SQL_004

CTES e Window Function

```
SELECT t1."date"
    ,t1."rank"
    ,t1.song
    ,t1.artist
    ,t1."last-week"
    ,t1."peak-rank"
    ,t1."weeks-on-board"
FROM PUBLIC."Billboard" AS t1 limit 100;


SELECT
    t1.artist
    ,t1.song
FROM PUBLIC."Billboard" AS t1
order by t1.artist
    ,t1.song;

SELECT
    t1.artist
    ,count(*) as qtd_artist
FROM PUBLIC."Billboard" AS t1
group by t1.artist
order by t1.artist
;
SELECT
    t1.song
    ,count(*) as qtd_song
FROM PUBLIC."Billboard" AS t1
group by t1.song
order by t1.song
;

SELECT t1.artist
    ,t2.qtd_artist
    ,t1.song
    ,t3.qtd_song
```

```
FROM PUBLIC."Billboard" AS t1
LEFT JOIN (
    SELECT t1.artist
        ,count(*) AS qtd_artist
    FROM PUBLIC."Billboard" AS t1
    GROUP BY t1.artist
    ORDER BY t1.artist
    ) AS t2 ON (t1.artist = t2.artist)
LEFT JOIN (
    SELECT t1.song
        ,count(*) AS qtd_song
    FROM PUBLIC."Billboard" AS t1
    GROUP BY t1.song
    ORDER BY t1.song
    ) AS t3 ON (t1.song = t3.song);

WITH cte_artist
AS (
    SELECT t1.artist
        ,count(*) AS qtd_artist
    FROM PUBLIC."Billboard" AS t1
    GROUP BY t1.artist
    ORDER BY t1.artist
    )
    ,cte_song
AS (
    SELECT t1.song
        ,count(*) AS qtd_song
    FROM PUBLIC."Billboard" AS t1
    GROUP BY t1.song
    ORDER BY t1.song
    )
SELECT t1.artist
    ,t2.qtd_artist
    ,t1.song
    ,t3.qtd_song
FROM PUBLIC."Billboard" AS t1
LEFT JOIN cte_artist AS t2 ON (t1.artist = t2.artist)
LEFT JOIN cte_song AS t3 ON (t1.song = t3.song);
```

CTEs

Common Table Expressions
OU
Expressões de Tabela Comuns

"Uma CTE tem o uso bem similar ao de uma subquery ou tabela derivada, com a vantagem do conjunto de dados poder ser utilizado mais de uma vez na consulta, ganhando performance (nessa situação) e também, melhorando a legibilidade do código. Por estes motivos, o uso da CTE tem sido bastante difundido como substituição à outras soluções citadas." Dirceu Resende

Sintaxe:

```
WITH expression_name [ ( column_name [,...n] ) ]

AS

( CTE_query_definition )
```

Window Function

Para que serve?

Poupar esforços em ações de ranqueamento e classificação dentro do banco de dados. PAra isso, o Postgres cria uma partição dos dados(window).

Elas complementam as funções de SUM, COUNT, AVG, MAX e MIN.

Podem ser:

- numeração de registros: ROW_NUMBER()
- ranqueamento: RANK(), DENSE_RANK(), PERCENT_RANK()
- subdivisão: NTILE(), LAG(), lead)
- recuperação de registros: FIRST_VALUE(), LAST_VALUE(), NTH_VALUE()
- distancia relativa: CUME_DIST()

```
with CTE_BILLBOARD as (
select distinct
    t1.artist
    ,t1.song
FROM PUBLIC."Billboard" AS t1
order by t1.artist
    ,t1.song
)
select *
    ,row_number() over(order by artist, song) as "row_number"
    ,row_number() over(partition by artist order by artist, song) as
```

```
"row_number_by_artist"
    ,rank() over(partition by artist order by artist, song) as
"rank_artist"
    ,lag(song, 1) over(order by artist, song) as "lag_song"
    ,lead(song, 1) over(order by artist, song) as "lead_song"
    ,first_value(song) over(partition by artist order by artist, song) as
"first_song"
    ,last_value(song) over(partition by artist order by artist, song
RANGE BETWEEN
            UNBOUNDED PRECEDING AND
            UNBOUNDED FOLLOWING) as "last_song"
    ,nth_value (song,2) over(partition by artist order by artist, song )
as "nth_song"
from CTE_BILLBOARD
```

Exemplo geral

```
WITH T(StyleID, ID, Nome)
     AS (SELECT 1,1, 'Rhuan' UNION ALL
         SELECT 1,1, 'Andre' UNION ALL
         SELECT 1,2, 'Ana' UNION ALL
         SELECT 1,2, 'Maria' UNION ALL
         SELECT 1,3, 'Letícia' UNION ALL
         SELECT 1,3, 'Lari' UNION ALL
         SELECT 1,4, 'Edson' UNION ALL
         SELECT 1,4, 'Marcos' UNION ALL
         SELECT 1,5, 'Rhuan' UNION ALL
         SELECT 1,5, 'Lari' UNION ALL
         SELECT 1,6, 'Daisy' UNION ALL
         SELECT 1,6, 'João'
         )
SELECT *,
       ROW_NUMBER() OVER(PARTITION BY StyleID ORDER BY ID) AS
"ROW_NUMBER",
       RANK() OVER(PARTITION BY StyleID ORDER BY ID)       AS "RANK",
       DENSE_RANK() OVER(PARTITION BY StyleID ORDER BY ID) AS
"DENSE_RANK",
       PERCENT_RANK() OVER(PARTITION BY StyleID ORDER BY ID) AS
"PERCENT_RANK",
       CUME_DIST() OVER(PARTITION BY StyleID ORDER BY ID) AS "CUME_DIST",
       CUME_DIST() OVER(PARTITION BY StyleID ORDER BY ID DESC) AS
"CUME_DIST_DESC",
```

```
       FIRST_VALUE(Nome) OVER(PARTITION by StyleID ORDER BY ID) AS
"FIRST_VALUE",
       LAST_VALUE(Nome) OVER(PARTITION by StyleID ORDER BY ID) AS
"LAST_VALUE",
       NTH_VALUE(Nome,5) OVER(PARTITION by StyleID ORDER BY ID) AS
"NTH_VALUE",
       NTILE (5) OVER (ORDER BY StyleID) as "NTILE_5",
       LAG(Nome, 1) over(order by ID) as "LAG_NOME",
       LEAD(Nome, 1) over(order by ID) as "LEAD_NOME"
FROM   T  ;
```