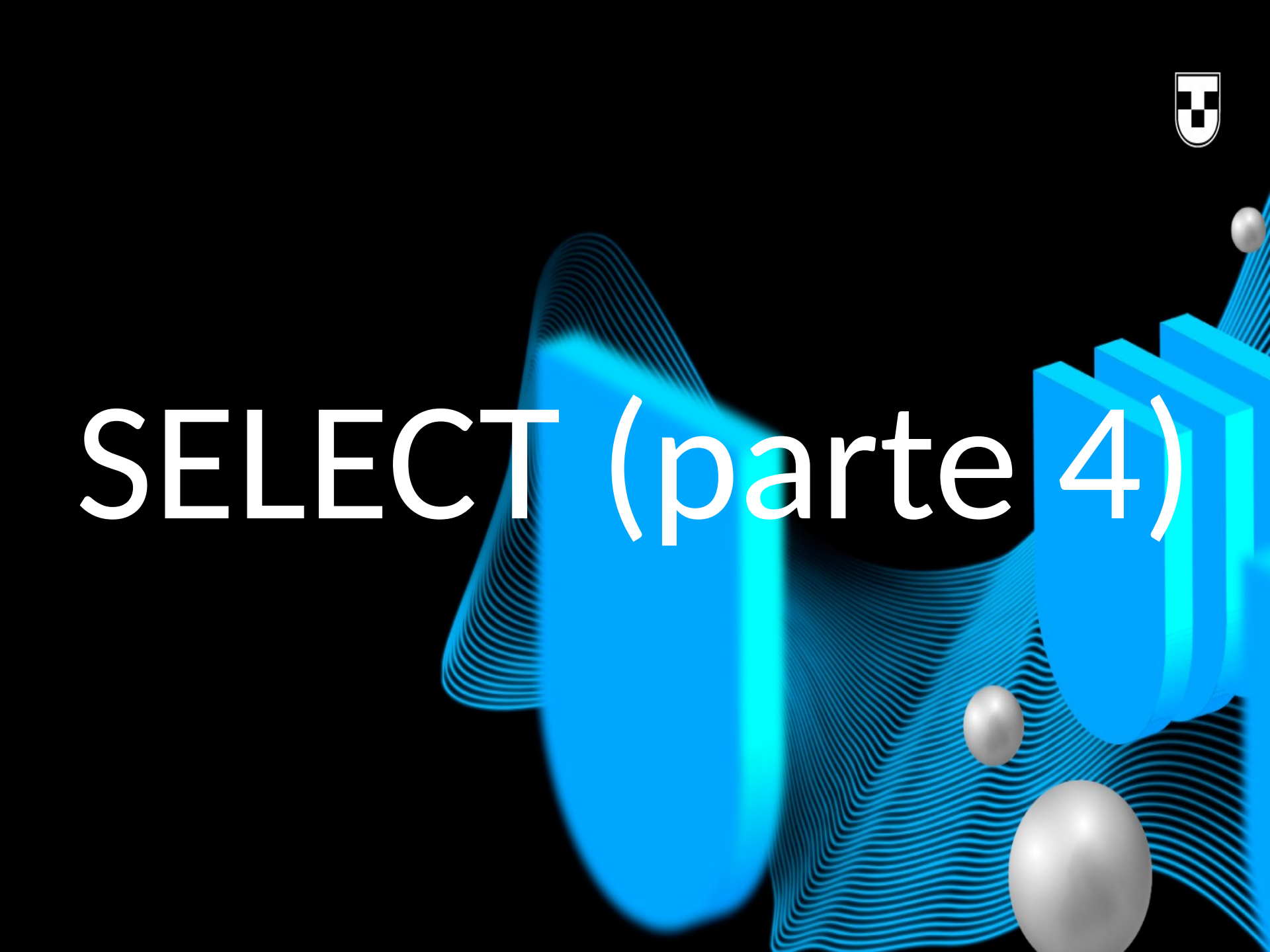

UNITAU
Universidade de Taubaté


SELECT (parte 4)

OFFSET

OFFSET start { ROW | ROWS }

Essa cláusula permite pular um certo número de linhas no resultado que será retornado.

Esta cláusula normalmente é utilizado com ORDER BY pois sem determinar uma ordem específica, não é possível se prever quais linhas serão puladas.

Exemplo:

```
SELECT nome, nascimento FROM funcionario ORDER BY nascimento
OFFSET 10 ROWS;
```

nome	nascimento
Marta	1979-11-15
Andreia	1982-07-21
Andre	1983-08-03
Sandra	1983-09-14
Luis	1984-05-18
Pedro	1984-10-05

OFFSET

A cláusula **OFFSET** foi incluída no padrão da linguagem SQL a partir da versão **SQL:2008**. O PostgreSQL já implementava esse recurso com a mesma cláusula **OFFSET** mas sem o uso do termo **ROW** ou **ROWS**. Por uma questão de compatibilidade, deve ser dada preferência para a sintaxe padrão.

Exemplo:

```
SELECT nome, nascimento FROM funcionario ORDER BY nascimento
OFFSET 10;
```

```
      nome        | nascimento
------------------+------------
 Marta            | 1979-11-15
 Andreia          | 1982-07-21
 Andre            | 1983-08-03
 Sandra           | 1983-09-14
 Luis             | 1984-05-18
 Pedro            | 1984-10-05
```

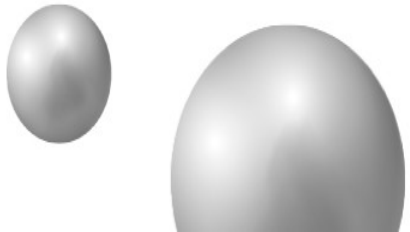
FETCH

FETCH { FIRST | NEXT } [count] { ROW | ROWS } { ONLY | WITH TIES }

Essa cláusula permite determinar o número máximo de linhas que será retornado. Se não for especificado o número de linhas a retornar, será usado 1 como default. Se o número de linhas a retornar for menor que o limite especificado, a cláusula não terá efeito.

Essa cláusula normalmente é utilizado com ORDER BY pois sem determinar uma ordem específica, não é possível se prever quais linhas serão retornadas.

O uso do termo FIRST ou NEXT e de ROW ou ROWS não interfere no resultado da consulta.

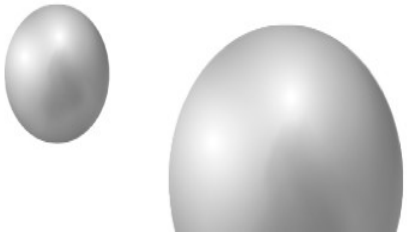


FETCH

Exemplo:

```
SELECT nome, nascimento FROM funcionario ORDER BY nascimento
FETCH FIRST 4 ROWS ONLY;
      nome        | nascimento
------------------+------------
 Claudia          | 1971-04-21
 Marcos           | 1971-04-21
 Nanci            | 1972-01-29
 Luana            | 1972-03-26
```

```
SELECT nome, nascimento FROM funcionario ORDER BY nascimento
FETCH FIRST ROW ONLY;
      nome        | nascimento
------------------+------------
 Claudia          | 1971-04-21
```



FETCH

A claúsula ONLY retorna apenas o número de linhas especificado. A cláusula WITH TIES retorna também as demais linhas que tenham o mesmo valor da expressão de ordenação da última linha selecionada.

Exemplo:

```
SELECT nome, nascimento FROM funcionario ORDER BY nascimento
FETCH FIRST ROW ONLY;
      nome        | nascimento
------------------+------------
 Claudia          | 1971-04-21
```

```
SELECT nome, nascimento FROM funcionario ORDER BY nascimento
FETCH FIRST ROW WITH TIES;
      nome        | nascimento
------------------+------------
 Marcos           | 1971-04-21
 Claudia          | 1971-04-21
```

LIMIT

A cláusula **FETCH** foi inclusa no padrão da linguagem SQL a partir do **SQL:2008**. O PostgreSQL já implementava o mesmo recurso com a cláusula **LIMIT** que ainda pode ser utilizada com o PostgreSQL, porém, por uma questão de compatibilidade, deve ser dada preferência para a cláusula padrão.

Exemplo:

```
SELECT nome, nascimento FROM funcionario ORDER BY nascimento
LIMIT 4;
```

```
      nome        | nascimento
------------------+------------
 Claudia          | 1971-04-21
 Marcos           | 1971-04-21
 Nanci            | 1972-01-29
 Luana            | 1972-03-26
```

OFFSET/FETCH

Segundo o padrão da linguagem SQL, se forem utilizadas as cláusulas OFFSET e FETCH na mesma consulta, a cláusula OFFSET deve vir antes da cláusula FETCH.

Por compatibilidade com a sintaxe usada com as cláusulas LIMIT e OFFSET, o PostgreSQL aceita que a cláusula FETCH seja colocada antes da cláusula OFFSET.

OFFSET/FETCH

Exemplo:

```
SELECT nome, nascimento FROM funcionario ORDER BY nascimento
OFFSET 5 ROWS FETCH FIRST 4 ROWS ONLY;
      nome        | nascimento
------------------+------------
 Ana              | 1973-12-04
 Raquel           | 1974-04-30
 Sandro           | 1974-10-09
 Luis             | 1976-08-12
```

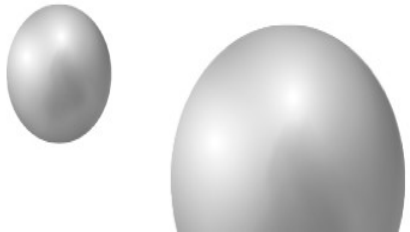
```
SELECT nome, nascimento FROM funcionario ORDER BY nascimento
FETCH FIRST 4 ROWS ONLY OFFSET 5 ROWS;
      nome        | nascimento
------------------+------------
 Ana              | 1973-12-04
 Raquel           | 1974-04-30
 Sandro           | 1974-10-09
 Luis             | 1976-08-12
```

Window Functions

Window functions ou funções de janelamento ou funções de janela, são um conjunto de funções que permitem realizar cálculos relacionando a linha corrente com outras linhas correlacionadas. O conjunto de linhas correlacionadas é considerado uma janela ou partição.

O uso de funções de janelamento permite a execução de análises que, de outra forma, exigiriam subconsultas ou tratamento com linguagens procedurais para serem obtidas.

As funções de janelamento foram introduzidas na versão SQL:2003 e detalhadas na versão SQL:2008.



Window Functions

Exemplo:

```
SELECT data,valor,SUM(valor) OVER(ORDER BY data) AS soma
FROM venda;
```

data	valor	soma
2010-01-07	28000.00	28000.00
2010-01-10	29500.00	57500.00
2010-01-21	15500.00	73000.00
2010-01-25	22100.00	95100.00
2010-02-05	17500.00	112600.00
2010-02-06	11500.00	124100.00
2010-02-15	28000.00	152100.00
2010-02-19	17500.00	169600.00
2010-02-23	31000.00	200600.00
2010-03-02	39500.00	240100.00
2010-03-10	24500.00	264600.00
2010-03-11	21500.00	286100.00
2010-03-12	42000.00	328100.00

Window Functions

Funções de agregação também permitem realizar cálculos sobre um grupo de linhas, porém, agrupam as linhas, apresentando um único resultado para cada grupo.

Funções de janelamento não agrupam as linhas em um único resultado mas permitem que a expressão seja calculada considerando todas as linhas da partição.

Window Functions

```
SELECT fabricante.nome, SUM(valor) AS total FROM venda,
automovel,fabricante WHERE automovel.codigo=venda.automovel AND
fabricante.codigo=automovel.fabricante AND data<='2010/02/28' GROUP BY
fabricante.nome ORDER BY nome;
    nome     |  total
-------------+----------
 Chevrolet   | 44500.00
 Fiat        | 28000.00
 Ford        | 39600.00
 Volkswagen | 88500.00
SELECT fabricante.nome, valor, SUM(valor) OVER(PARTITION BY
fabricante.nome) AS total FROM venda,automovel,fabricante WHERE
automovel.codigo=venda.automovel AND fabricante.codigo =
automovel.fabricante AND data<='2010/02/28' ORDER BY nome;
    nome     |  valor   |  total
-------------+----------+----------
 Chevrolet   | 15500.00 | 44500.00
 Chevrolet   | 17500.00 | 44500.00
 Chevrolet   | 11500.00 | 44500.00
 Fiat        | 28000.00 | 28000.00
 Ford        | 17500.00 | 39600.00
 Ford        | 22100.00 | 39600.00
 Volkswagen | 29500.00 | 88500.00
 Volkswagen | 31000.00 | 88500.00
 Volkswagen | 28000.00 | 88500.00
```

OVER

Para utilizar funções de janelamento é sempre obrigatório definir o particionamento das linhas com a cláusula **OVER**.

```
OVER( partition-clause order-clause frame-clause )
```

partition-clause permite a divisão das linhas em partições, se não for definida, todas as linhas serão parte de uma única partição, é definido por:

```
PARTITION BY expressão [, … ]
```

order-clause permite ordenar as linhas dentro da partição, é definido por:

```
ORDER BY expressão [ ASC | DESC | USING operator ] [ NULLS { FIRST | LAST } ] [, … ]
```

frame-clause define o frame dentro da partição que será utilizado para o cálculo da função de janelamento.

OVER

O uso de **OVER** sem nenhuma cláusula faz que todas as linhas sejam parte de uma única partição e um único frame.

Exemplo:

```
SELECT conta, data,valor,SUM( valor ) OVER() FROM lancamento
WHERE conta='01';
```

conta	data	valor	sum
01	2012-01-01	120.00	160.00
01	2012-01-30	500.00	160.00
01	2012-01-30	-600.00	160.00
01	2012-02-20	700.00	160.00
01	2012-02-28	-600.00	160.00
01	2012-03-27	700.00	160.00
01	2012-03-27	-110.00	160.00
01	2012-03-30	-600.00	160.00
01	2012-04-27	750.00	160.00
01	2012-04-30	-700.00	160.00

Window Frame

O frame é o conjunto de linhas dentro da partição, que será utilizado para calcular funções de janelamento que dependam do frame. Algumas funções de janelamento são calculadas sobre todas as linhas da partição, algumas apenas sobre as linhas do frame.

Atualmente, as funções que são calculadas sobre o frame são:

- funções de agregação usadas como funções de janelamento
- first_value
- last_value
- nth_value

Window Frame

As opções para definição do frame são:

```
[ RANGE | ROWS ] frame_start
[ RANGE | ROWS ] BETWEEN frame_start AND frame_end
```

As opções para definição do frame_start e frame_end são:

UNBOUNDED PRECEDING – inicia o frame na primeira linha da partição

value PRECEDING – inicia ou termina o frame com um offset acima da linha corrente

CURRENT ROW – incia ou termina o frame na linha corrente

value FOLLOWING – inicia ou termina o frame com um offset abaixo da linha corrente

UNBOUNDED FOLLOWING – termina o frame na última linha da partição

Window Frame

Exemplo:

```
SELECT data,valor,SUM(valor) OVER(ORDER BY data RANGE BETWEEN
UNBOUNDED PRECEDING AND CURRENT ROW), SUM(valor)
OVER(ORDER BY data RANGE BETWEEN CURRENT ROW AND
UNBOUNDED FOLLOWING) FROM venda;
```

data	valor	sum	sum
2010-01-07	28000.00	28000.00	328100.00
2010-01-10	29500.00	57500.00	300100.00
2010-01-21	15500.00	73000.00	270600.00
2010-01-25	22100.00	95100.00	255100.00
2010-02-05	17500.00	112600.00	233000.00
2010-02-06	11500.00	124100.00	215500.00
2010-02-15	28000.00	152100.00	204000.00
2010-02-19	17500.00	169600.00	176000.00
2010-02-23	31000.00	200600.00	158500.00
2010-03-02	39500.00	240100.00	127500.00
2010-03-10	24500.00	264600.00	88000.00
2010-03-11	21500.00	286100.00	63500.00
2010-03-12	42000.00	328100.00	42000.00

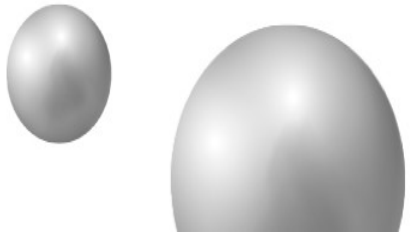
Window Frame

Se não for especificado o **frame_end**, será utilizado **CURRENT ROW**.

O **frame_end** não pode ser uma linha anterior a **frame_start**.

O uso de **OVER** sem **ORDER BY** e sem uma cláusula para frame faz com que toda a partição seja um único frame.

Se for especificada uma ordenação mas sem uma cláusula para frame, será usado como padrão **RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW**.



Window Frame

Exemplo:

```
SELECT conta,data,valor,SUM(valor) OVER(), SUM(valor)
OVER( PARTITION BY conta ), SUM(valor) OVER( PARTITION BY conta
ORDER BY data) FROM lancamento WHERE valor>0 AND conta IN
('01','04');
```

conta	data	valor	sum	sum	sum
01	2012-01-01	120.00	8070.00	2770.00	120.00
01	2012-01-30	500.00	8070.00	2770.00	620.00
01	2012-02-20	700.00	8070.00	2770.00	1320.00
01	2012-03-27	700.00	8070.00	2770.00	2020.00
01	2012-04-27	750.00	8070.00	2770.00	2770.00
04	2012-01-01	400.00	8070.00	5300.00	400.00
04	2012-01-20	900.00	8070.00	5300.00	1300.00
04	2012-02-20	1100.00	8070.00	5300.00	2400.00
04	2012-03-20	1600.00	8070.00	5300.00	4000.00
04	2012-04-20	1300.00	8070.00	5300.00	5300.00

RANGE / GROUPS

Se utilizada a cláusula RANGE ou GROUPS, CURRENT ROW corresponderá a primeira/última linha das linhas empatadas com a linha corrente, segundo o critério de ordenação.

RANGE / GROUPS

Exemplo:

```
SELECT data,valor,SUM(valor) OVER( ORDER BY valor RANGE BETWEEN
UNBOUNDED PRECEDING AND CURRENT ROW) FROM lancamento
WHERE conta='01' AND valor<0;
```

data	valor	sum
2012-04-30	-700.00	-700.00
2012-01-30	-600.00	-2500.00
2012-02-28	-600.00	-2500.00
2012-03-30	-600.00	-2500.00
2012-03-27	-110.00	-2610.00

```
SELECT data,valor,SUM(valor) OVER( ORDER BY valor GROUPS
BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) FROM lancamento
WHERE conta='01' AND valor<0;
```

data	valor	sum
2012-04-30	-700.00	-700.00
2012-01-30	-600.00	-2500.00
2012-02-28	-600.00	-2500.00
2012-03-30	-600.00	-2500.00
2012-03-27	-110.00	-2610.00

ROWS

Se utilizada com a cláusula ROWS, CURRENT ROW corresponderá exatamente a linha corrente.

Exemplo:

```
SELECT data,valor,SUM(valor) OVER( ORDER BY valor ROWS BETWEEN
UNBOUNDED PRECEDING AND CURRENT ROW) FROM lancamento
WHERE conta='01' AND valor<0;
```

data	valor	sum
2012-04-30	-700.00	-700.00
2012-01-30	-600.00	-1300.00
2012-02-28	-600.00	-1900.00
2012-03-30	-600.00	-2500.00
2012-03-27	-110.00	-2610.00

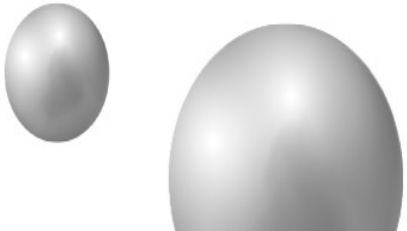
value PRECEDING/FOLLOWING

Se for utilizado um offset para o início ou final do frame, o offset será considerado segundo a cláusula do frame:

ROWS – o offset determina o número de linhas acima ou abaixo da linha corrente

GROUPS – o offset determina o número de grupos acima ou abaixo da linha corrente

RANGE – o offset determina o valor a ser subtraido/adicionado ao valor da expressao de ordenação da linha corrente, nesse caso, a expressão de ordenação só pode ter um único campo



value PRECEDING/FOLLOWING

Exemplo:

```
SELECT data,valor,SUM(valor) OVER( ORDER BY valor ROWS
BETWEEN 1 PRECEDING AND CURRENT ROW) FROM lancamento WHERE
conta='01' AND valor<0;
```

```
    data    |  valor  |   sum
------------+---------+----------
 2012-04-30 | -700.00 |  -700.00
 2012-01-30 | -600.00 | -1300.00
 2012-02-28 | -600.00 | -1200.00
 2012-03-30 | -600.00 | -1200.00
 2012-03-27 | -110.00 |  -710.00
```

```
SELECT  data,valor,SUM(valor)  OVER(  ORDER  BY  valor  GROUPS
BETWEEN 1 PRECEDING AND CURRENT ROW) FROM lancamento WHERE
conta='01' AND valor<0;
```

```
    data    |  valor  |   sum
------------+---------+----------
 2012-04-30 | -700.00 |  -700.00
 2012-01-30 | -600.00 | -2500.00
 2012-02-28 | -600.00 | -2500.00
 2012-03-30 | -600.00 | -2500.00
 2012-03-27 | -110.00 | -1910.00
```

value PRECEDING/FOLLOWING

Exemplo:

```
SELECT data,valor,SUM(valor) OVER( ORDER BY valor RANGE
BETWEEN 99 PRECEDING AND CURRENT ROW) FROM lancamento WHERE
conta='01' AND valor<0;
```

```
    data    |  valor  |   sum
------------+---------+----------
 2012-04-30 | -700.00 |  -700.00
 2012-01-30 | -600.00 | -1800.00
 2012-02-28 | -600.00 | -1800.00
 2012-03-30 | -600.00 | -1800.00
 2012-03-27 | -110.00 |  -110.00
```

```
SELECT data,valor,SUM(valor) OVER( ORDER BY valor RANGE
BETWEEN 100 PRECEDING AND CURRENT ROW) FROM lancamento WHERE
conta='01' AND valor<0;
```

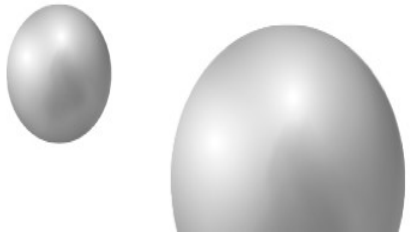
```
    data    |  valor  |   sum
------------+---------+----------
 2012-04-30 | -700.00 |  -700.00
 2012-01-30 | -600.00 | -2500.00
 2012-02-28 | -600.00 | -2500.00
 2012-03-30 | -600.00 | -2500.00
 2012-03-27 | -110.00 |  -110.00
```

Window Functions

As funções de agregação podem ser utilizadas como funções de janelamento desde que utilizadas com a cláusula **OVER**. Sem a cláusula **OVER**, estas funções atuam como funções de agregação regulares.

Existem várias funções de janelamento específicas implementadas no **PostreSQL**.

Também é possível utilizar funções definidas pelo usuário como funções de janelamento.



ROW_NUMBER

row_number() - retorna o número da linha corrente dentro da partição

Exemplo:

```
SELECT conta,data,valor,ROW_NUMBER() OVER( PARTITION BY
conta ORDER BY valor ) FROM lancamento WHERE valor>0 AND
conta IN ('01','04');
```

conta	data	valor	row_number
01	2012-01-01	120.00	1
01	2012-01-30	500.00	2
01	2012-02-20	700.00	3
01	2012-03-27	700.00	4
01	2012-04-27	750.00	5
04	2012-01-01	400.00	1
04	2012-01-20	900.00	2
04	2012-02-20	1100.00	3
04	2012-04-20	1300.00	4
04	2012-03-20	1600.00	5

RANK

rank() - retorna o rank da linha corrente dentro da partição, no caso de empates, as linhas empatadas terão o mesmo rank, causando um intervalo no rank.

Exemplo:

```
SELECT conta,data,valor,RANK() OVER( PARTITION BY conta
ORDER BY valor ) FROM lancamento WHERE valor>0 AND conta IN
('01','04');
```

conta	data	valor	rank
01	2012-01-01	120.00	1
01	2012-01-30	500.00	2
01	2012-02-20	700.00	3
01	2012-03-27	700.00	3
01	2012-04-27	750.00	5
04	2012-01-01	400.00	1
04	2012-01-20	900.00	2
04	2012-02-20	1100.00	3
04	2012-04-20	1300.00	4
04	2012-03-20	1600.00	5

DENSE_RANK

dense_rank() - retorna o rank da linha corrente sem intervalos, esta função conta por grupos.

Exemplo:

```
SELECT conta,data,valor,DENSE_RANK() OVER( PARTITION BY
conta ORDER BY valor ) FROM lancamento WHERE valor>0 AND
conta IN ('01','04');
```

conta	data	valor	dense_rank
01	2012-01-01	120.00	1
01	2012-01-30	500.00	2
01	2012-02-20	700.00	3
01	2012-03-27	700.00	3
01	2012-04-27	750.00	4
04	2012-01-01	400.00	1
04	2012-01-20	900.00	2
04	2012-02-20	1100.00	3
04	2012-04-20	1300.00	4
04	2012-03-20	1600.00	5

PERCENT_RANK

percent_rank() - retorna o rank relativo (entre 0 e 1) da linha corrente, é igual a (rank - 1) / (número de linhas totais – 1).

Exemplo:

```
SELECT conta,data,valor,PERCENT_RANK() OVER( PARTITION BY
conta ORDER BY valor ) FROM lancamento WHERE valor>0 AND
conta IN ('01','04');
```

conta	data	valor	percent_rank
01	2012-01-01	120.00	0
01	2012-01-30	500.00	0.25
01	2012-02-20	700.00	0.5
01	2012-03-27	700.00	0.5
01	2012-04-27	750.00	1
04	2012-01-01	400.00	0
04	2012-01-20	900.00	0.25
04	2012-02-20	1100.00	0.5
04	2012-04-20	1300.00	0.75
04	2012-03-20	1600.00	1

CUME_DIST

cume_dist() - retorna o rank relativo da linha corrente mas considerando o número de linhas que precedem ou são iguais a linha corrente, é igual a (número de linhas que precedem ou são iguais a linha corrente) / (numero de linhas totais).

Exemplo:

```
SELECT conta,data,valor,CUME_DIST() OVER( ORDER BY valor )
FROM lancamento WHERE valor>0 AND conta='01';
```

conta	data	valor	cume_dist
01	2012-01-01	120.00	0.2
01	2012-01-30	500.00	0.4
01	2012-02-20	700.00	0.8
01	2012-03-27	700.00	0.8
01	2012-04-27	750.00	1

NTILE

ntile(numero_de_grupos) - divide a partição em grupos com tamanhos o mais próximo possível.

Exemplo:

```
SELECT codigo,modelo,ano,preco,NTILE(5) OVER( ORDER BY preco
DESC ) FROM automovel WHERE ano>='2000';
```

codigo	modelo	ano	preco	ntile
02	Golf	2007	39000.00	1
09	Golf	2005	37000.00	1
11	Polo	2007	29000.00	1
15	Polo	2006	27500.00	2
10	Siena	2006	26000.00	2
01	Gol	2000	25000.00	2
13	Palio	2007	23000.00	3
06	Fiesta	2003	20000.00	3
04	Fiesta	2002	20000.00	4
12	Fiesta	2002	18000.00	4
14	Corsa Sedan	2002	16000.00	5
08	Palio	2002	15000.00	5

LAG

lag(expressão [, offset [, default]]) - retorna o resultado de **expressão** da linha situada **offset** linhas acima da linha corrente, se a linha não existir, retorna **default**, o valor padrão para **offset** é 1 e para **default** é NULL.

Exemplo:

```
SELECT data,valor,LAG( valor, 2) OVER( ORDER BY valor ) FROM
lancamento WHERE conta='01' AND valor<0;
```

```
    data    |  valor  |   lag
------------+---------+---------
 2012-04-30 | -700.00 |
 2012-01-30 | -600.00 |
 2012-02-28 | -600.00 | -700.00
 2012-03-30 | -600.00 | -600.00
 2012-03-27 | -110.00 | -600.00
```

LEAD

lead(expressão [, offset [, default]]) - retorna o resultado de **expressão** da linha situada **offset** linhas abaixo da linha corrente, se a linha não existir, retorna **default**, o valor padrão para **offset** é 1 e para **default** é NULL.

Exemplo:

```
SELECT data,valor,LEAD( valor, 2) OVER( ORDER BY valor )
FROM lancamento WHERE conta='01' AND valor<0;
```

```
    data    |  valor  |  lead
------------+---------+---------
 2012-04-30 | -700.00 | -600.00
 2012-01-30 | -600.00 | -600.00
 2012-02-28 | -600.00 | -110.00
 2012-03-30 | -600.00 |
 2012-03-27 | -110.00 |
```

FIRST_VALUE

first_value(expressão) – retorna o valor de **expressão** para a primeira linha do frame da linha corrente.

Exemplo:

```
SELECT conta, data,valor,FIRST_VALUE( valor )
OVER( PARTITION BY conta ORDER BY data ) FROM lancamento
WHERE conta IN ( '01','04' ) AND valor<0;
```

conta	data	valor	first_value
01	2012-01-30	-600.00	-600.00
01	2012-02-28	-600.00	-600.00
01	2012-03-27	-110.00	-600.00
01	2012-03-30	-600.00	-600.00
01	2012-04-30	-700.00	-600.00
04	2012-01-20	-1000.00	-1000.00
04	2012-01-26	-100.00	-1000.00
04	2012-02-20	-1200.00	-1000.00
04	2012-02-26	-100.00	-1000.00
04	2012-03-20	-1500.00	-1000.00
04	2012-04-20	-1200.00	-1000.00
04	2012-04-26	-120.00	-1000.00

LAST_VALUE

last_value(expressão) – retorna o valor de **expressão** para a última linha do frame da linha corrente.

Exemplo:

```
SELECT conta, data,valor,LAST_VALUE( valor ) OVER( PARTITION
BY conta ORDER BY data RANGE BETWEEN CURRENT ROW AND
UNBOUNDED FOLLOWING ) FROM lancamento WHERE conta IN
( '01','04' ) AND valor<0;
```

conta	data	valor	last_value
01	2012-01-30	-600.00	-700.00
01	2012-02-28	-600.00	-700.00
01	2012-03-27	-110.00	-700.00
01	2012-03-30	-600.00	-700.00
01	2012-04-30	-700.00	-700.00
04	2012-01-20	-1000.00	-120.00
04	2012-01-26	-100.00	-120.00
04	2012-02-20	-1200.00	-120.00
04	2012-02-26	-100.00	-120.00
04	2012-03-20	-1500.00	-120.00
04	2012-04-20	-1200.00	-120.00
04	2012-04-26	-120.00	-120.00

NTH_VALUE

nth_value(expressão, numero_de_linha) - retorna o valor de **expressão** para a linha **numero_de_linha** do frame da linha corrente ou nulo se a linha não existir

Exemplo:

```
SELECT data,valor,NTH_VALUE( valor,3 ) OVER( PARTITION BY
conta ) FROM lancamento WHERE conta='01' AND valor<0;
```

```
    data    |  valor  | nth_value
------------+---------+-----------
 2012-01-30 | -600.00 |   -110.00
 2012-02-28 | -600.00 |   -110.00
 2012-03-27 | -110.00 |   -110.00
 2012-03-30 | -600.00 |   -110.00
 2012-04-30 | -700.00 |   -110.00
```

Window Functions

Funções de janelamento são calculadas depois do processamento da cláusula HAVING e antes do processamento da cláusula ORDER BY. Portanto, são permitidas apenas na lista de campos do SELECT e na cláusula ORDER BY, não podem ser usadas nas demais cláusulas como GROUP BY, HAVING, WHERE. Funções de janelamento são processadas depois das funções de agregação regulares.

WINDOW

Se for usada a mesma partição para mais de uma função de janelamento, pode ser usada a cláusula WINDOW para definir a partição uma única vez.

Exemplo:

```
SELECT conta,data,valor,SUM(valor) OVER(w), AVG(valor)
OVER(w) FROM lancamento WHERE valor>0 AND conta IN
('01','04') WINDOW w AS (PARTITION BY conta);
```

conta	data	valor	sum	avg
01	2012-04-27	750.00	2770.00	554.000000000000000
01	2012-01-30	500.00	2770.00	554.000000000000000
01	2012-02-20	700.00	2770.00	554.000000000000000
01	2012-03-27	700.00	2770.00	554.000000000000000
01	2012-01-01	120.00	2770.00	554.000000000000000
04	2012-04-20	1300.00	5300.00	1060.000000000000000
04	2012-01-01	400.00	5300.00	1060.000000000000000
04	2012-01-20	900.00	5300.00	1060.000000000000000
04	2012-03-20	1600.00	5300.00	1060.000000000000000
04	2012-02-20	1100.00	5300.00	1060.000000000000000

WINDOW

É possível complementar a definição da partição para alguma das funções de janelamento que utilizem uma partição definida pela cláusula **WINDOW**.

Exemplo:

```
SELECT conta,data,valor,SUM(valor) OVER(w ORDER BY data),
AVG(valor) OVER(w) FROM lancamento WHERE valor>0 AND conta
IN ('01','04') WINDOW w AS (PARTITION BY conta);
```

conta	data	valor	sum	avg
01	2012-01-01	120.00	120.00	554.000000000000000
01	2012-01-30	500.00	620.00	554.000000000000000
01	2012-02-20	700.00	1320.00	554.000000000000000
01	2012-03-27	700.00	2020.00	554.000000000000000
01	2012-04-27	750.00	2770.00	554.000000000000000
04	2012-01-01	400.00	400.00	1060.000000000000000
04	2012-01-20	900.00	1300.00	1060.000000000000000
04	2012-02-20	1100.00	2400.00	1060.000000000000000
04	2012-03-20	1600.00	4000.00	1060.000000000000000
04	2012-04-20	1300.00	5300.00	1060.000000000000000

WITH

A cláusula **WITH** permite executar um ou mais comandos auxiliares que podem ser referenciados na consulta principal. Esses comandos auxiliares são chamados de **Common Table Expressions** ou **CTEs**

```
[ WITH [ RECURSIVE ] with_query [, ...] ]

SELECT ...
```

Os comandos auxiliares criarão tabelas temporárias durante a execução da consulta principal. Os comandos podem ser um **SELECT**, **VALUES**, **INSERT**, **UPDATE** ou **DELETE**.

É obrigatório definir um nome para cada comando auxiliar do **WITH**. Também é possível definir uma lista de nomes para as colunas resultantes do comando, caso contrário, as colunas terão os nomes de acordo com o resultado dao comando.

WITH

Exemplo:

```
WITH movimento AS ( SELECT data, produto, quantidade FROM
entrada UNION ALL SELECT data, produto, -1*quantidade FROM saida )

SELECT produto.codigo, produto.descricao, movimento.data,
SUM(movimento.quantidade) AS saldo FROM produto, movimento
WHERE produto.codigo=movimento.produto AND produto.codigo='01'
GROUP BY produto.codigo, movimento.data ORDER BY data;
```

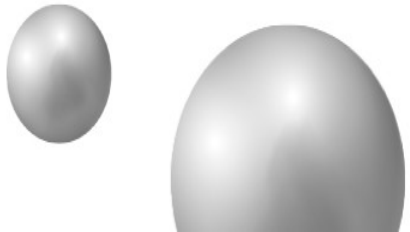
codigo	descricao	data	saldo
01	Caneta	2010-03-01	20
01	Caneta	2010-03-03	2
01	Caneta	2010-03-05	-4
01	Caneta	2010-03-10	-2
01	Caneta	2010-03-14	-1
01	Caneta	2010-03-20	8
01	Caneta	2010-03-25	-6

WITH

A cláusula **WITH** é utilizada para facilitar o uso de subconsultas, simplificando a estrutura da consulta principal.

Como os comandos auxiliares são executados uma única vez, o uso de **WITH** pode evitar que subconsultas sejam executadas mais de uma vez, o que poderia ocorrer dependendo da forma como a subconsulta fosse colocada na consulta principal.

Porém, em contrapartida, não é possível filtrar o resultado dos comandos auxiliares com elementos da consulta principal, o que pode levar a consultas menos eficientes.



WITH

Conceitualmente, a consulta principal e os comandos auxiliares são executadas ao mesmo tempo e os comandos auxiliares são executadas uma única vez, mesmo que a consulta principal referencie o comando mais de uma vez.

Porém, os comandos auxiliares podem utilizar os resultados dos comandos auxiliares anteriores, criando uma cadeia de comandos.

WITH

Exemplo:

```
WITH
a AS ( SELECT data, produto, quantidade FROM entrada),
b AS ( SELECT data, produto, -1*quantidade FROM saida ),
movimento AS ( SELECT * FROM a UNION ALL SELECT * FROM b ),
soma AS ( SELECT produto, data, SUM(quantidade) AS saldo
FROM movimento GROUP BY produto, data )
SELECT produto.codigo, produto.descricao, soma.data,
soma.saldo FROM produto, soma WHERE
produto.codigo=soma.produto AND produto.codigo='01' ORDER BY
data;
```

codigo	descricao	data	saldo
01	Caneta	2010-03-01	20
01	Caneta	2010-03-03	2
01	Caneta	2010-03-05	-4
01	Caneta	2010-03-10	-2
01	Caneta	2010-03-14	-1
01	Caneta	2010-03-20	8
01	Caneta	2010-03-25	-6

WITH

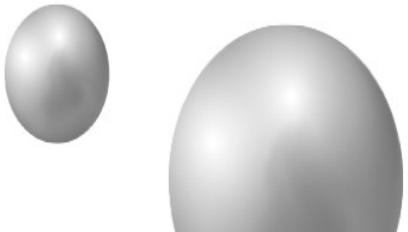
Os comandos auxiliares que alteram dados das tabelas envolvidas também são executadas uma única vez e todos ao mesmo tempo, portanto, se a consulta principal e um ou mais comandos auxiliares alterarem os mesmos dados, o resultado é imprevisível.

RECURSIVE

O uso da cláusula **RECURSIVE** permite que um comando auxiliar referencie a sua própria saída. Apesar do uso da palavra **RECURSIVE**, essa não e uma recursividade mas um processo iterativo, onde a saída de uma iteração é usada como entrada para a nova iteração.

A forma geral de uma consulta recursiva é sempre com um termo não recursivo e um **UNION** (ou **UNION ALL**) com um termo recursivo. Apenas o termo recursivo pode fazer referência à saída da própria consulta.

Em consultas recursivas, é importante que a parte recursiva da consulta retorne um resultado vazio em algum momento para interromper as iterações, senão, a consulta entrará em loop infinito.

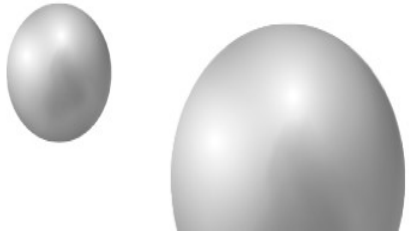


RECURSIVE

Exemplo:

```
WITH RECURSIVE t(n) AS
(
VALUES (1)
UNION ALL
SELECT n+1 FROM t WHERE n < 10
)
SELECT n FROM t;
```

```
 n
----
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
```



RECURSIVE

Consultas recursivas são utilizadas em dados hierárquicos ou organizados em árvores pois permitem uma pesquisa em profundidade.

Exemplo:

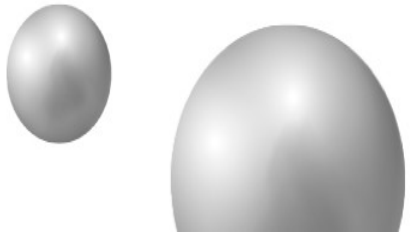
```
WITH RECURSIVE t(nivel, nome, vendas, superior, sequencia,
total) AS
(
SELECT 1, nome, vendas, superior, CAST(nome AS TEXT ),vendas
FROM organizacao WHERE superior IS NULL
UNION
SELECT nivel+1, organizacao.nome, organizacao.vendas,
organizacao.superior, sequencia || ' > ' ||
organizacao.nome,CAST(organizacao.vendas+t.total AS
NUMERIC(9,2) ) FROM organizacao, t WHERE
organizacao.superior=t.nome
)
SELECT nivel, nome, sequencia, total FROM t ORDER BY
sequencia;
```

RECURSIVE

nivel	nome	sequencia	total
1	Andreia	Andreia	8100.00
2	Fabio	Andreia > Fabio	15900.00
3	Raquel	Andreia > Fabio > Raquel	23000.00
2	Sandra	Andreia > Sandra	16500.00
3	Ana	Andreia > Sandra > Ana	23300.00
3	Silvia	Andreia > Sandra > Silvia	25400.00
1	Claudia	Claudia	7500.00
2	Andre	Claudia > Andre	15800.00
3	Luana	Claudia > Andre > Luana	21000.00
3	Marta	Claudia > Andre > Marta	23600.00
3	Sandro	Claudia > Andre > Sandro	23900.00
4	Luis	Claudia > Andre > Sandro > Luis	31100.00
4	Marcos	Claudia > Andre > Sandro > Marcos	32000.00
2	Nanci	Claudia > Nanci	15800.00
3	Paula	Claudia > Nanci > Paula	22700.00
2	Pedro	Claudia > Pedro	16700.00
1	Rodrigo	Rodrigo	8900.00

RECURSIVE

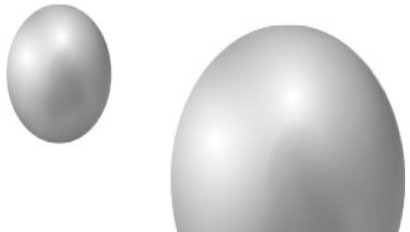
Ao utilizar consultas recursivas, deve-se garantir que a parte recursiva da consulta eventualmente não retorne nenhum registro para que a execução da consulta tenha fim. Se a parte recursiva da consulta sempre retornar novos registros, a consulta entrará em loop infinito. Limitar o resultado da consulta principal também evita o loop infinito, pois o PostgreSQL processa a consulta do WITH apenas o necessário para satisfazer a consulta principal. Porém esse comportamento pode não ser adotado em outros SGBDs.



RECURSIVE

Exemplo:

```
WITH RECURSIVE t(n) AS
(
VALUES (1)
UNION ALL
SELECT n+1 FROM t
)
SELECT n FROM t LIMIT 10;
 n
----
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
```



RECURSIVE

Utilize Window Functions e/ou WITH para resolver as questões abaixo:

01) Obter o código, data e quantidade da entrada e o total das entradas ate a data da entrada para o produto 01

02) Obter a data e valor da venda, modelo do automóvel e o total de vendas até a data da venda

03) Obter um extrato da conta 01 com a data, valor e descrição do grupo do lançamento e o saldo total da conta até a data do lançamento

04) Obter o código, data e quantidade da entrada e o total das entradas ate a entrada para o produto 01

05) Obter um extrato da conta 01 com a data, valor e descrição do grupo do lançamento e o saldo total da conta até o lançamento

RECURSIVE

06) Obter um demonstrativo das despesas com a data, valor, descrição do grupo de despesa e o total do grupo de despesa até a data, ordenado por descrição da despesa e data

07) Obter a data e valor da venda, nome do fabricante do automóvel e total das vendas dos automóveis do fabricante ate a data, ordenado por nome do fabricante e data

08) Obter um demonstrativo das despesas por conta, com o código da conta, data, valor, descrição do grupo de despesa e o total das despesa na conta até a data, ordenado pela conta e data

09) Obter um demonstrativo das despesas com a data, valor, descrição do grupo de despesa, total do grupo de despesa até a data e o total das despesas até a data, ordenado por data e descrição da despesa

RECURSIVE

10) Obter a data e valor da venda, nome do fabricante do automóvel, total das vendas dos automóveis do fabricante ate a data e total de todas as vendas ate a data, ordenado por data

11) Obter um demonstrativo das despesas por conta, com o código da conta, data, valor, descrição do grupo de despesa, total do grupo de despesas na conta até a data e total das despesa na conta até a data, ordenado pela conta, data e descrição da despesa

12) Obter o modelo e ano do automóvel, data e valor da venda e a diferença com relação a venda de maior valor

13) Obter o nome da seção, nome, salário do funcionário e a diferença entre o salário do funcionário e a média dos salários para os funcionários da diretoria de pessoal

RECURSIVE

14) Obter o código da seção, nome, salário do funcionário e a diferença entre o salário do funcionário e a média salarial da seção onde o funcionário trabalha, ordenado por seção e salário

15) Obter o código da seção, nome, salário do funcionário e a diferença entre o salário do funcionário e o maior e o menor salário da seção onde o funcionário trabalha, ordenado por seção e salário

16) Obter o nome do fabricante, modelo e ano do automóvel, data e valor da venda e a diferença com relação a venda de maior valor e a venda de menor valor dos automóveis do fabricante

17) Listar a seção, nome, salário do funcionário e a diferença com relação ao maior salário da seção e ao salãrio imediatamente superior na seção, ordenado por seção e salário

RECURSIVE

18) Obter um demonstrativo com a data da movimentação, total das receitas na data, total das receitas até a data, total das despesas na data, total das despesas até a data e total da movimentação até a data para os lançamentos de receitas e despesas, ordenados por data

19) Listar os modelos de carro, lucro total das vendas do modelo, média do total do lucro por modelo e a diferença com relação a média do total de lucro por modelo, ordenado por modelo

20) Obter a descrição do grupo de despesa, total de movimentação da despesa (inverter sinal dos lançamentos para melhor compreensão) e diferença entre o total da movimentação da despesa e a média das movimentações das despesa, ordenado pela total da movimentação

RECURSIVE

21) Listar os modelos de carro, lucro total das vendas do modelo, diferença com relação ao modelo de menor e de maior total de lucro, ordenado por modelo

22) Obter a descrição do grupo de despesa, total de movimentação da despesa (inverter sinal dos lançamentos para melhor compreensão) e diferença entre o total da movimentação da despesa e o menor e maior total de movimentação das despesa, ordenado pela total da movimentação

23) Obter o código da seção, nome e salário dos dois funcionários de maior salário de cada seção

24) Obter o nome do fabricante, modelo do automóvel, valor da venda para as duas vendas de maior valor de cada fabricante