**SQL Note**

SELECT Email, COUNT(\*)

FROM user\_log

Where email = “xxxx”

**GROUP** BY Email

**HAVING** COUNT(\*) > 1

ORDER BY UpdateDate ASC|DESC #always last one

You can establish the first condition by including a WHERE clause, which discards any publishers that are not in California, before calculating average prices. The second condition requires a HAVING clause, because the condition is based on the results of grouping and summarizing the data. The resulting SQL statement might look like this:

SELECT titles.pub\_id, AVG(titles.price)

FROM titles INNER JOIN publishers

ON titles.pub\_id = publishers.pub\_id

WHERE publishers.state = 'CA'

GROUP BY titles.pub\_id

HAVING AVG(price) > 10

**Basic windowing syntax**

SELECT start\_terminal, 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,

**ROW\_NUMBER() OVER (PARTITION BY** start\_terminal **ORDER BY** start\_time) AS row\_number,

**RANK() OVER (PARTITION BY** start\_terminal ORDER BY start\_time) AS rank,

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

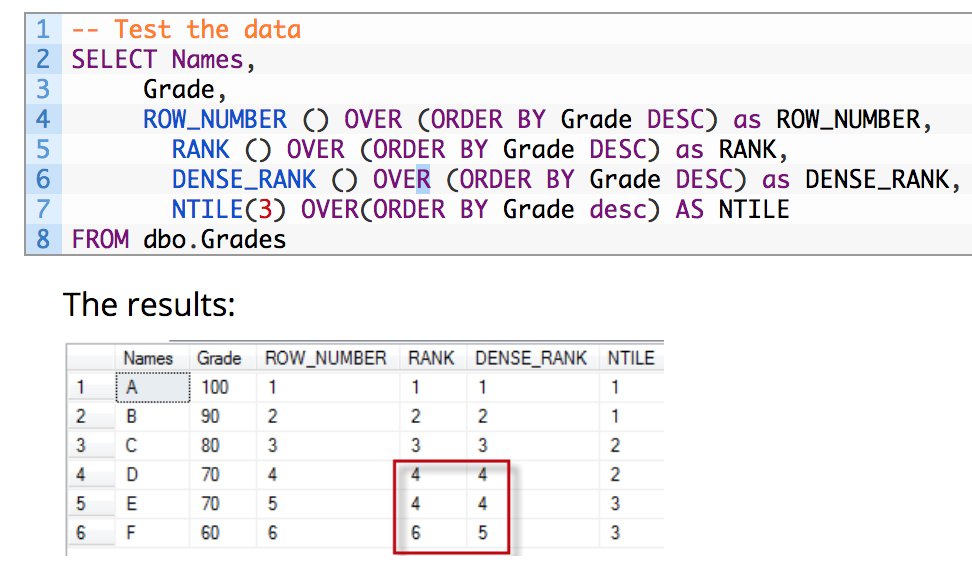
#calculate differences between rows, LAG pulls from previous rows and LEAD pulls from following rows

 FROM table

 WHERE start\_time < '2012-01-08' and difference **IS** **NOT** NULL

You can use LAG or LEAD to create columns that pull values from other rows—all you need to do is enter which column to pull from and how many rows away you’d like to do the pull. LAG pulls from previous rows and LEAD pulls from following rows:

The first row of the difference column is null because there is no previous row from which to pull. Similarly, using LEAD will create nulls at the end of the dataset. If you’d like to make the results a bit cleaner, you can wrap it in an outer query to remove nulls:

****

=================================================================

**CASE WHEN**

SELECT CustomerName, City, Country

FROM Customers

ORDER BY

(CASE WHEN City IS NULL THEN Country ELSE City

END);

======================================================================

SELECT conference,

SUM(players) AS total\_players,

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

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

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

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

FROM (

SELECT teams.conference AS conference,

players.year,

COUNT(1) AS players

FROM benn.college\_football\_players players

JOIN benn.college\_football\_teams teams

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

GROUP BY 1,2

) sub

GROUP BY 1

ORDER BY 2 DESC

========================================================================================

CASE ebv.db\_no

WHEN 22978 THEN 'WECS 9500'

WHEN 23218 THEN 'WECS 9500'

WHEN 23219 THEN 'WECS 9500'

ELSE 'WECS 9520'

END as wecs\_system

Otherwise, use:

CASE

WHEN ebv.db\_no IN (22978, 23218, 23219) THEN 'WECS 9500'

ELSE 'WECS 9520'

END as wecs\_system

 #median within name group

**PERCENTILE\_CONT(0.5) WITHIN GROUP (ORDER BY**response\_time **ASC) OVER(PARTITION BY** name**)** AS median\_time

#95 quantile within total population

**PERCENTILE\_CONT(0.95) WITHIN GROUP (ORDER BY** response\_time**ASC) OVER()** AS precent95\_time

**DELETE Duplicate**

DELETE p1 FROM Person p1,

    Person p2

WHERE

    p1.Email = p2.Email AND p1.Id > p2.Id

**SWAP VARIABLE**

**UPDATE** salary

SET

    sex = CASE sex WHEN 'm' THEN 'f'

        ELSE 'm'

    END;

SELECT select\_list

FROM table\_expression

[ ORDER BY sort\_expression1 [ASC | DESC] [, sort\_expression2 [ASC | DESC] ...] ]

[ LIMIT { number | ALL } ] [ OFFSET number ]

**OFFSET** says to skip that many rows before beginning to return rows.If both OFFSET and LIMIT appear, then OFFSET rows are skipped before starting to count the LIMIT rows that are returned.

① select \* from table limit 2,1; //含义是跳过2条取出1条数据，limit后面是从第2条开始读，读取1条信息，即读取第3条数 ② select \* from table limit 2 offset 1; //含义是从第1条（不包括）数据开始取出2条数据，limit后面跟的是2条数据，offset后面是从第1条开始读取，即读取第2,3条

**String Concatenation**

CONCAT('Sci' ,'ence’)

**SUBSTRING**(MyColumn, 1 , 1 ) for the first character

* *start\_pos:* The position to start extraction from. The first position in string is 1
* *number\_of\_chars:* The number of characters to extract

**LEFT**(MyColumn, 1)

**#POSITION** allows you to specify a substring, then returns a numerical value equal to the character number (counting from left) #where that substring first appears in the target string.

SELECT incidnt\_num, descript, **POSITION('A' IN descript)** AS a\_position FROM tutorial.sf\_crime\_incidents\_2014\_01

**#TRIM**is used to remove characters from the beginning and end of a string

SELECT **TRIM**( '.,! ' **FROM** '# test .') AS Result;

**Return the first non-null expression in a list:**use COALESCE to replace the null values when performing outer joins

**COALESCE**(descript, 'No Description')

SELECT JobTitle FROM HumanResources.Employee WHERE Gender = 'M’

**EXCEPT**

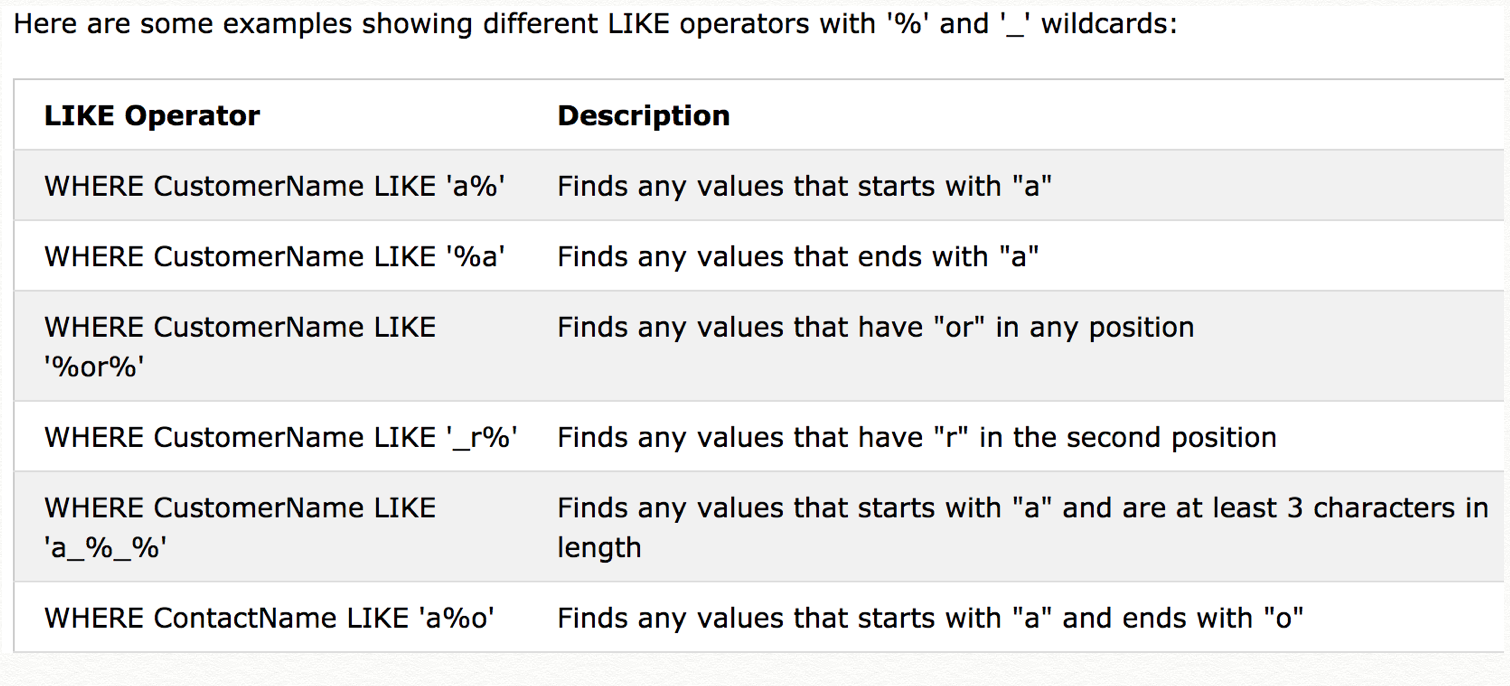
SELECT JobTitle FROM HumanResources.Employee WHERE Gender = ‘F’

等于

SELECT DISTINCT M.JobTitle FROM HumanResources.Employee AS M

WHERE M.Gender = 'M' AND

M.JobTitle**NOT IN** (SELECT F.JOBTITLE FROM HumanResources.Employee AS F WHERE F.Gender = 'F')



**Exclusive OR:**

 WHERE (

(condition1 OR condition2)

AND

NOT(condition1 AND condition2)

)

**ROUND** function: ROUND(NUM, DECIMAL\_PLACE)

ROUND(8923.79, 1)  -> 8923.8

ROUND(8923.79, 0)  -> 8924

ROUND(8923.79, -3) -> 9000

**FLOOR**(-2.7) -> -3

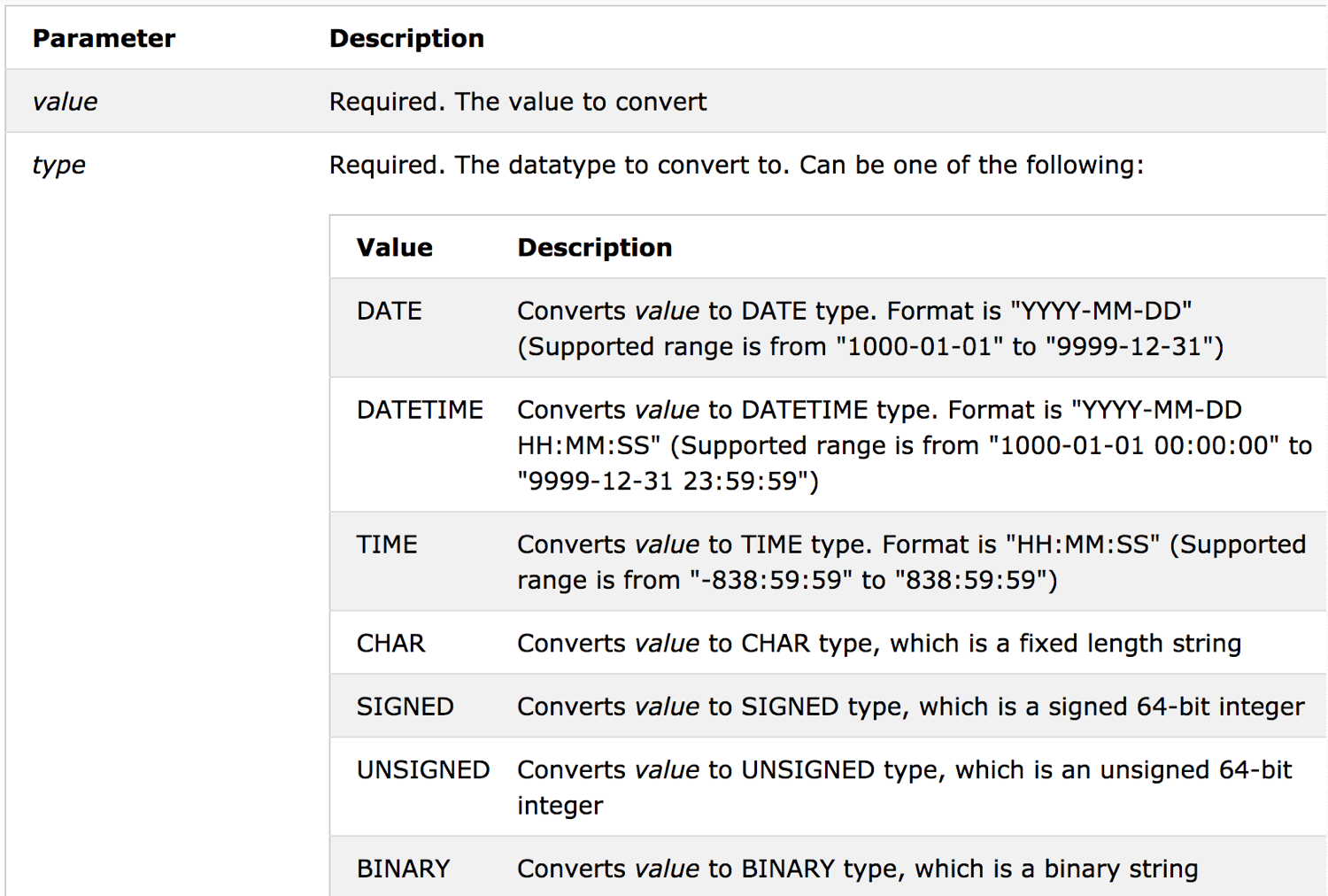
FLOOR(2.7) -> 2

**CEIL**(2.7) -> 3

CEIL(-2.7) -> -2

**MOD**(27, 10) -> 7

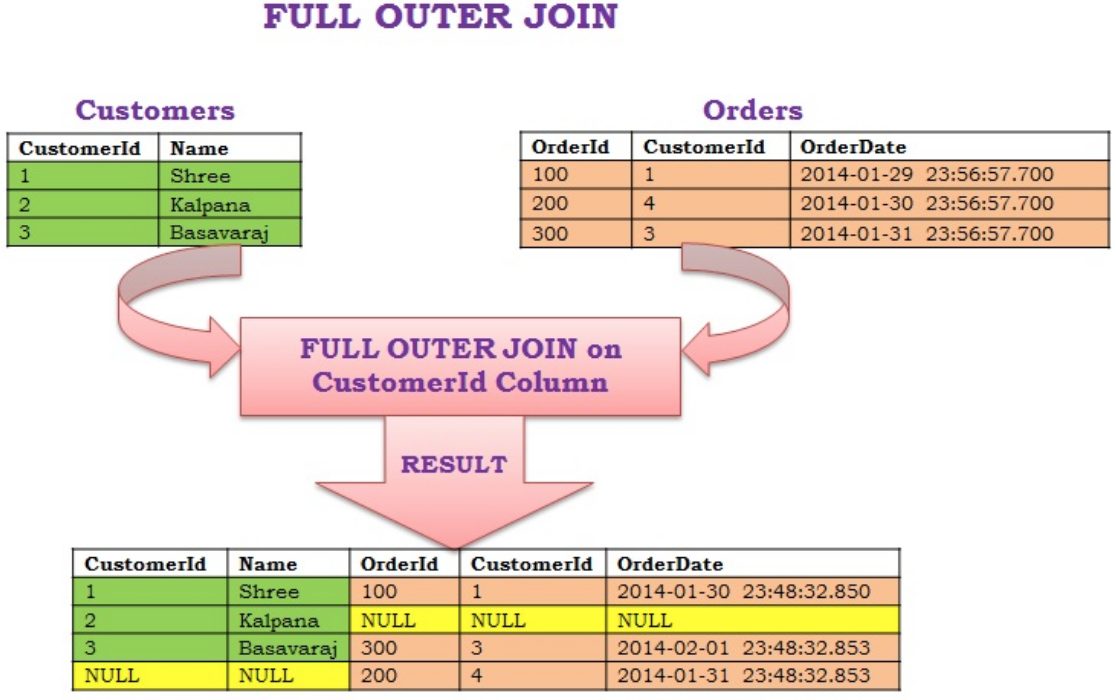
**CAST(expression AS data\_type(length))**



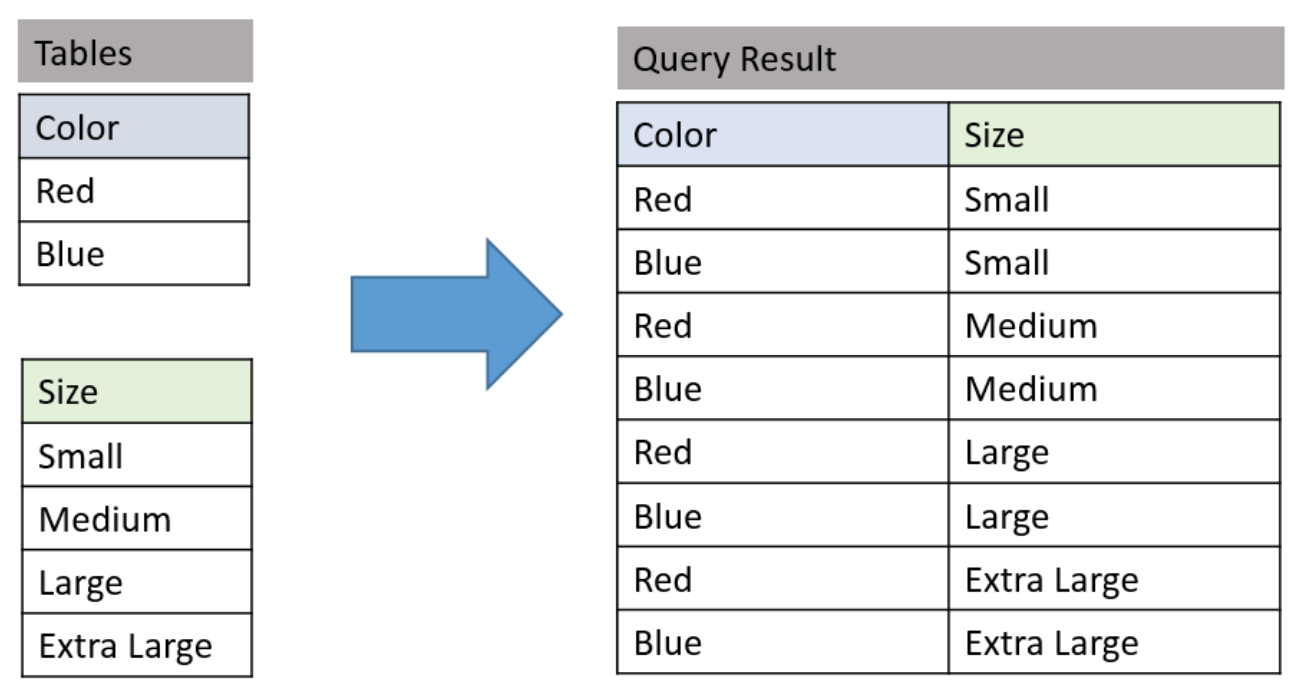
**FULL OUTER JOIN VS CROSS JOIN?**

Now for the full outer JOIN, this is like asking for a LEFT OUTER JOIN and a RIGHT OUTER JOIN in the same query. You are asking to get every row from Table1 matching what could be matched in Table2, for those that don’t match on the Table1 side, just display the Table2 side, with NULLs on the LEFT side, and for those that don’t match on the TABLE2 side, just display the Table1 side with NULLs on the RIGHT side.

|  |  |
| --- | --- |
| 1  2  3  4 | SELECT \*    FROM People p    FULL OUTER JOIN Colors c      ON p.id = c.id; |





****

**SELF JOIN**

#microsoft跳槽到google的人数#

select count(distinct a.member\_id)

from tbl as a join tbl as b

on a.member\_id=b.member\_id and a.company=Microsoft and b.company = Google and a.year\_start<b.year\_start

#R

inner\_join(x,x,by='member\_id') %>%

filter(company.x == "g" & company.y =="l" & date.x<date.y)

   (b). count members who directly moved from Microsoft to Google? sol: window function

   (M-L-G does’t count)

select count(distinct a.member\_id)

from tbl as a join tbl as b

on a.member\_id=b.member\_id and a.company=Microsoft and b.company = Google and a.year\_start<b.year\_start

where ROW\_NUMBER() over(partition by a.member\_id order by b.date desc) = 1

SELECT COUNT (DISTINCT c1.Member\_id)1point3acres缃�  
FROM company c1 JOIN company c2. 1point3acres.   
ON c1.Member\_id = c2.Member\_id.鐣欏璁哄潧-涓€浜�  
AND c1.Company = 'Microsoft'   
AND c2.Company = 'Google'  
**AND c1.Year\_Start < c2.Year\_Start**  
AND NOT EXISTS(  
SELECT c3.Member\_id  
FROM company c3.   
WHERE c3.Member\_id = c1.Member\_id  
AND c3.Year\_Start BETWEEN c1.Year\_Start AND c2.Year\_Start  
)

**Table with continent | country | population**

1. Find the country with largest population in each continent, with strictly output: continent, country, population. Consider corner case that two country have same largest population in the same continent.

SELECT continent, country, population

FROM (SELECT \*,

row\_number() over (partition by continent order by population desc) as rank1 #(unique) rank number

RANK () OVER (partition by continent ORDER BY population DESC) as RANK

#可并列 rank number

FROM table

) t

WHERE rank=1

1. Find the country with largest % of population in given continent.  only one line output.

SELECT \*, population/ continent\_popu as percent

FROM

(SELECT country, continent, population, sum() over (partition by continent) as continent\_popu

FROM table) t

ORDER BY population/continent\_popu

LIMIT 1

WITH conti\_population AS (

SELECT \*, population/continent\_popu as percent

FROM table as a left join

(SELECT continent, sum(population) as continent\_popu

FROM table

GROUP BY continent

) as b on a.continent = b.continent

)

SELECT \* FROM conti\_population where percent = (select max(percent) from conti\_population)

**R实现self join：**

**INNER JOIN**

df1.join(df2.set\_index('key'), lsuffix='\_A', rsuffix=‘\_B',on='key')

 pd.merge(d[['some\_col']], d, left\_index=True, right\_on='i', suffixes=['\_y','']).sort\_index()

select

case when b.id is not null then b.state else 'neither' end as state2,

case when a.id is not null then a.state else 'neither' end as state1,

count(\*)

from analytics\_bank\_account\_logs as a join analytics\_bank\_account\_logs as b

on a.id = b.id and a.date = "2017-01-01" and b.date = "2017-02-01"

group by a.state, b.state’

例子：

#How to find count of second degree followers in a database as below?

#Followee Follower

A B

B C

B D

B E

A F

F G

F H

B H

select count(distinct t1.Follower)

from t

join

t t1

on t.Follower=t1.Followee

group by t.Followee

SELECT \*

FROM T1

**CROSS JOIN** T2

等价于

SELECT \*

FROM T1, T2

<https://www.essentialsql.com/get-ready-to-learn-sql-server-19-introduction-to-sub-queries/>

A **subquery** may occur in :

* - A SELECT clause
* - A FROM clause
* - A WHERE clause

### SQL Joins Using WHERE or ON

ON： filter one or both of the tables **before**joining them. For example, you only want to create matches between the tables under certain circumstances. can think of it as a WHERE clause that only applies to one of the tables.

WHERE：If you move the same filter to the WHERE clause, you will notice that the filter happens **after** the tables are joined. The result is that the 1000memories row is joined onto the original table, but then it is filtered out entirely (in both tables) in the WHERE clause before displaying results.

EXAMPLES!!!

Write a query to print the sum of all total investment values in 2016 (**TIV\_2016**), to a scale of 2 decimal places, for all policy holders who meet the following criteria:

1. Have the same **TIV\_2015** value as one or more other policyholders.
2. Are not located in the same city as any other policyholder (i.e.: the (latitude, longitude) attribute pairs must be unique).

**EXISTS:** only need to find one satisfied，It returns TRUE whenever the subquery returns one or more values.

**NOT EXISTS:** need to check every one. It returns TRUE if zero rows are returned.

**IN** returns TRUE if the tested value is found in the comparison list

**NOT IN** returns TRUE if the tested value is not found. #不能含有NULL

**EXISTS** (SELECT NULL) 🡪 return TRUE #用exists注意不能含有NULL

**IN** (SELECT NULL) 🡪returns zero rows

select 'true' where 3 in (1, 2, 3, null) 🡪 return “true”

select 'true' where 3 not in (1, 2, null) 🡪 not return anything #用not in 时注意需要是不含NULL的list

**select** round(sum(i1.TIV\_2016), 2) as TIV\_2016  
from insurance i1  
**where EXISTS** (select \*  
from insurance i2  
where i1.PID <> i2.PID and i1.TIV\_2015 = i2.TIV\_2015)  
**and NOT EXISTS**(select \*  
from insurance i3  
where i1.PID <> i3.PID and i1.LAT = i3.LAT and i1.LON = i3.LON)

**select** sum(new.tiv\_2016) as TIV\_2016

from

(**select** distinct **i1**.pid, **i1**.tiv\_2016

from insurance **i1**, insurance **i2**

where **i1**.tiv\_2015 = **i2**.tiv\_2015 **and** **i1**.pid != **i2**.pid **and** not exists

(**select** \*

from insurance **i3**

where **i1**.lat = **i3**.lat **and** **i1**.lon = **i3**.lon **and** **i1**.pid != **i3**.pid)

) as new

select sum(I.TIV\_2016) as TIV\_2016

From Insurance I

**Where** I.Pid **In** (select I1.PID

From Insurance I1, Insurance I2

Where I1.TIV\_2015 = I2.TIV\_2015

and I1.PID <> I2.Pid)

**and** I.Pid **Not In** (select I1.PID

From Insurance I1, Insurance I2

Where I1.LAT = I2.LAT

and I1.LON = I2.LON

and I1.Pid <> I2.Pid)

UNION & UNION ALL

**SELECT** \*

**FROM** tutorial.crunchbase\_investments\_part1

**UNION**

**SELECT** \*

**FROM** tutorial.crunchbase\_investments\_part2

Note that UNION only appends distinct values. More specifically, when you use UNION, the dataset is appended, and any rows in the appended table that are exactly identical to rows in the first table are dropped. If you’d like to append all the values from the second table, use UNION ALL. You’ll likely use UNION ALL far more often than UNION.

**Wide Format to Long format**

select \*   
from. Waral 鍗氬鏈夋洿澶氭枃绔�,  
select date, 'a' as product\_name, qty\_a as quantity\_sold from t1   
union all. 鐗涗汉浜戦泦,涓€浜╀笁鍒嗗湴  
select date, 'b' as product\_name, qty\_b as quantity\_sold from t1 . 涓€浜�-涓夊垎-鍦帮紝鐙鍙戝竷  
union all  
select date, 'c' as product\_name, qty\_c as quantity\_sold from t1  
;

第一道题

from table a  (member\_id, email\_address)  to generate table b (member\_id, email\_1,email\_2)

for example:

table a

1,a

1,b

1,c

table b

1,a,b

1,a,c.

1,b,c.

SELECT distinct member\_id, a.email, b.email, CAST(member\_id AS VARCHAR(20)) + CAST(num AS VARCHAR(5)) AS columnz

FROM

(SELECT \*, ROW\_NUMBER() OVER(PARTITION BY member\_id ORDER BY email\_address) as num

FROM table as a join table as b on a.member\_id =  b.member\_id)

Table 1: Campaigns

Account\_id (AID) | Campaign\_id (CID)

1     123

1    234

2    235

Table 2:  Spend

Campaign\_id (CID) | Date | Spend\_amount | Currency

123  2017-08-01 200 USD

123 2017-08-02 150 USD

234 2017-09-01 500 USD

235 2017-07-01 100 CAD

Table 3: Exchange\_rate

Currency | Rate(to USD)

CAD  0.79

USD 1.00

.1point3acres缃�

q1: CID, total spend in USD

q2: AID, number of days from first spend date to highest spend date

1. 鐣欏鐢宠璁哄潧-涓€浜╀笁鍒嗗湴
2. -- Q1: CID, total spend in USD
3. SELECT c\_id, SUM(s\_amount \* e\_rate)
4. FROM Spend JOIN ExchangeRate ON s\_currency = e\_currency. more info on [1point3acres.com](http://1point3acres.com/)
5. GROUP BY c\_id;
6. -- Q2: AID, number of days from first spend date to highest spend date
7. WITH SummaryTable AS (
8. SELECT a\_id, s\_date, SUM(s\_amount \* e\_rate) AS t\_amount
9. FROM Campaigns C JOIN Spend S ON C.c\_id = S.c\_id
10. JOIN ExchangeRate ON s\_currency = e\_currency
11. GROUP BY a\_id, s\_date
12. ), FirstSpend AS (
13. SELECT a\_id, MIN(s\_date) AS s\_date
14. FROM SummaryTable
15. GROUP BY a\_id
16. ), HighestSpend AS (
17. SELECT a\_id, s\_date
18. FROM (-google 1point3acres
19. SELECT a\_id, s\_date,. Waral 鍗氬鏈夋洿澶氭枃绔�,
20. ROW\_NUMBER() OVER(PARTITION BY a\_id ORDER BY t\_amount DESC) AS rk
21. FROM SummaryTable
22. ) T
23. WHERE rk = 1
24. ) 鏉ユ簮涓€浜�.涓夊垎鍦拌鍧�.
25. SELECT F.a\_id, H.s\_date - F.s\_date AS days
26. FROM FirstSpend F JOIN HighestSpend H ON F.a\_id = H.a\_id;