8.10. Matrizes

O PostgreSQL permite que colunas de uma tabela sejam definidas como matrizes (*arrays*) multidimensionais de comprimento variável. Podem ser criadas matrizes de qualquer tipo de dado, nativo ou definido pelo usuário (Entretanto, ainda não são suportadas matrizes de tipos compostos ou domínios).

8.10.1. Declaração do tipo matriz

Para ilustrar a utilização do tipo matriz, é criada a tabela abaixo:

```
CREATE TABLE sal_emp (
    nome              text,
    pagamento_semanal  integer[],
    agenda             text[][]
);
```

Conforme visto, o tipo de dado matriz é identificado anexando colchetes ([]) ao nome do tipo de dado dos elementos da matriz. O comando acima cria uma tabela chamada sal_emp, contendo uma coluna do tipo *text* (nome), uma matriz unidimensional do tipo *integer* (pagamento_semanal), que representa o salário semanal do empregado, e uma matriz bidimensional do tipo *text* (agenda), que representa a agenda semanal do empregado.

A sintaxe de CREATE TABLE permite especificar o tamanho exato da matriz como, por exemplo:

```
CREATE TABLE jogo_da_velha (
    casa     integer[3][3]
);
```

Entretanto, a implementação atual não obriga que os limites de tamanho da matriz sejam respeitados — o comportamento é o mesmo das matrizes com comprimento não especificado.

Na verdade, a implementação atual também não obriga que o número de dimensões declarado seja respeitado. As matrizes de um determinado tipo de elemento são todas consideradas como sendo do mesmo tipo, não importando o tamanho ou o número de dimensões. Portanto, a declaração do número de dimensões ou tamanhos no comando CREATE TABLE é simplesmente uma documentação, que não afeta o comportamento em tempo de execução.

Pode ser utilizada uma sintaxe alternativa para matrizes unidimensionais, em conformidade com o padrão SQL:1999. A coluna pagamento_semanal pode ser definida como:

```
    pagamento_semanal  integer ARRAY[4],
```

Esta sintaxe requer uma constante inteira para designar o tamanho da matriz. Entretanto, como anteriormente, o PostgreSQL não obriga que os limites de tamanho da matriz sejam respeitados.

8.10.2. Entrada de valor matriz

Para escrever um valor matriz como uma constante literal, os valores dos elementos devem

ser envoltos por chaves ({}) e separados por vírgulas (Quem conhece C pode ver que não é diferente da sintaxe da linguagem C para inicializar estruturas). Podem ser colocadas aspas (") em torno de qualquer valor de elemento, sendo obrigatório caso o elemento contenha vírgulas ou chaves (abaixo são mostrados mais detalhes). Portanto, o formato geral de uma constante matriz é o seguinte:

```
'{ val1 delim val2 delim ... }'
```

onde *delim* é o caractere delimitador para o tipo, conforme registrado na sua entrada em pg_type. Entre todos os tipos de dado padrão fornecidos na distribuição do PostgreSQL, o tipo box usa o ponto-e-vírgula (;) mas todos os demais usam a vírgula (,). Cada *val* é uma constante do tipo do elemento da matriz, ou uma submatriz. Um exemplo de uma constante matriz é

```
'{{1,2,3},{4,5,6},{7,8,9}}'
```

Esta constante é uma matriz bidimensional, 3 por 3, formada por três submatrizes de inteiros.

(Estes tipos de constante matriz são, na verdade, apenas um caso especial do tipo genérico de constantes mostrado na Seção 4.1.2.5. A constante é inicialmente tratada como uma cadeia de caracteres e passada para a rotina de conversão de entrada de matriz. Pode ser necessária uma especificação explícita do tipo).

Agora podemos ver alguns comandos INSERT.

```
INSERT INTO sal_emp
    VALUES ('Bill',
    '{10000, 10000, 10000, 10000}',
    '{{"reunião", "almoço"}, {"reunião"}}');
ERRO:  matrizes multidimensionais devem ter expressões de matriz com dimensões correspondentes
```

Deve ser observado que as matrizes multidimensionais devem possuir tamanhos correspondentes para cada dimensão. Uma combinação errada causa uma mensagem de erro.

```
INSERT INTO sal_emp
    VALUES ('Bill',
    '{10000, 10000, 10000, 10000}',
    '{{"reunião", "almoço"}, {"treinamento", "apresentação"}}');

INSERT INTO sal_emp
    VALUES ('Carol',
    '{20000, 25000, 25000, 25000}',
    '{{"café da manhã", "consultoria"}, {"reunião", "almoço"}}');
```

Uma limitação da implementação atual de matriz, é que os elementos individuais da matriz não podem ser valores nulos SQL. Toda a matriz pode ser definida como nula, mas não pode existir uma matriz com alguns elementos nulos e outros não.

O resultado das duas inserções anteriores se parece com:

```
SELECT * FROM sal_emp;

 nome  |     pagamento_semanal     |                      agenda
-------+---------------------------+-------------------------------------------------
 Bill  | {10000,10000,10000,10000} | {{reunião,almoço},{treinamento,apresentação}}
 Carol | {20000,25000,25000,25000} | {{"café da manhã",consultoria},{reunião,almoço}}
(2 linhas)
```

Também pode ser utilizada a sintaxe de construtor de ARRAY:

```
INSERT INTO sal_emp
    VALUES ('Bill',
    ARRAY[10000, 10000, 10000, 10000],
    ARRAY[['reunião', 'almoço'], ['treinamento', 'apresentação']]);
```

```
INSERT INTO sal_emp
    VALUES ('Carol',
    ARRAY[20000, 25000, 25000, 25000],
    ARRAY[['café da manhã', 'consultoria'], ['reunião', 'almoço']]);
```

Deve ser observado que os elementos da matriz são constantes comuns ou expressões SQL; por exemplo, os literais cadeia de caracteres ficam entre apóstrofos, em vez de aspas como no caso de um literal matriz. A sintaxe do construtor de ARRAY é mostrada com mais detalhes na Seção 4.2.10.

8.10.3. Acesso às matrizes

Agora podemos efetuar algumas consultas na tabela. Primeiro, será mostrado como acessar um único elemento da matriz de cada vez. Esta consulta mostra os nomes dos empregados cujo pagamento mudou na segunda semana:

```
SELECT nome FROM sal_emp WHERE pagamento_semanal[1] <> pagamento_semanal[2];

 nome
-------
 Carol
(1 linha)
```

Os números dos índices da matriz são escritos entre colchetes. Por padrão, o PostgreSQL utiliza a convenção de numeração baseada em um, ou seja, uma matriz de *n* elementos começa por array[1] e termina por array[*n*].

Esta consulta mostra o pagamento da terceira semana de todos os empregados:

```
SELECT pagamento_semanal[3] FROM sal_emp;

 pagamento_semanal
-------------------
             10000
             25000
(2 linhas)
```

Também é possível acessar faixas retangulares arbitrárias da matriz, ou submatrizes. Uma faixa da matriz é especificada escrevendo *limite-inferior*:*limite-superior* para uma ou mais dimensões da matriz. Por exemplo, esta consulta mostra o primeiro item na agenda do Bill para os primeiros dois dias da semana:

```
SELECT agenda[1:2][1:1] FROM sal_emp WHERE nome = 'Bill';

          agenda
---------------------------
 {{reunião},{treinamento}}
(1 linha)
```

Também pode ser escrito

```
SELECT agenda[1:2][1] FROM sal_emp WHERE nome = 'Bill';
```

para obter o mesmo resultado. Uma operação com índices em matriz é sempre considerada como representando uma faixa da matriz, quando qualquer um dos índices estiver escrito na forma *inferior*:*superior*. O limite inferior igual a 1 é assumido para qualquer índice quando for especificado apenas um valor, como neste exemplo:

```
SELECT agenda[1:2][2] FROM sal_emp WHERE nome = 'Bill';

                    agenda
-----------------------------------------------
 {{reunião,almoço},{treinamento,apresentação}}
(1 linha)
```

Podem ser obtidas as dimensões correntes de qualquer valor matriz através da função array_dims:

```
SELECT array_dims(agenda) FROM sal_emp WHERE nome = 'Carol';

 array_dims
------------
 [1:2][1:2]
(1 linha)
```

A função array_dims produz um resultado do tipo *text*, conveniente para as pessoas lerem mas, talvez, nem tão conveniente para os programas. As dimensões também podem ser obtidas através das funções array_upper e array_lower, que retornam os limites superior e inferior da dimensão especificada da matriz, respectivamente.

```
SELECT array_upper(agenda, 1) FROM sal_emp WHERE nome = 'Carol';

 array_upper
-------------
           2
(1 linha)
```

8.10.4. Modificação de matrizes

Um valor matriz pode ser inteiramente substituído utilizando:

```
UPDATE sal_emp SET pagamento_semanal = '{25000,25000,27000,27000}'
    WHERE nome = 'Carol';
```

ou utilizando a sintaxe com a expressão ARRAY:

```
UPDATE sal_emp SET pagamento_semanal = ARRAY[25000,25000,27000,27000]
    WHERE nome = 'Carol';
```

Também pode ser atualizado um único elemento da matriz:

```
UPDATE sal_emp SET pagamento_semanal[4] = 15000
    WHERE nome = 'Bill';
```

ou pode ser atualizada uma faixa da matriz:

```
UPDATE sal_emp SET pagamento_semanal[1:2] = '{27000,27000}'
    WHERE nome = 'Carol';
```

Um valor matriz armazenado pode ser ampliado fazendo atribuição a um elemento adjacente aos já presentes, ou fazendo atribuição a uma faixa que é adjacente ou se sobrepõe aos dados já presentes. Por exemplo, se a matriz minha_matriz possui atualmente quatro elementos, esta matriz terá cinco elementos após uma atualização que faça uma atribuição a minha_matriz[5]. Atualmente, as ampliações desta maneira somente são permitidas para matrizes unidimensionais, não sendo permitidas em matrizes multidimensionais.

A atribuição de faixa de matriz permite a criação de matrizes que não utilizam índices baseados em um. Por exemplo, pode ser feita a atribuição minha_matriz[-2:7] para criar uma matriz onde os valores dos índices variam de -2 a 7.

Também podem ser construídos novos valores matriz utilizando o operador de concatenação ||.

```
SELECT ARRAY[1,2] || ARRAY[3,4];

 ?column?
```

```
-----------
 {1,2,3,4}
(1 linha)

SELECT ARRAY[5,6] || ARRAY[[1,2],[3,4]];

      ?column?
---------------------
 {{5,6},{1,2},{3,4}}
(1 linha)
```

O operador de concatenação permite colocar um único elemento no início ou no fim de uma matriz unidimensional. Aceita, também, duas matrizes *N*-dimensionais, ou uma matriz *N*-dimensional e outra *N+1*-dimensional.

Quando é colocado um único elemento no início de uma matriz unidimensional, o resultado é uma matriz com o limite inferior do índice igual ao limite inferior do índice do operando à direita, menos um. Quando um único elemento é colocado no final de uma matriz unidimensional, o resultado é uma matriz mantendo o limite inferior do operando à esquerda. Por exemplo:

```
SELECT ARRAY[2,3];

 array
-------
 {2,3}
(1 linha)

SELECT array_dims(ARRAY[2,3]);

 array_dims
------------
 [1:2]
(1 linha)

-- Adicionar no início da matriz

SELECT 1 || ARRAY[2,3];

 ?column?
----------
 {1,2,3}
(1 linha)

SELECT array_dims(1 || ARRAY[2,3]);

 array_dims
------------
 [0:2]
(1 linha)

-- Adicionar no final da matriz

SELECT ARRAY[1,2] || 3;

 ?column?
----------
 {1,2,3}
(1 linha)

SELECT array_dims(ARRAY[1,2] || 3);

 array_dims
------------
 [1:3]
(1 linha)
```

Quando duas matrizes com o mesmo número de dimensões são concatenadas, o resultado mantém o limite inferior do índice da dimensão externa do operando à esquerda. O resultado é uma matriz contendo todos os elementos do operando à esquerda seguido por todos os elementos do operando à direita. Por exemplo:

```
-- Concatenação de matrizes unidimensionais
```

```
SELECT ARRAY[1,2] || ARRAY[3,4,5];

  ?column?
-------------
 {1,2,3,4,5}
(1 linha)

SELECT array_dims(ARRAY[1,2] || ARRAY[3,4,5]);

 array_dims
------------
 [1:5]
(1 linha)

-- Concatenação de matrizes bidimensionais

SELECT ARRAY[[1,2],[3,4]] || ARRAY[[5,6],[7,8],[9,0]];

            ?column?
---------------------------------
 {{1,2},{3,4},{5,6},{7,8},{9,0}}
(1 linha)

SELECT array_dims(ARRAY[[1,2],[3,4]] || ARRAY[[5,6],[7,8],[9,0]]);

 array_dims
------------
 [1:5][1:2]
(1 linha)
```

Quando uma matriz *N*-dimensional é colocada no início ou no final de uma matriz *N+1*-dimensional, o resultado é análogo ao caso da matriz elemento acima. Cada submatriz *N*-dimensional se torna essencialmente um elemento da dimensão externa da matriz *N+1*-dimensional. Por exemplo:

```
-- Exemplo de matriz unidimensional concatenada com matriz bidimensional

SELECT ARRAY[1,2] || ARRAY[[3,4],[5,6]];

      ?column?
---------------------
 {{1,2},{3,4},{5,6}}
(1 linha)

SELECT array_dims(ARRAY[1,2] || ARRAY[[3,4],[5,6]]);

 array_dims
------------
 [0:2][1:2]
(1 linha)

-- Exemplo de matriz bidimensional concatenada com matriz tridimensional (N. do T.)

SELECT ARRAY[[-1,-2],[-3,-4]];

       array
-------------------
 {{-1,-2},{-3,-4}}
(1 linha)

SELECT array_dims(ARRAY[[-1,-2],[-3,-4]]);

 array_dims
------------
 [1:2][1:2]
(1 linha)

SELECT ARRAY[[[5,6],[7,8]],[[9,10],[11,12]],[[13,14],[15,16]]];

                       array
-----------------------------------------------------
 {{{5,6},{7,8}},{{9,10},{11,12}},{{13,14},{15,16}}}
(1 linha)

SELECT array_dims(ARRAY[[[5,6],[7,8]],[[9,10],[11,12]],[[13,14],[15,16]]]);

   array_dims
-----------------
 [1:3][1:2][1:2]
```

```
(1 linha)

SELECT ARRAY[[-1,-2],[-3,-4]] ||
       ARRAY[[[5,6],[7,8]],[[9,10],[11,12]],[[13,14],[15,16]]];

                                ?column?
------------------------------------------------------------------------
 {{{-1,-2},{-3,-4}},{{5,6},{7,8}},{{9,10},{11,12}},{{13,14},{15,16}}}
(1 linha)

SELECT array_dims(ARRAY[[-1,-2],[-3,-4]] ||
                  ARRAY[[[5,6],[7,8]],[[9,10],[11,12]],[[13,14],[15,16]]]);

   array_dims
-----------------
 [0:3][1:2][1:2]
(1 linha)

SELECT ARRAY[[[5,6],[7,8]],[[9,10],[11,12]],[[13,14],[15,16]]] ||
       ARRAY[[-1,-2],[-3,-4]];

                                ?column?
------------------------------------------------------------------------
 {{{5,6},{7,8}},{{9,10},{11,12}},{{13,14},{15,16}},{{-1,-2},{-3,-4}}}
(1 linha)

SELECT array_dims(ARRAY[[[5,6],[7,8]],[[9,10],[11,12]],[[13,14],[15,16]]] ||
                  ARRAY[[-1,-2],[-3,-4]]);

   array_dims
-----------------
 [1:4][1:2][1:2]
(1 linha)
```

Uma matriz também pode ser construída utilizando as funções array_prepend, array_append e array_cat. As duas primeiras suportam apenas matrizes unidimensionais, mas array_cat suporta matrizes multidimensionais. Deve ser observado que é preferível utilizar o operador de concatenação mostrado acima, em vez de usar diretamente estas funções. Na verdade, estas funções têm seu uso principal na implementação do operador de concatenação. Entretanto, podem ser úteis na criação de agregações definidas pelo usuário. Alguns exemplos:

```
SELECT array_prepend(1, ARRAY[2,3]);

 array_prepend
---------------
 {1,2,3}
(1 linha)

SELECT array_append(ARRAY[1,2], 3);

 array_append
--------------
 {1,2,3}
(1 linha)

SELECT array_cat(ARRAY[1,2], ARRAY[3,4]);

 array_cat
-----------
 {1,2,3,4}
(1 linha)

SELECT array_cat(ARRAY[[1,2],[3,4]], ARRAY[5,6]);

      array_cat
---------------------
 {{1,2},{3,4},{5,6}}
(1 linha)

SELECT array_cat(ARRAY[5,6], ARRAY[[1,2],[3,4]]);

      array_cat
---------------------
 {{5,6},{1,2},{3,4}}
```

8.10.5. Procura em matrizes

Para procurar um valor em uma matriz deve ser verificado cada valor da matriz. Pode ser feito à mão, se for conhecido o tamanho da matriz: Por exemplo:

```
SELECT * FROM sal_emp WHERE pagamento_semanal[1] = 10000 OR
                            pagamento_semanal[2] = 10000 OR
                            pagamento_semanal[3] = 10000 OR
                            pagamento_semanal[4] = 10000;
```

Entretanto, em pouco tempo se torna entediante para matrizes grandes, e não servirá se a matriz for de tamanho desconhecido. Um método alternativo está descrito na Seção 9.17. A consulta acima pode ser substituída por:

```
SELECT * FROM sal_emp WHERE 10000 = ANY (pagamento_semanal);
```

Além disso, podem ser encontradas as linhas onde a matriz possui todos os valores iguais a 10000 com:

```
SELECT * FROM sal_emp WHERE 10000 = ALL (pagamento_semanal);
```

> **Dica:** Matrizes não são conjuntos; a procura por determinados elementos da matriz pode ser um sinal de um banco de dados mal projetado. Considere a utilização de uma outra tabela, com uma linha para cada item que seria um elemento da matriz. Assim é mais fácil procurar e, provavelmente, vai se comportar melhor com um número grande de elementos.

8.10.6. Sintaxe de entrada e de saída das matrizes

A representação textual externa de um valor matriz é formada por itens que são interpretados de acordo com as regras de conversão de I/O para o tipo do elemento da matriz, mais os adornos que indicam a estrutura da matriz. Estes adornos consistem em chaves ({ e }) em torno do valor matriz, mais os caracteres delimitadores entre os itens adjacentes. O caractere delimitador geralmente é a vírgula (,), mas pode ser outro: é determinado pela definição de typdelim para o tipo do elemento da matriz (Entre os tipos de dado padrão fornecidos na distribuição do PostgreSQL o tipo box utiliza o ponto-e-vírgula (;), mas todos os outros utilizam a vírgula). Em uma matriz multidimensional cada dimensão (linha, plano, cubo, etc.) recebe seu nível próprio de chaves, e os delimitadores devem ser escritos entre entidades de chaves adjacentes do mesmo nível.

A rotina de saída de matriz coloca aspas em torno dos valores dos elementos caso estes sejam cadeias de caracteres vazias, ou se contenham chaves, caracteres delimitadores, aspas, contrabarras, ou espaços em branco. Aspas e contrabarras incorporadas aos valores dos elementos recebem o escape de contrabarra. No caso dos tipos de dado numéricos é seguro assumir que as aspas nunca vão estar presentes, mas para tipos de dado textuais deve-se estar preparado para lidar tanto com a presença quanto com a ausências das aspas (Esta é uma mudança de comportamento com relação às versões do PostgreSQL anteriores a 7.2).

Por padrão, o limite inferior do valor do índice de cada dimensão da matriz é definido como um. Se alguma das dimensões da matriz tiver um limite inferior diferente de um, um adorno adicional indicando as verdadeiras dimensões da matriz precede o adorno da estrutura da matriz. Este adorno é composto por colchetes ([]) em torno de cada limite inferior e superior da dimensão da matriz, com o caractere delimitador dois-pontos (:) entre estes. O adorno de dimensão da matriz é seguido pelo sinal de igual (=). Por exemplo:

```
SELECT 1 || ARRAY[2,3] AS array;

     array
---------------
 [0:2]={1,2,3}
(1 linha)

SELECT ARRAY[1,2] || ARRAY[[3,4]] AS array;

          array
```

```
--------------------------
 [0:1][1:2]={{1,2},{3,4}}
(1 linha)
```

Esta sintaxe também pode ser utilizada para especificar índices de matriz não padrão em um literal matriz. Por exemplo:

```
SELECT f1[1][-2][3] AS e1, f1[1][-1][5] AS e2
 FROM (SELECT '[1:1][-2:-1][3:5]={{{1,2,3},{4,5,6}}}'::int[] AS f1) AS ss;

 e1 | e2
----+----
  1 |  6
(1 linha)
```

Conforme mostrado anteriormente, ao escrever um valor matriz pode-se colocar aspas em torno de qualquer elemento individual da matriz. Isto *deve* ser feito se o valor do elemento puder, de alguma forma, confundir o analisador de valor matriz. Por exemplo, os elementos contendo chaves, vírgulas (ou qualquer que seja o caractere delimitador), aspas, contrabarras ou espaços em branco na frente ou atrás devem estar entre aspas. Para colocar aspas ou contrabarras no valor entre aspas do elemento da matriz, estes devem ser precedidos por uma contrabarra. Como alternativa, pode ser utilizado o escape de contrabarra para proteger qualquer caractere de dado que seria de outra forma considerado como sintaxe da matriz.

Podem ser escritos espaços em branco antes do abre chaves ou após o fecha chaves. Também podem ser escritos espaços em branco antes ou depois de qualquer item individual cadeia de caracteres. Em todos estes casos os espaços em branco são ignorado. Entretanto, espaços em branco dentro de elementos entre aspas, ou envoltos nos dois lados por caracteres de um elemento que não são espaços em branco, não são ignorados.

> **Nota:** Lembre-se que o que se escreve em um comando SQL é interpretado primeiro como um literal cadeia de caracteres e, depois, como uma matriz. Isto duplica o número de contrabarras necessárias. Por exemplo, para inserir um valor matriz do tipo *text* contendo uma contrabarra e uma aspa, deve ser escrito
>
> ```
> INSERT ... VALUES ('{"\\\\","\\""}');
> ```
>
> O processador de literais cadeias de caracteres remove um nível de contrabarras, portanto o que chega para o analisador de valor matriz se parece com {"\\","\""}. Por sua vez, as cadeias de caracteres introduzidas na rotina de entrada do tipo de dado *text* se tornam \ e ", respectivamente (Se estivéssemos trabalhando com um tipo de dado cuja rotina de entrada também tratasse as contrabarras de forma especial como, por exemplo, *bytea*, seriam necessárias oito contrabarras no comando para obter uma contrabarra armazenada no elemento da matriz). Pode ser utilizada a delimitação por cifrão (dollar quoting) (consulte a Seção 4.1.2.2) para evitar a necessidade de duplicar as contrabarras.
>
> **Dica:** Ao se escrever valores matrizes nos comandos SQL, geralmente é mais fácil trabalhar com a sintaxe do construtor de ARRAY (consulte a Seção 4.2.10) do que com a sintaxe do literal cadeia de caracteres. Em ARRAY, os valores dos elementos individuais são escritos da mesma maneira como seriam escritos caso não fossem membros de uma matriz.

Anterior
Tipos para cadeias de bits

Principal
Acima

Próxima
Tipos compostos