FUNÇÕES ANALÍTICAS

As funções analíticas são valiosas para todo tipo de processamento, desde suporte à decisão interativa até à geração de relatórios em lote. Além disso, melhoram o desempenho de consultas à base de dados e a produtividade dos programadores.
As otimizações incorporadas com as funções analíticas melhoram significativamente o desempenho das consultas.
Ações que antes requeriam múltiplos comandos SQL, ou o uso de linguagem procedimental, podem ser realizadas com comandos SQL simples, pela aplicação das funções analíticas.
As funções analíticas calculam um valor agregado com base num grupo de linhas. Porém, ao contrário das funções de agregação, as funções analíticas podem devolver várias linhas para cada grupo. Podem-se usar funções analíticas para calcular médias móveis*, totais acumulados, percentis ou os primeiros N resultados de um grupo.

As funções analíticas permitem que o conjunto resultante da consulta seja dividido em grupos ordenados de linhas chamados de partições. Essa divisão pode estar baseada em qualquer coluna ou expressão. Um resultado de consulta pode ter uma única partição, com todas as linhas, ou várias pequenas partições com poucas linhas. Cada cálculo executado com uma função analítica está baseado numa linha corrente dentro de uma partição.

Função RANK( )

As linhas (registos) dentro de uma partição que têm os mesmos valores receberão a mesma classificação. A classificação da primeira linha dentro de uma partição é um (1). No resultado da função RANK () as classificações podem não ser consecutivas.

SÍNTAXE:

```
RANK() OVER (
    [PARTITION BY partition_expression, ... ]
    ORDER BY sort_expression [ASC | DESC], ...
)
```

- A cláusula PARTITION BY divide as linhas nas partições do conjunto de resultados às quais a função é aplicada.
- A cláusula ORDER BY especifica a ordem de classificação lógica das linhas em cada partição à qual a função é aplicada.

*Por exemplo médias de vendas em cada mês do ano

Exemplo 1:

Considera a seguinte tabela, de nome rank_demo, com um campo V

	v
▶	A
	B
	B
	C
	C
	D
	E

```
SELECT v, RANK() OVER (ORDER BY v ASC) as rank_no
FROM rank_demo;
```

Resultado:

v	rank_no
A	1
B	2
B	2
C	4
C	4
D	6
E	7

A segunda e a terceira linhas recebem a mesma classificação porque têm o mesmo valor B. As quarta e quinta linhas obtêm a classificação 4 porque a função RANK () ignora a classificação 3 e as duas também têm o mesmos valores.
A função tratou todo o conjunto de resultados como uma única partição.

Exemplo 2: Considera a tabela ao lado.

	sales_employee	fiscal_year	sale
▶	Alice	2016	150.00
	Alice	2017	100.00
	Alice	2018	200.00
	Bob	2016	100.00
	Bob	2017	150.00
	Bob	2018	200.00
	John	2016	200.00
	John	2017	150.00
	John	2018	250.00

A instrução a seguir usa a função RANK () para classificar as vendas dos funcionários, por ordem decrescente, particionadas por ano fiscal:

```
SELECT
    sales_employee, fiscal_year, sale,
    RANK() OVER (PARTITION BY fiscal_year
                 ORDER BY sale DESC
                )as  sales_rank
FROM sales;
```

Neste exemplo:

- A cláusula PARTITION BY divide os conjuntos de resultados, em partições, por ano fiscal.
- a cláusula ORDER BY classifica os funcionários por vendas (sales) por ordem decrescente.

Resultado:

```
| sales_employee | fiscal_year | sale | sales_rank |
|----------------|-------------|------|------------|
|           John |        2016 |  200 |          1 |
|          Alice |        2016 |  150 |          2 |
|            Bob |        2016 |  100 |          3 |
|            Bob |        2017 |  150 |          1 |
|           John |        2017 |  150 |          1 |
|          Alice |        2017 |  100 |          3 |
|           John |        2018 |  250 |          1 |
|          Alice |        2018 |  200 |          2 |
|            Bob |        2018 |  200 |          2 |
```

Função DENSE_RANK( )

Atribui uma classificação a todas as linhas dentro de sua partição com base na cláusula ORDER BY. Se uma partição tiver duas ou mais linhas com o mesmo valor de classificação, cada uma dessas linhas receberá a mesma classificação, não havendo lacunas na sequência dos valores classificados.

SÍNTAXE:

```
DENSE_RANK() OVER (
    [PARTITION BY partition_expression, ... ]
    ORDER BY sort_expression [ASC | DESC], ...
)
```

Exemplo 3:

A instrução seguinte usa a função DENSE_RANK () para classificar os funcionários pelo valor da venda.

```
SELECT
    sales_employee,fiscal_year, sale,
    DENSE_RANK() OVER (PARTITION BY fiscal_year
                ORDER BY sale DESC
               ) AS sales_rank
FROM sales;
```

Resultado:

	sales_employee	fiscal_year	sale	sales_rank
▶	John	2016	200.00	1
	Alice	2016	150.00	2
	Bob	2016	100.00	3
	Bob	2017	150.00	1
	John	2017	150.00	1
	Alice	2017	100.00	2
	John	2018	250.00	1
	Alice	2018	200.00	2
	Bob	2018	200.00	2

Neste exemplo:

- A cláusula PARTITION BY dividiu os conjuntos de resultados em partições usando o ano fiscal.
- A cláusula ORDER BY especificou a ordem dos funcionários por vendas, por ordem decrescente.
- A função DENSE_RANK () é aplicada a cada partição com a ordem das linhas especificada pela cláusula ORDER BY, não havendo lacunas na sequência dos valores classificados (em 2017 há dois valores 1, passando o valor da linha seguinte para 2 e não para 3, como aconteceria na função rank()).

OVER e funções de agregação

A cláusula OVER é usada de duas maneiras principais:

• Junto com uma cláusula ORDER BY: para se definir um cálculo baseado na ordenação de um critério considerando toda a listagem;

• Junto com cláusulas PARTITION BY e OVER BY: para se definir um cálculo baseado na ordenação de um critério particionando o resultado conforme valores de uma ou mais colunas da listagem.

Como as **funções agregadas** da cláusula **GROUP BY**, as funções analíticas também operam num subconjunto de linhas, mas não reduzem o número de linhas devolvidas pela consulta.

Exemplo 4:

Por exemplo, a seguinte consulta devolve as vendas de cada funcionário, por partição de ano fiscal, juntamente com as vendas totais dos funcionários por ano fiscal:

```
SELECT
    fiscal_year, sales_employee, sale,
    SUM(sale) OVER (PARTITION BY fiscal_year) total_sales
FROM sales;
```

Resultado:

fiscal_year	sales_employee	sale	total_sales
2016	Alice	150	450
2016	Bob	100	450
2016	John	200	450
2017	John	150	400
2017	Bob	150	400
2017	Alice	100	400
2018	Alice	200	650
2018	Bob	200	650
2018	John	250	650

Se fosse usada apenas a função e agregação SUM poderíamos obter o total de vendas por ano fiscal, mas sem poder mostrar todas as linhas correspondentes a cada ano fiscal.

fiscal_year	total_sales
2016	450
2017	400
2018	650

Função FIRST_VALUE()

É uma função que permite selecionar a primeira linha de um quadro, partição ou conjunto de resultados da janela.

SÍNTAXE:

```
FIRST_VALUE(expression) OVER (
        [partition_clause]
        [order_clause]
        [frame_clause]
)
```

- A função FIRST_VALUE () devolve o valor da expressão da primeira linha referente ao resulltado da seleção.
- A cláusula partition_clause divide as linhas dos conjuntos de resultados em partições às quais a função se aplica independentemente.
- A cláusula order_clause especifica a ordem lógica das linhas em cada partição na qual a função FIRST_VALUE () opera.
- A frame_clause define o subconjunto (ou quadro) da partição atual.

Exemplo 5:

A declaração a seguir obtém o nome do funcionário, horas extras e o funcionário que tem menos horas extra:

```
SELECT
    employee_name,
    hours,
    FIRST_VALUE(employee_name) OVER (
        ORDER BY hours
    ) least_over_time
FROM
    overtime;
   sales;
```

Resultado:

employee_name	hours	least_over_time
Steve Patterson	29	Steve Patterson
Diane Murphy	37	Steve Patterson
Jeff Firrelli	40	Steve Patterson
Gerard Bondur	47	Steve Patterson
Loui Bondur	49	Steve Patterson
William Patterson	58	Steve Patterson
Barry Jones	65	Steve Patterson
Foon Yue Tseng	65	Steve Patterson
Anthony Bow	66	Steve Patterson
Gerard Hernandez	66	Steve Patterson
Mary Patterson	74	Steve Patterson

Neste exemplo, a cláusula ORDER BY ordenou as linhas no resultado definido por horas e o FIRST_VALUE () selecionou a primeira linha indicando o funcionário que tinha menos horas extras.

Exemplo 6: Localiza o funcionário que tem menos horas extras em todos os departamentos.

```
SELECT
    employee_name,department,hours
    FIRST_VALUE(employee_name) OVER (
        PARTITION BY department
        ORDER BY hours
    ) AS least_over_time
FROM overtime;
```

Resultado:

employee_name	department	hours	least_over_time
Diane Murphy	Accounting	37	Diane Murphy
Jeff Firrelli	Accounting	40	Diane Murphy
Mary Patterson	Accounting	74	Diane Murphy
Gerard Bondur	Finance	47	Gerard Bondur
William Patterson	Finance	58	Gerard Bondur
Anthony Bow	Finance	66	Gerard Bondur
Leslie Thompson	IT	88	Leslie Thompson
Leslie Jennings	IT	90	Leslie Thompson
Loui Bondur	Marketing	49	Loui Bondur
Gerard Hernandez	Marketing	66	Loui Bondur
George Vanauf	Marketing	89	Loui Bondur
Steve Patterson	Sales	29	Steve Patterson
Foon Yue Tseng	Sales	65	Steve Patterson
Julie Firrelli	Sales	81	Steve Patterson
Barry Jones	SCM	65	Barry Jones
Pamela Castillo	SCM	96	Barry Jones
Larry Bott	SCM	100	Barry Jones

Neste exemplo:

• A cláusula PARTITION BY dividiu os funcionários em partições por departamentos. Em outras palavras, cada partição consiste em funcionários que pertencem ao mesmo departamento.

• A cláusula ORDER BY especificou as ordens das linhas em cada partição.

• O FIRST_VALUE () opera em cada partição classificada por horas. Ele retornou a primeira linha em cada partição, que é o funcionário que tem menos horas extras no departamento.

EXERCÍCIOS:

Todos os resultados devem ser guardados notepad com o nome ficha9.

Usa o editor online : http://sqlfiddle.com/ , opção de servidor SQL server 2017 ou o WampServer, para resolver as seguintes questões:

1. Cria a tabela do exemplo2, da página 2, com tipo de dados adequados, como nome Vendas_emp.

2. Insere os dados na tabela vendas_emp.

3. Replica o exemplo2 e o exemplo 3, de utilização da função rank e Dense_rank, respetivamente.

4. Replica a querie do exemplo 4, no que se refere à utilização da cláusula over aplicada a partições.

   Seleciona a base de dados **vendascd**.
5. Tendo como exemplo as queries para utilização da função FIRST_VALUE() **e SUM(),** relativas aos exemplos 5, 6 e 4**, respetivamente,** executa as instruções SQL que te permita obter:
   a. O título do cd, preco e o CD com menor preço (usa um alias para o Cd_com_menor_preco);

   b. O nome da loja e a data, das lojas que têm menos quantidades vendidas em cada uma das Datas;

   c. A data, nome da loja e qtd, por partição de data, juntamente com o total de qtd vendidas, por data.

6. Usa a função rank() para atribuir uma classificação particionando por data, ordenada pelo valor da quantidade vendida e para valores acima dos 35. Mostrar o nome da loja, data e qtd e resultado do rank com o nome *vendas_sup_a_35*.

Bibiografia
https://docs.microsoft.com/pt-br/sql/t-sql/functions/functions?view=sql-server-2017#analytic-functions
https://www.embrapa.br/busca-de-publicacoes/-/publicacao/8644/funcoes-analiticas-da-linguagem-sql-do-oracle
http://www.sqlservertutorial.net/sql-server-window-functions/
http://www.sqlservertutorial.net/sql-server-window-functions
https://www.youtube.com/watch?v=beK2IhdFnXY