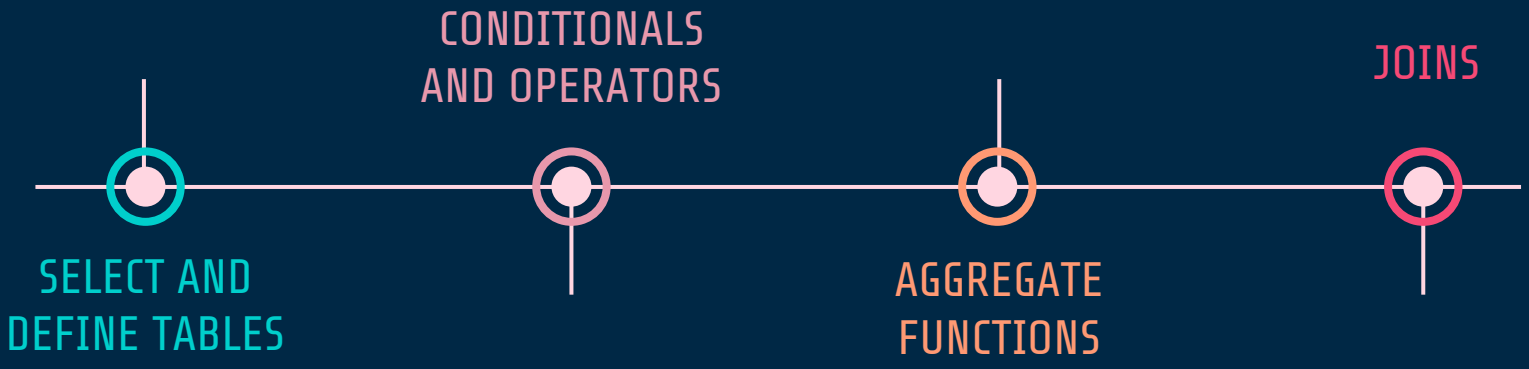
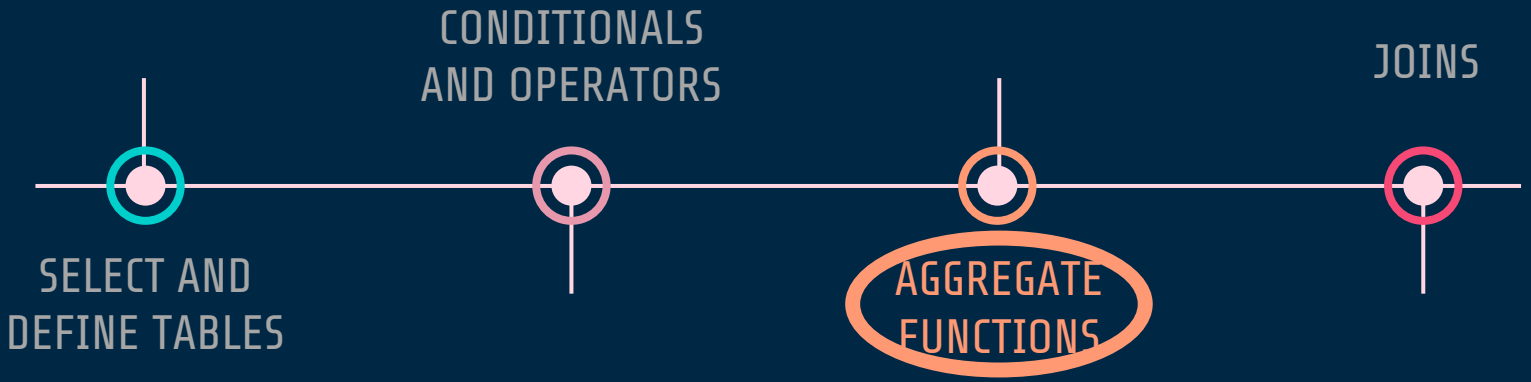
Bases de datos

Construcción de Queries

QUERIES

03


SELECT AND
DEFINE TABLES
CONDITIONALS
AND OPERATORS
AGGREGATE
FUNCTIONS
JOINS


SELECT AND
DEFINE TABLES
CONDITIONALS
AND OPERATORS
AGGREGATE
FUNCTIONS
JOINS

AGGREGATE FUNCTIONS

Common Aggregate functions :

- Avg (expression)
- Count (expression) or Count (*)
- Sum (expression)
- Min/Max (expression)

Statistics Aggregate functions:

- Corr (Y, X)
- Sttdev (expression)
- Variance (expression)

AGGREGATE FUNCTIONS

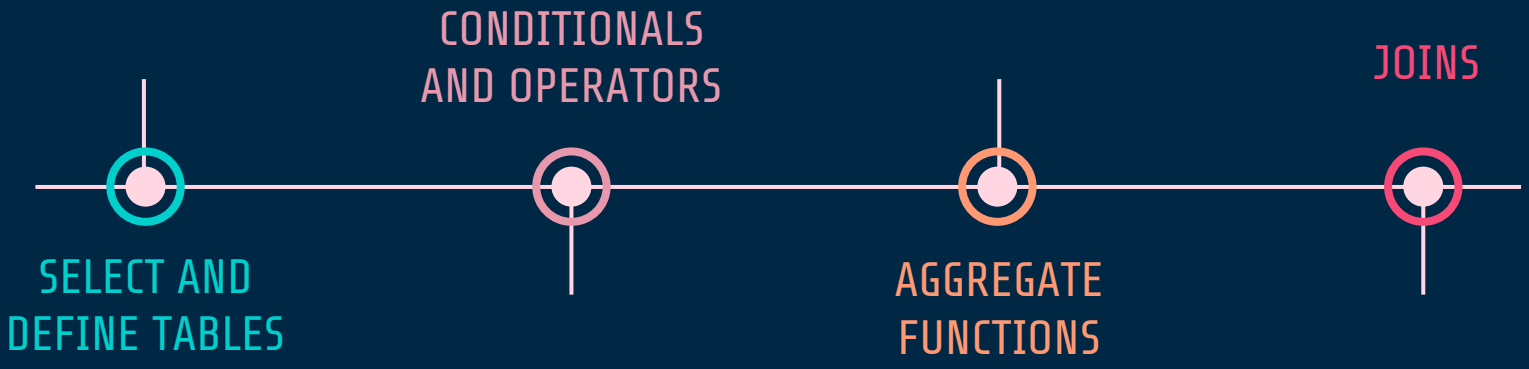
- Aggregate functions compute a single result from a set of input values.
- Syntax:

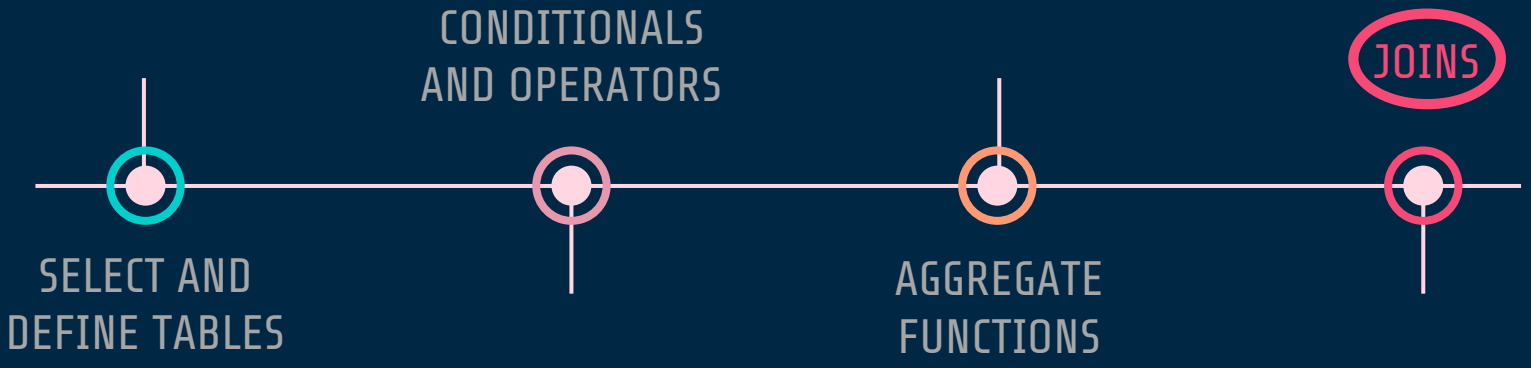
```
SELECT column_name(s), AGGREGATE_FUNCTION() AS column_name
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

* AGGREGATE FUNCTIONS need a GROUP BY clause always !!!!!!!!!!
* HAVING clause is use to conditionate aggregate functions.

¡EJERCICIO RÁPIDO: GROUP BY!

1. Sumar el neto de ventas de la tabla info.d01_ventas agrupándolo por mes y ordenar por mes. Tomar en cuenta solamente las fechas posteriores a 2018-01-01.
2. Agregar a la query anterior que muestre solamente los registros de los meses cuya suma del neto da por lo menos 700.000.000
3. Encontrar el mínimo, el máximo y el precio promedio de cualquier boleta para cada mes de las fechas posteriores a 2018-01-01


SELECT AND
DEFINE TABLES
CONDITIONALS
AND OPERATORS
AGGREGATE
FUNCTIONS
JOINS


SELECT AND
DEFINE TABLES
CONDITIONALS
AND OPERATORS
AGGREGATE
FUNCTIONS
JOINS

JOINS:
UNION CLAUSE AND CARTESIAN JOIN

UNION

- Syntax:

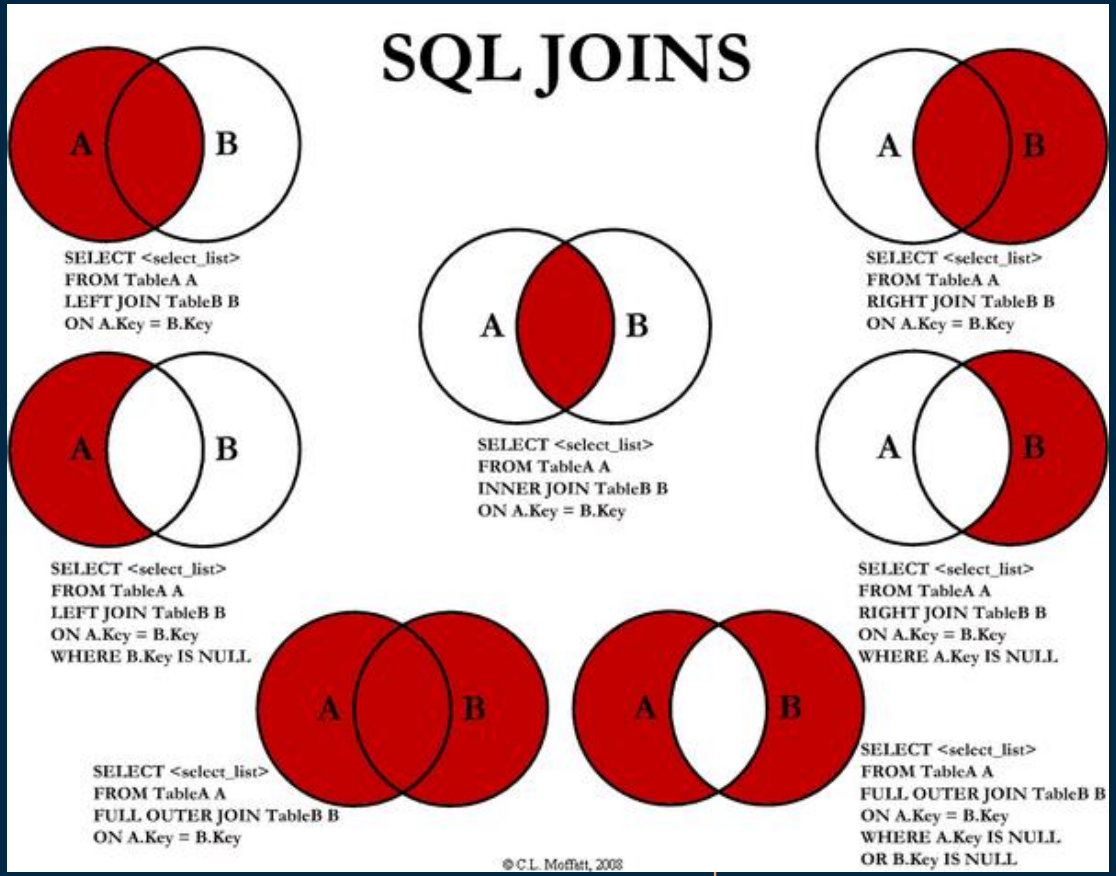
```
SELECT exp1,exp2,exp_n
FROM table_name
[WHERE condition]
UNION ALL
SELECT exp1,exp2,exp_n
FROM table_name
[WHERE condition];
```

CARTESIAN

- Syntax:

```
SELECT t1.a, t1.b, t2.aa, t2.bb
FROM table_name AS t1
[LEFT/INNER/etc] JOIN table_name AS t2
        ON t1.columnY=t2.colunmX;
```

JOINS


SQL JOINS
A
B
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
© C.L. Moffatt, 2008

CARTESIAN JOIN EXAMPLE

Customers

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

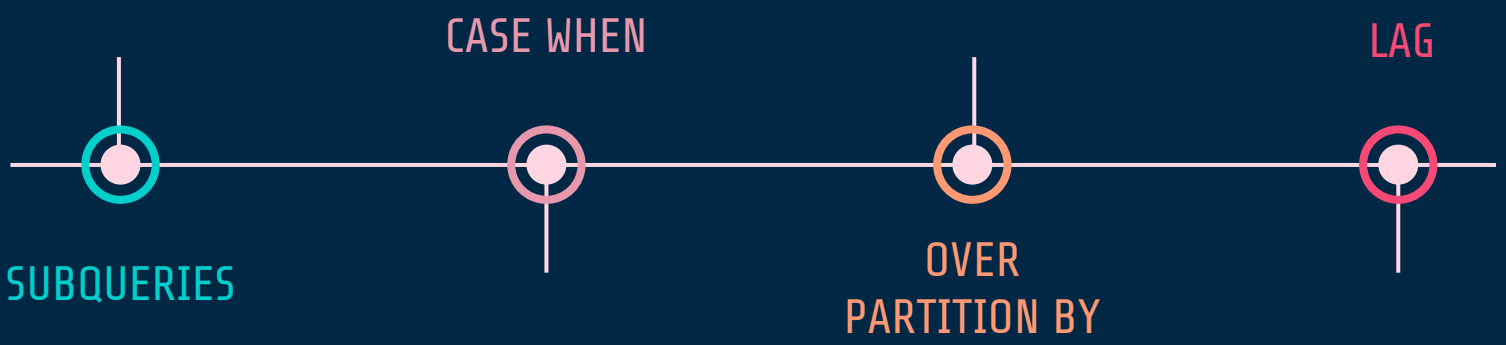
Orders

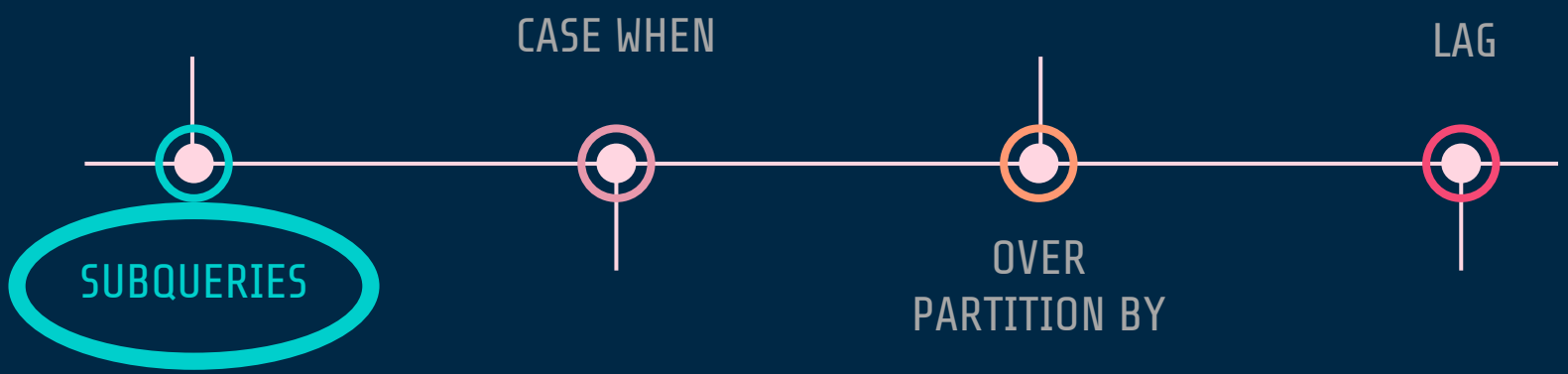
OID	DATE	CUSTOMER_ID	AMOUNT
102	2009-10-08 00:00:00	3	3000
100	2009-10-08 00:00:00	3	1500
101	2009-11-20 00:00:00	2	1560
103	2008-05-20 00:00:00	4	2060

ID	NAME	AMOUNT	DATE
1	Ramesh	3000	2009-10-08 00:00:00
1	Ramesh	1500	2009-10-08 00:00:00
1	Ramesh	1560	2009-11-20 00:00:00
1	Ramesh	2060	2008-05-20 00:00:00
2	Khilan	3000	2009-10-08 00:00:00
2	Khilan	1500	2009-10-08 00:00:00
2	Khilan	1560	2009-11-20 00:00:00
2	Khilan	2060	2008-05-20 00:00:00
3	kaushik	3000	2009-10-08 00:00:00
3	kaushik	1500	2009-10-08 00:00:00
3	kaushik	1560	2009-11-20 00:00:00
3	kaushik	2060	2008-05-20 00:00:00
4	Chaitali	3000	2009-10-08 00:00:00
4	Chaitali	1500	2009-10-08 00:00:00
4	Chaitali	1560	2009-11-20 00:00:00
4	Chaitali	2060	2008-05-20 00:00:00
5	Hardik	3000	2009-10-08 00:00:00
5	Hardik	1500	2009-10-08 00:00:00
5	Hardik	1560	2009-11-20 00:00:00
5	Hardik	2060	2008-05-20 00:00:00
6	Komal	3000	2009-10-08 00:00:00
6	Komal	1500	2009-10-08 00:00:00
6	Komal	1560	2009-11-20 00:00:00
6	Komal	2060	2008-05-20 00:00:00
7	Muffy	3000	2009-10-08 00:00:00
7	Muffy	1500	2009-10-08 00:00:00
7	Muffy	1560	2009-11-20 00:00:00
7	Muffy	2060	2008-05-20 00:00:00

¡EJERCICIO RÁPIDO: JOINS!

1. Seleccionar de la tabla info.d01_diccionario_sku todos los subdepartamentos que pertenezcan al departamento CUIDADO PERSONAL.
2. Asociar a cada subdepartamento la cantidad de ventas de la tabla info.d01_ventas posteriores a 2018-01-01 y ordenarlos de forma descendiente.
3. Seleccionar solamente las clases del subdepartamento BEBES que tengan ventas en la fecha 2018-01-31 y calcular el promedio de los precios de los productos que se hayan vendido ese día. Ordenar por el promedio de precios descendientemente.
4. Contar la cantidad de líneas en d00_diccionario_skus, contar la cantidad de skus únicos, contar la cantidad de skus repetidos y exportarlos en una misma consulta un dato abajo del otro con la leyenda correspondiente


CASE WHEN
LAG
SUBQUERIES
OVER
PARTITION BY


CASE WHEN
LAG
SUBQUERIES
OVER
PARTITION BY

NESTED

- Syntax:

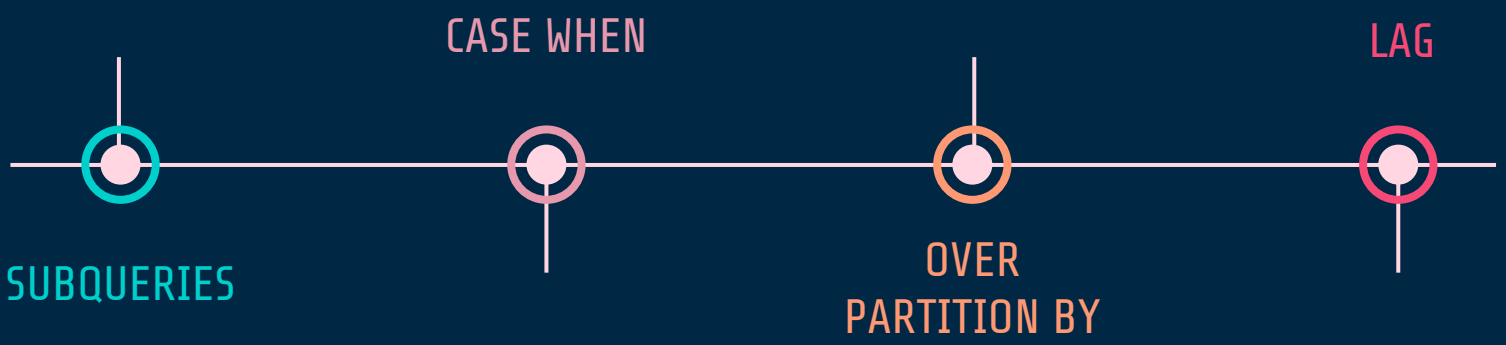
```
SELECT columnlist
FROM (SELECT columnlist FROM tablelist);

SELECT columnlist
FROM tablelist
WHERE columnlist in (SELECT columnlist FROM tablelist);

SELECT product, price,  (SELECT AVG(price) FROM tablelist) avg_price
FROM tablelist;
```

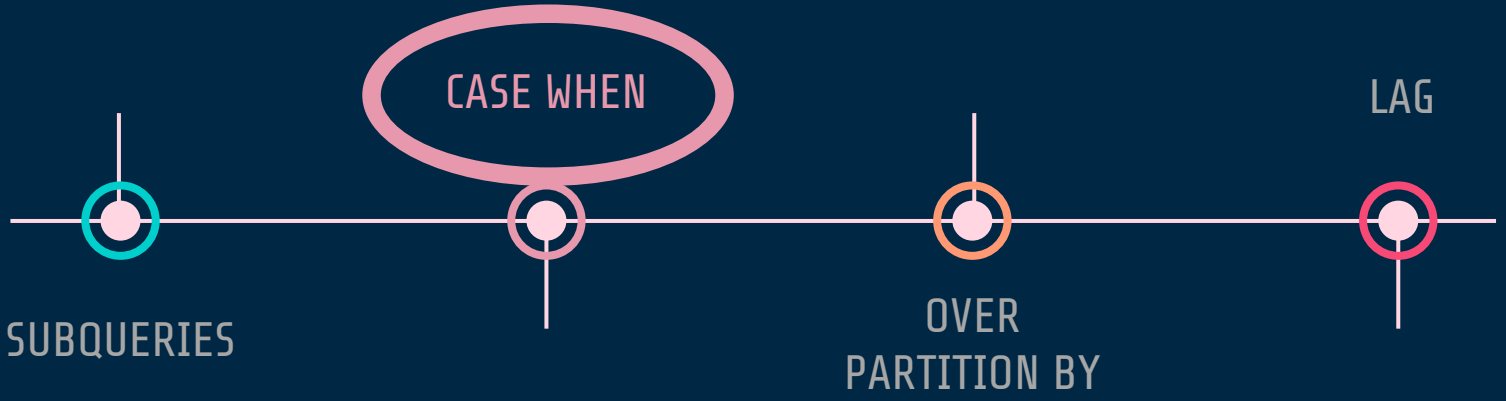
WITH

Permite hacer consultas intermedias (de la misma forma que una subquery) pero de forma ordenada y más simple cuando tenemos que hacer muchas. La idea es evitar anidar una subquery adentro de la otra y adentro de la otra y adentro de la otra, etc.

- Syntax:

```
WITH tabla1 AS (
        SELECT columnlist
        FROM tablelist)
    , tabla2 AS (
        SELECT columnlist
        FROM tablelist)
SELECT * FROM tabla1 INNER JOIN tabla2 ON tabla1.campo1 = tabla2.campo2;
```


CASE WHEN
LAG
SUBQUERIES
OVER
PARTITION BY


CASE WHEN
LAG
SUBQUERIES
OVER
PARTITION BY

CASE

The SQL CASE expression is a generic conditional expression, similar to if/else statements in other languages:

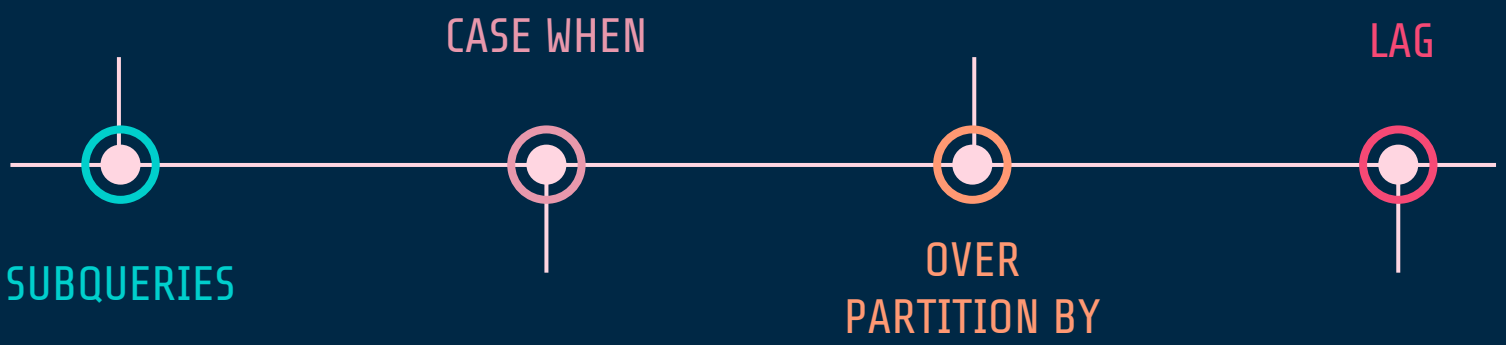
- Syntax:

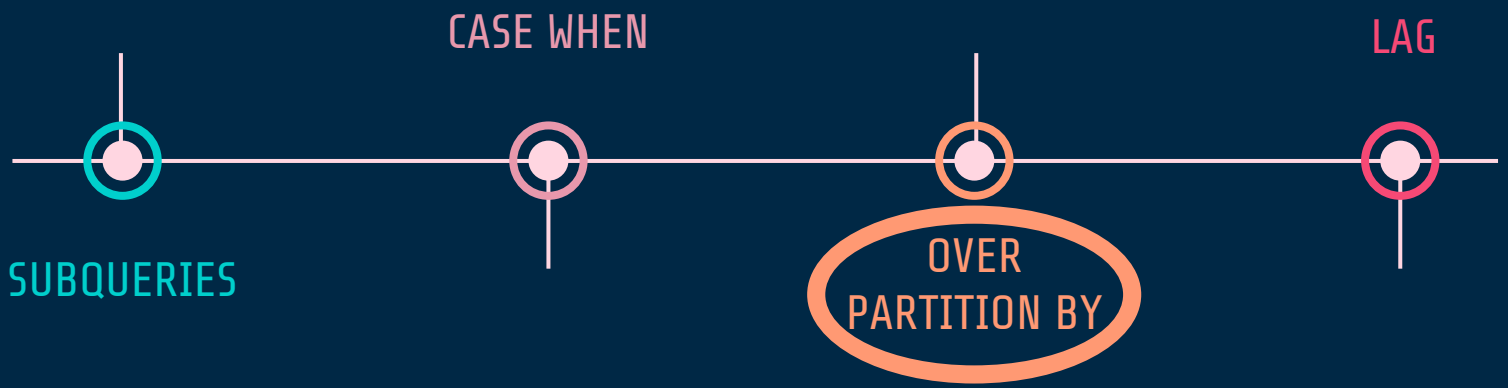
```
SELECT name,
CASE
        WHEN condition THEN result
        [WHEN ...]
        [ELSE result]
        END column_name
FROM tablelist;
```

¡COMPETENCIA EJERCICIO NO TAN RÁPIDO: SUBQUERY + CASE!

Describir a los clientes que compraron durante el último año (que haya en los datos de ventas) con un campo “activo” (booleano) si cumple que:

- Compró al menos 5 veces (frecuencia) en el último año y


CASE WHEN
LAG
SUBQUERIES
OVER
PARTITION BY


CASE WHEN
LAG
SUBQUERIES
OVER
PARTITION BY

OVER PARTITION BY ORDER BY

Es una función agregada aplicada a una partición o subconjunto del resultado de una consulta. Se la llama WINDOW FUNCTION

Diferencia con GROUP BY:

- GROUP BY se usa con las funciones de agregación (suma, promedio, etc.) para indicarle a la función todos los campos que debe tener en cuenta para el cálculo.

- OVER clause define una ventana sobre la que hacer el calculo que NO NECESARIAMENTE son todas los campos de la tabla.

OVER PARTITION BY

PARTITION BY: indica qué campos de la tabla mirar para hacer el cálculo solamente sobre los registros que estén dentro de esa Ventana.

- Syntax:

```
SELECT depname, empno, salary,
        AVG(salary) OVER (PARTITION BY depname)
FROM empsalary;
```

depname	empno	salary	avg
develop	11	5200	5020.0000000000000000
develop	7	4200	5020.0000000000000000
develop	9	4500	5020.0000000000000000
develop	8	6000	5020.0000000000000000
develop	10	5200	5020.0000000000000000
personnel	5	3500	3700.0000000000000000
personnel	2	3900	3700.0000000000000000
sales	3	4800	4866.6666666666666667
sales	1	5000	4866.6666666666666667
sales	4	4800	4866.6666666666666667

(10 rows)

+ info → diferencia entre aggregate functions y Windows functions

https://learnsql.com/blog/window-functions-vs-aggregate-functions/

ORDER BY

ORDER BY: se usa si estamos haciendo una función que necesita la información de campos anteriores o posteriores. Hay que especificar el orden de los campos para que la Ventana esté correctamente seleccionada.

- Syntax:

```
SELECT depname, empno, salary,
        rank() OVER (PARTITION BY depname
                     ORDER BY salary DESC)
FROM empsalary;
```

depname	empno	salary	rank
develop	8	6000	1
develop	10	5200	2
develop	11	5200	2
develop	9	4500	4
develop	7	4200	5
personnel	2	3900	1
personnel	5	3500	2
sales	1	5000	1
sales	4	4800	2
sales	3	4800	2

(10 rows)

ROWS: PRECEDING

ROW or RANGE: Permite que dentro de la ventana establecida por el PARTITION BY se tomen solamente ciertas filas o un rango en particular. Por ejemplo ROWS BETWEEN 3 PRECEEDING AND CURRENT ROW toma solamente los 3 anteriores, y la fila del cálculo .

Syntax:

```
SELECT Year, DepartmentID, Revenue,
        SUM(Revenue) OVER (PARTITION BY DepartmentID
                           ORDER BY [YEAR]
                           ROWS BETWEEN 3 PRECEDING AND CURRENT ROW)
                           AS CurrentAndPrev3
FROM revenue
ORDER BY DepartmentID, Year;
```

ROWS: FOLLOWING

ROW or RANGE: Permite que dentro de la ventana establecida por el PARTITION BY se tomen solamente ciertas filas o un rango en particular. Por ejemplo ROWS BETWEEN CURRENT ROW AND 3 FOLLOWING toma la fila del cálculo y las 3 posteriores.

- Syntax:

```
SELECT Year, DepartmentID, Revenue,
        SUM(Revenue) OVER (PARTITION BY DepartmentID
                           ORDER BY [YEAR]
                           ROWS BETWEEN CURRENT ROW AND 3 FOLLOWING)
                           AS CurrentAndNext3
FROM revenue
ORDER BY DepartmentID, Year;
```

ROWS:
PRECEDING AND FOLLOWING

ROW or RANGE: Permite que dentro de la ventana establecida por el PARTITION BY se tomen solamente ciertas filas o un rango en particular. Por ejemplo ROWS BETWEEN 1 PRECEEDING AND 1 FOLLOWING toma 1 fila anterior, la fila del cálculo y la posterior.

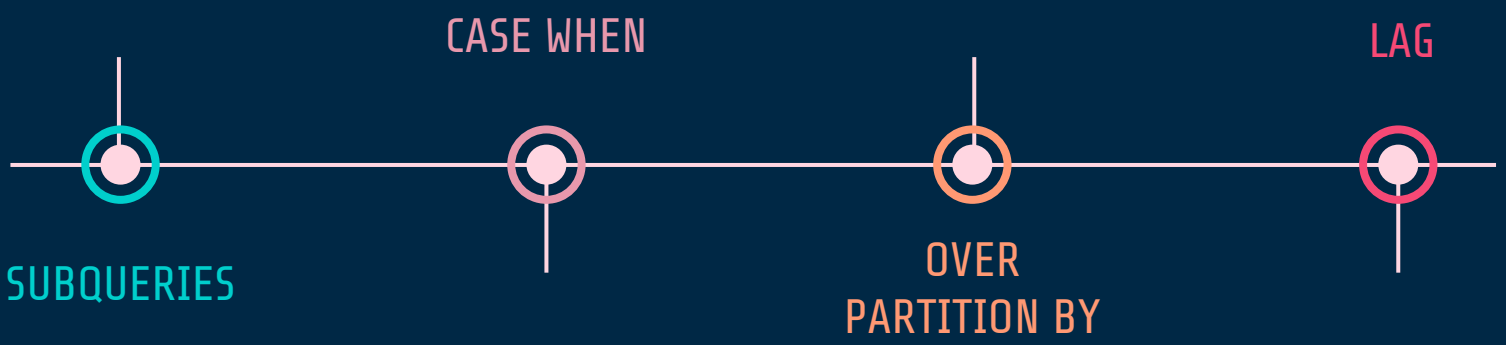
- Syntax:

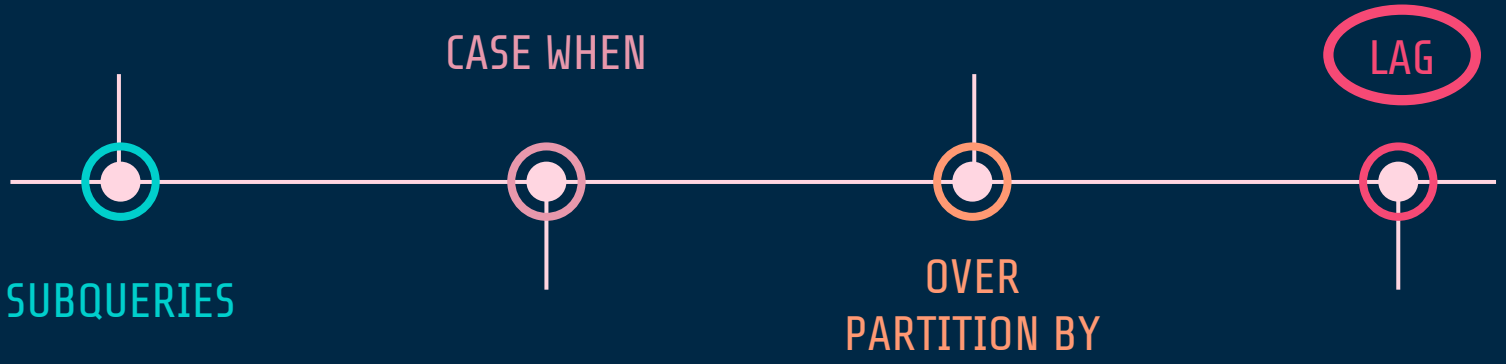
```
SELECT Year, DepartmentID, Revenue,
        SUM(Revenue) OVER (PARTITION BY DepartmentID
                           ORDER BY [YEAR]
                           ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING)
                           AS BeforeAndAfter
FROM revenue
ORDER BY DepartmentID, Year;
```

	Year	DepartmentID	Revenue	CurrentAndPrev3
1	1998	1	10030	10030
2	1999	1	20000	30030
3	2000	1	40000	70030
4	2001	1	30000	100030
5	2002	1	90000	180000
6	2003	1	10300	170300
7	2004	1	10000	140300
8	2005	1	20000	130300
9	2006	1	40000	80300
10	2007	1	70000	140000
11	2008	1	50000	180000
12	2009	1	20000	180000
13	2010	1	30000	170000

	Year	DepartmentID	Revenue	CurrentAndNext3
1	1998	1	10030	100030
2	1999	1	20000	180000
3	2000	1	40000	170300
4	2001	1	30000	140300
5	2002	1	90000	130300
6	2003	1	10300	80300
7	2004	1	10000	140000
8	2005	1	20000	180000
9	2006	1	40000	180000
10	2007	1	70000	170000
11	2008	1	50000	180000
12	2009	1	20000	140000

	Year	DepartmentID	Revenue	BeforeAndAfter
1	1998	1	10030	30030
2	1999	1	20000	70030
3	2000	1	40000	90000
4	2001	1	30000	160000
5	2002	1	90000	130300
6	2003	1	10300	110300
7	2004	1	10000	40300
8	2005	1	20000	70000
9	2006	1	40000	130000
10	2007	1	70000	160000
11	2008	1	50000	140000
12	2009	1	20000	100000
13	2010	1	30000	130000
14	2011	1	80000	120000


CASE WHEN
LAG
SUBQUERIES
OVER
PARTITION BY


CASE WHEN
LAG
SUBQUERIES
OVER
PARTITION BY

LAG AND LEAD FUNCTION

Lag da acceso a un registro previo específico , y Lead al siguiente registro.

- Syntax:

```
LAG(expression [,offset [,default_value]])
OVER ( [PARTITION BY partition_expression, ... ]
ORDER BY sort_expression [ASC | DESC], ...);
```

	year smallint	amount numeric	previous_year_sales numeric
1	2018	5021.00	[null]
2	2019	4944.00	5021.00
3	2020	5137.00	4944.00

- Ejemplo:

```
SELECT year, amount, LAG(amount,1) OVER (ORDER BY year ) prevs_year_sales
FROM table1;
```

¡EJERCICIO NO TAN RÁPIDO: OVER PARTITION BY!

1. Realizar un forecast mensual naive last value (tomar el último valor de compras registrado) por DEPARTAMENTO.
2. Realizar un forecast mensual haciendo un promedio móvil de los últimos tres datos (para cada mes que hay que forecastear tomo el promedio de los últimos tres datos) por DEPARTAMENTO.
3. Rankear por DEPARTAMENTO en orden descendente los clientes que más gastaron en el último mes de data de ventas.

¡TIPS – otras funciones!

CAST(campo as tipo) → convierte el formato del dato
campo::FORMATO → convierte el formato del dato (ej: sku::INT)
DATE_TRUNC('datepart', fecha) → truncar fechas
To_date → transforma a fecha
Date(dd,mm,yyyy) → crea fecha
TRIM() → quita espacios
EXTRACT('datepart' from fecha) → Extraer una parte/dato de la fecha
Group by 1,2,5 .... Hacer referencia a los campos segun orden de ocurrencia
FLOOR() – CEILING() → permite redondear un numero para abajo o para arriba respectivamente