39.10. PL / pgSQL debajo del capó

Esta sección analiza algunos detalles de implementación que con frecuencia son importantes para que los usuarios de PL / pgSQL pueda saber.

39.10.1. Sustitución Variable

Las declaraciones y expresiones SQL dentro de una función PL / pgSQL pueden referirse a variables y parámetros de función. Detrás de escena, PL / pgSQL sustituye parámetros de consulta para tales referencias. Los parámetros solo se sustituirá en lugares donde se permita sintácticamente una referencia de parámetro o columna. Como un caso extremo, considere este ejemplo de estilo de programación pobre:

```
INSERTAR EN LOS VALORES foo (foo) (foo);
```

La primera aparición de foo debe ser sintácticamente un nombre de tabla, por lo que no se sustituirá, incluso si el La función tiene una variable llamada foo. La segunda aparición debe ser el nombre de una columna de la tabla, por lo que tampoco será sustituido. Solo la tercera aparición es un candidato para ser una referencia a la función variable.

> Nota: las versiones de PostgreSQL anteriores a 9.0 intentarían sustituir la variable en los tres casos, lo que lleva a errores de sintaxis.

Dado que los nombres de las variables no son sintácticamente diferentes de los nombres de las columnas de la tabla, puede haber ambigüedad en las declaraciones que también se refieren a tablas: es un nombre de pila destinado a referirse a una columna de tabla, o ur ¿variable? Cambiemos el ejemplo anterior a

```
INSERTAR EN dest (col) SELECCIONE foo + bar DESDE src;
```

Aquí, dest y src deben ser nombres de tabla, y col debe ser una columna de dest, pero foo y bar podrían razonablemente ser variables de la función o columnas de src.

Por defecto, PL / pgSQL informará un error si un nombre en una declaración SQL puede referirse a una variable o a un columna de tabla Puede resolver este problema cambiando el nombre de la variable o columna, o calificando referencia ambigua, o diciéndole a PL / pgSQL qué interpretación preferir.

La solución más simple es cambiar el nombre de la variable o columna. Una regla de codificación común es usar un diferente convención de nomenclatura para las variables PL / pgSQL que la que usa para los nombres de columna. Por ejemplo, si tu constantemente nombrar variables de función v_ ***algo*** mientras que ninguno de los nombres de sus columnas comienza con v_, no se producirán conflictos.

https://www.postgresql.org/docs/9.1/plpgsql-implementation.html

Página 2

Alternativamente, puede calificar referencias ambiguas para aclararlas. En el ejemplo anterior, src.foo sería una referencia inequívoca a la columna de la tabla. Para crear una referencia inequívoca a un variable, declararlo en un bloque etiquetado y usar la etiqueta del bloque (ver Sección 39.2) Por ejemplo,

```
<<bloque>>
DECLARAR
        foo int;
EMPEZAR
        foo: = ...;
        INSERTAR EN dest (col) SELECCIONAR block.foo + bar DESDE src;
```

Aquí block.foo significa la variable incluso si hay una columna foo en src. Parámetros de función, así como Las variables especiales como FOUND pueden calificarse por el nombre de la función, porque están implícitamente declarado en un bloque externo etiquetado con el nombre de la función.

A veces no es práctico x todas las referencias ambiguas en un gran cuerpo de código PL / pgSQL. De tal En los casos, puede especificar que PL / pgSQL debe resolver referencias ambiguas como la variable (que es compatible con el comportamiento de PL / pgSQL antes de PostgreSQL 9.0), o como la columna de la tabla (que es compatible con algunos otros sistemas como Oracle).

Para cambiar este comportamiento en todo el sistema, establezca el parámetro de configuración plpgsql.variable_conflict a uno de error, use_variable o use_column (donde error es el predeterminado de fábrica). Este parámetro afecta las compilaciones posteriores de declaraciones en funciones PL / pgSQL, pero no declaraciones ya compiladas en la sesión actual. Para establecer el parámetro antes de que PL / pgSQL haya sido cargado, es necesario haber agregado "plpgsql" a la lista custom_variable_classes en postgresql.conf. Debido a que cambiar esta configuración puede causar cambios inesperados en el comportamiento de las funciones PL / pgSQL, puede solo puede ser cambiado por un superusuario.

También puede establecer el comportamiento función por función, insertando uno de estos comandos especiales al comienzo del texto de la función:

```
#variable_conflict error
#variable_conflict use_variable
#variable_conflict use_column
```

Estos comandos solo afectan la función en la que están escritos y anulan la configuración de plpgsql.variable_conflict. Un ejemplo es

```
CREAR FUNCIÓN stamp_user (id int, texto del comentario) DEVOLUCIONES nulo AS $$
    #variable_conflict use_variable
    DECLARAR
        marca de tiempo de tiempo de corte: = ahora ();
    EMPEZAR
        ACTUALIZAR usuarios SET last_modified = curtime, comment = comment
          DONDE users.id = id;
    FIN;
$$ IDIOMA plpgsql;
```

En el comando UPDATE, curtime, comment e id se referirán a las variables y parámetros de la función. si los usuarios tienen o no columnas de esos nombres. Tenga en cuenta que tuvimos que calificar la referencia a users.id en la cláusula WHERE para que haga referencia a la columna de la tabla. Pero no tuvimos que calificar el referencia para comentar como un objetivo en la lista ACTUALIZACIÓN, porque sintácticamente debe ser una columna de usuarios. Podríamos escribir la misma función sin depender de la configuración variable_conflict de esta manera:

https://www.postgresql.org/docs/9.1/plpgsql-implementation.html


```
CREAR FUNCIÓN stamp_user (id int, texto del comentario) DEVOLUCIONES nulo AS $$
    <<fn>>
    DECLARAR
        marca de tiempo de tiempo de corte: = ahora ();
    EMPEZAR
        ACTUALIZAR usuarios SET last_modified = fn.curtime, comment =
stamp_user.comment
          DONDE users.id = stamp_user.id;
    FIN;
$$ IDIOMA plpgsql;
```

La sustitución de variables no ocurre en la cadena de comando dada a EJECUTAR o una de sus variantes. Si debe insertar un valor variable en dicho comando, hacerlo como parte de la construcción del valor de cadena, o use USING, como se ilustra en la Sección 39.5.4 .

La sustitución de variables actualmente solo funciona en los comandos SELECT, INSERT, UPDATE y DELETE, porque El motor SQL principal permite parámetros de consulta solo en estos comandos. Para usar un nombre no constante o valor en otros tipos de instrucciones (genéricamente llamadas declaraciones de utilidad), debe construir la utilidad declaración como una cadena y EJECUTARLO.

39.10.2. Planificar el almacenamiento en caché

El intérprete PL / pgSQL analiza el texto fuente de la función y produce un árbol de instrucciones binarias internas. la primera vez que se llama a la función (dentro de cada sesión). El árbol de instrucciones traduce completamente el PL / pgSQL estructura de la declaración, pero las expresiones SQL individuales y los comandos SQL utilizados en la función no son traducido de inmediato.

Como cada expresión y comando SQL se ejecuta primero en la función, el intérprete PL / pgSQL crea un plan de ejecución preparado (utilizando las funciones SPI_prepare y SPI_saveplan del administrador SPI). Las visitas posteriores a esa expresión o comando reutilizan el plan preparado. Por lo tanto, una función con el código condicional que contiene muchas declaraciones para las cuales se pueden requerir planes de ejecución solo prepare y guarde los planes que realmente se usan durante la vida útil de la conexión de la base de datos. Esto puede reducir sustancialmente la cantidad de tiempo total requerida para analizar y generar planes de ejecución para declaraciones en una función PL / pgSQL. Una desventaja es que los errores en una expresión o comando específico no se puede detectar hasta que se alcance esa parte de la función en ejecución. (Los errores de sintaxis triviales serán detectado durante el paso de análisis inicial, pero no se detectará nada más profundo hasta la ejecución).

Un plan guardado se volverá a planificar automáticamente si hay algún cambio de esquema en cualquier tabla utilizada en la consulta, o si alguna función definida por el usuario utilizada en la consulta se redefine. Esto hace que la reutilización de los planes preparados transparente en la mayoría de los casos, pero hay casos de esquina en los que un plan obsoleto podría reutilizarse. Un ejemplo es que eliminar y volver a crear un operador definido por el usuario no afectará los planes ya almacenados en caché; ellos continuarán llame a la función subyacente del operador original, si eso no se ha cambiado. Cuando sea necesario, el caché se puede usar iniciando una nueva sesión de base de datos.

Debido a que PL / pgSQL guarda los planes de ejecución de esta manera, los comandos SQL que aparecen directamente en un PL / pgSC la función debe referirse a las mismas tablas y columnas en cada ejecución; es decir, no puedes usar un parámetro como el nombre de una tabla o columna en un comando SQL. Para evitar esta restricción, puede construir comandos dinámicos utilizando la instrucción PL / pgSQL EXECUTE, al precio de construir un nuevo plan de ejecución en cada ejecución.

Otro punto importante es que los planes preparados están parametrizados para permitir los valores de PL / pgSQL variables para cambiar de un uso a otro, como se discutió en detalle anteriormente. A veces esto significa que un el plan es menos eficiente de lo que sería si se generara para un valor variable específico. Como ejemplo, considere

https://www.postgresql.org/docs/9.1/plpgsql-implementation.html

23/10/2019 PostgreSQL: Documentación: 9.1: PL / pgSQL Under the Hood

Página 4

SELECCIONE * EN myrec DESDE el diccionario DONDE la palabra ME GUSTA search_term;

donde search_term es una variable PL / pgSQL. El plan en caché para esta consulta nunca usará un índice en palabra, ya que el planificador no puede asumir que el patrón LIKE estará anclado a la izquierda en el tiempo de ejecución. Para usar un indexar la consulta debe planificarse con un patrón LIKE constante constante proporcionado. Esta es otra situacion donde EJECUTAR se puede usar para forzar un nuevo plan que se generará para cada ejecución.

La naturaleza mutable de las variables de registro presenta otro problema a este respecto. Cuando los campos de un las variables de registro se usan en expresiones o declaraciones, los tipos de datos de los campos no deben cambiar de

una llamada de la función a la siguiente, ya que cada expresión se planificará utilizando el tipo de datos que es presente cuando la expresión se alcanza por primera vez. EJECUTAR puede usarse para solucionar este problema cuando necesario.

Si se usa la misma función como desencadenante para más de una tabla, PL / pgSQL prepara y almacena en caché los planes independientemente para cada una de esas tablas, es decir, hay una caché para cada función de disparo y tabla combinación, no solo para cada función. Esto alivia algunos de los problemas con diferentes tipos de datos; para Por ejemplo, una función de disparo podrá funcionar correctamente con una columna llamada clave incluso si sucede tener diferentes tipos en diferentes tablas.

Del mismo modo, las funciones que tienen tipos de argumentos polimórficos tienen un caché de plan separado para cada combinación de tipos de argumentos reales para los que se han invocado, de modo que las diferencias de tipos de datos no causen fallas inesperadas

El almacenamiento en caché del plan a veces puede tener efectos sorprendentes en la interpretación de valores sensibles al tiempo. por Por ejemplo, hay una diferencia entre lo que hacen estas dos funciones:

```
CREAR FUNCIÓN logfunc1 (texto logtxt) DEVUELVE nulo AS $$
        EMPEZAR
                INSERTAR EN VALORES registrables (logtxt, 'ahora');
        FIN;
$$ IDIOMA plpgsql;
```

y:

```
CREAR FUNCIÓN logfunc2 (texto logtxt) DEVUELVE nulo AS $$
        DECLARAR
                marca de tiempo de tiempo de corte;
        EMPEZAR
                curtime: = 'ahora';
                INSERTAR EN VALORES registrables (logtxt, curtime);
        FIN;
$$ IDIOMA plpgsql;
```

En el caso de logfunc1, el analizador principal de PostgreSQL sabe al preparar el plan para INSERTAR que la cadena 'ahora' debe interpretarse como marca de tiempo, porque la columna de destino de logtable es de ese tipo. Por lo tanto, 'ahora' se convertirá en una constante cuando se planea INSERTAR, y luego se usará en todos invocaciones de logfunc1 durante la duración de la sesión. No hace falta decir que esto no es lo que el programador querido.

En el caso de logfunc2, el analizador principal de PostgreSQL no sabe en qué tipo debería convertirse "ahora" y por lo tanto, devuelve un valor de datos de tipo texto que contiene la cadena ahora. Durante la asignación subsiguiente a la variable local curtime, el intérprete PL / pgSQL convierte esta cadena al tipo de marca de tiempo llamando al funciones text_out y timestamp_in para la conversión. Entonces, la marca de tiempo calculada se actualiza en cada ejecución como el programador espera.

https://www.postgresql.org/docs/9.1/plpgsql-implementation.html