A FORÇA DO SISTEMA FECOMÉRCIO AO SEU LADO.

Eixo Tecnológico: GESTÃO
Aula 03
Curso: Superior de Tecnologia em Análise e Desenvolvimento de Sistemas
Unidade Curricular / Unidade de Estudo: BANCO DE DADOS I - Manhã
Docente: Fábio Giulian Marques
Módulo/Semestre: 3º SEMESTRE

Tipos numéricos PostgreSQL

Os tipos numéricos consistem em inteiros de dois, quatro e oito bytes, números de ponto flutuante de quatro e oito bytes, e decimais de precisão selecionável. A Tabela 8-2 lista os tipos disponíveis.

Tabela 8-2. Tipos numéricos

Nome	Tamanho de armazenamento	Descrição	Faixa de valores
smallint	2 bytes	inteiro com faixa pequena	-32768 a +32767
integer	4 bytes	escolha usual para inteiro	-2147483648 a +2147483647
bigint	8 bytes	inteiro com faixa larga	-9223372036854775808 a 9223372036854775807
decimal	variável	precisão especificada pelo usuário, exato	sem limite
numeric	variável	precisão especificada pelo usuário, exato	sem limite
real	4 bytes	precisão variável, inexato	precisão de 6 dígitos decimais
double precision	8 bytes	precisão variável, inexato	precisão de 15 dígitos decimais
serial	4 bytes	inteiro com auto-incremento	1 a 2147483647
bigserial	8 bytes	inteiro grande com auto-incremento	1 a 9223372036854775807

A sintaxe das constantes para os tipos numéricos é descrita na Seção 4.1.2. Os tipos numéricos possuem um conjunto completo de operadores aritméticos e funções correspondentes. Consulte o Capítulo 9 para obter informações adicionais. As próximas seções descrevem os tipos em detalhe.

8.1.1. Tipos inteiros

Os tipos smallint, integer e bigint armazenam números inteiros, ou seja, números sem a parte fracionária, com faixas diferentes. A tentativa de armazenar um valor fora da faixa permitida resulta em erro.

O tipo integer é a escolha usual, porque oferece o melhor equilíbrio entre faixa de valores, tamanho de armazenamento e desempenho. Geralmente o tipo smallint só é utilizado quando o espaço em disco está muito escasso. O tipo bigint somente deve ser usado quando a faixa de valores de integer não for suficiente, porque este último é bem mais rápido.

O tipo bigint pode não funcionar de modo correto em todas as plataformas, porque depende de suporte no compilador para inteiros de oito bytes. Nas máquinas sem este suporte, o bigint age do mesmo modo que o integer (mas ainda ocupa oito bytes de armazenamento). Entretanto, não é de nosso conhecimento nenhuma plataforma razoável onde este caso se aplique.

O padrão SQL somente especifica os tipos inteiros integer (ou int) e smallint. O tipo bigint, e os nomes de tipo int2, int4 e int8 são extensões, também compartilhadas por vários outros sistemas de banco de dados SQL.

8.1.2. Números com precisão arbitrária

O tipo numeric pode armazenar números com precisão de até 1.000 dígitos e realizar cálculos exatos. É recomendado, especialmente, para armazenar quantias monetárias e outras quantidades onde se requeira exatidão. Entretanto, a aritmética em valores do tipo numeric é muito lenta se comparada com os tipos inteiros, ou com os tipos de ponto flutuante descritos na próxima seção.

São utilizados os seguintes termos: A *escala* do tipo numeric é o número de dígitos decimais da parte fracionária, à direita do ponto decimal. A *precisão* do tipo numeric é o número total de dígitos significativos de todo o número, ou seja, o número de dígitos nos dois lados do ponto decimal. Portanto, o número 23.5141 [5] possui precisão igual a 6 e escala igual a 4. Os inteiros podem ser considerados como tendo escala igual a zero.

Tanto a precisão máxima quanto a escala de uma coluna do tipo numeric podem ser configuradas. Para declarar uma coluna do tipo numeric é utilizada a sintaxe:

```
NUMERIC(precisão, escala)
```

A precisão deve ser um número positivo, enquanto a escala pode ser zero ou positiva. Como forma alternativa,

```
NUMERIC(precisão)
```

define a escala como sendo igual a 0. Especificando-se

```
NUMERIC
```

sem qualquer precisão ou escala é criada uma coluna onde podem ser armazenados valores numéricos com qualquer precisão ou escala, até a precisão limite da implementação. Uma coluna deste tipo não converte os valores de entrada para nenhuma escala em particular, enquanto as colunas do tipo numeric com escala declarada convertem os valores da entrada para esta escala (O padrão SQL requer a escala padrão igual a 0, ou seja, uma conversão para a precisão inteira. Isto foi considerado sem utilidade. Havendo preocupação com a portabilidade, a precisão e a escala devem ser sempre especificadas explicitamente).

Se a escala do valor a ser armazenado for maior que a escala declarada para a coluna, o sistema arredonda o valor para o número de dígitos fracionários especificado. Depois, se o número de dígitos à esquerda do ponto decimal exceder a precisão declarada menos a escala declarada, é gerado um erro.

Exemplo 8-1. Arredondamento em tipo *numeric*

Abaixo estão mostrados exemplos de inserção de dados em um campo do tipo numeric. No terceiro exemplo o arredondamento faz com que a precisão do campo seja excedida. [6]

A execução deste exemplo no SQL Server 2000 e no Oracle 10g produziu o mesmo resultado, mas o DB2 8.1 em vez de arredondar trunca as casas decimais e, por isso, a precisão não é excedida.

```
BEGIN;
CREATE TEMPORARY TABLE t ( c NUMERIC(6,3)) ON COMMIT DROP;
INSERT INTO t VALUES (998.9991);
INSERT INTO t VALUES (998.9999);
SELECT * FROM t;

    c
---------
 998.999
 999.000
(2 linhas)

INSERT INTO t VALUES (999.9999);

ERRO:  estouro de campo numérico
DETALHE:  O valor absoluto é maior ou igual a 10^3 para campo com
precisão 6, escala 3.

COMMIT;
ROLLBACK
```

Os valores numéricos são armazenados fisicamente sem zeros adicionais no início ou no final. Portanto, a precisão e a escala declaradas para uma coluna são as alocações máximas, e não fixas (Sob este aspecto o tipo numeric é mais semelhante ao tipo varchar(n) do que ao tipo char(n)).

Além dos valores numéricos ordinários o tipo numeric aceita o valor especial NaN, que significa "não-é-um-número" (*not-a-number*). Toda operação envolvendo NaN produz outro NaN. Para escrever este valor como uma constante em um comando SQL deve-se colocá-lo entre apóstrofos como, por exemplo, UPDATE tabela SET x = 'NaN'. Na entrada, a cadeia de caracteres NaN é reconhecida sem que haja distinção entre letras maiúsculas e minúsculas.

Os tipos decimal e numeric são equivalentes. Os dois tipos fazem parte do padrão SQL.

8.1.3. Tipos de ponto flutuante

Os tipos de dado real e double precision são tipos numéricos não exatos de precisão variável. Na prática, estes tipos são geralmente implementações do "Padrão IEEE 754 para Aritmética Binária de Ponto Flutuante" (de precisão simples e dupla, respectivamente), conforme suportado pelo processador, sistema operacional e compilador utilizados.

Não exato significa que alguns valores não podem ser convertidos exatamente para o formato interno, sendo armazenados como aproximações. Portanto, ao se armazenar e posteriormente imprimir um valor podem ocorrer pequenas discrepâncias. A gerência destes erros, e como se propagam através dos cálculos, é assunto de um ramo da matemática e da ciência da computação que não será exposto aqui, exceto os seguintes pontos:

- Se for necessário armazenamento e cálculos exatos (como em quantias monetárias), em vez de tipos de ponto flutuante deve ser utilizado o tipo numeric.
- Se for desejado efetuar cálculos complicados usando tipos de ponto flutuante para algo importante, especialmente dependendo de certos comportamentos em situações limites (infinito ou muito próximo de zero), a implementação deve ser avaliada cuidadosamente.
- A comparação de igualdade de dois valores de ponto flutuante pode funcionar conforme o esperado, ou não.

Na maioria das plataformas o tipo real possui uma faixa de pelo menos 1E-37 a 1E+37, com precisão de pelo menos 6 dígitos decimais. O tipo double precision normalmente possui uma faixa em torno de 1E-307 a 1E+308 com precisão de pelo menos 15 dígitos. Os valores muito pequenos ou muito grandes causam erro. O arredondamento pode acontecer se a precisão do número entrado for muito grande. Os números muito

próximos de zero, que não podem ser representados de forma distinta de zero, causam erro de *underflow*.

Além dos valores numéricos ordinários, os tipos de ponto flutuante possuem diversos valores especiais:

Infinity
-Infinity
NaN

Estes valores representam os valores especiais do padrão IEEE 754 "infinito", "infinito negativo" e "não-é-um-número", respectivamente (Nas máquinas cuja aritmética de ponto flutuante não segue o padrão IEEE 754, estes valores provavelmente não vão funcionar da forma esperada). Ao se escrever estes valores como constantes em um comando SQL deve-se colocá-los entre apóstrofos como, por exemplo, UPDATE tabela SET x = 'Infinity'. Na entrada, estas cadeias de caracteres são reconhecidas sem que haja distinção entre letras maiúsculas e minúsculas

O PostgreSQL também suporta a notação do padrão SQL float e float(p) para especificar tipos numéricos inexatos. Neste caso, p especifica a precisão mínima aceitável em dígitos binários. O PostgreSQL aceita de float(1) a float(24) como selecionando o tipo real, enquanto float(25) a float(53) selecionam double precision. Os valores de p fora da faixa permitida ocasionam erro. float sem precisão especificada é assumido como significando double precision.

Nota: Antes do PostgreSQL 7.4 a precisão em float(p) era considerada como significando a quantidade de dígitos decimais, mas foi corrigida para corresponder ao padrão SQL, que especifica que a precisão é medida em dígitos binários. A premissa que real e double precision possuem exatamente 24 e 53 bits na mantissa, respectivamente, está correta para implementações em acordo com o padrão IEEE para números de ponto flutuante. Nas plataformas não-IEEE pode ser um pouco diferente, mas para simplificar as mesmas faixas de p são utilizadas em todas as plataformas.

8.1.4. Tipos seriais

Os tipos de dado serial e bigserial não são tipos verdadeiros, mas meramente uma notação conveniente para definir colunas identificadoras únicas (semelhante à propriedade AUTO_INCREMENTO existente em alguns outros bancos de dados). Na implementação corrente especificar

```
CREATE TABLE nome_da_tabela (
    nome_da_coluna SERIAL
);
```

equivale a especificar:

```
CREATE SEQUENCE nome_da_tabela_nome_da_coluna_seq;
CREATE TABLE nome_da_tabela (
    nome_da_coluna integer DEFAULT
nextval('nome_da_tabela_nome_da_coluna_seq') NOT NULL
);
```

Conforme visto, foi criada uma coluna do tipo inteiro e feito o valor padrão ser atribuído a partir de um gerador de seqüência. A restrição NOT NULL é aplicada para garantir que não pode ser inserido o valor nulo explicitamente. Na maior parte das vezes, deve ser colocada uma restrição UNIQUE ou PRIMARY KEY para não permitir a inserção de valores duplicados por acidente, mas isto não é automático.

Nota: Antes do PostgreSQL 7.3 serial implicava UNIQUE, mas isto não é mais automático. Se for desejado que a coluna serial esteja em uma restrição de unicidade ou de chave primária isto deve ser especificado, da mesma forma como em qualquer outro tipo de dado.

Para inserir o próximo valor da seqüência em uma coluna do tipo serial deve ser especificada a atribuição do valor padrão à coluna serial, o que pode ser feito omitindo a coluna na lista de colunas no comando INSERT, ou através da utilização da palavra chave DEFAULT.

Os nomes de tipo serial e serial4 são equivalentes: ambos criam colunas do tipo integer. Os nomes de tipo bigserial e serial8 funcionam da mesma maneira, exceto por criarem uma coluna bigint. Deve ser utilizado bigserial se forem esperados mais de 2^{31} identificadores durante a existência da tabela.

A seqüência criada para a coluna do tipo serial é removida automaticamente quando a coluna que a definiu é removida, e não pode ser removida de outra forma (Isto não era verdade nas versões do PostgreSQL anteriores a 7.3. Deve ser observado que este vínculo de remoção automática não ocorre em uma seqüência criada pela restauração da cópia de segurança de um banco de dados pré-7.3; a cópia de segurança não contém as informações necessárias para estabelecer o vínculo de dependência). Além disso, a dependência entre a seqüência e a coluna é feita apenas para a própria coluna serial; se qualquer outra coluna fizer referência à seqüência (talvez chamando manualmente a função nextval()), haverá rompimento se a seqüência for removida. Esta forma de utilizar as seqüências das colunas serial é considerada um estilo ruim. Se for desejado suprir várias colunas a partir do mesmo gerador de seqüência, a seqüência deve ser criada como um objeto independente.

Exemplo 8-2. Alteração da seqüência da coluna serial

A seqüência criada para a coluna do tipo serial pode ter seus parâmetros alterados através do comando ALTER SEQUENCE, da mesma forma que qualquer outra seqüência criada através do comando CREATE SEQUENCE. Este exemplo mostra como proceder para fazer com que o valor inicial da coluna do tipo serial seja igual a 1000. [7]

```
=> CREATE TABLE t ( c1 SERIAL, c2 TEXT);

NOTA:  CREATE TABLE irá criar a seqüência implícita "t_c1_seq" para a
coluna "serial" "t.c1"
CREATE TABLE

=> \ds

             Lista de relações
 Esquema |   Nome    |   Tipo    |   Dono
---------+----------+-----------+----------
 public  | t_c1_seq | seqüência | postgres
(1 linha)

=> ALTER SEQUENCE t_c1_seq RESTART WITH 1000;

=> INSERT INTO t VALUES (DEFAULT, 'Primeira linha');

=> SELECT * FROM t;

  c1  |       c2
------+----------------
 1000 | Primeira linha
(1 linha)
```

Notas

[1] O SQL suporta três modalidades de tipos de dado: *tipos de dado pré-definidos*, *tipos construídos* e *tipos definidos pelo usuário*. Os tipos pré-definidos são algumas vezes chamados de "tipos nativos", mas não neste Padrão Internacional. Os tipos definidos pelo usuário podem ser definidos por um padrão, por uma implementação, ou por um aplicativo.

O tipo construído é especificado utilizando um dos construtores de tipo de dado do SQL: ARRAY, MULTISET, REF e ROW. O tipo construído é um tipo matriz, um tipo multi-conjunto, um tipo referência ou um tipo linha, se for especificado por ARRAY, MULTISET, REF e ROW, respectivamente. Os tipos matriz e multi-conjunto são conhecidos genericamente como tipos coleção.

(ISO-ANSI Working Draft) Foundation (SQL/Foundation), August 2003, ISO/IEC JTC 1/SC 32, 25-jul-2003, ISO/IEC 9075-2:2003 (E) (N. do T.)

[2] Todo tipo de dado inclui um valor especial, chamado de *valor nulo*, algumas vezes

denotado pela palavra chave NULL. Este valor difere dos demais valores com relação aos seguintes aspectos.

— Uma vez que o valor nulo está presente em todo tipo de dado, o tipo de dado do valor nulo implicado pela palavra chave NULL não pode ser inferido; portanto NULL pode ser utilizado para denotar o valor nulo apenas em certos contextos, e não em todos os lugares onde um literal é permitido.

— Embora o valor nulo não seja igual a qualquer outro valor, nem seja não igual a qualquer outro valor - é *desconhecido* se é igual ou não a qualquer outro valor - em alguns contextos, valores nulos múltiplos são tratados juntos; por exemplo, a <cláusula group by> trata todos os valores nulos juntos.

(ISO-ANSI Working Draft) Framework (SQL/Framework), August 2003, ISO/IEC JTC 1/SC 32, 25-jul-2003, ISO/IEC 9075-1:2003 (E) (N. do T.)

[3] *literal* — um valor usado exatamente da forma como é visto. Por exemplo, o número 25 e a cadeia de caracteres "Alô" são ambos literais. Os literais podem ser utilizados em expressões, e podem ser atribuídos literais para constantes ou variáveis no Visual Basic. Microsoft Glossary for Business Users (N. do T.)

[4] *deprecated* — Dito de um programa ou funcionalidade que é considerada em obsolescência e no processo de ter sua utilização gradualmente interrompida, geralmente em favor de uma determinada substituição. As funcionalidades em obsolescência podem, infelizmente, demorar muitos anos para desaparecer. The Jargon File (N. do T.)

[5] Padrão americano: o ponto, e não a vírgula, separando a parte fracionária. (N. do T.)

[6] Exemplo escrito pelo tradutor, não fazendo parte do manual original.

[7] Exemplo escrito pelo tradutor, não fazendo parte do manual original.

Anterior: LIMIT e OFFSET | Principal: Acima | Próxima: Tipos monetários

8.2. Tipos monetários

Nota: O tipo money está em obsolescência. Em seu lugar deve ser utilizado o tipo numeric ou decimal, em combinação com a função to_char.

O tipo money armazena a quantia monetária com uma precisão fracionária fixa; consulte a Tabela 8-3. A entrada é aceita em vários formatos, incluindo literais inteiros e de ponto flutuante, e também o formato monetário "típico", como '$1,000.00'. A saída geralmente é neste último formato, mas depende do idioma).

Tabela 8-3. Tipos monetários

Nome	Tamanho de Armazenamento	Descrição	Faixa
money	4 bytes	quantia monetária	-21474836.48 a +21474836.47

8.3. Tipos para cadeias de caracteres

A Tabela 8-4 mostra os tipos de propósito geral para cadeias de caracteres disponíveis no PostgreSQL.

Tabela 8-4. Tipos para cadeias de caracteres

Nome	Descrição
character varying(n), varchar(n)	comprimento variável com limite
character(n), char(n)	comprimento fixo, completado com brancos
text	comprimento variável não limitado

O SQL define dois tipos primários para caracteres: character varying(n) e character(n), onde n é um número inteiro positivo. Estes dois tipos podem armazenar cadeias de caracteres com comprimento de até n caracteres. A tentativa de armazenar uma cadeia de caracteres mais longa em uma coluna de um destes tipos resulta em erro, a não ser que os caracteres excedentes sejam todos espaços; neste caso a cadeia de caracteres será truncada em seu comprimento máximo (Esta exceção um tanto bizarra é requerida pelo padrão SQL). Se a cadeia de caracteres a ser armazenada for mais curta que o comprimento declarado, os valores do tipo character são completados com espaços; os valores do tipo character varying simplesmente armazenam uma cadeia de caracteres mais curta.

Se um valor for convertido explicitamente (*cast*) para character varying(n), ou para character(n), o excesso de comprimento será truncado para n caracteres sem gerar erro (isto também é requerido pelo padrão SQL).

Nota: Antes do PostgreSQL 7.2 as cadeias de caracteres muito longas eram sempre truncadas sem gerar erro, tanto no contexto de conversão explícita quanto no de implícita.

As notações varchar(n) e char(n) são sinônimos para character varying(n) e character(n), respectivamente. O uso de character sem especificação de comprimento equivale a

character(1); se for utilizado character varying sem especificador de comprimento, este tipo aceita cadeias de caracteres de qualquer tamanho. Este último é uma extensão do PostgreSQL.

Além desses o PostgreSQL disponibiliza o tipo text, que armazena cadeias de caracteres de qualquer comprimento. Embora o tipo text não esteja no padrão SQL, vários outros sistemas gerenciadores de banco de dados SQL também o possuem.

Os valores do tipo character são preenchidos fisicamente com espaços até o comprimento n especificado, sendo armazenados e mostrados desta forma. Entretanto, os espaços de preenchimento são tratados como não sendo significativos semanticamente. Os espaços de preenchimento são desconsiderados ao se comparar dois valores do tipo character, e são removidos ao converter um valor do tipo character para um dos outros tipos para cadeia de caracteres. Deve ser observado que os espaços no final *são* significativos semanticamente nos valores dos tipos character varying e text.

São necessários para armazenar dados destes tipos 4 bytes mais a própria cadeia de caracteres e, no caso do tipo character, mais os espaços para completar o tamanho. As cadeias de caracteres longas são comprimidas automaticamente pelo sistema e, portanto, o espaço físico necessário em disco pode ser menor. Os valores longos também são armazenados em tabelas secundárias, para não interferirem com o acesso rápido aos valores mais curtos da coluna. De qualquer forma, a cadeia de caracteres mais longa que pode ser armazenada é em torno de 1 GB (O valor máximo permitido para n na declaração do tipo de dado é menor que isto. Não seria muito útil mudar, porque de todo jeito nas codificações de caractere multibyte o número de caracteres e de bytes podem ser bem diferentes. Se for desejado armazenar cadeias de caracteres longas, sem um limite superior especificado, deve ser utilizado text ou character varying sem a especificação de comprimento, em vez de especificar um limite de comprimento arbitrário).

Dica: Não existe diferença de desempenho entre estes três tipos, a não ser pelo aumento do tamanho do armazenamento quando é utilizado o tipo completado com brancos. Enquanto o tipo character(n) possui vantagens no desempenho em alguns outros sistemas gerenciadores de banco de dados, não possui estas vantagens no PostgreSQL. Na maioria das situações deve ser utilizado text ou character varying em vez deste tipo.

Consulte a Seção 4.1.2.1 para obter informações sobre a sintaxe dos literais cadeias de caracteres, e o Capítulo 9 para obter informações sobre os operadores e funções. O conjunto de caracteres do banco de dados determina o conjunto de caracteres utilizado para armazenar valores textuais; para obter mais informações sobre o suporte a conjunto de caracteres consulte a Seção 20.2.

Existem dois outros tipos para cadeias de caracteres de comprimento fixo no PostgreSQL, mostrados na Tabela 8-5. O tipo name existe *apenas* para armazenamento de identificadores nos catálogos internos do sistema, não tendo por finalidade ser usado

pelos usuários comuns. Seu comprimento é definido atualmente como 64 bytes (63 caracteres utilizáveis mais o terminador) mas deve ser referenciado utilizando a constante NAMEDATALEN. O comprimento é definido quando é feita a compilação (sendo, portanto, ajustável para usos especiais); o padrão para comprimento máximo poderá mudar em uma versão futura. O tipo "char" (observe as aspas) é diferente de char(1), porque utiliza apenas um byte para armazenamento. É utilizado internamente nos catálogos do sistema como o tipo de enumeração do homem pobre (*poor-man's enumeration type*).

Tabela 8-5. Tipos especiais para caracteres

Nome	Tamanho de Armazenamento	Descrição
"char"	1 byte	tipo interno de um único caractere
name	64 bytes	tipo interno para nomes de objeto

Exemplo 8-3. Utilização dos tipos para cadeias de caracteres

```
=> CREATE TABLE teste1 (a character(4));
=> INSERT INTO teste1 VALUES ('ok');
=> SELECT a, char_length(a) FROM teste1; -- (1)


  a   | char_length
------+-------------
 ok   |           4

=> CREATE TABLE teste2 (b VARCHAR(5));
=> INSERT INTO teste2 VALUES ('ok');
=> INSERT INTO teste2 VALUES ('bom       '); -- (2)


=> INSERT INTO teste2 VALUES ('muito longo');
ERRO:  valor muito longo para o tipo character varying(5)
=> INSERT INTO teste2 VALUES (CAST('muito longo' AS VARCHAR(5))); --
truncamento explícito
=> SELECT b, char_length(b) FROM teste2;

   b    | char_length
-------+-------------
 ok    |           2
 bom   |           5
 muito |           5
```

(1)
A função char_length é mostrada na Seção 9.4.

(2)
O DB2 8.1 atua da mesma maneira que o PostgreSQL 8.0.0, truncando os espaços à direita que excedem o tamanho do campo, o SQL Server 2005

> também, mas como a função len exclui os espaços à direita o comprimento mostrado fica sendo igual a 3, enquanto o Oracle 10g não trunca os espaços à direita e gera mensagem de erro informando que o valor é muito longo, como no comando seguinte. (N. do T.)

Exemplo 8-4. Comparação de cadeias de caracteres com espaço à direita

Nestes exemplos faz-se a comparação de uma cadeia de caracteres com espaço à direita com outra cadeia de caracteres idêntica sem espaço à direita. Na tabela t1 é feita a comparação entre dois tipos char, na tabela t2 é feita a comparação entre dois tipos varchar, e na tabela t3 é feita a comparação entre os tipos char e varchar. O mesmo script foi executado no PostgreSQL, no Oracle, no SQL Server e no DB2. Abaixo está mostrado o script executado: [1] [2]

```
CREATE TABLE t1 ( c1 CHAR(10), c2 CHAR(10));
INSERT INTO t1 VALUES ('X', 'X ');
SELECT '''' || c1 || '''' AS c1,
       '''' || c2 || '''' AS c2,
       CASE WHEN (c1=c2) THEN 'igual' ELSE 'diferente' END AS
comparação
FROM t1;

CREATE TABLE t2 ( c1 VARCHAR(10), c2 VARCHAR(10));
INSERT INTO t2 VALUES ('X', 'X ');
SELECT '''' || c1 || '''' AS c1,
       '''' || c2 || '''' AS c2,
       CASE WHEN (c1=c2) THEN 'igual' ELSE 'diferente' END AS
comparação
FROM t2;

CREATE TABLE t3 ( c1 CHAR(10), c2 VARCHAR(10));
INSERT INTO t3 VALUES ('X', 'X ');
INSERT INTO t3 VALUES ('X ', 'X');
SELECT '''' || c1 || '''' AS c1,
       '''' || c2 || '''' AS c2,
       CASE WHEN (c1=c2) THEN 'igual' ELSE 'diferente' END AS
comparação
FROM t3;
```

A seguir estão mostrados os resultados obtidos:

PostgreSQL 8.0.0:

```
 c1  | c2  | comparação
-----+-----+------------
 'X' | 'X' | igual

 c1  |  c2  | comparação
-----+------+------------
 'X' | 'X ' | diferente
```

```
 c1  |  c2  | comparação
-----+------+------------
 'X' | 'X ' | igual
 'X' | 'X'  | igual
```

SQL Server 2000:

```
c1           c2           comparação
------------ ------------ ----------
'X          ' 'X          ' igual

c1           c2           comparação
------------ ------------ ----------
'X'          'X '         igual

c1           c2           comparação
------------ ------------ ----------
'X          ' 'X '         igual
'X          ' 'X'          igual
```

Oracle 10g:

```
C1           C2           COMPARAÇÃO
------------ ------------ ----------
'X          ' 'X          ' igual

C1           C2           COMPARAÇÃO
------------ ------------ ----------
'X'          'X '         diferente

C1           C2           COMPARAÇÃO
------------ ------------ ----------
'X          ' 'X '         diferente
'X          ' 'X'          diferente
```

DB2 8.1:

```
C1           C2           COMPARAÇÃO
------------ ------------ ----------
'X          ' 'X          ' igual

C1           C2           COMPARAÇÃO
------------ ------------ ----------
'X'          'X '         igual

C1           C2           COMPARAÇÃO
------------ ------------ ----------
'X          ' 'X '         igual
'X          ' 'X'          igual
```

Como pode ser visto, no SQL Server e no DB2 todas as comparações foram consideradas como sendo iguais. No Oracle só foi considerada igual a comparação entre dois tipos char, enquanto no PostgreSQL só foi considerada diferente a comparação entre dois tipos varchar.

8.4. Tipos de dado binários

O tipo de dado bytea permite o armazenamento de *cadeias binárias*; consulte a Tabela 8-6.

Tabela 8-6. Tipos de dado binários

Nome	Tamanho de Armazenamento	Descrição
bytea	4 bytes mais a cadeia binária	Cadeia binária de comprimento variável

A cadeia binária é uma seqüência de octetos (ou bytes). As cadeias binárias se distinguem das cadeias de caracteres por duas características: Em primeiro lugar, as cadeias binárias permitem especificamente o armazenamento de octetos com o valor zero e outros octetos "não-imprimíveis" (geralmente octetos fora da faixa 32 a 126), enquanto as cadeias de caracteres não permitem o octeto com valor zero, e também não permitem outros valores de octeto e seqüências de valores de octeto inválidos para o conjunto de caracteres da codificação selecionada para o banco de dados. Em segundo lugar, as operações nas cadeias binárias processam os bytes como estão armazenados, enquanto o processamento das cadeias de caracteres depende do idioma definido. Resumindo, as cadeias binárias são apropriadas para armazenar dados que os programadores imaginam como "bytes brutos" (*raw bytes* [1]), enquanto as cadeias de caracteres são apropriadas para armazenar texto.

Ao se entrar com valores para bytea, os octetos com certos valores *devem* ser colocados em uma seqüência de escape (porém, todos os valores de octeto *podem* ser colocados em uma seqüência de escape), quando utilizados como parte de um literal cadeia de bytes em uma declaração SQL. Em geral, para construir a seqüência de escape de um octeto o mesmo é convertido em um número octal de três dígitos equivalente ao valor decimal do octeto, e precedido por duas contrabarras. A Tabela 8-7 mostra os caracteres que devem ser colocados em uma seqüência de escape, e fornece seqüências de escape alternativas onde aplicável.

Tabela 8-7. Octetos com seqüência de escape para literais *bytea*

Valor decimal do octeto	Descrição	Representação da entrada com escape	Exemplo	Representação da saída

Valor decimal do octeto	Descrição	Representação da entrada com escape	Exemplo	Representação da saída
0	octeto zero	'\\000'	SELECT '\\000'::bytea;	\000
39	apóstrofo	'\'' ou '\\047'	SELECT '\''::bytea;	'
92	contrabarra	'\\\\' ou '\\134'	SELECT '\\\\'::bytea;	\\
0 a 31 e 127 a 255	octetos "não-imprimíveis"	'\\xxx' (valor octal)	SELECT '\\001'::bytea;	\001

A necessidade de colocar os octetos "não-imprimíveis" em uma seqüência de escape varia conforme o idioma definido. Em certas circunstâncias podem ser deixados fora de uma seqüência de escape. Deve ser observado que o resultado de todos os exemplos da Tabela 8-7 têm exatamente um octeto de comprimento, muito embora a representação de saída do octeto zero e da contrabarra possuam mais de um caractere.

Exemplo 8-5. Letra acentuada em *bytea*

Neste exemplo é feita a conversão explícita da cadeia de caracteres aàáâã para o tipo bytea. Os mesmos resultados são obtidos em bancos de dados com conjunto de caracteres LATIN1 e SQL_ASCII, desde que o conjunto de caracteres do cliente seja ISO-8859-1 ou 1252 (Windows). A letra a sem acento é mostrada literalmente, mas as letras acentuadas são mostradas através de valores octais. [2] [3]

```
=> \!chcp 1252
Active code page: 1252
=> SELECT cast('aàáâã' AS bytea);

       bytea
--------------------
 a\340\341\342\343
(1 linha)
```

O motivo pelo qual é necessário escrever tantas contrabarras, conforme mostrado na Tabela 8-7, é que uma cadeia de caracteres de entrada escrita como um literal cadeia de caracteres deve passar por duas fases de análise no servidor PostgreSQL. A primeira contrabarra de cada par é interpretada como um caractere de escape pelo analisador de literais cadeias de caracteres e portanto consumida, deixando a segunda contrabarra do par. A contrabarra remanescente é então reconhecida pela função de entrada de bytea como o início de um valor octal de três dígitos ou como escape de outra contrabarra. Por exemplo, o literal cadeia de caracteres passado para o servidor como '\\001' se torna '\001' após passar pelo analisador de literais cadeias de caracteres. O '\001' é então

enviado para a função de entrada de bytea, onde é convertido em um único octeto com valor decimal igual a 1. Deve ser observado que o caractere apóstrofo não recebe tratamento especial por bytea e, portanto, segue as regras usuais para literais cadeias de caracteres (Consulte também a Seção 4.1.2.1.)

Os octetos bytea também são transformados em seqüências de escape na saída. De uma maneira geral, cada octeto "não-imprimível" é convertido em seu valor octal equivalente de três dígitos, e precedido por uma contrabarra. Os octetos "imprimíveis" são, em sua maioria, representados através de sua representação padrão no conjunto de caracteres do cliente. O octeto com valor decimal 92 (contrabarra) possui uma representação de saída alternativa especial. Os detalhes podem ser vistos na Tabela 8-8.

Tabela 8-8. Saída dos octetos *bytea* com escape

Valor decimal do octeto	Descrição	Representação da saída com escape	Exemplo	Resultado de saída
92	contrabarra	\\	SELECT '\\134'::bytea;	\\
0 a 31 e 127 a 255	octetos "não-imprimíveis"	\xxx (valor octal)	SELECT '\\001'::bytea;	\001
32 a 126	octetos "imprimíveis"	representação no conjunto de caracteres do cliente	SELECT '\\176'::bytea;	~

Dependendo do programa cliente do PostgreSQL utilizado, pode haver trabalho adicional a ser realizado em relação a colocar e retirar escapes das cadeias bytea. Por exemplo, pode ser necessário colocar escapes para os caracteres de nova-linha e retorno-de-carro se a interface realizar a tradução automática destes caracteres.

O padrão SQL define um tipo de cadeia binária diferente, chamado BLOB ou BINARY LARGE OBJECT (objeto binário grande). O formato de entrada é diferente se comparado com bytea, mas as funções e operadores fornecidos são praticamente os mesmos.

8.5. Tipos para data e hora

O PostgreSQL suporta o conjunto completo de tipos para data e hora do SQL, mostrados na Tabela 8-9. As operações disponíveis para estes tipos de dado estão descritas na Seção 9.9.

Tabela 8-9. Tipos para data e hora

Nome	Tamanho de Armazenamento	Descrição	Menor valor	Maior valor	Resolução
timestamp [(p)] [without time zone]	8 bytes	tanto data quanto hora	4713 AC	5874897 DC	1 microssegundo / 14 dígitos
timestamp [(p)] with time zone	8 bytes	tanto data quanto hora, com zona horária	4713 AC	5874897 DC	1 microssegundo / 14 dígitos
interval [(p)]	12 bytes	intervalo de tempo	-178000000 anos	178000000 anos	1 microssegundo / 14 dígitos
date	4 bytes	somente data	4713 AC	32767 DC	1 dia
time [(p)] [without time zone]	8 bytes	somente a hora do dia	00:00:00.00	23:59:59.99	1 microssegundo / 14 dígitos
time [(p)] with time zone	12 bytes	somente a hora do dia, com zona horária	00:00:00.00+12	23:59:59.99-12	1 microssegundo / 14 dígitos

Nota: Antes do PostgreSQL 7.3, escrever apenas timestamp equivalia a escrever timestamp with time zone. Isto foi mudado para ficar em conformidade com o padrão SQL.

Os tipos time, timestamp, e interval aceitam um valor opcional de precisão p, que especifica o número de dígitos fracionários mantidos no campo de segundos. Por padrão não existe limite explícito para a precisão. O intervalo permitido para p é de 0 a 6 para os tipos timestamp e interval.

Nota: Quando os valores de timestamp são armazenados como números de ponto flutuante de precisão dupla (atualmente o padrão), o limite efetivo da precisão pode ser inferior a 6. Os valores de timestamp são armazenados como segundos antes ou após a meia-noite de 2000-01-01. A precisão de microssegundos é obtida para datas próximas a 2000-01-01 (alguns anos), mas a precisão degrada para datas mais afastadas. Quando os valores de timestamp são armazenadas como inteiros de oito bytes (uma opção de

compilação), a precisão de microssegundo está disponível para toda a faixa de valores. Entretanto, os valores de timestamp em inteiros de 8 bytes possuem uma faixa de tempo mais limitada do que a mostrada acima: de 4713 AC até 294276 DC. A mesma opção de compilação também determina se os valores de time e interval são armazenados como ponto flutuante ou inteiros de oito bytes. No caso de ponto flutuante, a precisão dos valores de interval degrada conforme o tamanho do intervalo aumenta.

Para os tipos time o intervalo permitido para p é de 0 a 6 quando armazenados em inteiros de oito bytes, e de 0 a 10 quando armazenados em ponto flutuante.

O tipo time with time zone é definido pelo padrão SQL, mas a definição contém propriedades que levam a uma utilidade duvidosa. Na maioria dos casos, a combinação de date, time, timestamp without time zone e timestamp with time zone deve fornecer uma faixa completa de funcionalidades para data e hora requeridas por qualquer aplicativo.

Os tipos abstime e reltime são tipos de menor precisão usados internamente. É desestimulada a utilização destes tipos em novos aplicativos, além de ser incentivada a migração dos aplicativos antigos quando apropriado. Qualquer um destes tipos internos pode desaparecer em uma versão futura, ou mesmo todos.

8.5.1. Entrada de data e hora

A entrada da data e da hora é aceita em praticamente todos os formatos razoáveis, incluindo o *ISO 8601*, o SQL-compatível, o POSTGRES tradicional, além de outros. Para alguns formatos a ordem do dia, mês e ano na entrada da data é ambíguo e, por isso, existe suporte para especificar a ordem esperada destes campos. Deve ser definido o parâmetro DateStyle como MDY para selecionar a interpretação mês-dia-ano, DMY para selecionar a interpretação dia-mês-ano, ou YMD para selecionar a interpretação ano-mês-dia.

O PostgreSQL é mais flexível no tratamento da entrada de data e hora do que o requerido pelo padrão SQL. Consulte o Apêndice B para conhecer as regras exatas de análise da entrada de data e hora e os campos texto reconhecidos, incluindo meses, dias da semana e zonas horárias.

Lembre-se que qualquer entrada literal de data ou hora necessita estar entre apóstrofos, como os textos das cadeias de caracteres. Consulte a Seção 4.1.2.5 para obter informações adicionais. O SQL requer a seguinte sintaxe

```
tipo [ (p) ] 'valor'
```

onde p, na especificação opcional da precisão, é um número inteiro correspondendo ao número de dígitos fracionários do campo de segundos. A precisão pode ser especificada

para os tipos time, timestamp e interval. Os valores permitidos estão mencionados acima. Se não for especificada nenhuma precisão na especificação da constante, a precisão do valor literal torna-se o padrão.

8.5.1.1. Datas

A Tabela 8-10 mostra algumas entradas possíveis para o tipo date.

Tabela 8-10. Entrada de data

Exemplo	Descrição
January 8, 1999	não-ambíguo em qualquer modo de entrada em datestyle
1999-01-08	ISO 8601; 8 de janeiro em qualquer modo (formato recomendado)
1/8/1999	8 de janeiro no modo MDY; 1 de agosto no modo DMY
1/18/1999	18 de janeiro no modo MDY; rejeitado nos demais modos
01/02/03	2 de janeiro de 2003 no modo MDY; 1 de fevereiro de 2003 no modo DMY; 3 de fevereiro de 2001 no modo YMD
1999-Jan-08	8 de janeiro e qualquer modo
Jan-08-1999	January 8 em qualquer modo
08-Jan-1999	8 de janeiro em qualquer modo
99-Jan-08	8 de janeiro no modo YMD, caso contrário errado
08-Jan-99	8 de janeiro, porém errado no modo YMD
Jan-08-99	8 de janeiro, porém errado no modo YMD
19990108	ISO 8601; 8 de janeiro de 1999 em qualquer modo
990108	ISO 8601; 8 de janeiro de 1999 em qualquer modo
1999.008	ano e dia do ano
J2451187	dia juliano
January 8, 99 BC	ano 99 antes da era comum [a]

Notas:
a. A Era Comum (EC), ou "Common Era (CE)" em inglês, é um termo relativamente novo que tem experimentado um aumento de utilização e se espera que, eventualmente, substitua AD. Este último é uma abreviação de "Anno Domini" em Latim, ou "Ano do Senhor". Este último se refere ao ano de nascimento aproximado de Yeshua de Nazaré (Jesus Cristo). EC, CE, AD e DC possuem o mesmo valor. The use of "CE" and "BCE" to identify dates (N. do T.)

8.5.1.2. Horas

Os tipos hora-do-dia são time [(p)] without time zone e time [(p)] with time zone. Escrever apenas time equivale a escrever time without time zone.

Entradas válidas para estes tipos consistem na hora do dia seguida por uma zona horária opcional (Consulte a Tabela 8-11 e a Tabela 8-12). Se for especificada a zona horária na entrada de time without time zone, esta é ignorada em silêncio.

Tabela 8-11. Entrada de hora

Exemplo	Descrição
04:05:06.789	ISO 8601
04:05:06	ISO 8601
04:05	ISO 8601
040506	ISO 8601
04:05 AM	o mesmo que 04:05; AM não afeta o valor
04:05 PM	o mesmo que 16:05; a hora entrada deve ser <= 12
04:05:06.789-8	ISO 8601
04:05:06-08:00	ISO 8601
04:05-08:00	ISO 8601
040506-08	ISO 8601
04:05:06 PST	zona horária especificada pelo nome

Tabela 8-12. Entrada de zona horária

Exemplo	Descrição
PST	Hora Padrão do Pacífico (Pacific Standard Time)
-8:00	deslocamento ISO-8601 para PST
-800	deslocamento ISO-8601 para PST
-8	deslocamento ISO-8601 para PST
zulu	Abreviatura militar para UTC
z	Forma abreviada de zulu

Consulte o Apêndice B para ver a lista de nomes de zona horária reconhecidos na entrada.

8.5.1.3. Carimbos do tempo

As entradas válidas para os tipos carimbo do tempo são formadas pela concatenação da data com a hora seguida, opcionalmente, pela zona horária, e seguida opcionalmente por AD ou BC (Como alternativa, AD ou BC pode aparecer antes da zona horária, mas esta não é a ordem preferida). Portanto,

```
1999-01-08 04:05:06
```

e

```
1999-01-08 04:05:06 -8:00
```

são valores válidos, que seguem o padrão *ISO 8601*. Além desses, é suportado o formato muito utilizado

```
January 8 04:05:06 1999 PST
```

O padrão SQL diferencia os literais timestamp without time zone de timestamp with time zone pela existência de "+"; ou "-". Portanto, de acordo com o padrão,

```
TIMESTAMP '2004-10-19 10:23:54'
```

é um timestamp without time zone, enquanto

```
TIMESTAMP '2004-10-19 10:23:54+02'
```

é um timestamp with time zone. O PostgreSQL difere do padrão requerendo que os literais timestamp with time zone sejam digitados explicitamente:

```
TIMESTAMP WITH TIME ZONE '2004-10-19 10:23:54+02'
```

Se o literal não for informado explicitamente como sendo timestamp with time zone, o PostgreSQL ignora em silêncio qualquer indicação de zona horária no literal. Ou seja, o valor resultante de data e hora é derivado dos campos data e hora do valor da entrada, não sendo ajustado conforme a zona horária.

Para timestamp with time zone, o valor armazenado internamente está sempre em UTC (Tempo Universal Coordenado, tradicionalmente conhecido por Hora Média de Greenwich, GMT [1]). Um valor de entrada possuindo a zona horária especificada explicitamente é convertido em UTC utilizando o deslocamento apropriado para esta zona horária. Se não for especificada nenhuma zona horária na cadeia de caracteres da entrada, pressupõe-se que está na mesma zona horária indicada pelo parâmetro do sistema timezone, sendo convertida em UTC utilizando o deslocamento da zona em timezone.

Quando um valor de timestamp with time zone é enviado para a saída, é sempre convertido de UTC para a zona horária corrente de timezone, e mostrado como hora local desta zona. Para ver a hora em outra zona horária, ou se muda timezone ou se usa a construção AT TIME ZONE (consulte a Seção 9.9.3).

As conversões entre timestamp without time zone e timestamp with time zone normalmente assumem que os valores de timestamp without time zone devem ser recebidos ou fornecidos como hora local da timezone. A referência para uma zona horária diferente pode ser especificada para a conversão utilizando AT TIME ZONE.

8.5.1.4. Intervalos

Os valores do tipo interval podem ser escritos utilizando uma das seguintes sintaxes:

```
[@] quantidade unidade [quantidade unidade...] [direção]
```

onde: quantidade é um número (possivelmente com sinal); unidade é second, minute, hour, day, week, month, year, decade, century, millennium, ou abreviaturas ou plurais destas unidades; direção pode ser ago (atrás) ou vazio. O sinal de arroba (@) é opcional. As quantidades com unidades diferentes são implicitamente adicionadas na conta com o sinal adequado.

As quantidades de dias, horas, minutos e segundos podem ser especificadas sem informar explicitamente as unidades. Por exemplo, '1 12:59:10' é lido do mesmo modo que '1 day 12 hours 59 min 10 sec'.

A precisão opcional p deve estar entre 0 e 6, sendo usado como padrão a precisão do literal da entrada.

8.5.1.5. Valores especiais

Por ser conveniente, o PostgreSQL também suporta vários valores especiais para entrada de data e hora, conforme mostrado na Tabela 8-13. Os valores infinity e -infinity possuem representação especial dentro do sistema, sendo mostrados da mesma maneira; porém, os demais são simplesmente notações abreviadas convertidas para valores comuns de data e hora ao serem lidos (Em particular, now e as cadeias de caracteres relacionadas são convertidas para um valor específico de data e hora tão logo são lidas). Todos estes valores devem ser escritos entre apóstrofos quando usados como constantes nos comandos SQL.

Tabela 8-13. Entradas especiais de data e hora

Cadeia de caracteres entrada	Tipos válidos	Descrição

Cadeia de caracteres entrada	Tipos válidos	Descrição
epoch	date, timestamp	1970-01-01 00:00:00+00 (hora zero do sistema Unix)
infinity	timestamp	mais tarde que todos os outros carimbos do tempo
-infinity	timestamp	mais cedo que todos os outros carimbos do tempo
now	date, time, timestamp	hora de início da transação corrente
today	date, timestamp	meia-noite de hoje
tomorrow	date, timestamp	meia-noite de amanhã
yesterday	date, timestamp	meia-noite de ontem
allballs	time	00:00:00.00 UTC

Também podem ser utilizadas as seguintes funções, compatíveis com o padrão SQL, para obter o valor corrente de data e hora para o tipo de dado correspondente: CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP, LOCALTIME e LOCALTIMESTAMP. As últimas quatro aceitam, opcionalmente, a especificação da precisão (consulte também a Seção 9.9.4). Entretanto, deve ser observado que são funções SQL, *não* sendo reconhecidas como cadeias de caracteres de entrada de dados.

Exemplo 8-6. Utilização das entradas especiais de data e hora

Neste exemplo são mostradas utilizações das entradas especiais de data e hora para o tipo timestamp with time zone. [2]

```
BEGIN;
CREATE TEMPORARY TABLE t ( c1 TEXT, c2 TIMESTAMP WITH TIME ZONE ) ON
COMMIT DROP;
INSERT INTO t VALUES ('epoch', 'epoch');
INSERT INTO t VALUES ('infinity', 'infinity');
INSERT INTO t VALUES ('-infinity', '-infinity');
INSERT INTO t VALUES ('now', 'now');
INSERT INTO t VALUES ('today', 'today');
INSERT INTO t VALUES ('tomorrow', 'tomorrow');
INSERT INTO t VALUES ('yesterday', 'yesterday');
INSERT INTO t VALUES ('CURRENT_TIMESTAMP', CURRENT_TIMESTAMP);
SELECT * FROM t;
COMMIT;
```

```
        c1          |             c2
--------------------+------------------------------
 epoch              | 1969-12-31 21:00:00-03
 infinity           | infinity
```

```
 -infinity          | -infinity
 now                | 2007-03-04 09:10:20.234-03
 today              | 2007-03-04 00:00:00-03
 tomorrow           | 2007-03-05 00:00:00-03
 yesterday          | 2007-03-03 00:00:00-03
 CURRENT_TIMESTAMP  | 2007-03-04 09:10:20.234-03
(8 linhas)
```

8.5.2. Saídas de data e hora

Utilizando o comando SET datestyle o formato de saída para os tipos de data e hora pode ser definido como um dos quatro estilos *ISO 8601*, SQL (Ingres), POSTGRES tradicional e German. O padrão é o formato ISO (o padrão SQL requer a utilização do formato *ISO 8601*; o nome do formato de saída "SQL" é um acidente histórico). A Tabela 8-14 mostra exemplo de cada um dos estilos de saída. A saída dos tipos date e time obviamente utilizam apenas a parte da data ou da hora de acordo com os exemplos fornecidos.

Tabela 8-14. Estilos de saída de data e hora

Especificação de estilo	Descrição	Exemplo
ISO	ISO 8601/padrão SQL	2005-04-21 18:39:28.283566-03
SQL	estilo tradicional	04/21/2005 18:39:28.283566 BRT
POSTGRES	estilo original	Thu Apr 21 18:39:28.283566 2005 BRT
German	estilo regional	21.04.2005 18:39:28.283566 BRT

Nos estilos SQL e POSTGRES, o dia vem antes do mês se a ordem de campo DMY tiver sido especificada, senão o mês vem antes do dia (veja na Seção 8.5.1 como esta especificação também afeta a interpretação dos valores de entrada). A Tabela 8-15 mostra um exemplo.

Tabela 8-15. Convenções de ordem na data

Definição de datestyle	Ordem de entrada	Exemplo de saída
SQL, DMY	dia/mês/ano	21/04/2005 18:39:28.283566 BRT
SQL, MDY	mês/dia/ano	04/21/2005 18:39:28.283566 BRT
Postgres, DMY	dia/mês/ano	Thu 21 Apr 18:39:28.283566 2005 BRT

A saída do tipo interval se parece com o formato da entrada, exceto que as unidades como century e week são convertidas em anos e dias, e que ago é convertido no sinal apropriado. No modo ISO a saída se parece com

```
[ quantidade unidade [ ... ] ] [ dias ] [ horas:minutos:segundos ]
```

Os estilos de data e hora podem ser selecionados pelo usuário utilizando o comando SET datestyle, o parâmetro DateStyle no arquivo de configuração `postgresql.conf`, ou a variável de ambiente PGDATESTYLE no servidor ou no cliente. A função de formatação to_char (consulte a Seção 9.8) também pode ser utilizada como uma forma mais flexível de formatar a saída de data e hora.

8.5.3. Zonas horárias

Zonas horárias e convenções de zonas horárias são influenciadas por decisões políticas, e não apenas pela geometria da Terra. As zonas horárias em torno do mundo se tornaram um tanto padronizadas durante o século XX, mas continuam propensas a mudanças arbitrárias, particularmente com relação a horários de inverno e de verão. Atualmente o PostgreSQL suporta as regras de horário de inverno e verão (daylight-savings rules) no período de tempo que vai de 1902 até 2038 (correspondendo à faixa completa de tempo do sistema Unix convencional). Datas fora desta faixa são consideradas como estando na "hora padrão" da zona horária selecionada, sem importar em que parte do ano se encontram.

O PostgreSQL se esforça para ser compatível com as definições do padrão SQL para o uso típico. Entretanto, o padrão SQL possui uma combinação única de tipos e funcionalidades para data e hora. Os dois problemas óbvios são:

- Embora o tipo date não possua zona horária associada, o tipo time pode possuir. As zonas horárias do mundo real possuem pouco significado a menos que estejam associadas a uma data e hora, porque o deslocamento pode variar durante o ano devido ao horário de verão.
- A zona horária padrão é especificada como um deslocamento numérico constante em relação ao UTC. Não é possível, portanto, fazer ajuste devido ao horário de verão ao se realizar aritmética de data e hora entre fronteiras do horário de verão (DST). [3]

Para superar estas dificuldades, recomenda-se utilizar tipos de data e hora contendo tanto a data quanto a hora quando utilizar zonas horárias. Recomenda-se *não* utilizar o tipo time with time zone (embora seja suportado pelo PostgreSQL para os aplicativos legados e para conformidade com o padrão SQL). O PostgreSQL assume a zona horária local para qualquer tipo contendo apenas a data ou a hora.

Todas as datas e horas com zona horária são armazenadas internamente em UTC. São convertidas para a hora local na zona especificada pelo parâmetro de configuração timezone antes de serem mostradas ao cliente.

O parâmetro de configuração timezone pode ser definido no arquivo `postgresql.conf`, ou por qualquer outro meio padrão descrito na Seção 16.4. Existem, também, várias outras formas especiais de defini-lo:

- Se timezone não for especificada no arquivo de configuração `postgresql.conf`, nem como uma chave da linha de comando do postmaster, o servidor tenta utilizar o valor da variável de ambiente TZ como a zona horária padrão. Se TZ não estiver definida, ou não contiver nenhum nome de zona horária conhecido pelo PostgreSQL, o servidor tenta determinar a zona horária padrão do sistema operacional verificando o comportamento da função localtime() da biblioteca C. A zona horária padrão é selecionada através da correspondência mais próximas entre as zonas horárias conhecidas pelo PostgreSQL.
- O comando SQL SET TIME ZONE define a zona horária para a sessão. Esta é uma forma alternativa de SET TIMEZONE TO com uma sintaxe mais compatível com a especificação SQL.
- Se a variável de ambiente PGTZ estiver definida no cliente, é utilizada pelos aplicativos libpq para enviar o comando SET TIME ZONE para o servidor durante a conexão.

Consulte o Apêndice B para obter a lista de zonas horárias disponíveis.

8.5.4. Internamente

O PostgreSQL utiliza datas Julianas [4] para todos os cálculos de data e hora, porque possuem a boa propriedade de predizer/calcular corretamente qualquer data mais recente que 4713 AC até bem distante no futuro, partindo da premissa que o ano possui 365,2425 dias.

As convenções de data anteriores ao século 19 são uma leitura interessante, mas não são suficientemente consistentes para permitir a codificação em rotinas tratadoras de data e hora.

8.6. Tipo booleano

O PostgreSQL disponibiliza o tipo SQL padrão boolean. O tipo boolean pode possuir apenas um dos dois estados: "verdade" ou "falso". O terceiro estado, "desconhecido", é representado pelo valor nulo do SQL. [1] [2]

Os valores literais válidos para o estado "verdade" são:

TRUE

't'

'true'

'y'

'yes'

'1'

Para o estado "falso" podem ser utilizados os seguintes valores:

FALSE

'f'

'false'

'n'

'no'

'0'

A utilização das palavras chave TRUE e FALSE é preferida (e em conformidade com o padrão SQL).

Exemplo 8-7. Utilização do tipo *boolean*

```
CREATE TABLE teste1 (a boolean, b text);
INSERT INTO teste1 VALUES (TRUE, 'sic est');
INSERT INTO teste1 VALUES (FALSE, 'non est');
SELECT * FROM teste1;

 a |    b
---+---------
 t | sic est
 f | non est

SELECT * FROM teste1 WHERE a;

 a |    b
---+---------
 t | sic est
```

O Exemplo 8-7 mostra que os valores do tipo boolean são exibidos utilizando as letras t e f.

Dica: Os valores do tipo boolean não podem ser convertidos diretamente em outros tipos (por exemplo, CAST (valor_booleano AS integer) não funciona). A conversão pode ser feita utilizando a expressão CASE: CASE WHEN valor_booleano THEN 'valor se for verdade' ELSE 'valor se for falso' END. Consulte a Seção 9.13.

O tipo boolean utiliza 1 byte para seu armazenamento.

Exemplo 8-8. Classificação do tipo *boolean*

Segundo o padrão SQL o valor verdade é maior que o valor falso. O PostgreSQL considera o valor nulo maior que estes dois, conforme mostrado neste exemplo. [3]

```
\pset null -
BEGIN;
CREATE TEMPORARY TABLE t (b BOOLEAN) ON COMMIT DROP;
INSERT INTO t VALUES(true);
INSERT INTO t VALUES(false);
INSERT INTO t VALUES(null);
SELECT * FROM t ORDER BY b;
COMMIT;

 b
---
 f
 t
 -
(3 linhas)
```

8.7. Tipos geométricos

Os tipos de dado geométricos representam objetos espaciais bidimensionais. A Tabela 8-16 mostra os tipos geométricos disponíveis no PostgreSQL. O tipo mais fundamental, o ponto, forma a base para todos os outros tipos.

Tabela 8-16. Tipos geométricos

Nome	Tamanho de Armazenamento	Descrição	Representação
point	16 bytes	Ponto no plano	(x,y)
line	32 bytes	Linha infinita (não totalmente implementado)	((x1,y1),(x2,y2))
lseg	32 bytes	Segmento de linha finito	((x1,y1),(x2,y2))
box	32 bytes	Caixa retangular	((x1,y1),(x2,y2))
path	16+16n bytes	Caminho fechado (semelhante ao polígono)	((x1,y1),...)
path	16+16n bytes	Caminho aberto	[(x1,y1),...]
polygon	40+16n bytes	Polígono (semelhante ao caminho fechado)	((x1,y1),...)
circle	24 bytes	Círculo	<(x,y),r> (centro e

Nome	Tamanho de Armazenamento	Descrição	Representação
			raio)

Está disponível um amplo conjunto de funções e operadores para realizar várias operações geométricas, como escala, translação, rotação e determinar interseções, conforme explicadas na Seção 9.10.

8.7.1. Pontos

Os pontos são os blocos de construção bidimensionais fundamentais para os tipos geométricos. Os valores do tipo point são especificados utilizando a seguinte sintaxe:

```
( x , y )
  x , y
```

onde x e y são as respectivas coordenadas na forma de números de ponto flutuante.

8.7.2. Segmentos de linha

Os segmentos de linha (lseg) são representados por pares de pontos. Os valores do tipo lseg são especificado utilizando a seguinte sintaxe:

```
( ( x1 , y1 ) , ( x2 , y2 ) )
  ( x1 , y1 ) , ( x2 , y2 )
    x1 , y1   ,   x2 , y2
```

onde (x1,y1) e (x2,y2) são os pontos das extremidades do segmento de linha.

8.7.3. Caixas

As caixas são representadas por pares de pontos de vértices opostos da caixa. Os valores do tipo box são especificados utilizando a seguinte sintaxe:

```
( ( x1 , y1 ) , ( x2 , y2 ) )
  ( x1 , y1 ) , ( x2 , y2 )
    x1 , y1   ,   x2 , y2
```

onde (x1,y1) e (x2,y2) são quaisquer vértices opostos da caixa.

As caixas são mostradas utilizando a primeira sintaxe. Os vértices são reordenados na entrada para armazenar o vértice direito superior e, depois, o vértice esquerdo inferior.

Podem ser especificados outros vértices da caixa, mas os vértices esquerdo inferior e direito superior são determinados a partir da entrada e armazenados.

8.7.4. Caminhos

Os caminhos são representados por listas de pontos conectados. Os caminhos podem ser *abertos*, onde o primeiro e o último ponto da lista não são considerados conectados, e *fechados*, onde o primeiro e o último ponto são considerados conectados.

Os valores do tipo path são especificados utilizando a seguinte sintaxe:

```
( ( x1 , y1 ) , ... , ( xn , yn ) )
[ ( x1 , y1 ) , ... , ( xn , yn ) ]
  ( x1 , y1 ) , ... , ( xn , yn )
  ( x1 , y1   , ... ,   xn , yn )
    x1 , y1   , ... ,   xn , yn
```

onde os pontos são os pontos das extremidades dos segmentos de linha que compõem o caminho. Os colchetes ([]) indicam um caminho aberto, enquanto os parênteses (()) indicam um caminho fechado.

Os caminhos são mostrados utilizando a primeira sintaxe.

8.7.5. Polígonos

Os polígonos são representados por uma lista de pontos (os vértices do polígono). Provavelmente os polígonos deveriam ser considerados equivalentes aos caminhos fechados, mas são armazenados de forma diferente e possuem um conjunto próprio de rotinas de suporte.

Os valores do tipo polygon são especificados utilizando a seguinte sintaxe:

```
( ( x1 , y1 ) , ... , ( xn , yn ) )
  ( x1 , y1 ) , ... , ( xn , yn )
  ( x1 , y1   , ... ,   xn , yn )
    x1 , y1   , ... ,   xn , yn
```

onde os pontos são os pontos das extremidades dos segmentos de linha compondo a fronteira do polígono.

Os polígonos são mostrados utilizando a primeira sintaxe.

8.7.6. Círculos

Os círculos são representados por um ponto central e um raio. Os valores do tipo circle são especificado utilizando a seguinte sintaxe:

```
< ( x , y ) , r >
( ( x , y ) , r )
  ( x , y ) , r
    x , y   , r
```

onde (x,y) é o centro e r é o raio do círculo.

Os círculos são mostrados utilizando a primeira sintaxe.

8.8. Tipos para endereço de rede

O PostgreSQL disponibiliza tipos de dado para armazenar endereços IPv4, IPv6 e MAC, conforme mostrado na Tabela 8-17. É preferível utilizar estes tipos em vez dos tipos de texto puro, porque estes tipos possuem verificação de erro na entrada, além de vários operadores e funções especializadas (consulte Seção 9.11).

Tabela 8-17. Tipos para endereço de rede

Nome	Tamanho de Armazenamento	Descrição
cidr	12 ou 24 bytes	redes IPv4 e IPv6
inet	12 ou 24 bytes	hospedeiros e redes IPv4 e IPv6
macaddr	6 bytes	endereço MAC

Ao ordenar os tipos de dado inet e cidr, os endereços IPv4 vêm sempre na frente dos endereços IPv6, inclusive os endereços IPv4 encapsulados ou mapeados em endereços IPv6, tais como ::10.2.3.4 ou ::ffff::10.4.3.2.

8.8.1. *inet*

O tipo de dado inet armazena um endereço de hospedeiro IPv4 ou IPv6 e, opcionalmente, a identificação da sub-rede onde se encontra, tudo em um único campo. A identificação da sub-rede é representada declarando quantos bits do endereço do hospedeiro representam o endereço de rede (a "máscara de rede"). Se a máscara de rede for 32 e o endereço for IPv4, então o valor não indica uma sub-rede, e sim um único hospedeiro. No IPv6 o comprimento do endereço é de 128 bits e, portanto, 128 bits especificam o endereço de um único hospedeiro. Deve ser observado que se for

desejado aceitar apenas endereços de rede, deve ser utilizado o tipo cidr em vez do tipo inet.

O formato de entrada para este tipo é endereço/y, onde endereço é um endereço IPv4 ou IPv6, e y é o número de bits da máscara de rede. Se a parte /y for deixada de fora, então a máscara de rede será 32 para IPv4 e 128 para IPv6, e o valor representa um único hospedeiro apenas. Ao ser mostrado, a porção /y é suprimida se a máscara de rede especificar apenas um único hospedeiro.

8.8.2. *cidr*

O tipo cidr armazena uma especificação de rede IPv4 ou IPv6. Os formatos de entrada e de saída seguem as convenções do Classless Internet Domain Routing [1] O formato para especificar redes é endereço/y, onde endereço é a rede representada por um endereço IPv4 ou IPv6, e y é o número de bits da máscara de rede. Se y for omitido, será calculado utilizando as premissas do sistema de numeração com classes antigo, exceto que será pelo menos suficientemente grande para incluir todos os octetos escritos na entrada. É errado especificar endereço de rede contendo bits definidos à direita da máscara de rede especificada.

A Tabela 8-18 mostra alguns exemplos.

Tabela 8-18. Exemplos de entrada para o tipo *cidr*

Entrada cidr	Saída cidr	abbrev(cidr)
192.168.100.128/25	192.168.100.128/25	192.168.100.128/25
192.168/24	192.168.0.0/24	192.168.0/24
192.168/25	192.168.0.0/25	192.168.0.0/25
192.168.1	192.168.1.0/24	192.168.1/24
192.168	192.168.0.0/24	192.168.0/24
128.1	128.1.0.0/16	128.1/16
128	128.0.0.0/16	128.0/16
128.1.2	128.1.2.0/24	128.1.2/24
10.1.2	10.1.2.0/24	10.1.2/24
10.1	10.1.0.0/16	10.1/16
10	10.0.0.0/8	10/8
10.1.2.3/32	10.1.2.3/32	10.1.2.3/32
2001:4f8:3:ba::/64	2001:4f8:3:ba::/64	2001:4f8:3:ba::/64
2001:4f8:3:ba:2e0:81ff:fe22:	2001:4f8:3:ba:2e0:81ff:fe22:	2001:4f8:3:ba:2e0:81ff:fe

Entrada cidr	Saída cidr	abbrev(cidr)
d1f1/128	d1f1/128	22:d1f1
::ffff:1.2.3.0/120	::ffff:1.2.3.0/120	::ffff:1.2.3/120
::ffff:1.2.3.0/128	::ffff:1.2.3.0/128	::ffff:1.2.3.0/128

8.8.3. *inet* versus *cidr*

A diferença essencial entre os tipos de dado inet e cidr é que inet aceita valores com bits diferente de zero à direita da máscara de rede, enquanto cidr não aceita.

Dica: Caso não se goste do formato de saída para os valores de inet ou cidr, deve-se tentar utilizar as funções host(), text() e abbrev().

8.8.4. *macaddr*

O tipo macaddr armazena endereços de MAC [2] , ou seja, endereços de hardware da placa Ethernet (embora os endereços de MAC sejam utilizados para outras finalidades também). A entrada é aceita em vários formatos habituais incluindo

'08002b:010203'
'08002b-010203'
'0800.2b01.0203'
'08-00-2b-01-02-03'
'08:00:2b:01:02:03'
sendo que todos especificam o mesmo endereço. Letras maiúsculas e minúsculas são aceitas para os dígitos de a a f. A saída é sempre na última forma mostrada.

Na distribuição do código fonte do PostgreSQL, o diretório `contrib/mac` contém ferramentas que podem ser utilizadas para fazer a correspondência entre endereços de MAC e nomes de fabricantes de hardware.