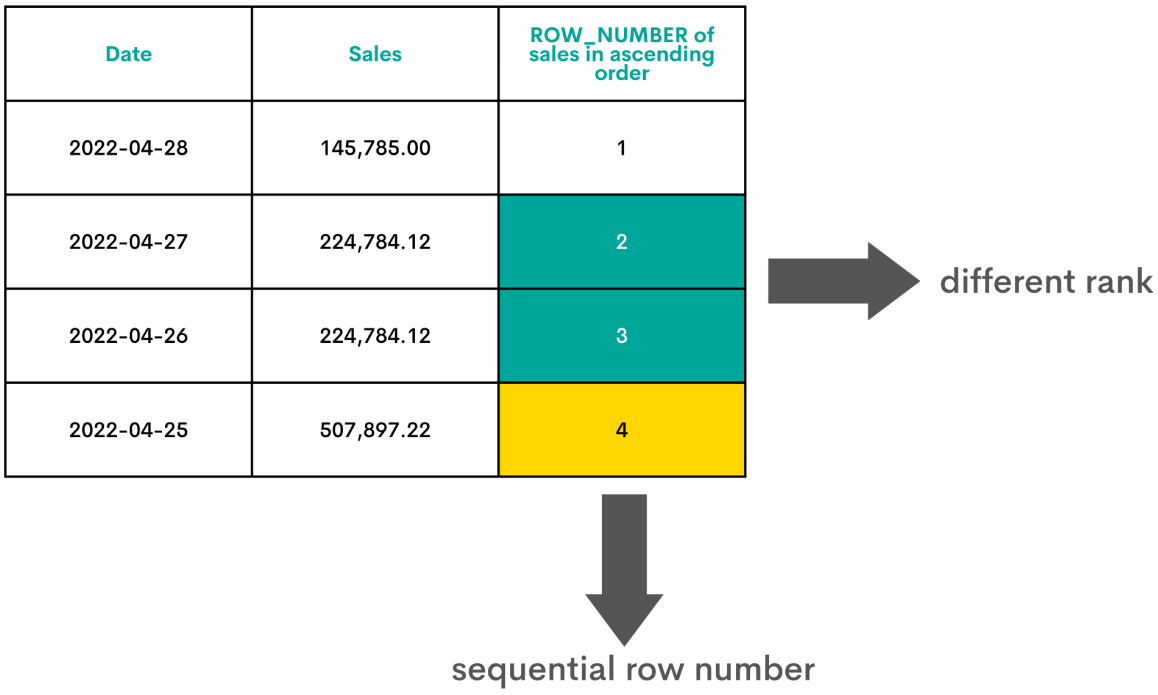
The DENSE_RANK() function is introduced here, also using the OVER() clause. There's no PARTITION BY clause, which means the ranking will be executed across the whole table. The data will be ranked by the sum of the number of beds from the highest to the lowest.

## ROW_NUMBER

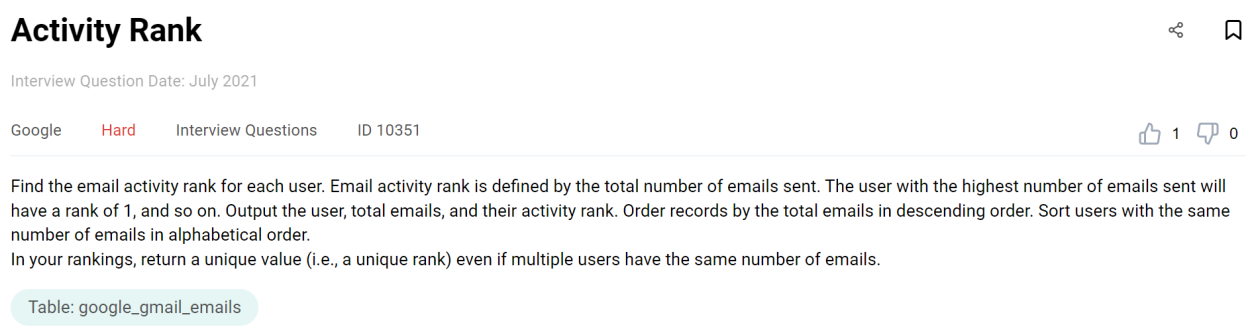
### Explanation

The third ranking function allocates the row number to the rows. The row number is allocated sequentially, which also includes the tied values. It's also possible that the ROW_NUMBER doesn't return the same row number each time. This happens when the values or the combination of values in the columns used in PARTITION BY and ORDER BY are not unique.

### Graphical Representation



Interview Question




Link to the question: https://platform.stratascratch.com/coding/10351-activity-rank

**Solution:**

```
SELECT from_user,
       COUNT(*) AS total_emails,
       ROW_NUMBER() OVER (
                           ORDER BY count(*) DESC, from_user ASC)
FROM google_gmail_emails
GROUP BY from_user
ORDER BY 2 DESC,
         1;
```

Due to the question's request to return a unique ranking value even when the users have the same number of emails, the ROW_NUMBER() is used for ranking. The function name is followed by the OVER() clause as in the previous examples. The ranking is here performed by the number of emails in descending order and alphabetically if there are ties.

# SQL Cheat Sheet: Set Operators

These operators allow for combining the outputs of multiple queries into a single data set.

The two most used operators are:

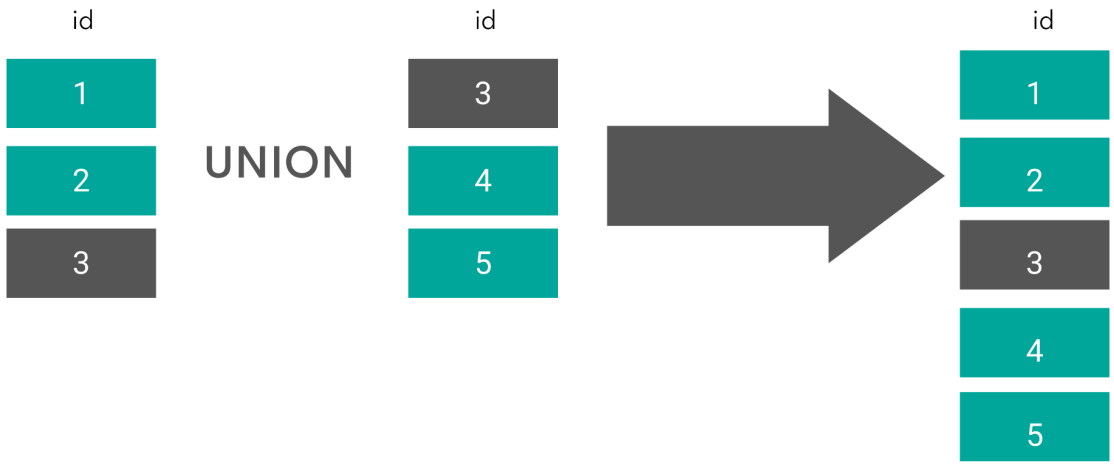
- UNION
- UNION ALL

# UNION

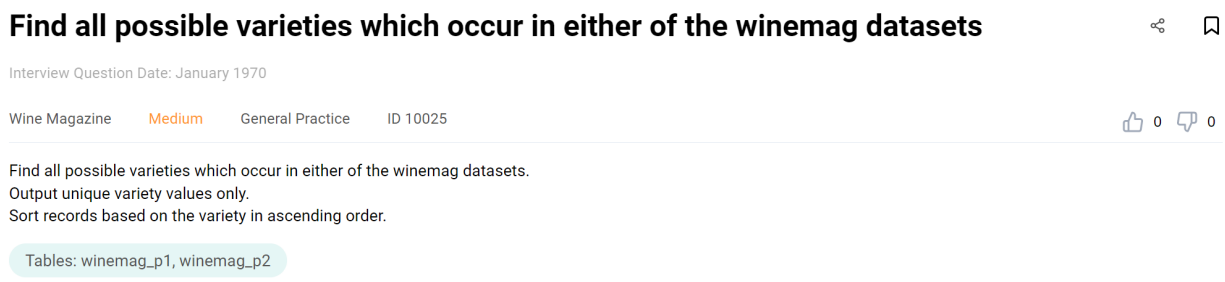
## Explanation

The UNION operator returns only the unique values from the outputs of two or more queries. There must be an equal number of columns in all queries for this operator to work. The columns have to be of the same data type between them, too.

## Graphical Representation




## Interview Question




Link to the question:
https://platform.stratascratch.com/coding/10025-find-all-possible-varieties-which-occur-in-either-of-the-winemag-datasets

**Solution:**

```
SELECT DISTINCT variety
FROM winemag_p1
UNION
SELECT DISTINCT variety
FROM winemag_p2
WHERE variety IS NOT NULL
ORDER BY variety;
```

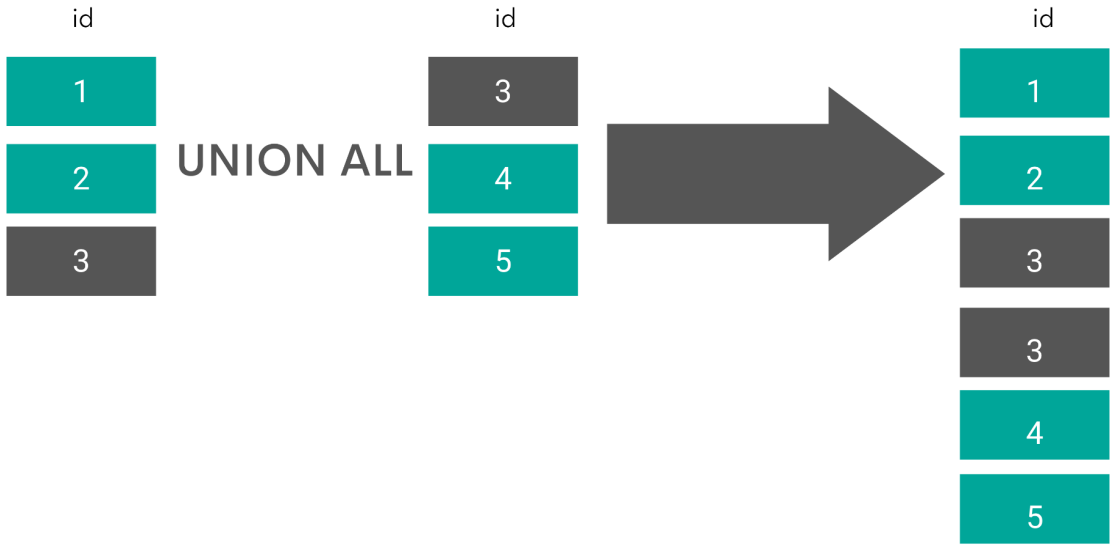
The distinct wine varieties have to be found separately using two SELECT statements. The first one will query the table winemag_p1, the second one will query the table winemag_p2. Since the question asks to output only the unique varieties, the UNION operator needs to be used: it is simply written between the two SELECT statements.

## UNION ALL

### Explanation

The UNION ALL operator does the same as the UNION operator, with one important difference: UNION ALL returns all data from all queries, including the duplicate values.

## Graphical Representation




## Interview Question

**Sum Of Numbers**

Interview Question Date: January 1970

Uber Medium General Practice ID 10008 0 0

Find the sum of numbers whose index is less than 5 and the sum of numbers whose index is greater than 5. Output each result on a separate row.

Table: transportation_numbers

Link to the question: https://platform.stratascratch.com/coding/10008-sum-of-numbers

**Solution:**

```
SELECT SUM(number)
FROM transportation_numbers
WHERE index < 5
UNION ALL
```

```
SELECT SUM(number)
FROM transportation_numbers
WHERE index > 5;
```

The first query will return the sum of numbers where index is less than 5. The second query will do the same, but for the index above 5. To merge the outputs of both queries, the UNION ALL is written between the SELECT statements. Why UNION ALL instead of UNION? Because the results of both queries can be the same. In fact, they are the same! If UNION was used, the result would have been shown only in one row. This would mean not getting the correct result because the question asks to output results in separate rows.