## Sql Tutorial

Base de datos **adventureWorks2019**

<https://mode.com/sql-tutorial/sql-count>

<https://app.mode.com/editor/sqlmariano/reports/9ea845594d0a/queries/79d7a42e19c0>

Contenido

[Sql Tutorial 1](#_Toc189752523)

[Intermediate SQL 2](#_Toc189752524)

[Aggregate functions 2](#_Toc189752525)

[Counting individual columns 3](#_Toc189752526)

[Sql sum 3](#_Toc189752527)

[Min and Max 3](#_Toc189752528)

[AVG 3](#_Toc189752529)

[Group by 4](#_Toc189752530)

[GROUP BY column numbers 4](#_Toc189752531)

[Using GROUP BY with ORDER BY 4](#_Toc189752532)

[HAVING 5](#_Toc189752533)

[query clause order 5](#_Toc189752534)

[CASE 5](#_Toc189752535)

[Adding multiple conditions to a CASE statement 5](#_Toc189752536)

[Using CASE with aggregate functions 7](#_Toc189752537)

[Using SQL DISTINCT for viewing unique values 10](#_Toc189752538)

[Using DISTINCT in aggregations 11](#_Toc189752539)

[DISTINCT performance 11](#_Toc189752540)

[Joins 12](#_Toc189752541)

[Inner Join 12](#_Toc189752542)

[Filtering in the ON clause 16](#_Toc189752543)

[El AND se ejecuta antes del join, es como un where para una sola tabla 17](#_Toc189752544)

[The SQL FULL JOIN command 21](#_Toc189752545)

[The SQL UNION operator 22](#_Toc189752546)

[Using comparison operators with joins 24](#_Toc189752547)

[Self joining tables 25](#_Toc189752548)

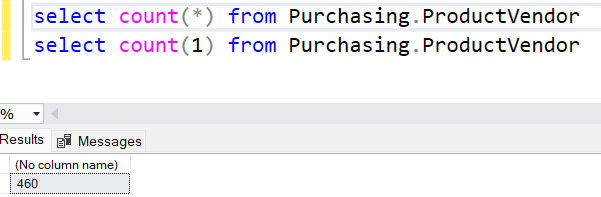
# [Intermediate SQL](https://mode.com/sql-tutorial/intro-to-intermediate-sql)

## Aggregate functions

* [COUNT](https://mode.com/sql-tutorial/sql-count) counts how many rows are in a particular column.
* [SUM](https://mode.com/sql-tutorial/sql-sum) adds together all the values in a particular column.
* [MIN](https://mode.com/sql-tutorial/sql-min-max) and [MAX](https://mode.com/sql-tutorial/sql-min-max) return the lowest and highest values in a particular column, respectively.
* [AVG](https://mode.com/sql-tutorial/sql-avg) calculates the average of a group of selected values.

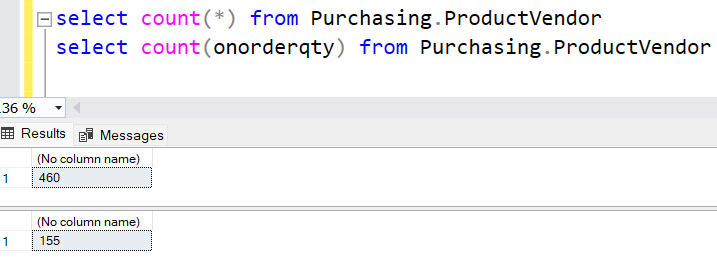
Count(1) == count(\*)

Sql top 100 , otros limit 100

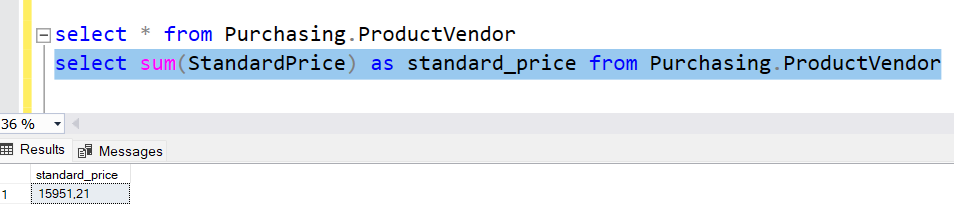


### Counting individual columns

Count columna cuenta solo los que no son nulos de la columna.



### Sql sum

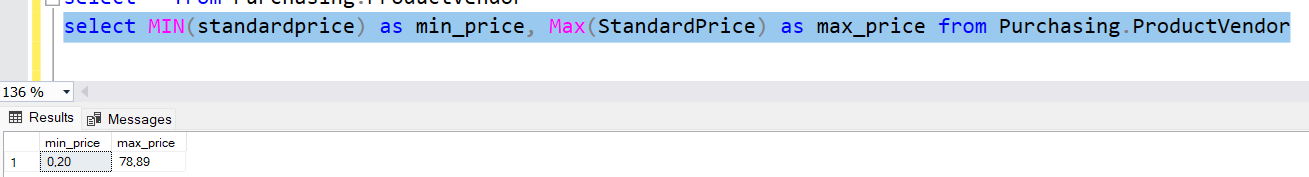


### Min and Max

Pueden ser usados con no numerical columns,

MIN will return the lowest number, earliest date, close alphabetically to "A" as possible.

MAX returns the highest number, the latest date, or the non-numerical value closest alphabetically to "Z."



### AVG

it can only be used on numerical columns

Ignores nulls completely.

SELECT AVG(high)

FROM tutorial.aapl\_historical\_stock\_price

WHERE high IS NOT NULL

Es lo mismo

SELECT AVG(high)

FROM tutorial.aapl\_historical\_stock\_price

### Group by

SELECT year,

COUNT(\*) AS count

FROM tutorial.aapl\_historical\_stock\_price

GROUP BY year

SELECT year,

month,

COUNT(\*) AS count

FROM tutorial.aapl\_historical\_stock\_price

GROUP BY year, month

### GROUP BY column numbers

SELECT year,

month,

COUNT(\*) AS count

FROM tutorial.aapl\_historical\_stock\_price

GROUP BY 1, 2

### Using GROUP BY with ORDER BY

select year, month, count(\*) as count

from aapl\_historical\_stock\_price

group by year, month

order by year, month

### HAVING

is the 'clean' way to filter a query that has been agregated,

but this is also commonly done using a subquery

select year, month, max(high) as month\_high

from aapl\_historical\_stock\_price

group by year, month

having max(high) > 400

order by year, month

### query clause order

group by

having

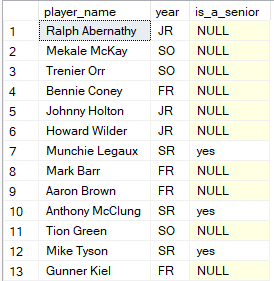
order by

### CASE

select player\_name, year,

case when year = 'SR' then 'yes' else null end as is\_a\_senior

from college\_football\_players



### Adding multiple conditions to a CASE statement

select player\_name, year,

weight,

case when weight > 250 then 'over 250'

when weight > 200 AND weight <= 250 then '201 - 250'

when weight > 175 AND weight <= 200 then '176 - 200'

else '175 or under' end as weight\_group

from college\_football\_players



select player\_name,

case when year = 'FR' and position = 'WR' then 'frosh\_wr'

else null end as sample\_case\_statement

from college\_football\_players

Tabla

Descripción generada automáticamente

Write a query that selects all columns from benn.college\_football\_players

and adds an additional column that displays the player's name if that player is a junior or senior.

select full\_school\_name, school\_name, player\_name, position,

height, weight, year, hometown, state, id,

case when year = 'SR' or year = 'JR' then player\_name

ELSE null end as player\_name

from college\_football\_players



### Using CASE with aggregate functions

-- para poder contar también los nulos

select

case when year = 'FR' then 'FR' else 'NOT FR' END AS year\_group,

count(1) as count

from college\_football\_players

group by case when year = 'FR' then 'FR' else 'NOT FR' END

select year, count(1) as count

from college\_football\_players

group by year

--counting multiple conditions

select case when year = 'FR' then 'FR'

when year = 'SO' then 'SO'

when year = 'JR' then 'JR'

when year = 'SR' then 'SR'

ELSE 'NO YEAR DATA' END AS year\_group,

count(1) as count

from college\_football\_players

group by case when year = 'FR' then 'FR'

when year = 'SO' then 'SO'

when year = 'JR' then 'JR'

when year = 'SR' then 'SR'

ELSE 'NO YEAR DATA' END

/\*

Write a query that counts the number of 300lb+ players for each of the following regions:

West Coast (CA, OR, WA), Texas, and Other (everywhere else).

\*/

select

case when state IN ('CA', 'OR', 'WA') THEN 'West Coast'

when state = 'TX' THEN 'TEXAS'

ELSE 'OTHER' end AS regions,

count(1) count

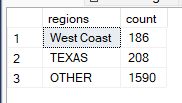
from college\_football\_players

where weight >= 300

group by case when state IN ('CA', 'OR', 'WA') THEN 'West Coast'

when state = 'TX' THEN 'TEXAS'

ELSE 'OTHER' end



--Write a query that calculates the combined weight of all underclass players (FR/SO) in California

--as well as the combined weight of all upperclass players (JR/SR) in California.

select

case when year in ('FR','SO') THEN 'underclass players'

when year in ('JR','SR') THEN 'upperclass players'

else '' end as players,

sum(weight) combined\_weight

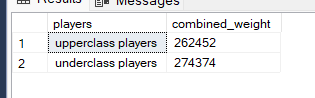
from college\_football\_players

where state = 'CA'

group by case when year in ('FR','SO') THEN 'underclass players'

when year in ('JR','SR') THEN 'upperclass players'

else '' end



--Using CASE inside of aggregate functions

--want to show data horizontally.

SELECT CASE WHEN year = 'FR' THEN 'FR'

WHEN year = 'SO' THEN 'SO'

WHEN year = 'JR' THEN 'JR'

WHEN year = 'SR' THEN 'SR'

ELSE 'No Year Data' END AS year\_group,

COUNT(1) AS count

FROM college\_football\_players

GROUP BY CASE WHEN year = 'FR' THEN 'FR'

WHEN year = 'SO' THEN 'SO'

WHEN year = 'JR' THEN 'JR'

WHEN year = 'SR' THEN 'SR'

ELSE 'No Year Data' END

Tabla

Descripción generada automáticamente

--And re-orient it horizontally

select count(case when year = 'FR' THEN 'FR' ELSE NULL END ) as fr\_count,

count(case when year = 'SO' THEN 'SO' ELSE NULL END ) as so\_count,

count(case when year = 'JR' THEN 'JR' ELSE NULL END ) as jr\_count,

count(case when year = 'SR' THEN 'SR' ELSE NULL END ) as sr\_count

FROM college\_football\_players

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

--Write a query that displays the number of players in each state, with FR, SO, JR, and SR players in separate columns

--and another column for the total number of players.

--Order results such that states with the most players come first.

SELECT

state ,

COUNT(CASE WHEN year = 'FR' then 'FR' else null end) as fr\_count,

COUNT(CASE WHEN year = 'SO' then 'SO' else null end) as so\_count,

COUNT(CASE WHEN year = 'JR' then 'JR' else null end) as jr\_count,

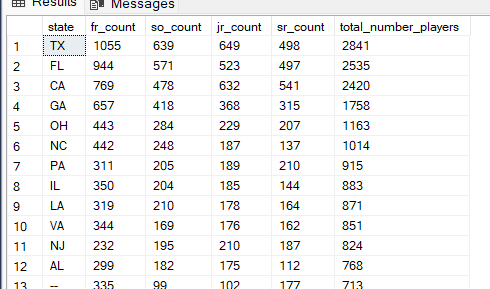
COUNT(CASE WHEN year = 'SR' then 'SR' else null end) as sr\_count,

count(1) total\_number\_players

FROM college\_football\_players

group by state

order by total\_number\_players desc



--Write a query that shows the number of players at schools with names that start with A through M,

--and the number at schools with names starting with N - Z.

select

count(case when school\_name < 'n' then school\_name ELSE NULL end) as 'A-M',

count(case when school\_name >= 'n' then school\_name ELSE NULL end) as 'N-Z'

FROM college\_football\_players

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

### Using SQL DISTINCT for viewing unique values

--when you want to look at only the unique values in a particular column

SELECT DISTINCT month

FROM aapl\_historical\_stock\_price

SELECT DISTINCT year, month

FROM aapl\_historical\_stock\_price

order by year, month

--Write a query that returns the unique values in the year column, in chronological order.

select distinct year

from aapl\_historical\_stock\_price

order by year

### Using DISTINCT in aggregations

--counts the unique values in the month

select count(distinct month) as unique\_month

from aapl\_historical\_stock\_price

### DISTINCT performance

--using DISTINCT, particularly in aggregations, can slow your queries down quite a bit.

--Write a query that counts the number of unique values in the month column for each year.

select year,

count(distinct month) as month\_count

FROM aapl\_historical\_stock\_price

group by year

Imagen que contiene Tabla

Descripción generada automáticamente

--Write a query that separately counts the number of unique values in the month column and

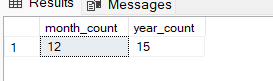
--the number of unique values in the `year` column.

select

count(distinct month) as month\_count,

count(distinct year) as year\_count

FROM aapl\_historical\_stock\_price



## Joins

### Inner Join

 inner join is the *intersection* of the two tables.

--Write a query that displays player names, school names and conferences for schools in the "FBS (Division I-A Teams)" division.

select players.player\_name, players.school\_name, teams.conference

from college\_football\_players players

inner join college\_football\_teams teams

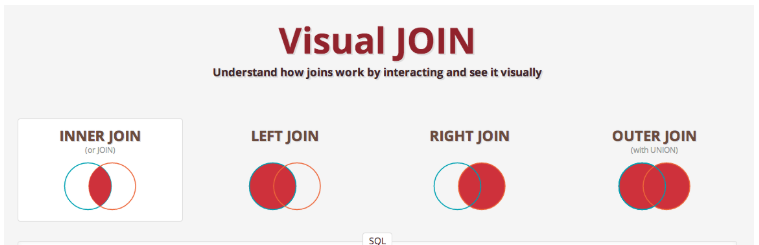
on players.school\_name = teams.school\_name

where division = 'FBS (Division I-A Teams)'

Outer joins

Outer joins are joins that return matched values and unmatched values from either or both tables. There are a few types of outer joins:

* [LEFT JOIN returns only unmatched rows from the left table](https://mode.com/sql-tutorial/sql-left-join), as well as matched rows in both tables.
* [RIGHT JOIN returns only unmatched rows from the right table](https://mode.com/sql-tutorial/sql-right-join) , as well as matched rows in both tables.
* [FULL OUTER JOIN returns unmatched rows from both tables,](https://mode.com/sql-tutorial/sql-full-outer-join)as well as matched rows in both tables.



select \* from crunchbase\_acquisitions

where company\_permalink = '/company/280-north'



SELECT companies.permalink AS companies\_permalink,

companies.name AS companies\_name,

acquisitions.company\_permalink AS acquisitions\_permalink,

acquisitions.acquired\_at AS acquired\_date

FROM crunchbase\_companies companies

JOIN crunchbase\_acquisitions acquisitions

ON companies.permalink = acquisitions.company\_permalink

where companies.permalink = '/company/280-north'

Tabla

Descripción generada automáticamente

SELECT companies.permalink AS companies\_permalink,

companies.name AS companies\_name,

acquisitions.company\_permalink AS acquisitions\_permalink,

acquisitions.acquired\_at AS acquired\_date

FROM crunchbase\_companies companies

LEFT JOIN crunchbase\_acquisitions acquisitions

ON companies.permalink = acquisitions.company\_permalink



--Write a query that performs an inner join between the tutorial.crunchbase\_acquisitions table and the tutorial.crunchbase\_companies table,

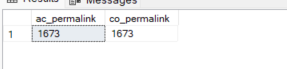
--but instead of listing individual rows, count the number of non-null rows in each table.

select count(ac.company\_permalink) ac\_permalink , count(co.permalink) as co\_permalink

from crunchbase\_acquisitions ac

inner join crunchbase\_companies co

on co.permalink = ac.company\_permalink



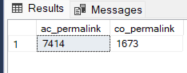
--Modify the query above to be a LEFT JOIN. Note the difference in results.

select count(ac.company\_permalink) ac\_permalink , count(co.permalink) as co\_permalink

from crunchbase\_acquisitions ac

left join crunchbase\_companies co

on co.permalink = ac.company\_permalink



--Count the number of unique companies (don't double-count companies) and unique acquired companies by state.

--Do not include results for which there is no state data,

--and order by the number of acquired companies from highest to lowest.

select company\_state\_code, count(distinct company\_name) as company\_name, count(distinct name) as ac\_name

from crunchbase\_companies co

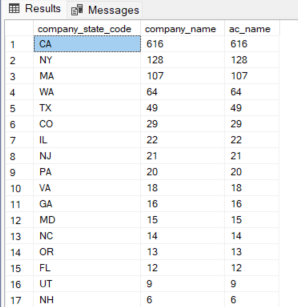
inner join crunchbase\_acquisitions ac

on co.permalink = ac.company\_permalink

where company\_state\_code <> ''

group by company\_state\_code

order by ac\_name desc



--Rewrite the previous practice query in which you counted total and acquired companies by state,

--but with a RIGHT JOIN instead of a LEFT JOIN. The goal is to produce the exact same results.

select company\_state\_code, count(distinct company\_name) as company\_name, count(distinct name) as ac\_name

from crunchbase\_acquisitions ac

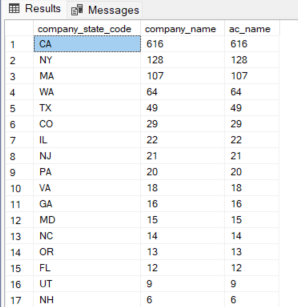
right join crunchbase\_companies co

on ac.company\_permalink = co.permalink

where company\_state\_code <> ''

group by company\_state\_code

order by ac\_name desc



### Filtering in the ON clause

SELECT companies.permalink AS companies\_permalink,

companies.name AS companies\_name,

acquisitions.company\_permalink AS acquisitions\_permalink,

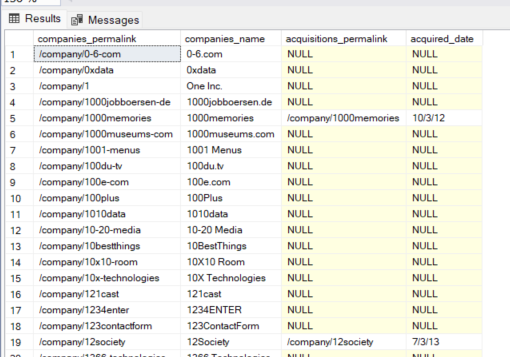
acquisitions.acquired\_at AS acquired\_date

FROM crunchbase\_companies companies

LEFT JOIN crunchbase\_acquisitions acquisitions

ON companies.permalink = acquisitions.company\_permalink

ORDER BY 1



SELECT companies.permalink AS companies\_permalink,

companies.name AS companies\_name,

acquisitions.company\_permalink AS acquisitions\_permalink,

acquisitions.acquired\_at AS acquired\_date

FROM crunchbase\_companies companies

LEFT JOIN crunchbase\_acquisitions acquisitions

ON companies.permalink = acquisitions.company\_permalink

AND acquisitions.company\_permalink != '/company/1000memories'

ORDER BY 1

### El AND se ejecuta antes del join, es como un where para una sola tabla

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

--Filtering in the WHERE clause

SELECT companies.permalink AS companies\_permalink,

companies.name AS companies\_name,

acquisitions.company\_permalink AS acquisitions\_permalink,

acquisitions.acquired\_at AS acquired\_date

FROM crunchbase\_companies companies

LEFT JOIN crunchbase\_acquisitions acquisitions

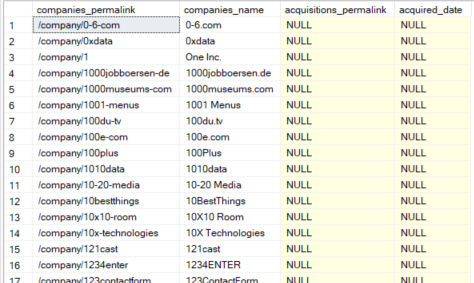
ON companies.permalink = acquisitions.company\_permalink

WHERE acquisitions.company\_permalink != '/company/1000memories'

OR acquisitions.company\_permalink IS NULL

ORDER BY 1

filter happens after the tables are joined



--Write a query that shows a company's name, "status" (found in the Companies table),

--and the number of unique investors in that company.

--Order by the number of investors from most to fewest.

--Limit to only companies in the state of New York.

select co.name as company, co.status, count(distinct inv.investor\_name) AS count\_inversors

from crunchbase\_companies co

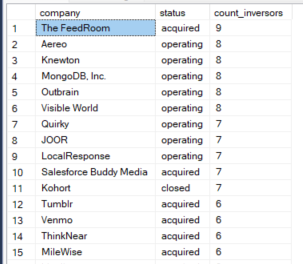
left join crunchbase\_investments inv

on co.permalink = inv.company\_permalink

where co.state\_code = 'NY'

group by co.name, co.status

order by 3 desc



-- Write a query that lists investors based on the number of companies in which they are invested.

-- Include a row for companies with no investor,

-- and order from most companies to least.

select

case when investor\_name is null then 'No Inversors' else investor\_name end as inversor,

count(distinct co.permalink) as count\_companies

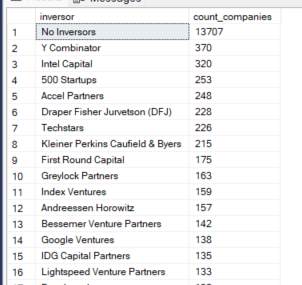
from crunchbase\_companies co

left join crunchbase\_investments inv

on co.permalink = inv.company\_permalink

group by investor\_name

order by 2 desc



### The SQL FULL JOIN command

Returns unmatched rows from both tables.

SELECT COUNT(CASE WHEN companies.permalink IS NOT NULL AND acquisitions.company\_permalink IS NULL

THEN companies.permalink ELSE NULL END) AS companies\_only,

COUNT(CASE WHEN companies.permalink IS NOT NULL AND acquisitions.company\_permalink IS NOT NULL

THEN companies.permalink ELSE NULL END) AS both\_tables,

COUNT(CASE WHEN companies.permalink IS NULL AND acquisitions.company\_permalink IS NOT NULL

THEN acquisitions.company\_permalink ELSE NULL END) AS acquisitions\_only

FROM crunchbase\_companies companies

FULL JOIN crunchbase\_acquisitions acquisitions

ON companies.permalink = acquisitions.company\_permalink



--Write a query that joins crunchbase\_companies and crunchbase\_investments\_part1 using a FULL JOIN.

--Count up the number of rows that are matched/unmatched as in the example above.

SELECT COUNT(CASE WHEN co.permalink IS NOT NULL AND inv.company\_permalink IS NULL

THEN co.permalink ELSE NULL END) AS companies\_only,

COUNT(CASE WHEN co.permalink IS NOT NULL AND inv.company\_permalink IS NOT NULL

THEN co.permalink ELSE NULL END) AS both\_tables,

COUNT(CASE WHEN co.permalink IS NULL AND inv.company\_permalink IS NOT NULL

THEN inv.company\_permalink ELSE NULL END) AS investments\_only

FROM crunchbase\_companies co

FULL JOIN crunchbase\_investments inv

ON co.permalink = inv.company\_permalink

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

### The SQL UNION operator

UNION allows you to write two separate SELECT statements,

and to have the results of one statement display in the same table as the results from the other statement.

UNION only appends distinct values.

 If you'd like to append all the values from the second table, use UNION ALL.

SQL has strict rules for appending data:

1. Both tables must have the same number of columns
2. The columns must have the same data types in the same order as the first table

--Write a query that appends the two crunchbase\_investments datasets above (including duplicate values).

--Filter the first dataset to only companies with names that start with the letter "T",

--and filter the second to companies with names starting with "M" (both not case-sensitive).

--Only include the company\_permalink, company\_name, and investor\_name columns.

select co.permalink, co.name, inv.investor\_name

FROM crunchbase\_companies co

inner JOIN crunchbase\_investments inv

ON co.permalink = inv.company\_permalink

and co.name >= 'M'

and co.name < 'N'

UNION ALL

select co.permalink, co.name, inv.investor\_name

FROM crunchbase\_companies co

inner JOIN crunchbase\_investments inv

ON co.permalink = inv.company\_permalink

and co.name >= 'T'

and co.name < 'U'

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

select 'DATA1' as dataset, co.status , count(distinct inv.investor\_name) as count\_investors

FROM crunchbase\_companies co

left JOIN crunchbase\_investments inv

ON co.permalink = inv.company\_permalink

group by co.status

union all

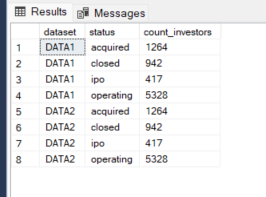
select 'DATA2' as dataset, co.status , count(distinct inv.investor\_name) as count\_investors

FROM crunchbase\_companies co

left JOIN crunchbase\_investments inv

ON co.permalink = inv.company\_permalink

group by co.status



### Using comparison operators with joins

--using > to join only investments that occurred more than 5 years after each company's founding year:

SELECT companies.permalink,

companies.name,

companies.status,

COUNT(investments.investor\_permalink) AS investors

FROM crunchbase\_companies companies

LEFT JOIN crunchbase\_investments investments

ON companies.permalink = investments.company\_permalink

AND investments.funded\_year > companies.founded\_year + 5

GROUP BY companies.permalink,

companies.name,

companies.status

--Joining on multiple keys

--the results of the following query will be the same with or without the last line.

--However, it is possible to optimize the database such that the query runs more quickly with the last line included:

SELECT companies.permalink,

companies.name,

investments.company\_name,

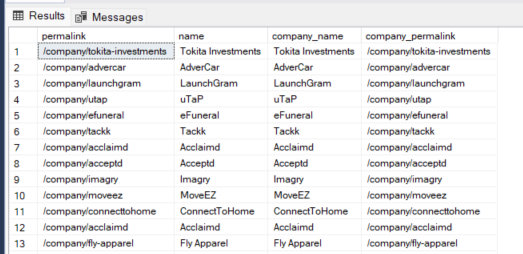
investments.company\_permalink

FROM crunchbase\_companies companies

LEFT JOIN crunchbase\_investments investments

ON companies.permalink = investments.company\_permalink

AND companies.name = investments.company\_name



## Self joining tables

--Identify companies that received an investment from Great Britain following an investment from Japan.

SELECT DISTINCT japan\_investments.company\_name,

japan\_investments.company\_permalink

FROM crunchbase\_investments japan\_investments

JOIN crunchbase\_investments gb\_investments

ON japan\_investments.company\_name = gb\_investments.company\_name

AND gb\_investments.investor\_country\_code = 'GBR'

AND gb\_investments.funded\_at > japan\_investments.funded\_at

WHERE japan\_investments.investor\_country\_code = 'JPN'

ORDER BY 1

# [Advanced SQL](https://mode.com/sql-tutorial/intro-to-advanced-sql)

## Changing a column's data type

--Changing a column's data type

--can use CAST or CONVERT to change the data type

--Using SQL String Functions to Clean Data

SELECT \* FROM sf\_crime\_incidents\_2014\_01

--Cleaning strings

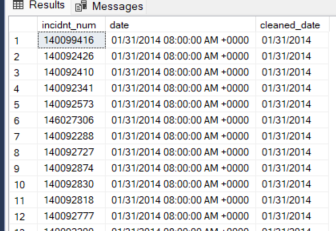
--LEFT, RIGHT, and LENGTH

SELECT incidnt\_num,

date,

LEFT(date, 10) AS cleaned\_date

FROM sf\_crime\_incidents\_2014\_01



--RIGHT cuenta desde atrás para adelante

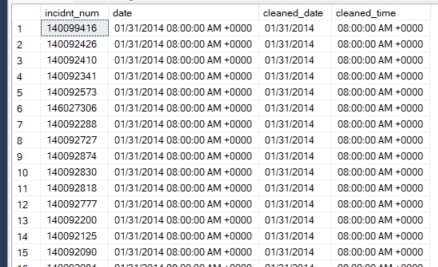
SELECT incidnt\_num,

date,

LEFT(date, 10) AS cleaned\_date,

RIGHT(date, 17) AS cleaned\_time

FROM sf\_crime\_incidents\_2014\_01



--The LEN function returns the length of a string

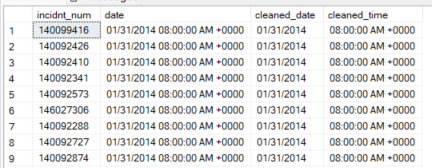
SELECT incidnt\_num,

date,

LEFT(date, 10) AS cleaned\_date,

RIGHT(date, LEN(date) - 11) AS cleaned\_time

FROM sf\_crime\_incidents\_2014\_01



-- TRIM

--TRIM( \\[ [{LEADING | TRAILING | BOTH}\\] [removal\_character]

--FROM ] target\_string )

--LEADING, TRAILING, or BOTH defines whether to remove the removal\_character at the beginning, end,

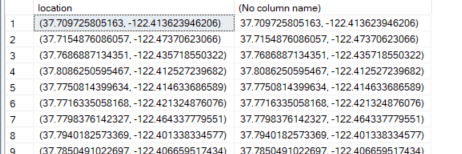
--or on both sides of target\_string, respectively.

--If omitted, this defaults to BOTH.

SELECT location,

TRIM(both '()' FROM location)

FROM sf\_crime\_incidents\_2014\_01



SELECT TRIM('#! ' FROM ' #SQL Tutorial! ') AS TrimmedString;



--CHARINDEX -- EN VEZ DE POSITION and STRPOS

--position and strpos no son soportados por sql.

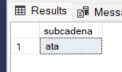
-- SELECT POSITION( 'DELHI' IN 'NEW DELHI') AS POSITION\_DELHI;

SELECT CHARINDEX('a', 'database' ) AS position



--SUBSTRING EN VEZ DE SUBSTR

SELECT SUBSTRING('database', 2, 3) as subcadena;



--Write a query that separates the `location` field into separate fields for latitude and longitude.

--You can compare your results against the actual `lat` and `lon` fields in the table.

select location,

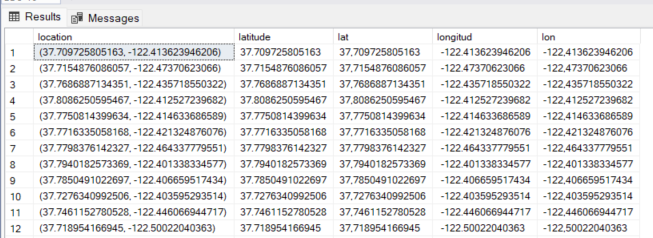
substring(location, 2, CHARINDEX(',', location)-2) as latitude,

lat,

substring(location, CHARINDEX(',', location)+1, len(location)-(CHARINDEX(',', location)+1) ) as longitud,

lon

FROM sf\_crime\_incidents\_2014\_01



--CONCAT

SELECT CONCAT('W3Schools', '.com');

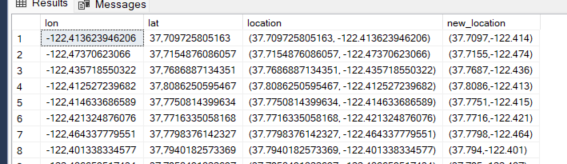


--Concatenate the lat and lon fields to form a field that is equivalent to the location field. (Note that the answer will have a different decimal precision.)

select lon, lat, location,

concat('(',lat, ',', lon, ')') as new\_location

FROM sf\_crime\_incidents\_2014\_01



--Write a query that creates a date column formatted YYYY-MM-DD.

select date,

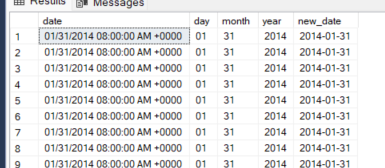
substring(date, 1,2) as day,

substring(date, 4,2) as month,

substring(date, 7,4) as year,

concat(substring(date, 7,4), '-', substring(date, 1,2), '-', substring(date, 4,2)) as new\_date

FROM sf\_crime\_incidents\_2014\_01



SELECT incidnt\_num,

address,

UPPER(address) AS address\_upper,

LOWER(address) AS address\_lower

FROM sf\_crime\_incidents\_2014\_01

Tabla

Descripción generada automáticamente

-- COALESCE

-- replace the nulls values

SELECT incidnt\_num,

descript,

COALESCE(descript, 'No Description')

FROM tutorial.sf\_crime\_incidents\_cleandate

ORDER BY descript DESC

--The COALESCE() function returns the first non-null value in a list.

SELECT COALESCE(NULL, NULL, NULL, 'W3Schools.com', NULL, 'Example.com');



SELECT

first\_name,

last\_name,

COALESCE(marital\_status,'Unknown')

FROM persons

## Subquery basics

--Write a query that selects all Warrant Arrests from the sf\_crime\_incidents\_2014\_01 dataset,

--then wrap it in an outer query that only displays unresolved incidents.

select arrests.\*

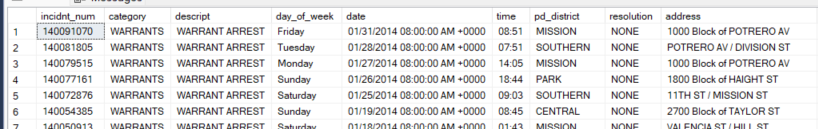
from (

select \* from sf\_crime\_incidents\_2014\_01

where descript = 'WARRANT ARREST'

) as arrests

where arrests.resolution = 'NONE'



--Write a query that displays the average number of monthly incidents for each category.

--Hint: use tutorial.sf\_crime\_incidents\_cleandate to make your life a little easier.

select incidents.category, AVG(incidents.cantidad) as average

from (

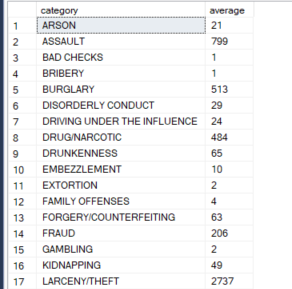
select category, substring(date, 1, 2) as month, count(\*) as cantidad

from sf\_crime\_incidents\_cleandate

group by category, substring(date, 1, 2)

) as incidents

group by incidents.category



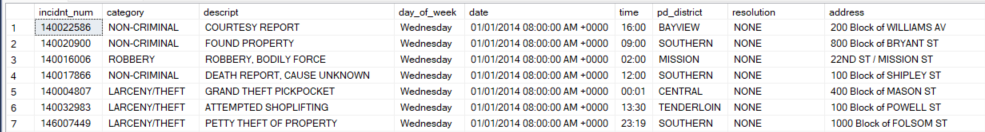
### Subqueries in conditional logic

--all of the entries from the earliest date

select \*

from sf\_crime\_incidents\_2014\_01

where date = ( select min(date) from sf\_crime\_incidents\_2014\_01)



--Most conditional logic will work with subqueries containing one-cell results

select \*

from sf\_crime\_incidents\_2014\_01

where date in (

select top 5 date

from sf\_crime\_incidents\_2014\_01

order by date

)

--Joining subqueries

--es la misma query que la anterior.

select \*

from sf\_crime\_incidents\_2014\_01 incidents

join (

select top 5 date

from sf\_crime\_incidents\_2014\_01

order by date

)sub

on incidents.date = sub.date

select incidents.\*,

sub.incidents as incidents\_that\_day

from sf\_crime\_incidents\_2014\_01 incidents

join (

select date,

count(incidnt\_num) as incidents

from sf\_crime\_incidents\_2014\_01

group by date

) sub

on incidents.date = sub.date

order by sub.incidents desc, time

Interfaz de usuario gráfica, Tabla

Descripción generada automáticamente

--Write a query that displays all rows from the three categories with the fewest incidents reported.

select inc.\*, sub.incident\_number

from sf\_crime\_incidents\_2014\_01 as inc

join (

select top 3 category, count(incidnt\_num) as incident\_number

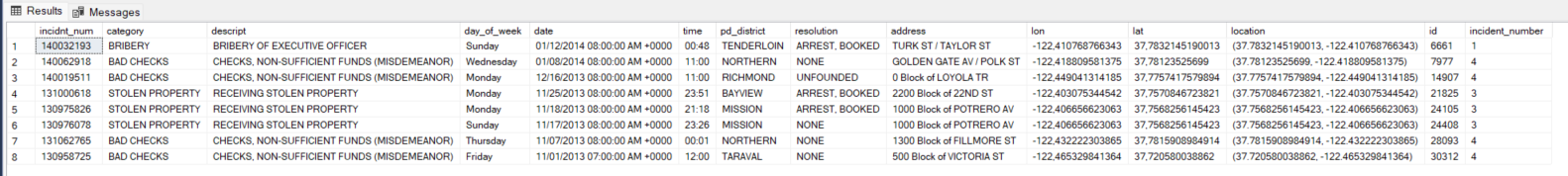
from sf\_crime\_incidents\_2014\_01

group by category

order by count(incidnt\_num)

)sub

on inc.category = sub.category



--Write a query that counts the number of companies founded and acquired by quarter starting in Q1 2012.

--Create the aggregations in two separate queries, then join them.

select coalesce(com.founded\_quarter,ac.acquired\_quarter) as quarter, com.count\_founded\_quarter, ac.count\_acq\_quarter

from

(

select founded\_quarter, count(founded\_quarter) as count\_founded\_quarter

from crunchbase\_companies

where founded\_quarter >= '2012-Q1'

group by founded\_quarter

) as com

join (

select acquired\_quarter, count(acquired\_quarter) as count\_acq\_quarter

from crunchbase\_acquisitions

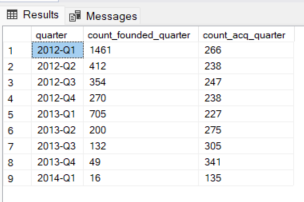
where acquired\_quarter >= '2012-Q1'

group by acquired\_quarter

) AS ac

on ac.acquired\_quarter = com.founded\_quarter

order by 1



--Write a query that ranks investors from the combined dataset above by the total number of investments they have made.

select investor\_name, count(investor\_name) as total\_number\_investments

from (

select \*

from crunchbase\_investments

union all

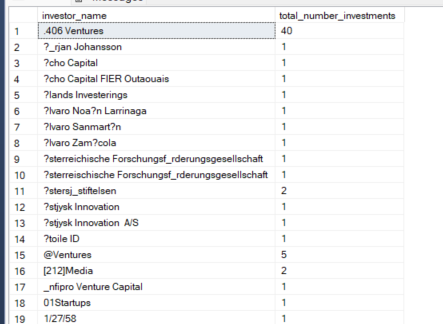
select \*

from crunchbase\_investments\_part2

)as inv

group by investor\_name

order by investor\_name



--Write a query that does the same thing as in the previous problem,

--except only for companies that are still operating.

--Hint: operating status is in tutorial.crunchbase\_companies.

select investor\_name, count(investor\_name) as total\_number\_investments

from (

select investor\_name

from crunchbase\_investments inv

inner join crunchbase\_companies co

on co.permalink = inv.company\_permalink

where status = 'operating'

union all

select investor\_name

from crunchbase\_investments\_part2 inv2

inner join crunchbase\_companies com

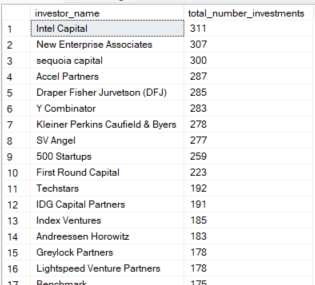
on com.permalink = inv2.company\_permalink

where status = 'operating'

) as invest

group by investor\_name

order by 2 desc



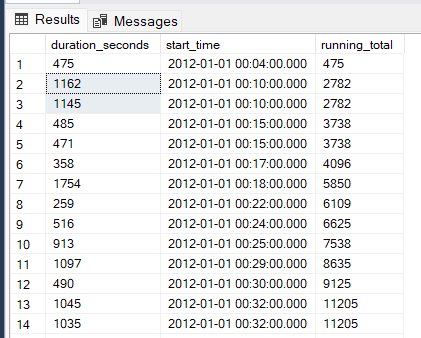
## Window Functions

SELECT duration\_seconds, start\_time,

SUM(duration\_seconds) OVER (ORDER BY start\_time) AS running\_total

FROM dc\_bikeshare\_q1\_2012

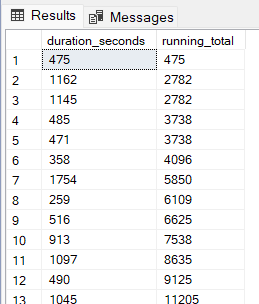
order by start\_time



SELECT duration\_seconds,

SUM(duration\_seconds) OVER (ORDER BY start\_time) AS running\_total

FROM dc\_bikeshare\_q1\_2012



--suma los segundos agrupados por terminal y ordenados por start\_time

select start\_terminal, start\_time, duration\_seconds,

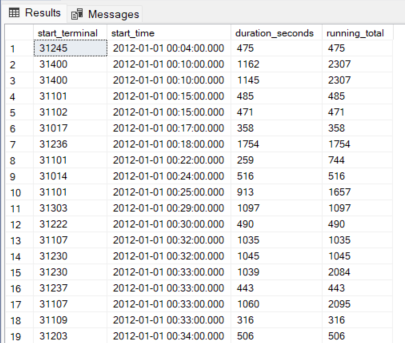
sum(duration\_seconds) over

(partition by start\_terminal order by start\_time) as running\_total

from dc\_bikeshare\_q1\_2012

where start\_time < '2012-01-08'

order by start\_time



-- you can't include window functions in a GROUP BY clause.

--Write a query modification of the above example query that

--shows the duration of each ride as a percentage of the total time accrued by riders

--from each start\_terminal

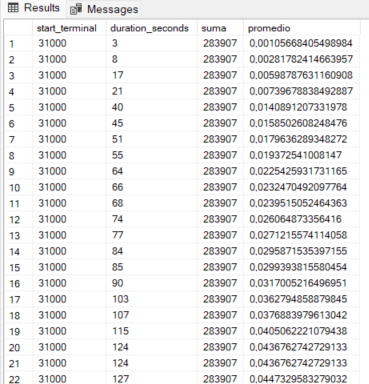
select start\_terminal, duration\_seconds,

sum(duration\_seconds) over(partition by start\_terminal) as suma,

(duration\_seconds / sum(duration\_seconds) over(partition by start\_terminal ) ) \*100 as promedio

from dc\_bikeshare\_q1\_2012

order by 1,4



SELECT start\_terminal,

duration\_seconds,

SUM(duration\_seconds) OVER

(PARTITION BY start\_terminal) AS running\_total,

COUNT(duration\_seconds) OVER

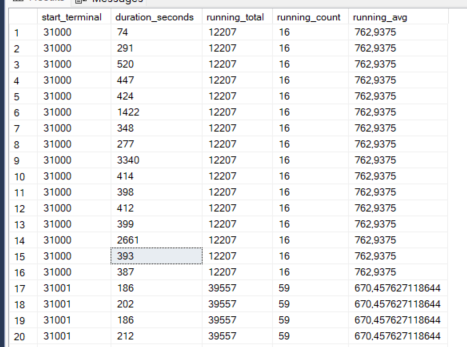
(PARTITION BY start\_terminal) AS running\_count,

AVG(duration\_seconds) OVER

(PARTITION BY start\_terminal) AS running\_avg

FROM dc\_bikeshare\_q1\_2012

WHERE start\_time < '2012-01-08'



-- with order by

SELECT start\_terminal,

start\_time,

duration\_seconds,

SUM(duration\_seconds) OVER

(PARTITION BY start\_terminal ORDER BY start\_time)

AS running\_total,

COUNT(duration\_seconds) OVER

(PARTITION BY start\_terminal ORDER BY start\_time)

AS running\_count,

AVG(duration\_seconds) OVER

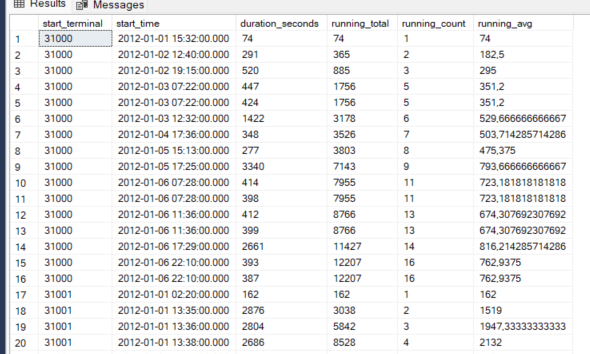
(PARTITION BY start\_terminal ORDER BY start\_time)

AS running\_avg

FROM dc\_bikeshare\_q1\_2012

WHERE start\_time < '2012-01-08'

order by 1, 2



-- Write a query that shows a running total of the duration of bike rides (similar to the last example),

--but grouped by end\_terminal,

--and with ride duration sorted in descending order.

SELECT end\_terminal,

duration\_seconds,

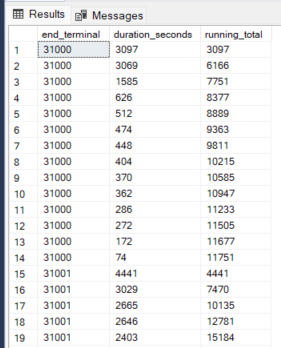
SUM(duration\_seconds) OVER

(PARTITION BY end\_terminal ORDER BY duration\_seconds desc)

AS running\_total

FROM dc\_bikeshare\_q1\_2012

WHERE start\_time < '2012-01-08'



--ROW\_NUMBER()

Select

start\_terminal,

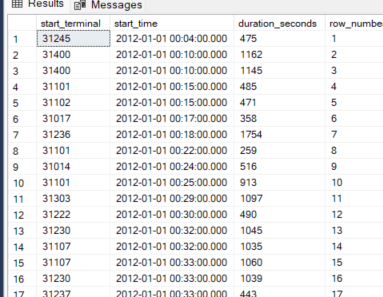
start\_time,

duration\_seconds,

row\_number() over (order by start\_time) as row\_number

FROM dc\_bikeshare\_q1\_2012

WHERE start\_time < '2012-01-08'



Select

start\_terminal,

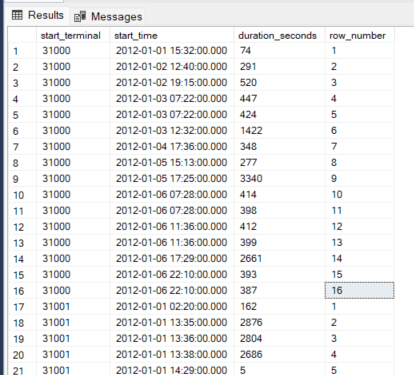
start\_time,

duration\_seconds,

row\_number() over (partition by start\_terminal order by start\_time) as row\_number

FROM dc\_bikeshare\_q1\_2012

WHERE start\_time < '2012-01-08'



### RANK() and DENSE\_RANK()

Select

start\_terminal,

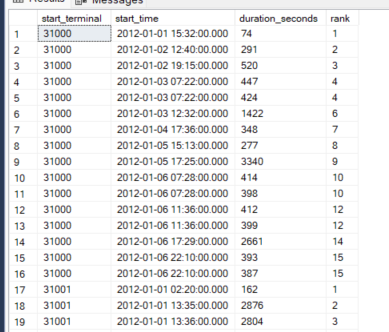
start\_time,

duration\_seconds,

rank() over (partition by start\_terminal order by start\_time) as rank

FROM dc\_bikeshare\_q1\_2012

WHERE start\_time < '2012-01-08'



Select

start\_terminal,

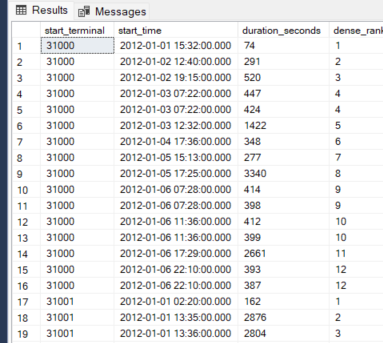
start\_time,

duration\_seconds,

dense\_rank() over (partition by start\_terminal order by start\_time) as dense\_rank

FROM dc\_bikeshare\_q1\_2012

WHERE start\_time < '2012-01-08'



--Write a query that shows the 5 longest rides from each starting terminal,

--ordered by terminal,

--and longest to shortest rides within each terminal.

--Limit to rides that occurred before Jan. 8, 2012.

select \*

from

(

select start\_terminal,

duration\_seconds,

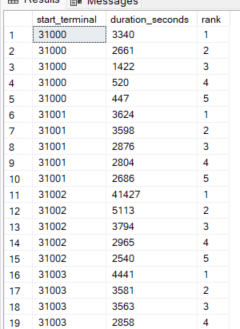
rank() over (partition by start\_terminal order by duration\_seconds desc ) as rank

FROM dc\_bikeshare\_q1\_2012

WHERE start\_time < '2012-01-08'

) sub

where sub.rank <= 5



--NTILE

-- divide la partición en el número que tiene entre parentesis.

SELECT start\_terminal,

duration\_seconds,

NTILE(4) OVER

(PARTITION BY start\_terminal ORDER BY duration\_seconds)

AS quartile,

NTILE(5) OVER

(PARTITION BY start\_terminal ORDER BY duration\_seconds)

AS quintile,

NTILE(100) OVER

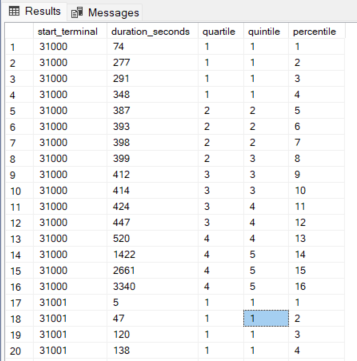
(PARTITION BY start\_terminal ORDER BY duration\_seconds)

AS percentile

FROM dc\_bikeshare\_q1\_2012

WHERE start\_time < '2012-01-08'

ORDER BY start\_terminal, duration\_seconds



-- Write a query that shows only the duration of the trip and

--the percentile into which that duration falls (across the entire dataset—not partitioned by terminal).

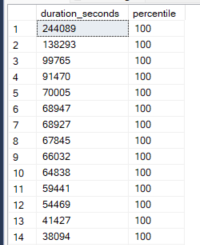
select duration\_seconds,

NTILE(100) over(order by duration\_seconds) as percentile

FROM dc\_bikeshare\_q1\_2012

WHERE start\_time < '2012-01-08'

order by 1 desc



-- LAG and LEAD

-- lag pone el registro anterior y lead el registro posterior.

SELECT start\_terminal,

duration\_seconds,

LAG(duration\_seconds, 1) OVER

(PARTITION BY start\_terminal ORDER BY duration\_seconds) AS lag,

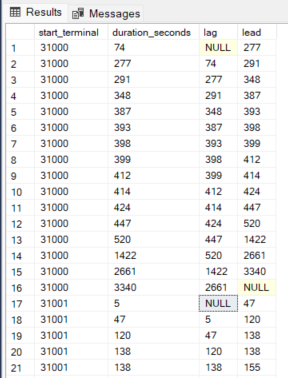
LEAD(duration\_seconds, 1) OVER

(PARTITION BY start\_terminal ORDER BY duration\_seconds) AS lead

FROM dc\_bikeshare\_q1\_2012

WHERE start\_time < '2012-01-08'

ORDER BY start\_terminal, duration\_seconds



--This is especially useful if you want to calculate differences between rows:

SELECT start\_terminal,

duration\_seconds,

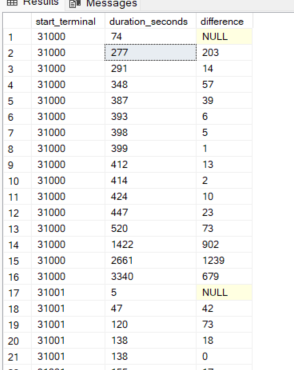
duration\_seconds -

LAG(duration\_seconds, 1) OVER (PARTITION BY start\_terminal ORDER BY duration\_seconds) AS difference

FROM dc\_bikeshare\_q1\_2012

WHERE start\_time < '2012-01-08'

ORDER BY start\_terminal, duration\_seconds



--Defining a window alias

SELECT start\_terminal,

duration\_seconds,

NTILE(4) OVER ntile\_window AS quartile,

NTILE(5) OVER ntile\_window AS quintile,

NTILE(100) OVER ntile\_window AS percentile

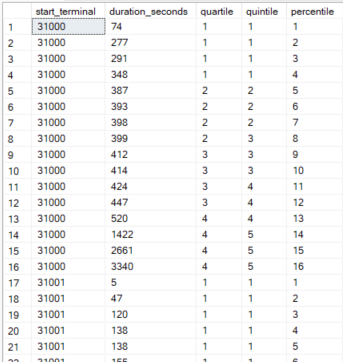
FROM dc\_bikeshare\_q1\_2012

WHERE start\_time < '2012-01-08'

WINDOW ntile\_window AS

(PARTITION BY start\_terminal ORDER BY duration\_seconds)

ORDER BY start\_terminal, duration\_seconds



--Pivoting rows to columns

SELECT teams.conference AS conference,

players.year,

COUNT(1) AS players

FROM college\_football\_players players

JOIN college\_football\_teams teams

ON teams.school\_name = players.school\_name

GROUP BY teams.conference, players.year

ORDER BY 1,2



SELECT conference,

SUM(CASE WHEN year = 'FR' THEN players ELSE 0 END) AS fr,

SUM(CASE WHEN year = 'SO' THEN players ELSE 0 END) AS so,

SUM(CASE WHEN year = 'JR' THEN players ELSE 0 END) AS jr,

SUM(CASE WHEN year = 'SR' THEN players ELSE 0 END) AS sr

FROM (

SELECT teams.conference,

players.year,

COUNT(1) AS players

FROM college\_football\_players players

JOIN college\_football\_teams teams

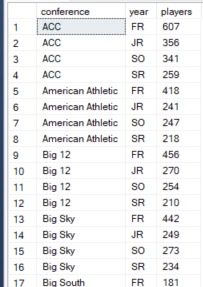
ON teams.school\_name = players.school\_name

GROUP BY teams.conference,players.year

) sub

GROUP BY conference

ORDER BY 1



SELECT conference, [FR], [SO], [JR], [SR]

FROM

(

SELECT teams.conference,

players.year,

COUNT(1) AS players

FROM college\_football\_players players

JOIN college\_football\_teams teams

ON teams.school\_name = players.school\_name

GROUP BY teams.conference,players.year

) AS source

PIVOT

(

SUM(players)

FOR year IN([FR], [SO], [JR], [SR])

) AS pivot\_table;

