Universidad Nacional Autónoma de México

Facultad de Ingeniería

Bases de Datos

Profesor(a): *Ing. Fernando Arreola Franco*
Semestre 2025-2

Tarea 6

Nombre de la Tarea
Tipos de datos en PostgreSQL

Grupo: 1

Alumno:
Nava Benítez David Emilio
No. de Cuenta:
320291599

Cd. Universitaria a 8 de febrero de 2025

Tipos Numéricos:

Tipos enteros

Los tipos smallint , entire y bigint almacenan números enteros, es decir, sin componentes fraccionarios, de varios rangos. Intentar almacenar valores fuera del rango permitido generará un error.

El tipo entero es la opción habitual, ya que ofrece el mejor equilibrio entre rango, tamaño de almacenamiento y rendimiento. El tipo smallint generalmente solo se usa si el espacio en disco es limitado. El tipo bigint solo debe usarse si el rango entero no es suficiente, ya que este último es definitivamente más rápido.

El tipo bigint podría no funcionar correctamente en todas las plataformas, ya que depende de la compatibilidad del compilador con enteros de ocho bytes. En una máquina sin dicha compatibilidad, bigint funciona igual que un entero (pero aún ocupa ocho bytes de almacenamiento). Sin embargo, no conocemos ninguna plataforma razonable donde esto sea así.

SQL solo especifica los tipos enteros «integer» (o «int») y «smallint» . El tipo «bigint» y los nombres de tipo «int2» , «int4» e «int8» son extensiones compartidas con otros sistemas de bases de datos SQL .

Números de precisión arbitraria

El tipo numérico puede almacenar números con una precisión de hasta 1000 dígitos y realizar cálculos con exactitud. Se recomienda especialmente para almacenar cantidades monetarias y otras que requieren precisión. Sin embargo, la aritmética con valores numéricos es muy lenta en comparación con los tipos enteros o los tipos de punto flotante que se describen en la siguiente sección.

En adelante, usaremos estos términos: la *escala* de un número numérico es el número de dígitos decimales en la parte fraccionaria, a la derecha de la coma decimal. La *precisión* de un número numérico es el número total de dígitos significativos en el número entero, es decir, el número de dígitos a ambos lados de la coma decimal. Por lo tanto, el número 23.5141 tiene una precisión de 6 y una escala de 4. Se puede considerar que los números enteros tienen una escala de cero.

Se pueden configurar tanto la precisión máxima como la escala máxima de una columna numérica . Para declarar una columna de tipo numérico , utilice la sintaxis

NUMERIC(***precision***, ***scale***)

La precisión debe ser positiva, la escala cero o positiva. Alternativamente,

NUMERIC(***precision***)

Tipos de punto flotante

Los tipos de datos de precisión real y doble son tipos numéricos inexactos de precisión variable. En la práctica, estos tipos suelen ser implementaciones del estándar IEEE 754 para aritmética binaria de punto flotante (precisión simple y doble, respectivamente), siempre que el procesador, el sistema operativo y el compilador subyacentes lo admitan.

Tipos de serie

Los tipos de datos serial y bigserial no son tipos verdaderos, sino simplemente una conveniencia de notación para configurar columnas de identificadores únicos. En la implementación actual, especificar

CREATE TABLE ***tablename*** (

colname SERIAL

);

es equivalente a especificar:

CREATE SEQUENCE ***tablename_colname***_seq;

CREATE TABLE ***tablename*** (

colname integer DEFAULT nextval('***tablename_colname***_seq') NOT NULL

);

Por lo tanto, hemos creado una columna de enteros y configurado sus valores predeterminados para que se asignen desde un generador de secuencias. Se aplica una restricción NOT NULL para garantizar que no se pueda insertar explícitamente un valor nulo. En la mayoría de los casos, también se recomienda añadir una restricción UNIQUE o PRIMARY KEY para evitar la inserción accidental de valores duplicados, pero esto no es automático.

Para insertar el siguiente valor de la secuencia en la columna serial , especifique que se le asigne su valor predeterminado . Esto puede hacerse excluyendo la columna de la lista de columnas en la instrucción INSERT o usando la palabra clave DEFAULT .

Tipos de fecha

PostgreSQL admite todos los tipos de fecha y hora de SQL como se muestra en la Tabla.

Name	Storage Size	Description	Low Value	High Value	Resolution
`timestamp [ (p) ] [ without time zone ]`	8 bytes	both date and time	4713 BC	5874897 AD	1 microsecond / 14 digits
`timestamp [ (p) ] with time zone`	8 bytes	both date and time, with time zone	4713 BC	5874897 AD	1 microsecond / 14 digits
`interval [ (p) ]`	12 bytes	time intervals	-178000000 years	178000000 years	1 microsecond / 14 digits
`date`	4 bytes	dates only	4713 BC	5874897 AD	1 day
`time [ (p) ] [ without time zone ]`	8 bytes	times of day only	00:00:00	24:00:00	1 microsecond / 14 digits
`time [ (p) ] with time zone`	12 bytes	times of day only, with time zone	00:00:00+1359	24:00:00-1359	1 microsecond / 14 digits

time , timestamp e interval aceptan un valor de precisión opcional ***p*** , que especifica la cantidad de dígitos fraccionarios que se conservan en el campo de segundos. Por defecto, no hay un límite explícito para la precisión. El rango permitido de ***p*** va de 0 a 6 para los tipos timestamp e interval .

Tipos de Carácter

Name	Description
`character varying(n), varchar(n)`	variable-length with limit
`character(n), char(n)`	fixed-length, blank padded
`text`	variable unlimited length

SQL define dos tipos de caracteres principales: character varying(***n***) y character(***n***) , donde ***n*** es un entero positivo. Ambos tipos pueden almacenar cadenas de hasta ***n*** caracteres. Intentar almacenar una cadena más larga en una columna de estos tipos generará un error, a menos que los caracteres sobrantes sean espacios, en cuyo caso la cadena se truncará a la longitud máxima. (Esta excepción, un tanto peculiar, es requerida por el estándar SQL). Si la cadena que se va a almacenar es más corta que la longitud declarada, los valores de tipo character se rellenarán con espacios; los valores de tipo character varying simplemente almacenarán la cadena más corta.

Table 8-5. Special Character Types

Name	Storage Size	Description
"char"	1 byte	single-character internal type
name	64 bytes	internal type for object names

Interesantes:

Tipos Money

El *money type* almacena una cantidad de moneda con una precisión fraccionaria fija.

Se aceptan valores de entrada en diversos formatos, incluyendo literales enteros y de punto flotante, así como el formato de moneda típico , como '$1,000.00' . La salida generalmente se presenta en este último formato, pero depende de la configuración regional.

Name	Storage Size	Description	Range
money	4 bytes	currency amount	-21474836.48 to +21474836.47

Binary Data Types

Name	Storage Size	Description
bytea	4 bytes plus the actual binary string	variable-length binary string

Una cadena binaria es una secuencia de octetos (o bytes). Las cadenas binarias se distinguen de las cadenas de caracteres por dos características: en primer lugar, las cadenas binarias permiten almacenar específicamente octetos de valor cero y otros octetos "no imprimibles" (normalmente, octetos fuera del rango de 32 a 126). Las cadenas de caracteres no permiten octetos de valor cero, ni tampoco cualquier otro valor de octeto y secuencia de valores de octetos que no sean válidos según la codificación del conjunto de caracteres seleccionado en la base de datos. En segundo lugar, las operaciones con cadenas binarias procesan los bytes reales, mientras que el procesamiento de las cadenas de caracteres depende de la configuración regional. En resumen, las cadenas binarias son adecuadas para almacenar datos que el programador considera como "bytes sin procesar" , mientras que las cadenas de caracteres son adecuadas para almacenar texto.

Al introducir valores bytea , los octetos de ciertos valores deben escaparse (aunque todos los valores de octetos pueden escaparse) cuando se utilizan como parte de una cadena literal en una sentencia SQL . En general, para escapar un octeto, se

convierte en el número octal de tres dígitos equivalente a su valor decimal, precedido por dos barras invertidas.

Bibliografía:

[1] “Data types,” *PostgreSQL Documentation*, Jan. 01, 2012. https://www.postgresql.org/docs/8.1/datatype.html

Cd. Universitaria a 8 de febrero de 2025