UNION

UNION une o resultado de duas consultas.

```
consulta1 UNION [ ALL ] consulta2
```

As duas consultas devem retornar o mesmo número de colunas, e as colunas correspondentes devem ser de tipos compatíveis.

Exemplo:

```
SELECT produto FROM entrada WHERE data >='2010/03/05' AND
data <= '2010/03/07' UNION SELECT produto FROM saida
WHERE data >='2010/03/05' AND data <= '2010/03/07';
```

```
SELECT entrada.produto,produto.descricao FROM
entrada,produto WHERE produto.codigo=entrada.produto AND
entrada.data >='2010/03/05' AND entrada.data <=
'2010/03/07' UNION SELECT saida.produto,produto.descricao
FROM saida,produto WHERE produto.codigo=saida.produto AND
saida.data >='2010/03/05' AND saida.data <= '2010/03/07';
```

UNION

```
SELECT produto FROM entrada WHERE data >='2010/03/05' AND
data <= '2010/03/07' UNION SELECT produto FROM saida
WHERE data >='2010/03/05' AND data <= '2010/03/07';
```

```
 produto
---------
 04
 05
 01
 06
```

UNION

```
SELECT entrada.produto,produto.descricao FROM
entrada,produto WHERE produto.codigo=entrada.produto AND
entrada.data >='2010/03/05' AND entrada.data <=
'2010/03/07' UNION SELECT saida.produto,produto.descricao
FROM saida,produto WHERE produto.codigo=saida.produto AND
saida.data >='2010/03/05' AND saida.data <= '2010/03/07';
```

```
 produto | descricao
---------+-----------
 06      | Caderno
 01      | Caneta
 04      | Borracha
 05      | Regua
```

UNION

As linhas duplicadas serão removidas do resultado, a menos que se utilize UNION ALL.

Exemplo:

```
SELECT entrada.produto,produto.descricao FROM entrada,
produto WHERE data>='2010/03/15' AND produto.codigo =
entrada.produto UNION ALL SELECT saida.produto,
produto.descricao FROM saida,produto WHERE data >=
'2010/03/15' AND produto.codigo=saida.produto;
```

UNION

```
SELECT entrada.produto,produto.descricao FROM entrada,
produto WHERE data>='2010/03/15' AND produto.codigo =
entrada.produto UNION ALL SELECT saida.produto,
produto.descricao FROM saida,produto WHERE data >=
'2010/03/15' AND produto.codigo=saida.produto;
 produto | descricao
---------+------------
 01      | Caneta
 01      | Caneta
 02      | Lapis
 04      | Borracha
 01      | Caneta
 01      | Caneta
 01      | Caneta
 02      | Lapis
 02      | Lapis
 04      | Borracha
 04      | Borracha
 04      | Borracha
 06      | Caderno
 06      | Caderno
 06      | Caderno
 06      | Caderno
```

UNION

As combinações de consultas podem ser encadeadas.

Exemplo:

```
SELECT produto FROM saida WHERE data='2010/03/03' UNION
SELECT produto FROM saida WHERE data='2010/03/05' UNION
SELECT produto FROM saida WHERE data='2010/03/12';
```

Essa consulta é equivalente a:

```
( SELECT produto FROM saida WHERE data='2010/03/03' UNION
SELECT produto FROM saida WHERE data='2010/03/05' ) UNION
SELECT produto FROM saida WHERE data='2010/03/12';
```

UNION

```
SELECT produto FROM saida WHERE data='2010/03/03' UNION
SELECT produto FROM saida WHERE data='2010/03/05' UNION
SELECT produto FROM saida WHERE data='2010/03/12';
```

```
 produto
---------
 02
 01
 06
 04
```

INTERSECT

INTERSECT retorna a interseção entre os resultado de duas consultas (linhas presentes nas duas consultas).

```
consulta1 INTERSECT [ ALL ] consulta2
```

As duas consultas devem retornar o mesmo número de colunas, e as colunas correspondentes devem ser de tipos compatíveis. As linhas duplicadas serão removidas do resultado, a menos que se utilize **INTERSECT ALL**.

Exemplo:

```
SELECT entrada.produto,produto.descricao FROM
entrada,produto WHERE produto.codigo=entrada.produto AND
entrada.data >='2010/03/05' AND entrada.data <=
'2010/03/07' INTERSECT SELECT
saida.produto,produto.descricao FROM saida,produto WHERE
produto.codigo=saida.produto AND saida.data
>='2010/03/05' AND saida.data <= '2010/03/07';
 produto | descricao
---------+-----------
 04      | Borracha
```

EXCEPT

EXCEPT retorna a diferença entre os resultado de duas consultas (linhas presentes na primeira consulta mas que não estão presentes na segunda consulta).

```
consulta1 EXCEPT [ ALL ] consulta2
```

As duas consultas devem retornar o mesmo número de colunas, e as colunas correspondentes devem ser de tipos compatíveis. As linhas duplicadas serão removidas do resultado, a menos que se utilize **EXCEPT ALL**.

Exemplo:

```
SELECT entrada.produto,produto.descricao FROM
entrada,produto WHERE produto.codigo=entrada.produto AND
entrada.data >='2010/03/05' AND entrada.data <=
'2010/03/07' EXCEPT SELECT
saida.produto,produto.descricao FROM saida,produto WHERE
produto.codigo=saida.produto AND saida.data
>='2010/03/05' AND saida.data <= '2010/03/07';
```

```
 produto | descricao
---------+-----------
 05      | Regua
```

Funções de Agregação

As funções de agregação retornam um único valor como resultado de um conjunto de valores.

COUNT(expressão) – número de valores de entrada, para os quais a expressão não é nula, para se contar todos os elementos da entrada, é usado * como expressão.

Exemplo:

```
SELECT COUNT(*) FROM aluno;
 count
-------
    10
SELECT COUNT(curso) FROM aluno;
 count
-------
     9
```

Funções de Agregação

Pode ser usado DISTINCT para contar quantos valores distintos não nulos são retornados pela expressão.

Exemplo:

```
SELECT COUNT(DISTINCT curso) FROM aluno;
 count
-------
     2
```

Funções de Agregação

Pode ser usado CASE para agrupar o resultado.

Exemplo:

```
SELECT COUNT(CASE WHEN serie='1' THEN 1 ELSE NULL END) AS
primeiro, COUNT(CASE WHEN serie='2' THEN 1 ELSE NULL END)
AS segundo FROM aluno;

 primeiro | segundo
----------+---------
        8 |       1
```

Funções de Agregação

SUM(expressão) – soma o valor da expressão para os valores de entrada.

Exemplo:

```
SELECT SUM(quantidade) FROM entrada WHERE produto='01';
 sum
-----
  35
```

Funções de Agregação

MAX(expressão) – retorna o valor máximo da expressão para os valores de entrada.

Exemplo:

```
SELECT MAX(quantidade) FROM entrada WHERE produto='01';
 max
-----
  15
```

Funções de Agregação

MIN(expressão) – retorna o valor mínimo da expressão para os valores de entrada.

Exemplo:

```
SELECT MIN(quantidade) FROM entrada WHERE produto='01';
 min
-----
   4
```

Funções de Agregação

AVG(expressão) – retorna a média aritmética da expressão para os valores de entrada.

Exemplo:

```
SELECT AVG(quantidade) FROM entrada WHERE produto='01';
        avg
--------------------
 7.000000000000000
```

Funções de Agregação

STDDEV(expressão) – retorna o desvio padrão da expressão para os valores de entrada.

Exemplo:

```
SELECT STDDEV(quantidade) FROM entrada WHERE
produto='01';
        stddev
--------------------
 4.5276925690687083
```

Funções de Agregação

VARIANCE(expressão) – retorna a variância da expressão para os valores de entrada.

Exemplo:

```
SELECT VARIANCE(quantidade) FROM entrada WHERE
produto='01';
      variance
---------------------
 20.500000000000000
```

GROUP BY

A cláusula GROUP BY agrupa em uma única linha, todas as linhas com o mesmo valor para uma expressão.

Exemplo:

```
SELECT curso FROM aluno GROUP BY curso;
 curso
-------

 0001
 0002
```

GROUP BY

Usualmente, a cláusula GROUP BY é utilizada com as funções de agregação. Quando se utiliza a cláusula GROUP BY, não é valido a lista de expressões da saída fazer referências a colunas que não são usadas na expressão de agrupamento, exceto em funções de agregação ou se for um campo com dependência funcional dos campos da expressão de agrupamento.

É importante saber que a cláusula GROUP BY é aplicada após a seleção das linhas pela cláusula WHERE, portanto a expressão de seleção da cláusula WHERE não pode fazer referências as expressões de agregação.

Exemplo:

```
SELECT curso, COUNT(*) FROM aluno GROUP BY curso;
 curso | count
-------+-------
       |     1
 0001  |     6
 0002  |     3
```

GROUP BY

Campos com dependência funcional dos campos da expressão de agrupamento são campos que pertencem a tabela onde os campos da expressão de agrupamento, ou parte deles, sejam a chave primária da tabela.

Exemplo:

```
SELECT curso.nome, COUNT(*) FROM curso,aluno WHERE
aluno.curso=curso.codigo GROUP BY curso.codigo;
```

```
            nome              | count
------------------------------+-------
 Engenharia da Computacao     |     3
 Tecnologia da Informacao     |     6
```

GROUP BY

Pode ser usado CASE como expressão de agrupamento.

Exemplo:

```
SELECT CASE WHEN funcao=1 OR funcao=4 THEN 'experiente'
WHEN funcao='3' THEN 'intermediario' WHEN funcao=2 OR
funcao=5 THEN 'iniciante' END AS experiencia, count(*)
FROM funcionario GROUP BY CASE WHEN funcao=1 OR funcao=4
THEN 'experiente' WHEN funcao='3' THEN 'intermediario'
WHEN funcao=2 OR funcao=5 THEN 'iniciante' END;
```

```
  experiencia  | count
---------------+-------
 intermediario |     3
 experiente    |     8
 iniciante     |     5
```

GROUP BY

A expressão de agrupamento pode fazer referências a colunas da lista de saída pela posição da coluna.

Exemplo:

```
SELECT CASE WHEN funcao=1 OR funcao=4 THEN 'experiente'
WHEN funcao='3' THEN 'intermediario' WHEN funcao=2 OR
funcao=5 THEN 'iniciante' END AS experiencia, count(*)
FROM funcionario GROUP BY 1;
```

```
  experiencia  | count
---------------+-------
 intermediario |     3
 experiente    |     8
 iniciante     |     5
```

GROUP BY

A expressão de agrupamento pode fazer referências a colunas da lista de saída pelo alias da coluna.

Exemplo:

```
SELECT CASE WHEN funcao=1 OR funcao=4 THEN 'experiente'
WHEN funcao='3' THEN 'intermediario' WHEN funcao=2 OR
funcao=5 THEN 'iniciante' END AS experiencia, count(*)
FROM funcionario GROUP BY experiencia;
```

```
  experiencia  | count
---------------+-------
 intermediario |     3
 experiente    |     8
 iniciante     |     5
```

HAVING

A cláusula HAVING é aplicada após a cláusula GROUP BY para filtrar as linhas do resultado utilizando as expressões de agregação e as expressões de agrupamento.

Exemplo:

```
SELECT curso, COUNT(*) FROM aluno GROUP BY curso HAVING
count(*)<4;
 curso | count
-------+-------
       |     1
 0002  |     3
SELECT curso, COUNT(*) FROM aluno GROUP BY curso HAVING
count(*)<4 AND curso IS NOT NULL;
 curso | count
-------+-------
 0002  |     3
```

ORDER BY

A cláusula ORDER BY permite determinar a ordem das linhas do resultado.

Sem a determinação de uma ordem específica, não é possível esperar que as linhas venham em alguma ordem, pois não é possível determinar a ordem que as linhas serão selecionadas pelo algoritmo do SGBD.

Exemplo:

```
SELECT matricula, nome, curso FROM aluno ORDER BY nome;
 matricula |      nome        | curso
-----------+------------------+-------
         6 | Alvaro           | 0001
         1 | Ana Lucia        | 0001
         8 | Andrea           | 0002
         9 | Carla            | 0002
         7 | Claudio          | 0002
         4 | Debora           | 0001
        10 | Fernanda         |
         5 | Fernanda         | 0001
         2 | Luis Claudio     | 0001
         3 | Marcelo          | 0001
```

ORDER BY

Quando utilizada mais de uma expressão para a ordenação, a primeira expressão avaliada será a expressão mais a esquerda e assim por diante.

Por default, o valor nulo é classificado em uma posição mais alta do que qualquer outro valor.

Exemplo:

```
SELECT matricula, nome, curso FROM aluno ORDER BY curso,
nome;
 matricula |      nome        | curso
-----------+------------------+-------
         6 | Alvaro           | 0001
         1 | Ana Lucia        | 0001
         4 | Debora           | 0001
         5 | Fernanda         | 0001
         2 | Luis Claudio     | 0001
         3 | Marcelo          | 0001
         8 | Andrea           | 0002
         9 | Carla            | 0002
         7 | Claudio          | 0002
        10 | Fernanda         |
```

ORDER BY

Para alterar a posição de classificação do valor nulo pode ser utilizado NULLS FIRST. A opção NULLS LAST é equivalente ao comportamento default.

Exemplo:

```
SELECT matricula, nome, curso FROM aluno ORDER BY curso
NULLS FIRST, nome;
```

matricula	nome	curso
10	Fernanda	
6	Alvaro	0001
1	Ana Lucia	0001
4	Debora	0001
5	Fernanda	0001
2	Luis Claudio	0001
3	Marcelo	0001
8	Andrea	0002
9	Carla	0002
7	Claudio	0002

ORDER BY

Para ordenação decrescente, utiliza-se DESC em frente a expressão que deve ser avaliada em ordem decrescente. Pode se utilizar ASC para especificar a ordem ascendente, porém não é necessário por ser este o padrão.

Exemplo:

```
SELECT matricula, nome, curso FROM aluno ORDER BY curso
DESC, nome;

SELECT matricula, nome, curso FROM aluno ORDER BY curso,
nome DESC;

SELECT matricula, nome, curso FROM aluno ORDER BY curso
DESC, nome DESC;
```

ORDER BY

```
SELECT matricula, nome, curso FROM aluno ORDER BY curso
DESC, nome;
```

matricula	nome	curso
10	Fernanda	
8	Andrea	0002
9	Carla	0002
7	Claudio	0002
6	Alvaro	0001
1	Ana Lucia	0001
4	Debora	0001
5	Fernanda	0001
2	Luis Claudio	0001
3	Marcelo	0001

ORDER BY

```
SELECT matricula, nome, curso FROM aluno ORDER BY curso,
nome DESC;
```

matricula	nome	curso
3	Marcelo	0001
2	Luis Claudio	0001
5	Fernanda	0001
4	Debora	0001
1	Ana Lucia	0001
6	Alvaro	0001
7	Claudio	0002
9	Carla	0002
8	Andrea	0002
10	Fernanda	

ORDER BY

```
SELECT matricula, nome, curso FROM aluno ORDER BY curso
DESC, nome DESC;
```

matricula	nome	curso
10	Fernanda	
7	Claudio	0002
9	Carla	0002
8	Andrea	0002
3	Marcelo	0001
2	Luis Claudio	0001
5	Fernanda	0001
4	Debora	0001
1	Ana Lucia	0001
6	Alvaro	0001

ORDER BY

Para as colunas com a expressão DESC, o padrão passa a ser NULLS FIRST. Se for necessário que os valores nulos passem para o final da ordenação pode ser utilizado NULLS LAST.

Exemplo:

```
SELECT matricula, nome, curso FROM aluno ORDER BY curso
DESC NULLS LAST, nome;
```

matricula	nome	curso
8	Andrea	0002
9	Carla	0002
7	Claudio	0002
6	Alvaro	0001
1	Ana Lucia	0001
4	Debora	0001
5	Fernanda	0001
2	Luis Claudio	0001
3	Marcelo	0001
10	Fernanda	

ORDER BY

A expressão de ordenação pode fazer referências as colunas da saída pela posição.

Exemplo:

```
SELECT matricula, nome, curso FROM aluno ORDER BY 3, 2;
 matricula |      nome        | curso
-----------+------------------+-------
         6 | Alvaro           | 0001
         1 | Ana Lucia        | 0001
         4 | Debora           | 0001
         5 | Fernanda         | 0001
         2 | Luis Claudio     | 0001
         3 | Marcelo          | 0001
         8 | Andrea           | 0002
         9 | Carla            | 0002
         7 | Claudio          | 0002
        10 | Fernanda         |
```

ORDER BY

Quando utilizado com combinação de consultas, a cláusula ORDER BY deve ser colocada após a última consulta e será aplicada a todo o resultado após a combinação do resultado de cada consulta.

Exemplo:

```
SELECT entrada.produto,produto.descricao FROM entrada,
produto WHERE data>='2010/03/15' AND produto.codigo =
entrada.produto UNION SELECT saida.produto,
produto.descricao FROM saida,produto WHERE data >=
'2010/03/15' AND produto.codigo=saida.produto ORDER BY 2;
 produto | descricao
---------+-----------
 04      | Borracha
 06      | Caderno
 01      | Caneta
 02      | Lapis
```

ORDER BY

Para aplicar a cláusula ORDER BY apenas a uma das consultas da combinação, essa consulta deve ser colocada entre parênteses.

Exemplo:

```
( SELECT entrada.produto,produto.descricao FROM entrada,
produto WHERE data>='2010/03/15' AND produto.codigo =
entrada.produto ORDER BY 1 ) UNION SELECT saida.produto,
produto.descricao FROM saida,produto WHERE data >=
'2010/03/15' AND produto.codigo=saida.produto;
```

```
 produto | descricao
---------+-----------
 02      | Lapis
 01      | Caneta
 04      | Borracha
 06      | Caderno
```

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

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.

Exemplo:

```
SELECT nome, nascimento FROM funcionario ORDER BY nascimento FETCH FIRST 4 ROWS ONLY;
```

```
SELECT nome, nascimento FROM funcionario ORDER BY nascimento FETCH FIRST ROW ONLY;
```

FETCH

```
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
```

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.

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
```

Exercícios

01) Obter uma lista da movimentação dos produtos com a data, código do produto, descrição do produto, tipo do movimento ('E' para entrada e 'S' para saída) e quantidade, ordenado por data e código do produto

02) Listar o total de créditos e total de débitos da conta 01, cada total em uma linha

03) Listar a descrição da fase e nome dos participantes do projeto 01 ordenados por fase, incluir a supervisão

04) Listar o total de horas de supervisão dos projetos

05) Listar a quantidade de automóveis

06) Obter o número de entradas do produto '02'

07) Listar o número de programadores participantes do projeto 02

08) Listar a quantidade de automóveis produzidos no Brasil

09) Obter o número de entradas para cada produto

10) Listar o país e o total de veículos produzidos no pais

Exercícios

11) Obter a soma das entradas para cada produto

12) Listar o país e total de valor pago nos automóveis fabricados no país

13) Listar o nome do fabricante e o total de automóveis de cada fabricante

14) Listar a descrição da plataforma e o total de horas gasto com fases dessa plataforma

15) Listar a descrição do projeto, descrição da plataforma e o total de horas gasto com fases dessa plataforma para cada projeto

16) Listar o nome do fabricante e o total do valor de venda dos automóveis do fabricante

17) Obter a média das quantidades das saídas de cada produto, ordenado pelo código do produto

18) Obter a maior quantidade saída de cada produto, ordenada por produto

Exercícios

19) Obter os produtos e total de saídas para os produtos que tiveram total de saídas maior que 20, ordenado por código do produto

20) Listar o nome das revendas e total do valor de venda para as revendas com total de vendas acima de R$ 50.000,00

21) Listar a descrição dos produtos que tiveram mais de 3 entradas

22) Listar o total de horas de cada fase dos projetos (incluindo supervisão) ordenados pelo total de horas em ordem decrescente

23) Listar a descrição do projeto e o custo total do projeto

24) Obter o código do produto, data e quantidade(s) da(s) ultima(s) saída(s) de cada produto, considerar apenas a data da saida

25) Obter o código, descrição do produto, total de entradas e total de saídas do produtos, ordenado pela descrição dos produtos

Exercícios

26) Obter o código, descrição e o saldo da movimentação dos produtos, ordenados pelo código dos produtos

27) Obter o código, descrição e o saldo da movimentação dos produtos que tem saldo menor ou igual a 15, ordenados pela descrição dos produtos

28) Obter o código, descrição e total de entradas dos 3 produtos com maior total de entradas

29) Obter o código, descrição, data e saldo da movimentação do produto na data ordenado por produto e data

30) Obter o código, descrição e total dos salários das diretorias com total de salários maior que $12000,00

31) Obter o total dos maiores salários de cada diretoria

32) Obter para cada produto o código e saldo da movimentação na primeira quinzena e na segunda quinzena do mês 03/2010.