SQL Notes

SQL

- Select: Questions (Query) (Columns)
  - What data do you want to retrieve?

- From: Specifies the table to use.
  - تحديد الجدول المستخدم → Specifies the table being used.

- Order By: Sorts results in a specified order, default is descending.
  - Order applied after FROM and before LIMIT clauses.
  - ترتيب تنازلي افتراضي → Default descending order
  - Example: ORDER BY column_name.

- Where: Filters a set of results based on a condition.
  - التصفية للنتائج → Filters the results
  - Must be used before ORDER BY; LIMIT applies after FROM.
  - Examples:
    - WHERE account_id = 4251
    - For non-numeric data: WHERE name = 'Ahmed'

Logical Operators

- LIKE: Used when you may not know exactly what you are searching for. It acts similarly to the equals (=) operator but allows pattern matching.

- IN: Allows you to specify multiple values in a WHERE clause. Similar to using multiple OR conditions.

- NOT: Reverses the result of LIKE or IN.

- AND & BETWEEN: Combines conditions where all must be true for the row to be included.

- OR: At least one of the conditions must be true for the row to be included.
  - Example: WHERE url LIKE '%google%'.

- IN: Allows filtering data based on multiple values.
  - Example: WHERE name IN ('Walmart', 'APPLE')
  - NOT IN: Reverses the result for IN.

SQL Joins

- Joins: Combine rows from two or more tables based on a related column.
  - Used to pull data from multiple tables.
  - مثال:
    - Example: ON account_id = accounts.id

- Example: SELECT orders.*, accounts.id

- Inner Join: Returns rows that have matching values in both tables.
  - النقاط المشتركة → Returns matching rows from both tables

- Left Join: Returns all rows from the left table and the matching rows from the right table. If no match, NULL is returned.

- Right Join: Returns all rows from the right table and the matching rows from the left table. If no match, NULL is returned.

- Select Distinct: Retrieves unique values only (no duplicates).
  - تظهر القيم مرة واحدة → Shows each unique value only once.

- Union, Union All, Cross Join, Self Join
  - Combine or relate data across multiple tables in various ways.

Aggregation Functions

- We can't use SUM(*) like COUNT because SUM requires numeric columns, while COUNT can work with any column.

- SUM treats NULL values as 0.
  - Use SUM only on quantitative data columns.

- MIN and MAX ignore NULL values, as do other aggregate functions like SUM and COUNT.

- AVG: Calculates the average of the data, typically used with numeric columns.

Group By

- Allows segmenting results into groups.
  - تتكون من عدة مجموعات → Composed of multiple groups

- Must appear between the FROM/WHERE and ORDER BY clauses.

- Any column not used in an aggregation function and included in SELECT must be listed in the GROUP BY clause.

- تجدر الإشارة إلى أن استخدامDISTINCT ، خاصة في التجميعات، يمكن أن يبطئ استعلاماتك قليلاً .

- و بتعفي عن الربط و الاستخدام لجداول مختلفه كمثال دا :

```
/*select distinct a.id as "account id", r.id as "region id",
a.name as "account name", r.name as "region name"
from accounts a
join sales_reps s
on s.id = a.sales_rep_id
join region r
on r.id = s.region_id
order by a.id
-------or------------
SELECT DISTINCT id, name
FROM accounts;*/
/*select s.id , a.name , COUNT(*) num_accounts
from sales_reps s
join accounts a
on a.sales_rep_id = s.id
GROUP BY s.id, a.name
order by s.id desc
--------------------or ------------
SELECT DISTINCT id, name
FROM sales_reps;*/
```

- في حاله الجمع او العمليات زي sum او غيرها مينفعش نستخدم عليها where , بنستخدم having

- WHERE subsets the returned data based on a logical condition.
- WHERE appears after the FROM, JOIN, and ON clauses, but before GROUP BY.
- HAVING appears after the GROUP BY clause, but before the ORDER BY clause.
- HAVING is like WHERE, but it works on logical statements involving aggregations.

بشكل تاني :

- WHERE:

- **Function** : Subsets the returned data based on a logical condition.

- **Position** : Appears after the `FROM`, `JOIN`, and `ON` clauses, but before `GROUP BY`.

- دي بتفلتر البيانات وترجعلنا بس اللي يطابق الشرط اللي بنكتبه.

- HAVING:

- **Function**: Used after the `GROUP BY` clause to filter groups based on a condition.

- **Position**: Appears after `GROUP BY`, but before `ORDER BY`.

- دي بتستخدم بعد ما نعمل تجميع للبيانات بـ`GROUP BY` ، وبنقدر من خلالها نفلتر التجميعات نفسها.

Dates :

في الداتا بيز بنتعامل مع التواريخ بالصيغه دي : YY-MM-DD

- عشان التوقيت بيكون من السنين للثواني فبالتالي بيكون صعب نجمعهم فبنستخدم توحيد لهم زي مبنكون عاوزين و هو DATE_TRUNK() ... يعني لو عاوزين نخلي الساعات للثواني نفسهم عشان يبقو نفس اليوم مثلا :

DATE_TRUNK('day' , occurred_at) → هيخلي الدقايق و الساعات و الثواني زي بعض لنفس اليوم

2017–04-01 12:15:01	DATE_TRUNC ('second', 2017-04-01 12:15:01)
2017–04-01 00:00:00	DATE_TRUNC ('day', 2017-04-01 12:15:01)
2017–04-01 00:00:00	DATE_TRUNC ('month', 2017-04-01 12:15:01)
2017–01-01 00:00:00	DATE_TRUNC ('year', 2017-04-01 12:15:01)

- طب لو عاوزين ناخد جزء بس من التاريخ بنستخدم DATE_PART() و هنا بناخد الجزء المطلوب بس زي كدا :

1	DATE_PART ('second', 2017-04-01 12:15:01)
1	DATE_PART ('day', 2017-04-01 12:15:01)
4	DATE_PART ('month', 2017-04-01 12:15:01)
2017	DATE_PART ('year', 2017-04-01 12:15:01)

- لكن لاحظ سحب month أو يوم من أيام الأسبوع (dow) يعني أنك لم تعد تحافظ على ترتيب السنوات. بل أنت تقوم بالتجميع لمكونات معينة بغض النظر عن السنة التي تنتمي إليها.

- و هنا (dow) فيها 0 يبقي الحد و 6 السبت .

**** دي مش موجوده مباشره في mysql و لكن بنقدر نعملها بطرق غير مباشره :**

In MySQL:

1. `DATE_TRUNC()`: MySQL doesn't have a native `DATE_TRUNC()` function like some other databases. However, you can achieve similar functionality using functions like `DATE_FORMAT()` or `TIMESTAMP()` to truncate a date to a specific part (e.g., year, month, day). Here are a few examples:

sql — Copy code

```sql
-- Truncate to year
SELECT DATE_FORMAT(NOW(), '%Y-01-01') AS truncated_year;

-- Truncate to month
SELECT DATE_FORMAT(NOW(), '%Y-%m-01') AS truncated_month;

-- Truncate to day
SELECT DATE(NOW()) AS truncated_day;
```

٢ . بديل DATE_PART() :

*** للفهم اكتر دا ChatGPT هيكون أوضح و تفسير اكبر : click here

CASE Statements

```
SELECT id,
       account_id,
       occurred_at,
       channel,
       CASE WHEN channel = 'facebook' OR channel = 'direct' THEN 'yes' ELSE 'no' END AS is_facebook
  FROM demo.web_events_full
 ORDER BY occurred_at
```

	id	account_id	occurred_at	channel	is_facebook
1	2471	2861	2013-12-04 04:18:00	direct	yes
2	4193	4311	2013-12-04 04:44:00	direct	yes
3	8825	4311	2013-12-04 08:27:00	adwords	no
4	6994	2861	2013-12-04 18:22:00	facebook	yes
5	294	1281	2013-12-05 20:17:00	direct	yes
6	4728	1281	2013-12-05 21:22:00	adwords	no
7	1998	2481	2013-12-06 02:03:00	direct	yes

هنا بتستخدمها بنعمل كولوم جديد بيكون مثلا لو حاجة عاوزينها زي اخر صورة بنحددها ب case when بتظهر ف الكولوم الجديد باللي بنحدده زي yes

end اللي يظهر لو اللي مطلوب مش موجود else اللي يظهر في حاله وجود المطلوب then شرط or الشرط Case when

و ممكن نستخدمها في حالات مختلفه و متعدده لو عايزين نتايج مختلفه زي دي :

```
SELECT account_id,
       occurred_at,
       total,
       CASE WHEN total > 500 THEN 'Over 500'
            WHEN total > 300 AND total <= 500 THEN '301 - 500'
            WHEN total > 100 AND total <= 300 THEN '101 - 300'
            ELSE '100 or under' END AS total_group
  FROM demo.orders
```

لعدد غير محدود من الشروط زي ما نعوز

Subqueries

```sql
SELECT channel,
       AVG(event_count) AS avg_event_count
FROM
(SELECT DATE_TRUNC('day',occurred_at) AS day,
       channel,
       COUNT(*) as event_count
  FROM demo.web_events_full
 GROUP BY 1,2
 ) sub
  GROUP BY 1
  ORDER BY 2 DESC
```

بنستخدمه جوا from و بنستخرج داتا معينه و ف الكويري اللي برة بنعمل من خلالها عمليات و بنحدد اللي محتاجينه يظهر و دا بيكون بشكل محدود و ادق بشكل كبير في النتيجه

و ممكن نستخدمه كمان كشرط مع Where و في الحاله دي بنستخدم الفاليو اللي جواه بس ف مش بنحتاج نسمي زي م بنحتاج اجباري في الحاله اللي فاتت و دا مثال :

```sql
SELECT AVG(standard_qty) avg_std, AVG(gloss_qty) avg_gls, AVG(poster_qty) avg
FROM orders
WHERE DATE_TRUNC('month', occurred_at) =
     (SELECT DATE_TRUNC('month', MIN(occurred_at)) FROM orders);

SELECT SUM(total_amt_usd)
FROM orders
WHERE DATE_TRUNC('month', occurred_at) =
      (SELECT DATE_TRUNC('month', MIN(occurred_at)) FROM orders);
```

او مع having لو عايزين نستخدمه ك شرط و نساوي مثلا النتيجة مع الكويري الحالي او يكون اكبر او اصغر زي كدا مثلا :

```sql
What is the lifetime average amount spent in terms of **total_amt_usd**,
 including only the companies that spent more per order,
 on average, than the average of all orders.*/
select avg(avg_amt)
from
( select a.name ,  o.account_id , avg(total_amt_usd) avg_amt
from orders o
join accounts a
on o.account_id = a.id
group by 1 ,2
having avg(o.total_amt_usd) >
( select avg(o.total_amt_usd) avg_all
from orders o ) ) sub1
```

Common table expression (CTE)

و دي بنستخدمها عشان منطولش الكويري و تبقي مكرره و بنشوف ايه الأساس او المكرر و بنستخدمه فيها بحيث نقلل المكتوب و يكون منظم اكتر و قابل للاستخدام اكتر من برة براحتك .

- وبداية مع With

```
WITH events AS (
          SELECT DATE_TRUNC('day',occurred_at) AS day,
                        channel, COUNT(*) as events
          FROM web_events
          GROUP BY 1,2)

SELECT channel, AVG(events) AS average_events
FROM events
GROUP BY channel
ORDER BY 2 DESC;
```

LEFT & RIGHT & LENGTH

هنا بنقدر ناخد او نقطع الجزء سواء من اليمين او الشمال و يتحط ف كولوم يعني لو 123 و كتبنا : LEFT(NUM,2) هيطلع لنا اول رقمين من الجهة الشمال 12 و هكذا من اليمين

و لكن LENGTH هنا بتطلع عدد الحروف و الأرقام في السيل او الرو المكتوبه جوا الفانكشن يعني لو 123 هيطلع في كولوم انه 3 عشان عدد الحروف 3

دا مثال :

```
Consider vowels as a, e, i, o, and u.
 What proportion of company names start with a vowel, and what percent start with anything else?*/
select vowels , counts_ , ( counts_ * 100 / total ) as percentage
from (select case when left(a.name , 1 ) in ('a' , 'e' , 'i' , 'o','u') then 'with' else 'not' end as vowels ,
count(*) counts_ , (SELECT COUNT(*) FROM accounts) total
from accounts a
group by 1) sub1
```

1 POSITION , 2 STRPOS , 3 LOWER & 4 UPPER :

1 هنا من خلالها بتقدر تستقبل حرف او كولوم و من خلالها بتقدر تحدد مثلا مكان الحرف المطلوب في الكولوم ك انديكس او مكان زي مثال هنا :

هنا جوا الكولوم هيدور علي مكان الفاصله في كل رو و هيطلع المكان ك انديكس و ليكن 11 . POSITION(',' IN city_state)

2 نفس الوضع بس بتكون مختلفه في التركيب و دا مثلا : **STRPOS(city_state, ',')**

3 و 4 بنكبر او نصغر كل الحروف في الكولوم

خد بالك : كل من POSITION و STRPOS حساسان لحالة الأحرف ، لذا فإن البحث عن A يختلف عن البحث عن a .

CONCAT

هنا بتجمع قيم عمودين في عمود واحد و في mysql بنستخدمها بس و مينفعش || فيها : **CONCAT(first_name, ' ', last_name)**

REPLACE

هنا بنحدد كولوم بنكون عاوزين نشيل حاجة لو قابلتنا في القيمه او نبدلها زي في المثال دا REPLACE(name, ' ', '') بنكون عاوزين نشيل المسافه ف هنبدلها بنقطين مفيش بينهم مسافات بحيث الاسمين يلزقو ف بعض بدون مسافات :

email
Tamara.Tuma@Walmart.com
Sung.Shields@ExxonMobil.com
Jodee.Lupo@Apple.com
Serafina.Banda@BerkshireHathaway.com

CAST

يُستخدم لتحويل البيانات بين الأنواع المختلفة من داتا تايب لتانية **CAST(date_column AS DATE)** هنا بنحول string لتاريخ و عشان نحول بردو من string لتاريخ ممكن نستخدم STR_TO_DATE('07-11-2024', '%d-%m-%Y')

COALESCE & IFNULL()

هنا بتستخدم لما نحب نسترجع من قائمه القيم اول قيمه غير فارغه يعني متكونش null و دا مثال هيوضح

```
SELECT COALESCE(NULL, NULL, 'Some value') ;
```

-- النتيجة: 'Some value'

و كمان لو عاوزين في عمود أي قيمه null نبدلها بقيمه عاوزينها بدل م تكون فارغه و دا مثال :

```
SELECT id, COALESCE(first_name, 'Unknown') AS first_name,
    COALESCE(last_name, 'Unknown') AS last_name
FROM employees;
```

في هذه الحالة، إذا كان أي من first_name أو last_name يحتوي على NULL ، سيتم استبداله بـ 'Unknown'

Window Functions

1 OVER & 2 PARTITION BY & 3 Rank – Dense_Rank()

1 نفس وظيفة **(GROUP BY)** الفرق الأساسي هو أن **OVER** يتيح الاحتفاظ بالتفاصيل الأصلية لكل صف مع إضافة القيم المحسوبة

يعني بتخلي الداتا الاصليه موجودة و لكن بيتضاف القيمه المحسوبه حسب التقسيمه زي مهنشوف دلوقتي OVER بدون تقسيم (PARTITION) :

```
SELECT name, salary,
    ROW_NUMBER() OVER (ORDER BY salary DESC) AS rank
FROM employees;
```

- **النتيجة** : سيتم إعطاء ترتيب تصاعدي لكل صف بناءً على الراتب، مع أعلى راتب في المرتبة الأولى.

OVER مع PARTITION BY , حساب ترتيب الرواتب لكل قسم (department):

```
SELECT department, name, salary,
    ROW_NUMBER() OVER (PARTITION BY department ORDER BY salary DESC) AS rank
FROM employees;
```

PARTITION BY department : يقسم البيانات حسب الأقسام.

ORDER BY salary DESC : يرتب الرواتب داخل كل قسم تنازليًا.

النتيجة : يتم إعطاء ترتيب الرواتب لكل قسم على حدة.

طيب هنشوف الفرق بين **PARTITION BY** , group by :

الفرق بين `PARTITION BY` و `GROUP BY` :

`GROUP BY`	`PARTITION BY`
يجمع البيانات ويعرض صفًا واحدًا لكل مجموعة.	يحتفظ بجميع الصفوف مع تطبيق العملية لكل مجموعة.
يستخدم مع دوال التجميع فقط.	يستخدم مع الدوال التحليلية فقط.
مثال: `SUM(salary) GROUP BY department`	مثال: `SUM(salary) OVER (PARTITION BY department)`

في المثال الجاي دا .. هنشوف order by موجوده و الرانك مختلف و القيم تمام لانه بيقيس بالشهر مثلا :

```sql
SELECT id,
       account_id,
       standard_qty,
       month(occurred_at) AS month,
       DENSE_RANK() OVER (PARTITION BY account_id ORDER BY month(occurred_at)) AS dense_rank_,
       SUM(standard_qty) OVER (PARTITION BY account_id ORDER BY month(occurred_at)) AS sum_std_qty,
       COUNT(standard_qty) OVER (PARTITION BY account_id ORDER BY month(occurred_at)) AS count_std_qty,
       AVG(standard_qty) OVER (PARTITION BY account_id ORDER BY month(occurred_at)) AS avg_std_qty,
       MIN(standard_qty) OVER (PARTITION BY account_id ORDER BY month(occurred_at)) AS min_std_qty,
       MAX(standard_qty) OVER (PARTITION BY account_id ORDER BY month(occurred_at)) AS max_std_qty
FROM orders
```

3 زي م خدنا بالنا الرانك بيختلف عن الدينس رانك في حاجة واحده و هي ان مثلا لو الرانك مدي 1 2 2 4 و دا هنا لانه لو في نفس القيم فهو بيسكب الرقم اللي بعده لكن لو دينس رانك 1223 بيدي رانك و مش بيعدي ارقام زي الرانك العادي

Comparing a Row to Previous Row

LAG & LEAD function :

- **column_name** : العمود الذي يتم استرجاع القيم منه.
- **offset** : عدد الصفوف السابقة (الافتراضي = 1).
- **default_value** : القيمة الافتراضية إذا لم يكن هناك صف سابق.
- **PARTITION BY** : تقسيم البيانات إلى مجموعات (اختياري). - **ORDER BY** : ترتيب الصفوف داخل كل مجموعة.

أمثلة:

1. استخدام `LAG` لاسترجاع قيمة الصف السابق:

Copy code

```sql
SELECT id, name, salary,
       LAG(salary, 1, 0) OVER (ORDER BY id) AS previous_salary
FROM employees;
```

`LAG(salary, 1, 0)` : تسترجع قيمة الراتب من الصف السابق.

`0` : القيمة الافتراضية إذا لم يكن هناك صف سابق.

`ORDER BY id` : ترتيب الصفوف حسب العمود `id`.

نتيجة:

Previous_Salary	Salary	Name	ID
0	5000	John	1
5000	7000	Jane	2
7000	[illegible]	Bob	3

2. استخدام `LEAD` لاسترجاع قيمة الصف التالي:

Copy code

```sql
SELECT id, name, salary,
       LEAD(salary, 1, 0) OVER (ORDER BY id) AS next_salary
FROM employees;
```

`LEAD(salary, 1, 0)` : تسترجع قيمة الراتب من الصف التالي.

`0` : القيمة الافتراضية إذا لم يكن هناك صف لاحق.

نتيجة:

Next_Salary	Salary	Name	ID
7000	5000	John	1
4000	7000	Jane	2
0	4000	Bob	3

Percentiles

NTILE()

تُستخدم الدالة **NTILE()** لتقسيم الصفوف إلى عدد متساوٍ تقريبًا من المجموعات (Buckets) بناءً على ترتيب معين. هذه الدالة مفيدة في تحليل البيانات عند تقسيمها إلى **فئات أو شرائح**، مثل تقسيم الرواتب إلى **رباعيّات (Quartiles)** أو **مئينات (Percentiles)** .

كيفية عمل NTILE()

- **يقوم بتوزيع الصفوف بالتساوي قدر الإمكان على عدد محدد من المجموعات.**
- **إذا لم يكن بالإمكان تقسيم الصفوف بالتساوي تمامًا، يتم توزيع الفائض على المجموعات الأولى.**

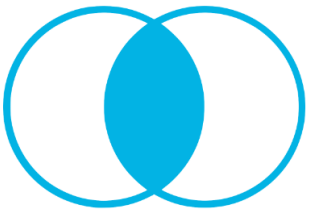
```
/* Quiz: Percentiles -- Percentiles with Partitions*/
/* 1
Use the NTILE functionality to divide the accounts into 4 levels in terms of the amount of standard_qty for their orders.
Your resulting table should have the account_id, the occurred_at time for each order, the total amount of standard_qty paper purchased,
and one of four levels in a standard_quartile column.*/

select account_id,occurred_at,standard_qty,ntile(4) over (PARTITION BY account_id order by standard_qty ) as standard_quartile
from orders o
ORDER BY account_id DESC;
/* 2
Use the NTILE functionality to divide the accounts into two levels in terms of the amount of gloss_qty for their orders.
 Your resulting table should have the account_id, the occurred_at time for each order,
 the total amount of gloss_qty paper purchased, and one of two levels in a gloss_half column.*/

 select account_id ,
        occurred_at ,
        gloss_qty ,
        ntile(2) over ( partition by account_id order by gloss_qty) as gloss_half
 from orders o
 order by 1 desc ;
/* 3
Use the NTILE functionality to divide the orders for each account into 100 levels in terms of the amount of total_amt_usd for their orders.
Your resulting table should have the account_id, the occurred_at time for each order, the total amount of total_amt_usd paper purchased,
and one of 100 levels in a total_percentile column. */
select account_id ,
       occurred_at ,
       total_amt_usd ,
       ntile(100) over ( partition by account_id order by total_amt_usd ) as total_percentile
from orders
order by 1 desc
```

[Advanced] SQL Advanced JOINs & Performancen Tuning

FULL OUTER JOIN

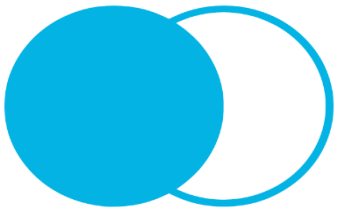


INNER JOIN Venn Diagram

- وهنا بتجيب الداتا فقط المشتركة بينهم اللي متواجدة و مربوطة في الجدولين و ليهم قيمه

```
SELECT column_name(s)
FROM Table_A
INNER JOIN Table_B ON Table_A.column_name = Table_B.column_name;
```

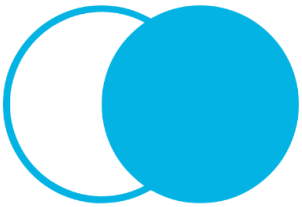
Left joins also include unmatched rows from the left table, which is indicated in the "FROM" clause.



`LEFT JOIN` Venn Diagram

```
SELECT column_name(s)
FROM Table_A
LEFT JOIN Table_B ON Table_A.column_name = Table_B.column_name;
```

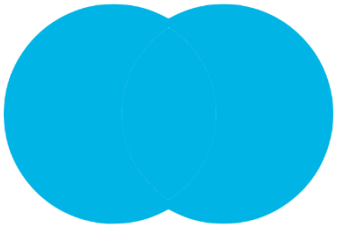
Right joins are similar to left joins, but include unmatched data from the right table -- the one that's indicated in the JOIN clause.



`RIGHT JOIN` Venn Diagram

```
SELECT column_name(s)
FROM Table_A
RIGHT JOIN Table_B ON Table_A.column_name = Table_B.column_name;
```

In some cases, you might want to include unmatched rows from *both* tables being joined. You can do this with a full outer join.



`FULL OUTER JOIN` Venn Diagram

```
SELECT column_name(s)
FROM Table_A
FULL OUTER JOIN Table_B ON Table_A.column_name = Table_B.column_name;
```

- هنا بنجيب كل الداتا اللي ف الجدولين كاملين حتي اللي مش مربوطة مع غيرها من الجدول التاني و كل الجدولين .

* زي م قولنا مش موجوده مباشر في Mysql بس و تعويضها ذكرناه تحت

و هنا مثال مثلا :

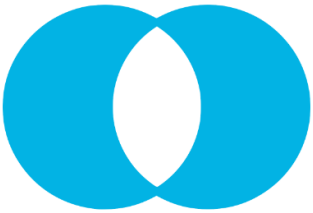
عند ضم جدولين في طابع زمني. لنفترض أن لديك جدولًا يحتوي على عدد الصنف 1 المباع كل يوم، وآخر يحتوي على عدد الصنف 2 المباع. إذا كان تاريخ معين، مثل 1 يناير 2018، موجودًا في الجدول الأيسر ولكن ليس الجدول الأيمن، بينما يوجد تاريخ آخر، مثل 2 يناير 2018، في الجدول الأيمن ولكن ليس الجدول الأيسر:

- ستؤدي Left join إلى إسقاط الصف الذي يحتوي على 2 يناير 2018 من مجموعة النتائج .
- ستؤدي Right join إلى إسقاط الصف الذي يحتوي على 1 يناير 2018 من مجموعة النتائج .

الطريقة الوحيدة للتأكد من وصول كل من 1 يناير 2018 و2 يناير 2018 إلى النتائج هي إجراء Full outer join . تقوم Full outer join بإرجاع السجلات غير المتطابقة في كل جدول بقيم فارغة للأعمدة الواردة من الجدول المقابل.

و لو عايز ترجع القيم ال unmatched بس بنستخدم الكويري : **WHERE** Table_A.column_name IS NULL OR Table_B.column_name IS NULL

* بس هنا في مشكله في mysql و مش موجوده في SQL Server
و هي انه مش بيدعم FULL OUTER JOIN مباشر ف بنعوضها باتحاد left مرة و right مرة
ونعملهم اتحاد زي كدا :



`FULL OUTER JOIN` with `WHERE A.Key IS NULL OR B.Key IS NULL`

```
SELECT e.emp_id, e.name, d.dept_id, d.dept_name
FROM employees e
LEFT JOIN departments d ON e.emp_id = d.emp_id

UNION

SELECT e.emp_id, e.name, d.dept_id, d.dept_name
FROM employees e
RIGHT JOIN departments d ON e.emp_id = d.emp_id;
```

Self JOINs

و هنا بنستخدمه لما نكون عايزين الجدول بتريتب معين يعني مثلا عاوزين اوردر حصل انهاردة لاكونت ف عاوزين اوردراته في خلال 5 أيام من انهارده و كذلك

في هنا دا مثال :

```
SELECT we1.id AS we_id,
       we1.account_id AS we1_account_id,
       we1.occurred_at AS we1_occurred_at,
       we1.channel AS we1_channel,
       we2.id AS we2_id,
       we2.account_id AS we2_account_id,
       we2.occurred_at AS we2_occurred_at,
       we2.channel AS we2_channel
  FROM web_events we1
 LEFT JOIN web_events we2
   ON we1.account_id = we2.account_id
  AND we1.occurred_at > we2.occurred_at
  AND we1.occurred_at <= we2.occurred_at + INTERVAL '1 day'
ORDER BY we1.account_id, we2.occurred_at
```

- هنا نبحث عن الأحداث **التي تحدث بعد حدث معين (w2)** وفي غضون يوم واحد فقط.
- نريد أن نرى جميع السجلات من الجدول الأول **w1**، مع محاولة ربط كل سجل بأحداث سابقة من الجدول الثاني **w2**تحدث في غضون يوم واحد.
- نبحث عن الأحداث السابقة قبل **(w1.occurred_at)** ، لذا نبدأ من **w1** ونبحث في **w2** عن الأحداث التي سبقت **w1** بفترة يوم واحد.

فبالتالي عاوز أقول كلمتين أوضح بيهم حاجة :

ترتيب الجداول في الـ JOIN يعتمد على المنطق التحليلي المطلوب :

- إذا كنت تبحث عن الأحداث أو السجلات التي تحدث **بعد** نقطة معينة ← تبدأ من الجدول الأقدم وتربطه بالجدول الأحدث.
- إذا كنت تبحث عن الأحداث أو السجلات التي تحدث **قبل** نقطة معينة ← تبدأ من الجدول الأحدث وتربطه بالجدول الأقدم.

UNION

دي بنجمع جدولين مع بعض و تلقائي بيتحذف المتشابهين و بيتجمع قيم الجداول و لكن لو في expressions لازم يكونو نفس العدد في select الاتنين , نفس الاعمده ف عددها و نفس الdata types بنفس الترتيب في الجدولين .
و UNION ALL مش بتشيل المتكرر .

ممكن نستخدم where في واحده منهم عادي و لكن الجدول التاني مش هيلتزم بيها و هيعرض الداتا فيه كلها .

ممكن في المثال دا و بديله جمبه :

```
select * from accounts a1
where a1.name = 'Walmart'
union all
select * from accounts a2
where a2.name = 'Disney'
```

```
SELECT * FROM accounts WHERE name = 'Walmart' OR name = 'Disney'
```

- ممكن نستخدم explain عشان نعرف الخطوات اللي اتنفذ بيها الكويري و وقته و هكذا
- الأفضل نقلل العمليات عشان نحسن من الأداء و ممكن فيها نقلل الداتا اللي بنستخدمها بدل م نستدعي الداتا كامله في from لا نعمل subquery و نحدد الداتا اللي محتاجينها بحيث مستندعيش ملايين او الاف الداتا و نهدر طاقه و وقت
- ممكن نستخدم الجداول بشكل منفصل و نربطهم زي كدا مثلا :

```
SELECT COALESCE(orders.date, web_events.date) AS date,
       orders.active_sales_reps,
       orders.orders,
       web_events.web_visits
 FROM (
SELECT DATE_TRUNC('day',o.occurred_at) AS date,
       COUNT(a.sales_rep_id) AS active_sales_reps,
       COUNT(o.id) AS orders
  FROM demo.accounts a
  JOIN demo.orders o
    ON o.account_id = a.id
 GROUP BY 1
 ) orders

 FULL JOIN

(
SELECT DATE_TRUNC('day',we.occurred_at) AS date,
       COUNT(we.id) AS web_visits
  FROM demo.web_events_full we
 GROUP BY 1
 ) web_events

   ON web_events.date = orders.date
 ORDER BY 1 DESC
```

دا كود خاص ب postgresql
لو عاوزين الخاص ب mysql استخدم شات جي بي تي لان الكود طويل جدا اني احطه هنا
لان للأسف مش بيدعم full join ف بنستخدم right left union
و الكود في الصفحه الجايه ------

```
SELECT
   COALESCE(orders.date, web_events.date) AS date,
   orders.active_sales_reps,
   orders.orders,
   web_events.web_visits
FROM (
   SELECT
      DATE(o.occurred_at) AS date,
      COUNT(a.sales_rep_id) AS active_sales_reps,
      COUNT(o.id) AS orders
   FROM demo.accounts a
   JOIN demo.orders o
   ON o.account_id = a.id
   GROUP BY DATE(o.occurred_at)
) orders
LEFT JOIN ( SELECT
      DATE(we.occurred_at) AS date,
      COUNT(we.id) AS web_visits
   FROM demo.web_events_full we
   GROUP BY DATE(we.occurred_at)
) web_events
ON web_events.date = orders.date

UNION

SELECT
   COALESCE(orders.date, web_events.date) AS date,
   orders.active_sales_reps,
   orders.orders,
   web_events.web_visits
FROM (
   SELECT
      DATE(o.occurred_at) AS date,
      COUNT(a.sales_rep_id) AS active_sales_reps,
      COUNT(o.id) AS orders
   FROM demo.accounts a
   JOIN demo.orders o
   ON o.account_id = a.id
   GROUP BY DATE(o.occurred_at)
) orders
RIGHT JOIN (
   SELECT
      DATE(we.occurred_at) AS date,
      COUNT(we.id) AS web_visits
   FROM demo.web_events_full we
   GROUP BY DATE(we.occurred_at)
) web_events
ON web_events.date = orders.date
ORDER BY date DESC;
```