Ruiz Aguilar Cristian Jair
Grupo 01
Bases de Datos

Tarea 10: tipos de datos de Postgres

Tipos de datos numéricos			
Nombre	Tamaño de almacenamiento	Descripción	Rango
smallint	2 bytes	Entero de rango pequeño	De -32,768 a 32,767
integer	4 bytes	Elección típica de un entero	De -2147483648 a +2147483647
bigint	8 bytes	Entero de rango largo	De -9223372036854775808 a +9223372036854775807
decimal	Variable	Exacto, precisión determinada por el usuario	Hasta 131072 dígitos antes del punto decimal; hasta 16383 dígitos después del punto decimal.
numeric	Variable	Exacto, precisión determinada por el usuario	Hasta 131072 dígitos antes del punto decimal; hasta 16383 dígitos después del

			punto decimal.
real	4 bytes	Precisión variable, inexacto.	Precisión de 6 dígitos decimales.
double precision	8 bytes	Precisión variable, inexacto.	Precisión de 15 dígitos decimales
smallserial	2 bytes	Entero de incremento automático (pequeño)	De 1 a 32767
serial	4 bytes	Entero de incremento automático	De 1 a 2147483647
bigserial	8 bytes	Entero de incremento automático (largo)	De 1 a 922337203685477 5807

Tipo entero

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

El tipo entero es la opción común, 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 escaso. El tipo bigint está diseñado para usarse cuando el rango del tipo entero es insuficiente.

SQL solo especifica los tipos enteros integer (o int), smallint y bigint. Los nombres de tipo int2, int4 e int8 son extensiones, que también utilizan otros sistemas de bases de datos SQL.

Tipo precisión arbitraria

El tipo numérico puede almacenar números con una gran cantidad de dígitos. Está especialmente recomendado para almacenar cantidades monetarias y otras cantidades donde se requiera exactitud. Los cálculos con valores numéricos arrojan resultados exactos cuando es posible, por ejemplo, suma, resta, multiplicación. Sin embargo, los cálculos de valores numéricos son muy lentos en comparación con los tipos de números enteros o los tipos de coma flotante que se describen en la siguiente sección.

Usamos los siguientes términos a continuación: La precisión de un número es el conteo total de dígitos significativos en el número entero, es decir, el número de dígitos a ambos lados del punto decimal. La escala de un numérico es el conteo de dígitos decimales en la parte fraccionaria, a la derecha del punto decimal. Entonces, 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 puede configurar tanto la precisión máxima como la escala máxima de una columna numérica.

Punto flotante

Los tipos de datos real y de doble precisión son tipos numéricos inexactos de precisión variable. En todas las plataformas admitidas actualmente, estos tipos son implementaciones del estándar IEEE 754 para aritmética de punto flotante binario (precisión simple y doble, respectivamente), en la medida en que el procesador subyacente, el sistema operativo y el compilador lo admitan.

Inexacto significa que algunos valores no se pueden convertir exactamente al formato interno y se almacenan como aproximaciones, por lo que almacenar y recuperar un valor puede mostrar ligeras discrepancias. El manejo de estos errores y cómo se propagan a través de los cálculos es el tema de toda una rama de las

matemáticas y la informática y no se discutirá aquí, excepto por los siguientes puntos:

- Si necesita almacenamiento y cálculos exactos (como para cantidades monetarias), utilice el tipo numérico en su lugar.
- Si desea realizar cálculos complicados con estos tipos para algo importante, especialmente si confía en cierto comportamiento en casos límite (infinito, subdesbordamiento), debe evaluar la implementación con cuidado.
- La comparación de dos valores de punto flotante para la igualdad puede no funcionar siempre como se esperaba.

Tipo serial

Los tipos de datos smallserial, serial y bigserial no son tipos verdaderos, sino simplemente una conveniencia notacional para crear columnas de identificador único (similar a la propiedad AUTO_INCREMENT admitida por algunas otras bases de datos).

Tipos de datos caracteres	
Nombre	Descripción
character varying(n), varchar(n)	Longitud variable con límite
character(n), char(n)	Longitud determinada, espacios en blanco
text	Longitud variable sin límite

SQL define dos tipos de caracteres principales: character varying(n) y character(n), donde n es un número entero positivo. Ambos tipos pueden almacenar cadenas de hasta n caracteres (no bytes) de longitud. Un intento de almacenar una cadena más larga en una columna de estos tipos generará un error, a menos que los caracteres sobrantes sean todos espacios, en cuyo caso la cadena se truncará a la longitud máxima. (Esta excepción un tanto extraña 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 carácter se rellenarán con espacios; los valores de tipo carácter que varían simplemente almacenarán la cadena más corta.

Si uno convierte explícitamente un valor en un carácter que varying(n) o un character(n), entonces un valor de longitud excesiva se truncará en n caracteres sin generar un error. (Esto también es requerido por el estándar SQL).

Las notaciones varchar(n) y char(n) son alias de carácter varying(n) y character(n), respectivamente. Si se especifica, la longitud debe ser mayor que cero y no puede exceder 10485760. El carácter sin especificador de longitud es equivalente al character(1). Si se utiliza la variación de caracteres sin el especificador de longitud, el tipo acepta cadenas de cualquier tamaño. Este último es una extensión de PostgreSQL.

Además, PostgreSQL proporciona el tipo de texto, que almacena cadenas de cualquier longitud. Aunque el tipo de texto no está en el estándar SQL, varios otros sistemas de administración de bases de datos SQL también lo tienen.

Tipos de fechas/hora					
Nombre	Tamaño de almacenamiento	Descripción	Valor menor	Valor mayor	Resolución
timestamp [(p)] [without time zone]	8 bytes	fecha y hora (sin zona horaria)	4713 BC	294276 AD	1 microsegundo
timestamp [(p)] [with time zone]	8 bytes	fecha y hora, con zona horaria	4713 BC	294276 AD	1 microsegundo
date	4 bytes	fecha (sin la hora del día)	4713 BC	5874897 AD	1 día
time [(p)] [without time zone]	8 bytes	hora (sin fecha)	00:00:00	24:00:00	1 microsegundo
time [(p)] [with time zone]	12 bytes	hora del día (sin fecha), con zona horaria	00:00:00+1559	24:00:00-1559	1 microsegundo
interval [fields] [(p)]	16 bytes	intervalo de hora.	-178000000 años	178000000 años	1 microsegundo

time, timestamp e interval aceptan un valor de precisión opcional **p** que especifica el número de dígitos fraccionarios retenidos en el campo de segundos. De forma

predeterminada, no hay un límite explícito en la precisión. El rango permitido de p es de 0 a 6.

La entrada de fecha y hora se acepta en casi cualquier formato razonable, incluido ISO 8601, compatible con SQL, POSTGRES tradicional y otros. Para algunos formatos, el orden de día, mes y año en la entrada de fecha es ambiguo y existe soporte para especificar el orden esperado de estos campos. Establezca el parámetro DateStyle en MDY para seleccionar la interpretación mes-día-año, DMY para seleccionar la interpretación día-mes-año o YMD para seleccionar la interpretación año-mes-día.

PostgreSQL es más flexible en el manejo de la entrada de fecha/hora de lo que requiere el estándar SQL. Consulte el Apéndice B para conocer las reglas de análisis exactas de la entrada de fecha/hora y los campos de texto reconocidos, incluidos los meses, los días de la semana y las zonas horarias.

Tipos monetarios			
Nombre	Tamaño de almacenamiento	Descripción	Rango
money	8 bytes	cantidad de moneda	De -92233720368547 758.08 a +92233720368547 758.07

El rango que se muestra en la tabla asume que hay dos dígitos fraccionarios. La entrada se acepta en una variedad de formatos, incluidos los literales enteros y de punto flotante, así como el formato de moneda típico, como “$1,000.00”. La salida es generalmente en la última forma, pero depende de la configuración regional.

Tipos de datos		
Tipo JSON primitivo	Tipo en PostgreSQL	Notas
string	text	\u0000 is disallowed, as are Unicode escapes representing characters not available in the database encoding
number	numeric	NaN and infinity values are disallowed
boolean	boolean	Only lowercase true and false spellings are accepted
null	(none)	SQL NULL is a different concept

PostgreSQL ofrece dos tipos para almacenar datos JSON: json y jsonb. Para implementar mecanismos de consulta eficientes para estos tipos de datos, PostgreSQL también proporciona el tipo de datos jsonpath.

Los tipos de datos json y jsonb aceptan conjuntos de valores casi idénticos como entrada. La principal diferencia es la eficiencia. El tipo de datos json almacena una copia exacta del texto de entrada, cuyas funciones de procesamiento deben volver a analizar en cada ejecución; mientras que los datos jsonb se almacenan en un formato binario descompuesto que hace que sea un poco más lento de ingresar debido a la sobrecarga de conversión adicional, pero significativamente más rápido de procesar, ya que no se necesita volver a analizar. jsonb también admite la indexación, lo que puede ser una ventaja significativa.

Debido a que el tipo json almacena una copia exacta del texto de entrada, conservará los espacios en blanco semánticamente insignificantes entre los tokens, así como el orden de las claves dentro de los objetos JSON. Además, si un objeto JSON dentro del valor contiene la misma clave más de una vez, se conservan todos

los pares clave/valor. (Las funciones de procesamiento consideran el último valor como el operativo). Por el contrario, jsonb no conserva los espacios en blanco, no conserva el orden de las claves de objeto y no conserva las claves de objeto duplicadas. Si se especifican claves duplicadas en la entrada, solo se conserva el último valor.

En general, la mayoría de las aplicaciones deberían preferir almacenar datos JSON como jsonb, a menos que haya necesidades bastante especializadas, como suposiciones heredadas sobre el orden de las claves de objeto.

Referencia

PostgreSQL Global Development Group (1996). PostgreSQL. Estados Unidos. Disponible en: www.postgresql.org