Universidad Nacional Autónoma de México

Facultad de Ingeniería

Bases de Datos
Profesor: Ing. Fernando Arreola Franco
Tarea 6

Alumnos
Rodríguez Zacarías Iván

Fecha de entrega: 5 de abril, 2025

Tipos de datos en Postgres

En postgres se tienen definidos de manera nativo diversos tipos de datos disponibles para que sean utilizados por los usuarios, también se pueden crear nuevos tipos de datos utilizando CREATE TYPE. En esta tarea se abordarán algunas características de unos tipos de datos interesantes e importantes para el estudio de la asignatura de Bases de Datos mencionando su forma de representación, su capacidad de representación y sus limitantes o desventajas.

Tipos numéricos

En los tipos numérciso podemos utilizar los que se muestran en la siguiente tabla.

Name	Storage Size	Description	Range
`smallint`	2 bytes	small-range integer	-32768 to +32767
`integer`	4 bytes	usual choice for integer	-2147483648 to +2147483647
`bigint`	8 bytes	large-range integer	-9223372036854775808 to 9223372036854775807
`decimal`	variable	user-specified precision, exact	no limit
`numeric`	variable	user-specified precision, exact	no limit
`real`	4 bytes	variable-precision, inexact	6 decimal digits precision
`double precision`	8 bytes	variable-precision, inexact	15 decimal digits precision
`serial`	4 bytes	autoincrementing integer	1 to 2147483647
`bigserial`	8 bytes	large autoincrementing integer	1 to 9223372036854775807

Tipos de datos numéricos enteros en PostgreSQL

Smallint: Este tipo se utiliza para almacenar números enteros sin componentes fraccionarios, puede representar valores en el rango de -32,768 a 32,767, siendo ideal para manejar cifras pequeñas. Tiene un rango limitado y es preferido cuando es necesario ahorrar espacio en disco.

Integer (Int): Representa números enteros y no permite valores fraccionarios. Soporta un rango de -2,147,483,648 a 2,147,483,647. Es el tipo usualmente más utilizado debido a que ofrece un buen equilibrio entre capacidad o rango, almacenamiento y rendimiento. No es útil para almacenar valores que excedan este rango.

Bigint: Diseñado para almacenar números enteros más grandes que superan el límite del tipo integer. Tiene un rango que abarca desde -9,223,372,036,854,775,808 hasta 9,223,372,036,854,775,807. Es más lento en comparación con el tipo integer y puede depender del soporte del compilador para manejar enteros de ocho bytes. Aunque este inconveniente no es común en plataformas actuales.

Tipos de datos numéricos con precisión arbitraria

Los datos numeric permiten almacenar valores con una precisión que alcanza hasta 1000 dígitos. Se pueden definir mediante la **precisión** (total de dígitos) y la **escala** (dígitos decimales), usando las siguientes opciones:

- **NUMERIC (precisión, escala)**: Define tanto la precisión como la escala.
- **NUMERIC (precisión)**: Utiliza una escala predeterminada de 0.
- **NUMERIC**: Admite valores sin restricciones específicas de precisión ni escala, respetando los límites de la implementación.

Este tipo también admite el valor especial **NaN** ("no es un número"), que resulta en NaN tras cualquier operación. En SQL, debe especificarse entre comillas. El tipo de dato decimal es equivalente al numeric.

Es ideal para aplicaciones que requieren alta precisión, como en cálculos monetarios. Algunas de las limitantes de este tipo de dato son:

1. **Rendimiento**: Las operaciones con este tipo de datos son más lentas que las realizadas con números de punto flotante o enteros.
2. **Errores de límites**: Exceder la escala o precisión configurada puede causar errores o redondeos de los valores.
3. **Espacio de almacenamiento**: Cada grupo de cuatro dígitos usa 2 bytes más 8 bytes adicionales.

Tipos de datos de punto flotante

Los tipos **real** y **double precision** son numéricos con precisión variable e inexacta, basados generalmente en el estándar IEEE 754 para aritmética de punto flotante (con precisión simple y doble). Dado que estos valores se almacenan como aproximaciones, pueden surgir diferencias al guardar y recuperar los mismos datos.

Capacidades de representación:

El tipo **real** cubre valores entre 1E-37 y 1E+37, con una precisión mínima de 6 dígitos, mientras el **double precision** maneja valores entre 1E-307 y 1E+308, con al menos 15 dígitos de precisión.

- Los valores **float(p)**, según el estándar SQL, definen la precisión mínima en dígitos binarios:
    - De **float(1)** a **float(24)** corresponden al tipo real.
    - De **float(25)** a **float(53)** se asignan al tipo double precision.
    - Si no se especifica p, se asume **double precision**.
- Permiten cálculos avanzados y un rango extenso de valores.

Limitantes o desventajas:

Algunos valores no pueden representarse con total exactitud, lo que afecta cálculos y comparaciones. Los valores extremadamente grandes o pequeños provocan errores y los números cercanos a cero, indistinguibles de este, generan errores de desbordamiento. Además, si la precisión de entrada supera los límites permitidos, se efectúan redondeos.

Tipos de datos seriales

Los tipos **serial** y **bigserial** no son tipos de datos propiamente dichos, sino una herramienta abreviada para definir columnas que generan identificadores únicos. Su funcionamiento equivale a:

1. Crear una secuencia con CREATE SEQUENCE.
2. Establecer una columna entera que utilice esta secuencia como valor predeterminado mediante DEFAULT nextval().
3. Aplicar la restricción NOT NULL para evitar que se inserten valores nulos.

Capacidad de Representación:

- **serial**: Define columnas de tipo entero (integer), capaces de manejar hasta $231-12^{31}$ - 1 identificadores únicos.
- **bigserial**: Genera columnas de tipo bigint, ideales para tablas que requieren más de $231-12^{31}$ - 1 valores únicos.
- Las secuencias asociadas a columnas serial se eliminan automáticamente si la columna es eliminada.

Algunas desventajas de estos tipos de datos:

A partir de PostgreSQL 7.3, las columnas serial ya no incluyen la restricción UNIQUE por defecto, por lo que debe declararse explícitamente. Adicionalmente, si otras columnas usan manualmente la secuencia de una columna serial, estas referencias fallarán si la secuencia es eliminada, lo que se considera una mala práctica. Además, al usar una secuencia para varias columnas puede causar problemas de gestión y provocar inconsistencias en los datos.

Datos de tipo caracter

PostgreSQL ofrece tres principales tipos para gestionar cadenas de texto:

1. **character varying(n)** o **varchar(n)**: Almacena texto de longitud variable, con un límite de hasta **n** caracteres.
2. **character(n)** o **char(n)**: Almacena texto de longitud fija, completando con espacios si el contenido tiene menos de **n** caracteres.

3. **text**: Permite almacenar texto sin límites de longitud.

Además, incluye dos tipos especiales:

- **"char"**: Representa un único carácter, ocupando 1 byte.
- **name**: Se utiliza internamente para nombres de objetos; admite hasta 64 bytes de longitud.

Los valores del tipo **character(n)** contienen espacios al final que son semánticamente insignificantes en comparaciones. En contraste, los espacios finales tienen relevancia en **character varying** y **text**.

Capacidades de representación:

- **character(n)** y **character varying(n)** admiten cadenas de hasta **n** caracteres. Intentar guardar cadenas más largas genera errores, aunque las cadenas con exceso de espacios se truncarán.
- El tipo **text** permite almacenar cadenas de cualquier tamaño, siendo adecuado para valores extensos y sin restricciones específicas.
- La longitud máxima para almacenar una cadena es de aproximadamente 1 GB.

Algunas consideraciones a tener en cuenta de los datos de tipo caracter

No hay diferencias sustanciales en el rendimiento entre **text** y **character varying**, salvo por el espacio adicional que requiere **character(n)** debido al relleno de espacios. Intentar almacenar valores muy largos en **character(n)** o **character varying(n)** genera errores, mientras que **text** carece de límites. Cada valor ocupa 4 bytes adicionales más el tamaño de la cadena. Las cadenas largas son comprimidas automáticamente y almacenadas en tablas secundarias.

Por lo general, se recomienda utilizar **text** o **character varying(n)**, ya que ofrecen mayor flexibilidad y eficiencia en comparación con **character(n)**.

Tipos de dato de fecha

PostgreSQL acepta múltiples formatos para representar y trabajar con fechas. Los principales tipos y sus características son los siguientes:

Formas de representación

- **Fechas (Date):** Utilizan el formato literal 'YYYY-MM-DD', siguiendo el estándar **ISO 8601** recomendado. Ejemplo: '1999-01-08'.
- **Horas (Time):** Se expresan en formatos como 'HH:MM:SS' y pueden incluir precisión opcional para los segundos.
- **Marcas de tiempo (Timestamp):** Combina fecha y hora en un solo dato, con posibilidad de incluir o excluir zona horaria.
- **Intervalos:** Representan periodos de tiempo (como '2 years 3 months').

Capacidades de representación

1. **Compatibilidad con diversos estándares y estilos:**
    - ISO 8601, reconocido globalmente.
    - Otros estilos tradicionales, como **POSTGRES**, **SQL** y **German**.
    - Flexibilidad para interpretar formatos como '1/8/1999' (MDY) o '08-Jan-1999'.
2. **Manejo opcional de zonas horarias:** Los datos de tipo tiempo pueden incluir zonas horarias, aunque en ciertos casos son ignoradas.
3. **Configuración de precisión:** Es posible definir la precisión en segundos para formatos como time(p).

Limitaciones o desventajas

- **Ambigüedad en formatos:** Algunos valores, como '01/02/03', dependen de la configuración de DateStyle. Esto puede causar confusión si no se establece correctamente.
- **Zonas horarias no procesadas:** En columnas definidas como time without time zone, las zonas horarias especificadas son descartadas.
- **Flexibilidad que puede complicar:** Aunque se aceptan múltiples formatos, esto puede generar dificultades en contextos donde no se define claramente el estándar

Tipos de datos binarios

Forma de representación

- El tipo de dato bytea sirve para almacenar cadenas binarias.
- Estas cadenas están compuestas por secuencias de octetos (bytes) que incluyen valores "no imprimibles," como el octeto cero.

Capacidades de representación

1. Permite guardar octetos con valores entre 0 y 255, lo que incluye caracteres no imprimibles, como aquellos fuera del rango de 32 a 126.
2. Opera directamente sobre los bytes, sin depender de codificaciones de caracteres ni configuraciones locales.
3. Es posible representar ciertos valores mediante escapes, como \\000 para el octeto cero o \\\\ para la barra invertida (\\).

Limitantes o desventajas

Los literales binarios requieren un proceso adicional de escapado, lo que puede resultar engorroso. La necesidad de escapar los octetos no imprimibles depende de las configuraciones regionales. Algunas interfaces requieren un tratamiento adicional, por ejemplo, para manejar saltos de línea o retornos de carro.

Tipos de datos monetarios (Monetary Data Types)

Forma de representación

- El tipo de dato money se utiliza para almacenar valores monetarios.
- Admite diversas entradas, como números enteros, valores de punto flotante o cadenas con formato típico, por ejemplo, $1,000.00.

Capacidades de representación

1. Gestiona cantidades monetarias con una precisión fraccional fija.
2. Permite el uso de diversos formatos de entrada, ajustándose a las configuraciones locales.
3. Puede manejar montos desde -21474836.48 hasta +21474836.47.

Limitantes o desventajas

Se recomienda reemplazar el tipo money por numeric o decimal en conjunto con la función to_char. La representación de los valores depende del formato local, lo que puede generar inconsistencias. Tiene un rango más limitado que otros tipos numéricos, como numeric.

Referencias

[1] Data types. (2012, 1 enero). PostgreSQL Documentation. https://www.postgresql.org/docs/8.1/datatype.html?_x_tr_sl=en&_x_tr_tl=es&_x_tr_hl=es&_x_tr_pto=tc