TAREA 10. TIPOS DE DATOS EN POSTGRES

GARCÍA SANCHÉZ LUIS MANUEL

Para hacer este análisis vamos a agrupar los tipos en tres categorías las cuales son:

- Caracteres
- Números
- Fechas y Hora

Tipos para caracteres

En el caso de texto, tenemos 3 tipos de datos principales, los cuales son: `char`, `varchar` y `text`.

- `char(n)`

Un columna de longitud fija, la cual es especificada por n. Un columna con un tipo por ejemplo char(20) utiliza 20 caracteres por columna independientemente si se están utilizando o no. Por ejemplo, “gato” ocupará 20 espacios de memoria.

- `varchar(n)`

Un columna de longitud variable, donde la máxima longitud es especificada por n. A diferencia del tipo anterior, si se tiene una columna de este tipo, se utiliza solamente la longitud del dato que quiera ser guardado en ella. Por ejemplo, “gato” solo ocupara 4 espacios de memoria.

Este tipo de dato, también puede ser definido sin `n`, de manera que tenga longitud ilimitada.

- `text`

Una columna de longitud variable, donde no existe limitación para la longitud de esta. Aunque este tipo de dato, a diferencia de los anteriores, no es parte del estándar de SQL.

A continuación, un ejemplo de como se definiría una columna con cada uno de los tipos que mencionamos previamente

```
CREATE TABLE char_types (
 type_char char(10),
 type_varchar_lim varchar(10),
 type_varchar_unlim varchar,
```

```
  type_text text
);
```

Comparación tipos para caracteres

Ahora, realizaremos una comparación de lo que sucedería con los distintos tipos de datos al insertar el mismo texto en diferentes columnas con el tipo.

Dato	char(10)	varchar(10)	varchar	text
'prueba'	'prueba '	'prueba'	'prueba'	'prueba'
‘string mas larga’	Fuera de rango	Fuera de rango	‘string mas larga’	‘string mas larga’

Comparación de tipos para caracteres
Como podemos ver, `char` es el único tipo que guardaría espacios vacíos para completar los 10 espacios con los que fue definido. Para el resto de los tipos, no se aprecian muchas diferencias al usar valores pequeño para guardar en ellos.

Una práctica común es evitar el uso del tipo `char`. Normalmente, se hace uso de `varchar`, y los casos donde se quiere limitar la longitud de un campo, se utiliza `varchar(n)` (por ejemplo cadenas de caracteres de longitud específica podría ser el número de placa de un automóvil).

Tipos para números

Para los números tenemos más opciones entre las cuales podemos elegir. Si alguna vez has programado en lenguajes como C, esta sección te será muy familiar. Para iniciar, PostgreSQL nos proporciona tipos para números enteros y números decimales.

Números enteros

Para números enteros, tenemos tres opciones: `smallint`, `integer` y `bigint`. La única diferencia que existe entre ellos, es el tamaño que ocupan memoria, causando que las columnas de distintos tipos de enteros, puedan o no, guardar determinados números.

Para mostrar una comparativa fácil entre los tipos y sus rangos, presento la siguiente tabla:

Tipo de dato	Tamaño en memoria	Rango
`smallint`	2 bytes	-32768 a 32767
`integer`	4 bytes	-2147483648 a 2147483647
`bigint`	8 bytes	-9223372036854775808 a 9223372036854775807

Comparación de tipos de datos para enteros
Si se trata de insertar un número fuera del rango del tipo de la columna, PostgreSQL detendría la operación de inserción y nos regresaría un error mencionando que estamos fuera del rango.

En este apartado, podemos encontrar columna de auto incremento, las cuales son una implementación propia de PostgreSQL, la cual es los `seriales`. Cada uno de los tipos que vimos anteriormente tiene si propio serial:

- `smallint => smallserial`
- `integer => serial`
- `bigint => bigserial`

Estos auto incrementos, es muy común verlos siendo usados en columnas que corresponderán a una llave primaria de una tabla por ejemplo. Su funcionamiento, es muy similar a los tipos con los que están asociados, con la única diferencia es que al realizar un `INSERT` a una tabla con una columna de este tipo, no es necesario especificar el valor para dicha columna: este será asignado automáticamente.

A continuación un ejemplo de como son usados en la definición de columnas

```
CREATE TABLE integer_types (
 column_1 smallint,
 column_2 smallserial,
 column_3 integer,
 column_4 serial,
 column_5 bigint,
 column_6 bigserial,
```

```
);
```

Elegir entre el tipo de entero que vaya a ser utilizado depende del conocimiento que se tenga sobre los datos que se desee guardar en dicha columna. Si se sabe que los enteros que se van a almacenar, estarán dentro del rango del `smallint`, se da preferencia a este con el mejorar el manejo de memoria. Un ejemplo de esto podría ser una columna que almacenará la edad de una persona.

Por otro lado, es común utilizar `bigserial` para todos las columnas que almacenarán un `id`, como podría ser llaves primarias y foráneas.

Números decimales

Para números decimales, tenemos menos fundamentalmente dos representaciones que son implementadas en PostgreSQL.

- Número de Punto Fijo
- Números de Punto flotante

Para entender estos dos tipos es necesario entender la diferencia entre estos dos tipos de números.

Los tipos de punto fijo, son almacenados junto con la información de la precisión y escala que utilizarán. Este tipo de número, tiene en PostgreSQL un tamaño variable, ya que este depende de la precisión y la escala. Estos números utilizan los tipos `numeric` o `decimal`, que pueden ser considerados como el mismo.

Los números de Punto flotante, por el contrario no permiten configurar la precisión y la escala, pero tienen un tamaño constante, y un rango de valores que pueden representar. Existen dos tipos para estos números los cuales son `real` y `double` (estos tipos son muy similares al funcionamiento que tendrían `float` y `double` en lenguaje C).

La principal diferencia entre los números de punto fijo y los de punto flotante es la precisión de estos, y el rango de valores que pueden representar. En la siguiente tabla, podemos hacer una comparación entre los distintos tipos.

Comparación tipos para números decimales

Tipo de dato	Tamaño en memoria	Rango
`numeric`, `decimal`	variable	hasta 131072 dígitos antes del punto; hasta 16383 dígitos después del punto

Tipo de dato	Tamaño en memoria	Rango
real	4 bytes	6 dígitos decimales de precisión
double	8 bytes	15 dígitos decimales de precisión

Comparación de tipos de datos para decimales

Ahora bien, ¿cómo elegir que tipo de datos utilizar para decimales? Pues como todo en la vida depende. Si el objetivo del valor que será en la almacenado en la columna es el de realizar operaciones con este, y se espera que los resultados sean exactos, siempre es mejor utilizar los tipos `numeric` o `decimal`.

La razón, es que los tipos guardados como datos de punto flotante tiene limitaciones en la precisión, debido a lo cuál tienen a tener una cantidad de error en los cálculos que se realizan con estos. Analizaremos el porqué en un nuevo artículo posteriormente.

A continuación, puedes ver un ejemplo de como pueden ser usados en la definición de las columnas

```
CREATE TABLE decimal_types (
  column_1 numeric(5, 2),
  column_2 numeric(5),
  column_3 numeric,
  column_4 real,
  column_5 double
);
```

Ahora bien, para los valores de `precision` y `scale` en la definición del tipo `numeric`, se tienen un par extra de posibilidades. La diferencia entre ellas, son los valores asignados por defecto estos. Cuando `scale` no ha sido especificado (p.e. `numeric(5)`), este es considerado como 0, por lo tanto la parte decimal es ignorada. Cuando `precision` tampoco es especificado, (p.e. `numeric`), `scale` permanece siendo 0, y `precision` se considera el máximo valor posible, el cual es 131072.

Para observar rápidamente las diferencias, en la siguiente tabla podrás observar como sería conservado un dato en los diferentes columnas.

Valor	`numeric(5,2)`	`numeric(5)`	`numeric`
300.279	300.28	300	300
30,000	Fuera de rango	30,000	30,000
300,000	Fuera de rango	Fuera de rango	300,000

Comparación entre las definición del tipo `numeric`

Tipos para fecha y hora

Para las fechas, tenemos varios tipos interesantes que analizaremos ahora, aunque algunos de ellos los podemos considerar como parte del tipo anterior.

Las opciones que tenemos son

- `timestamp`: que es fecha y hora, y el cuál puede incluir la zona horaria.
- `date`: solamente la fecha
- `time`: solamente la hora, y puede incluir la zona horaria
- `interval`: el cual puede guardar un intervalo de tiempo en días, horas, años, etc.

Un hecho a considerar, para el formato de fechas, es que es recomendable el utilizar el formato ISO, el cual es `YYYY-MM-DD HH:MM:SS`, aunque PostgreSQL soporta otro tipo de formatos, al hacer un query que incluya la información de estas columnas, por defecto será mostrado en el formato recomendado.

Comparación tipos de fecha y hora

A continuación podemos observar una comparativa entre los tamaños y los rangos que permiten los diferentes tipos

Tipo de dato	Tamaño en memoria	Rango
`timestamp`	8 bytes	4713 BC a 294276 AD

Tipo de dato	Tamaño en memoria	Rango
`date`	4 bytes	4713 BC a 5874891 AD
`time`	8 bytes	00:00:00 a 24:00:00
`interval`	16 bytes	+/- 178,000,000 años

Comparativa entre tipos para fecha y hora

El tipo `intervalo` no es muy común verlo usado como un tipo de una columna, como si lo es `timestamp` por ejemplo, pero cada vez que se realiza una operación entre fechas o horas, el resultado de dicha operación será del tipo `interval`.

Y como con los grupos anteriores, a continuación puedes ver pequeño ejemplo de como podrían ser usados durante la definición de una tabla.

```
CREATE TABLE time_and_hour_types (
  column_1 timestamp,
  column_2 timestamp with time zone,
  column_3 date,
  column_4 time,
  column_5 time with time zone,
  column_6 interval
);
```

El uso de las zonas horarias, será cubierto en mayor detalle en un artículo posterior ya que existen múltiples formatos en los que se pueden registrar, similar a los múltiplos formatos disponibles para las fechas.

- `2003-04-12 04:05:06 America/New_York`
- `04:05:06 PST`

Y bueno, finalmente hemos llegado a la conclusión de este artículo. Como se mencionó a lo largo del mismo, siempre es importante analizar los datos que se desean guardar con detenimiento antes de definir la tabla que se usará para ellos, sobre todo en aplicaciones dónde se espera guardar muchas filas por tabla, ya que puede ser complicado cambiar el tipo de la columna, una vez que esta está llena de información.

Tipos de datos

Postgres tiene un rico conjunto de tipos de datos nativos disponibles para los usuarios. Los usuarios pueden agregar nuevos tipos a Postgres usando el comando DEFINE TYPE.

En las siguientes secciones se analizarán el cumplimiento de los estándares SQL, los problemas de portabilidad y el uso. Algunos tipos de Postgres corresponden directamente a los tipos compatibles con SQL92. En otros casos, los tipos de datos definidos por la sintaxis SQL92 se asignan directamente a los tipos nativos de Postgres. Muchos de los tipos incorporados tienen formatos externos obvios. Sin embargo, varios tipos son exclusivos de Postgres, como las rutas abiertas y cerradas, o tienen varias posibilidades de formatos, como los tipos de fecha y hora.[1]

En este post, discutiremos sobre los tipos de datos utilizados en PostgreSQL. Al crear la tabla, para cada columna, especifica un tipo de datos, es decir, qué tipo de datos desea almacenar en los campos de la tabla.

Esto permite varios beneficios:

Consistencia
: las operaciones contra columnas del mismo tipo de datos dan resultados consistentes y generalmente son las más rápidas.

Validación
: el uso adecuado de los tipos de datos implica la validación del formato de los datos y el rechazo de los datos fuera del alcance del tipo de datos.

Compacidad
: como una columna puede almacenar un solo tipo de valor, se almacena de forma compacta.

Rendimiento
: el uso adecuado de los tipos de datos proporciona el almacenamiento de datos más eficiente. Los valores almacenados se pueden procesar rápidamente, lo que mejora el rendimiento.

Tipos numéricos

Los tipos numéricos consisten en números enteros de dos bytes, cuatro bytes y ocho bytes, números de coma flotante de cuatro y ocho bytes y decimales de precisión seleccionables. La siguiente tabla enumera los tipos disponibles.

Tipos entero

Los tipos smallint, integer y bigint almacenan números enteros, es decir, números sin componentes fraccionales, 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 de letra pequeña 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.

Tipos punto flotante

Los tipos de datos real, double y precision son tipos numéricos inexactos de precisión variable. En todas las plataformas compatibles actualmente, estos tipos son implementaciones de IEEE Standard 754 para aritmética de punto flotante binario (precisión simple y doble, respectivamente), en la medida en que el procesador, el sistema operativo y el compilador subyacentes 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. Gestionar estos los errores y la forma en que 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 montos monetarios), use el tipo numeric en su lugar.
- Si desea hacer cálculos complicados con estos tipos para algo importante, especialmente si confía en cierto comportamiento en casos límite (infinito, subflujo), debe evaluar la implementación cuidadosamente.
- La comparación de dos valores de punto flotante para la igualdad podría no funcionar siempre como se esperaba.

En todas las plataformas compatibles actualmente, el tipo real tiene un rango de alrededor de 1E-37 a 1E + 37 con una precisión de al menos 6 dígitos decimales. El tipo de doble precisión tiene un rango de alrededor de 1E-307 a 1E + 308 con una

precisión de al menos 15 dígitos. Los valores que son demasiado grandes o demasiado pequeños causarán un error. El redondeo puede tener lugar si la precisión de un número de entrada es demasiado alta. Los números demasiado cercanos a cero que no son representables como distintos de cero causarán un error de flujo insuficiente.

Por defecto, los valores de coma flotante se emiten en forma de texto en su representación decimal precisa más corta; el valor decimal producido está más cerca del verdadero valor binario almacenado que de cualquier otro valor representable en la misma precisión binaria. (Sin embargo, el valor de salida actualmente nunca está exactamente a mitad de camino entre dos valores representables, para evitar un error generalizado donde las rutinas de entrada sí lo hacen no respeta adecuadamente la regla de redondeo a par más cercano). Este valor utilizará como máximo 17 dígitos decimales significativos para los valores float8 y como máximo 9 dígitos para los valores float4.

Tipos de serie

Los tipos de datos smallserial, serial y bigserial no son tipos verdaderos, sino simplemente una conveniencia de notación para reconectar columnas de identificador único (similar a la propiedad AUTO_INCREMENT admitida por algunas otras bases de datos). En la implementación actual, especificando:

```
CREATE TABLE tablename (
 colname SERIAL
);

--Es equivalente a:

CREATE SEQUENCE tablename_colname_seq AS integer;
CREATE TABLE tablename (
 colname integer NOT NULL DEFAULT
 nextval('tablename_colname_seq')
);
ALTER SEQUENCE tablename_colname_seq OWNED BY tablename.colname;
```

C++

Copy

Por lo tanto, hemos creado una columna entera y organizada para que sus valores predeterminados se asignen desde un generador de secuencia. Se aplica una restricción NOT NULL para garantizar que no se pueda insertar un valor nulo.

(En la mayoría de los casos, también querrá adjuntar una restricción de CLAVE ÚNICA o PRIMARIA para evitar que se inserten valores duplicados por accidente, pero esto no es utomático). Por último, la secuencia se marca como "propiedad de" la columna, de modo que se descartará si se descarta la columna o tabla.

Tipos monetarios

El tipo de dinero almacena una cantidad de moneda con una precisión fraccional fija. Los valores de los tipos de datos numéricos, int y bigint se pueden convertir en dinero . No se recomienda utilizar números de punto flotante para manejar dinero debido a la posibilidad de errores de redondeo.

Nombre	Tamaño de almacenamiento	Descripción	Distancia
dinero	8 bytes	cantidad de moneda	-92233720368547758.08 a +92233720368547758

Tipos de caracteres

La siguiente tabla enumera los tipos de caracteres de uso general disponibles en PostgreSQL.

String	Nombre y descripción
1	**character varying(n), varchar(n)** Longitud variable con límite
2	**character(n), char(n)** Longitud fija, acolchada en blanco
3	**text** Longitud ilimitada variable

Tipos de datos binarios

El tipo de datos bytea permite el almacenamiento de cadenas binarias como en la tabla siguiente.

Nombre	Tamaño de almacenamiento	Descripción
bytea	1 o 4 bytes más la cadena binaria real	cadena binaria de longitud variable

Tipos de fecha / hora

PostgreSQL admite un conjunto completo de tipos de fecha y hora de SQL, como se muestra en la tabla a continuación. Las fechas se cuentan según el calendario gregoriano. Aquí, todos los tipos tienen una resolución de 1 microsegundo / 14 dígitos, excepto el tipo de fecha, cuya resolución es el día.

Nombre	Tamaño de almacenamiento	Descripción
timestamp [(p)] [without time zone]	8 bytes	Tanto la fecha como la hora (sin zona horaria)
TIMESTAMPTZ	8 bytes	Tanto la fecha como la hora, con zona horaria
date	4 bytes	Fecha (sin hora del día)
time [(p)] [without time zone]	8 bytes	Hora del día (sin fecha)
time [(p)] with time zone	12 bytes	Solo horas del día, con zona horaria
interval [fields] [(p)]	12 bytes	Intervalo de tiempo

Tipo booleano

PostgreSQL proporciona el tipo SQL estándar Boolean. El tipo de datos booleanos puede tener los estados verdadero , falso y un tercer estado, desconocido , que está representado por el valor nulo de SQL.

Nombre	Tamaño de almacenamiento	Descripción
boolean	1 byte	El estado es true o false

Tipo enumerado

Los tipos enumerados (enum) son tipos de datos que comprenden un conjunto de valores estático y ordenado. Son equivalentes a los tipos de enumeración admitidos en varios lenguajes de programación.

A diferencia de otros tipos, los tipos enumerados deben crearse utilizando el comando CREATE TYPE. Este tipo se utiliza para almacenar un conjunto de valores estático y ordenado. Por ejemplo, direcciones de la brújula, es decir, NORTE, SUR, ESTE y OESTE o días de la semana como se muestra a continuación:

```
CREATE TYPE week AS ENUM ('Mon', 'Tue', 'Wed', 'Thu', 'Fri',
'Sat', 'Sun');
```

C++

Copy

Enumerado, una vez creado, se puede usar como cualquier otro tipo.

Tipo geométrico

Los tipos de datos geométricos representan objetos espaciales bidimensionales. El tipo más fundamental, el punto, forma la base de todos los otros tipos.

Tipo de dirección de red

PostgreSQL ofrece tipos de datos para almacenar direcciones IPv4, IPv6 y MAC. Es mejor usar estos tipos en lugar de los tipos de texto sin formato para almacenar direcciones de red, porque estos tipos ofrecen verificación de errores de entrada y operadores y funciones especializados.

Nombre	Tamaño de almacenamiento	Descripción
cidr	7 or 19 bytes	IPv4 y IPv6 redes
inet	7 or 19 bytes	IPv4 y IPv6 hosts y redes
macaddr	6 bytes	MAC addresses

Tipo de cadena de bits

Los tipos de cadenas de bits se utilizan para almacenar máscaras de bits. Son 0 o 1. Hay dos tipos de bits SQL: bit (n) y bit variable (n) , donde n es un entero positivo.

Tipo de búsqueda de texto

Este tipo admite la búsqueda de texto completo, que es la actividad de buscar en una colección de documentos en lenguaje natural para localizar los que mejor se ajustan a una consulta. Hay dos tipos de datos para esto:

Número deString	Nombre y descripción
1	**tsvector** Esta es una lista ordenada de palabras distintas que se han normalizado para fusionar diferentes variantes de la misma palabra, llamadas "lexemas".
2	**tsquery** Esto almacena los lexemas que se deben buscar y los combina en honor a los operadores booleanos y (AND), | (OR y ! (NOT). Los paréntesis se pueden usar para imponer la agrupación de los operadores.

Esto almacena los lexemas que se deben buscar y los combina en honor a los operadores booleanos & (AND), | (O y ! (NO). Los paréntesis se pueden usar para imponer la agrupación de los operadores.

Tipo de UUID

Un UUID (identificadores únicos universales) se escribe como una secuencia de dígitos hexadecimales en minúsculas, en varios grupos separados por guiones, específicamente un grupo de ocho dígitos, seguido de tres grupos de cuatro dígitos, seguidos por un grupo de 12 dígitos, para un total de 32 dígitos que representan los 128 bits.

Un ejemplo de UUID es: 550e8400-e29b-41d4-a716-446655440000

Tipo XML

El tipo de datos XML se puede usar para almacenar datos XML. Para almacenar datos XML, primero debe crear valores XML utilizando la función xmlparse de la siguiente manera:

```
XMLPARSE (DOCUMENT '<?xml version="1.0"?>
    <tutorial>
        <title>Código Electrónica</title>
    <topics>...</topics>
    </tutorial>')
XMLPARSE (CONTENT 'xyz<foo>bar</foo><bar>foo</bar>')
```

C++

Tipo JSON

El tipo de datos json se puede usar para almacenar datos JSON (JavaScript Object Notation). Dichos datos también se pueden almacenar como texto , pero el tipo de datos json tiene la ventaja de verificar que cada valor almacenado es un valor JSON válido. También hay disponibles funciones de soporte relacionadas, que se pueden usar directamente para manejar el tipo de datos JSON de la siguiente manera.

Ejemplo	
array_to_json('{{1,5},{99,100}}'::int[])	[[1,5],[99,1
row_to_json(row(1,'foo'))	{"f1":1,"f2"

Tipo de matriz

PostgreSQL ofrece la oportunidad de definir una columna de una tabla como una matriz multidimensional de longitud variable. Se pueden crear matrices de cualquier tipo base incorporado o definido por el usuario, tipo de enumeración o tipo compuesto.

Declaración de matrices

El tipo de matriz se puede declarar como

```
CREATE TABLE monthly_savings (
   name text,
   saving_per_quarter integer[],
   scheme text[][]
);
o usando la palabra clave "ARRAY" como

CREATE TABLE monthly_savings (
   name text,
   saving_per_quarter integer ARRAY[4],
   scheme text[][]
);
```

C++

Insertar valores

Los valores de matriz se pueden insertar como una constante literal, encerrando los valores de los elementos entre llaves y separándolos por comas. A continuación se muestra un ejemplo:

```
INSERT INTO monthly_savings
VALUES ('Manisha',
'{20000, 14600, 23500, 13250}',
'{{"FD", "MF"}, {"FD", "Property"}}');
```

C++

Acceso a matrices

A continuación se muestra un ejemplo para acceder a las matrices. El comando dado a continuación seleccionará a las personas cuyos ahorros son más en el segundo trimestre que en el cuarto trimestre.

```
SELECT name FROM monhly_savings WHERE saving_per_quarter[2] >
saving_per_quarter[4];
```

C++

Modificación de matrices

Un ejemplo de modificación de matrices es como se muestra a continuación.

```
UPDATE monthly_savings SET saving_per_quarter =
'{25000,25000,27000,27000}'
WHERE name = 'Manisha';
o usando la sintaxis de expresión ARRAY -


UPDATE monthly_savings SET saving_per_quarter =
ARRAY[25000,25000,27000,27000]
WHERE name = 'Manisha';
```

C++

Copy

Buscando matrices

Un ejemplo de búsqueda de matrices es el que se muestra a continuación.

```
SELECT * FROM monthly_savings WHERE saving_per_quarter[1] = 10000
OR
saving_per_quarter[2] = 10000 OR
saving_per_quarter[3] = 10000 OR
saving_per_quarter[4] = 10000;
```

C++

Si se conoce el tamaño de la matriz, se puede utilizar el método de búsqueda proporcionado anteriormente. De lo contrario, el siguiente ejemplo muestra cómo buscar cuando no se conoce el tamaño.

```
SELECT * FROM monthly_savings WHERE 10000 = ANY
(saving_per_quarter);
```

C++

Tipos compuestos

Este tipo representa una lista de nombres de campo y sus tipos de datos, es decir, la estructura de una fila o registro de una tabla.

Declaración de tipos compuestos

El siguiente ejemplo muestra cómo declarar un tipo compuesto

```
CREATE TYPE inventory_item AS (
```

```
    name text,
    supplier_id integer,
    price numeric
);
```

C++

Este tipo de datos se puede usar en las tablas de creación de la siguiente manera:

```
CREATE TABLE on_hand (
    item inventory_item,
    count integer
);
```

C++

Copy

Entrada de valor compuesto

Los valores compuestos se pueden insertar como una constante literal, encerrando los valores de campo entre paréntesis y separándolos por comas. A continuación se muestra un ejemplo:

```
INSERT INTO on_hand VALUES (ROW('fuzzy dice', 42, 1.99), 1000);
```

C++

Esto es válido para el elemento de inventario definido anteriormente. La palabra clave ROW es realmente opcional siempre que tenga más de un campo en la expresión.

Acceso a tipos compuestos

Para acceder a un campo de una columna compuesta, use un punto seguido del nombre del campo, al igual que seleccionar un campo de un nombre de tabla. Por ejemplo, para seleccionar algunos subcampos de nuestra tabla de ejemplo on_hand, la consulta sería como se muestra a continuación:

```
SELECT (item).name FROM on_hand WHERE (item).price > 9.99;
```

C++

Incluso puede usar el nombre de la tabla también (por ejemplo, en una consulta multitarea), de esta manera:

```
SELECT (on_hand.item).name FROM on_hand WHERE (on_hand.item).price
> 9.99;
```

C++

Tipos de rango

Los tipos de rango representan tipos de datos que usan un rango de datos. El tipo de rango puede ser rangos discretos (por ejemplo, todos los valores enteros del 1 al 10) o rangos continuos (por ejemplo, cualquier punto en el tiempo entre las 10:00 a.m. y las 11:00 a.m.).

Los tipos de rango integrados disponibles incluyen los siguientes rangos:

- int4range - Rango de enteros
- int8range - Rango de bigint
- numrange - Rango de numérico
- tsrange - Rango de marca de tiempo sin zona horaria
- tstzrange - Rango de marca de tiempo con zona horaria
- daterange - Gama de la fecha

Se pueden crear tipos de rango personalizados para hacer que estén disponibles nuevos tipos de rangos, como los rangos de direcciones IP que usan el tipo de entrada como base, o los rangos flotantes que usan el tipo de datos flotante como base.

Los tipos de rango admiten límites de rango inclusivos y exclusivos utilizando los caracteres [] y (), respectivamente. Por ejemplo, '[4,9)' representa todos los enteros que comienzan desde 4 e incluyen 4 hasta, pero no incluyen 9.

BIBLIOGRAFIA:

- "Postgresql tipo de datos | CodigoElectronica". Inicio | CódigoElectrónica. http://codigoelectronica.com/blog/postgresql-tipo-de-datos (accedido el 24 de marzo de 2023).