**Capítulo 10**

En este capítulo, vamos a examinar algunas de las formas en las que podemos ampliar y mejorar la características de PostgreSQL introducidos hasta ahora. Gran parte del material de este capítulo es específico de PostgreSQL, aunque la mayoría de los sistemas de gestión de bases de datos relacionales comerciales, tales como Oracle, incluye características similares

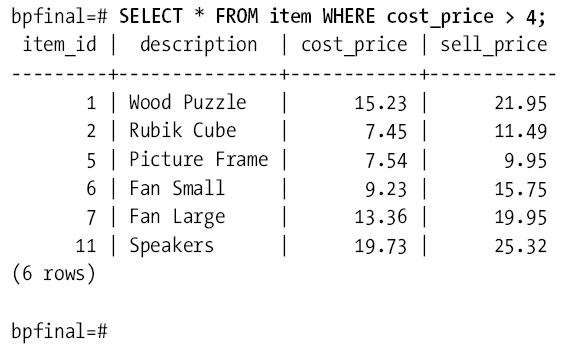
Para empezar, vamos a ver más de los operadores que PostgreSQL soporta dentro SELECT   
declaraciones, incluyendo a juego avanzado y operadores matemáticos que nos permiten construir pruebas sofisticadas cláusulas where. A continuación, vamos a ver cómo los operadores se implementan como funciones en PostgreSQL y mirar a unas pocas funciones adicionales que se suman a la fuerza expresiva de nuestra sentencias SELECT

PostgreSQL permite a un desarrollador para extender la funcionalidad de servidor de base de datos mediante la creación de nuevos funciones utilizando el lenguaje de programación C y cargarlos en el servidor cuando la base de datos se pone en marcha. Una extensión puede ser tan simple como una sola función adicional o tan complejo como una completa lenguaje de programación en su propio derecho. Varias de estas extensiones, conocidos como cargable procesal idiomas, están incluidos en la distribución estándar de PostgreSQL. Estas lenguas nos permiten para crear nuestras propias funciones, conocida como procedimientos almacenados, de forma rápida y más fácilmente que la escritura en C. Vamos a echar un breve vistazo a uno de los idiomas que se pueden cargar, PL / pgSQL, en este capítulo. PL / pgSQL es Idiomas-PostgreSQL específico, pero similar están disponibles en otras bases de datos. Por ejemplo, Oracle tiene PL / SQL y Sybase tiene Transact-SQL.

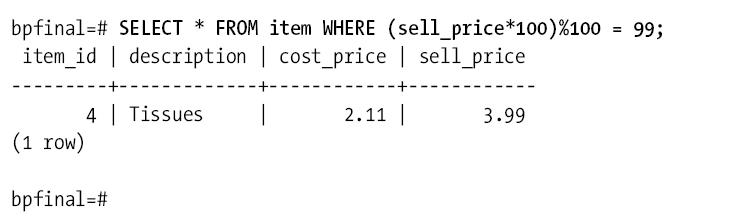
Los procedimientos almacenados también se pueden ejecutar de forma automática por el servidor PostgreSQL cuando condiciones particulares surgen dentro de la base de datos. Por ejemplo, cuando una eliminación de una fila de una tabla en grado de tentativa, de un procedimiento almacenado puede ejecutar para exigir la integridad referencial mediante la supresión de filas relacionados en otras tablas, o tal vez la prevención de la supresión que se produzcan. Estos autónoma acciones se conocen como factores desencadenantes, y vamos a verlos en acción aquí también

En este capítulo, vamos a cubrir los siguientes temas:   
• Operadores PostgreSQL   
• PostgreSQL funciones incorporadas   
• Lenguajes de procedimiento, específicamente PL / pgSQL   
• Los procedimientos almacenados   
• Funciones de SQL   
• Los disparadores

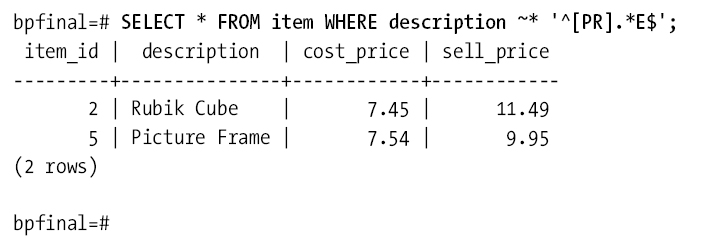
Operadores   
Ya hemos visto y utilizado algunos operadores simples instrucciones SELECT, a partir de   
Capítulo 4. Por ejemplo, podemos utilizar un operador de comparación numérica para limitar una selección de filas que obedecen a una condición, como los elementos que tienen un mayor precio de costo de $ 4



Aquí, el operador> se aplica entre el atributo cost\_price y un número dado.   
Podemos ir más allá e incluir otros atributos y operadores para crear más compleja   
condiciones:



Aquí, hemos utilizado el operador de multiplicación en conjunción con un resto a la lista de la   
elementos que tienen un precio de venta que termina en 0,99.   
También podemos utilizar los operadores para realizar búsquedas de coincidencias de expresiones regulares:

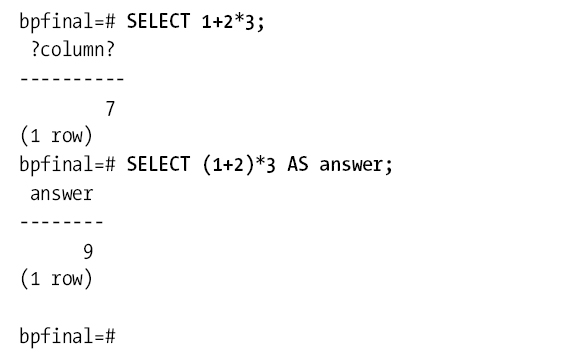


En este caso, el operador realiza una coincidencia de expresión regular entre mayúsculas y minúsculas para descubrir qué   
artículos tienen descripciones que empiezan con una minúscula o mayúscula p o r y terminan con e

Como puede ver, hay muchos operadores soportados por PostgreSQL. De hecho, si se cuenta el diferentes variaciones de un mismo operador (es decir, consideran que la comparación de números enteros como un sistema distinto de la comparación de números de punto flotante o la comparación de cadenas), hay alrededor de 600 operadores disponible

**Precedencia y asociatividad de operadores**

Muchos de los operadores de PostgreSQL se ven y actúan muy parecido a los operadores aritméticos normales que se encuentra en muchos lenguajes de programación. Los operadores tienen una prioridad modificable en el analizador que determina el orden en el que los operadores se ejecutan en el compuesto expresiones. Como de costumbre, la precedencia se puede reemplazar el uso de paréntesis. PostgreSQL permite el uso de operadores y funciones fuera de las cláusulas WHERE de SELECT declaraciones:



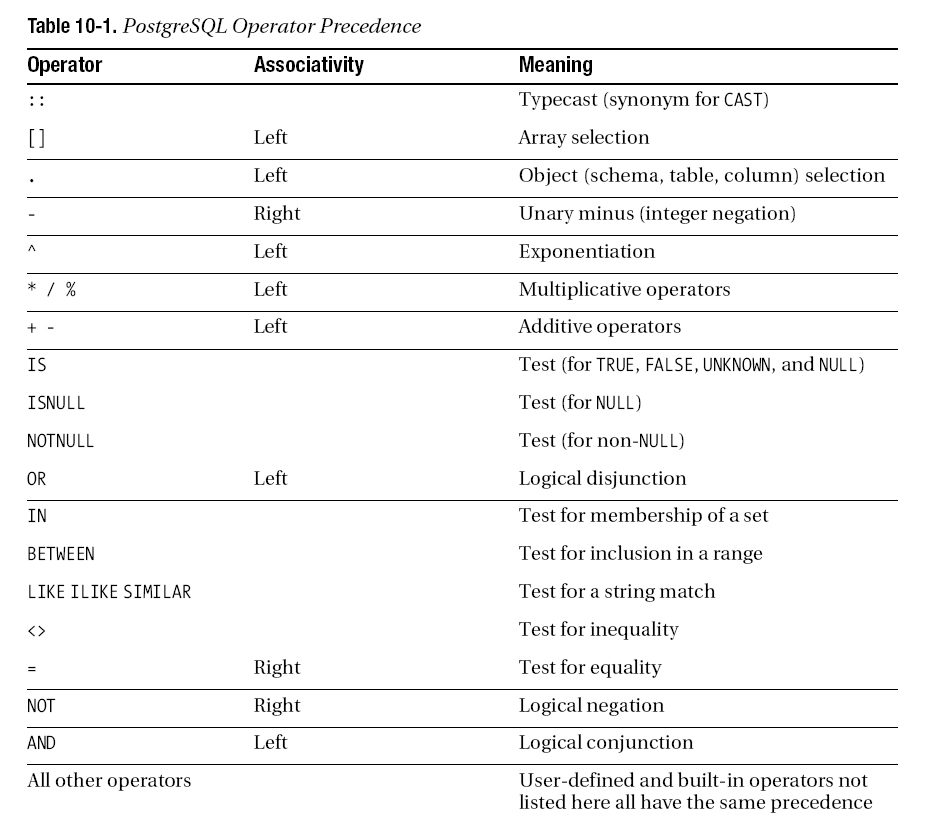
Aquí, podemos ver que el resultado de la expresión 1 +2 \* 3 se reporta como 7, muestra como un desconocido columna y dado el nombre de la columna? de forma predeterminada. En el segundo ejemplo, la prioridad de los operadores se anula y el resultado es nombrado como respuesta

Aunque la Mayoría de los Operadores en sí comportan exactamente de como Cabria Esperar Si Se ha Programado en C o Cualquier Otro Lenguaje de programación, ALGUNAS de las reglas de Prioridad de Operador Puede Ser Sorprendente.

Al Igual Que en C, los Operadores booleanos Tienen Una Menor precedencia Que El los Operadores aritméticos, Paréntesis ASI una Menudo sí require párrafo Obtener la Orden de ejecución celebra del Operador Deseado. En Caso de Duda, realizar el Pedido Explicita estafa Paréntesis

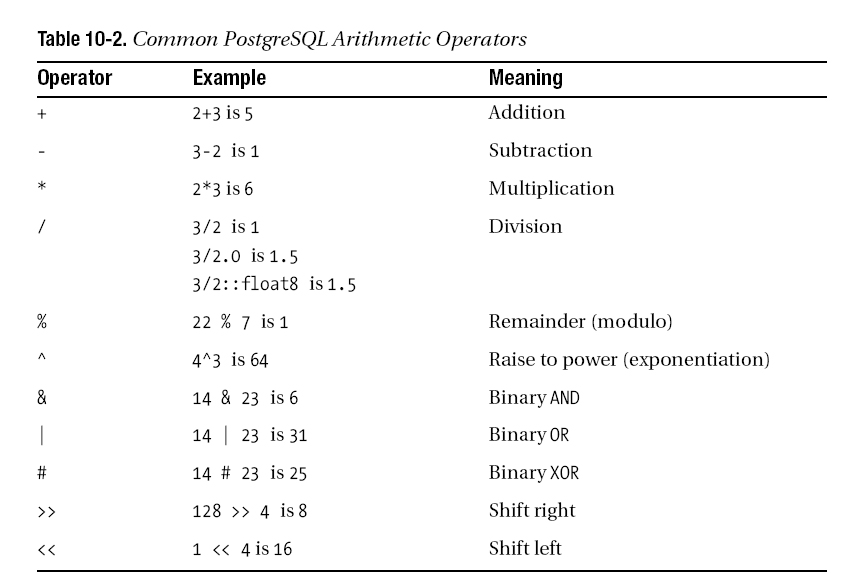
Operadores de PostgreSQL también muestran la asociatividad, ya sea a la derecha oa la izquierda, que determina la   
orden en que se evalúan los operadores de la misma precedencia. Los operadores aritméticos como suma y resta se dejan asociativa, por lo que 1 2-3 evalúa como si hubiera sido escrito (1 2) -3. Otros, como el operador de igualdad de Boole, son asociativos derecha, de modo que x = y = z se evalúa como x = (y = z).

Tabla 10-1 lista la precedencia léxica (en orden descendente) de los más comunes operadores de PostgreSQL

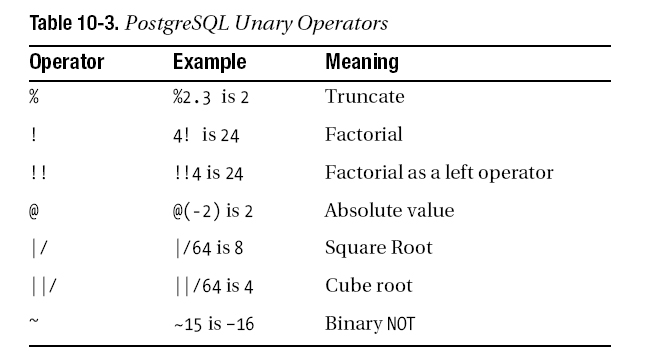


**Nota** Las versiones anteriores de los operadores de PostgreSQL admitidos para el cálculo de los logaritmos naturales y antilogaritmos (: Y;), pero éstos son obsoletos y que se han retirado de las últimas versiones de PostgreSQL, incluyendo 8.0. Utilice las funciones ln y exp instead.Table 10-1 lista la precedencia léxica (en orden descendente) de los más comunes   
operadores de PostgreSQL

**Operadores aritméticos**PostgreSQL proporciona una gama de operadores aritméticos. Los operadores aritméticos más comunes se enumeran en la Tabla 10-2. Todos ellos tienen la misma precedencia y son asociativos por la izquierda



También hay un número de operadores aritméticos unarios, se muestra en la Tabla 10-3

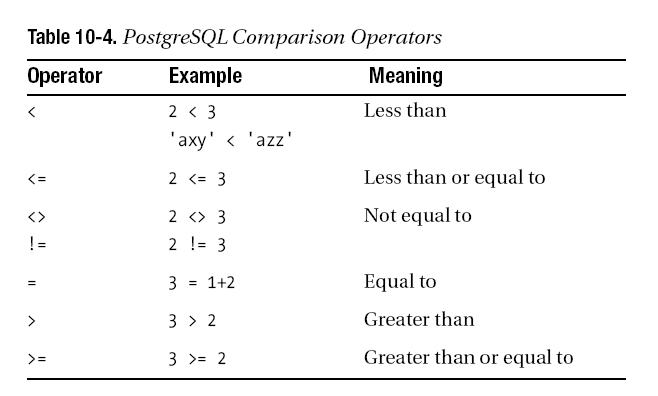


En general, los operadores aritméticos funcionan como deberían. PostgreSQL usará la versión del operador que coincide con el argumento utilizado. Por lo tanto, cuando se divide un número entero por otro, obtendrá un número entero que resulta. Cuando se divide un número de punto flotante por otro, se quiere obtener un resultado de coma flotante. Como se muestra en la Tabla 10-2, para forzar un resultado de coma flotante, uno de los argumentos deben ser emitidos como un número de punto flotante

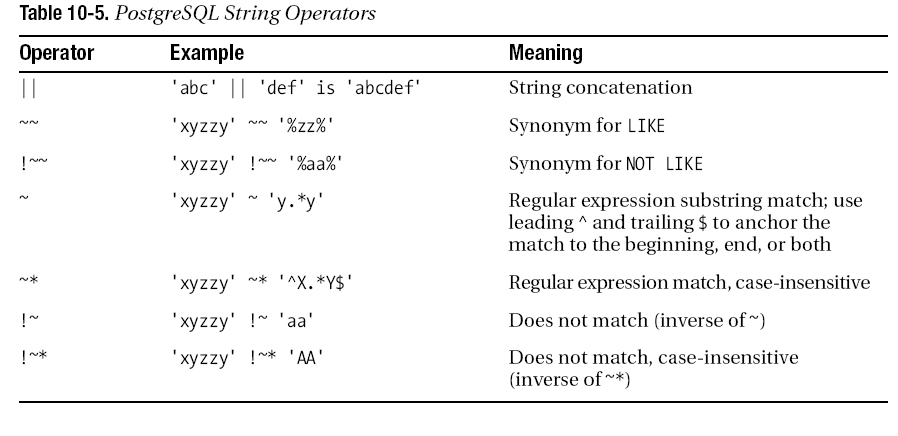
**Comparación y Operadores de Cadena**

PostgreSQL proporciona la variedad usual de operadores de comparación, como menor que y GreaterThan. Estos operadores trabajan en muchos de los tipos que PostgreSQL soporta, así que por ejemplo, se puede utilizar el operador mayor que para la prueba de orden alfabético la cadena, así como en relación tamaños de valores numéricos

El resultado de un operador de comparación es verdadera o falsa, que psql mostrará como t o f. Un listado de los operadores de comparación disponibles se presentan en la Tabla 10-4.



Cuerdas tienen su propio conjunto de operadores de PostgreSQL. Hay operadores para concatenar cadenas y para las cadenas coincidentes de acuerdo con diversas normas. Ellos se resumen en la Tabla 10-5



En una coincidencia de expresión regular, una cadena se compara con una expresión similar a la utilizada en la utilidad grep de UNIX o los operadores de los partidos de Perl

**Otros operadores**

PostgreSQL soporta una gran cantidad de operadores adicionales para comparar y manipular el Tipos de datos PostgreSQL específicos tales como puntos, círculos, intervalos de tiempo, y las direcciones IP. para más información, consulte la Sección II de la documentación de PostgreSQL

Documentación del Consejo El PostgreSQL se puede instalar como un conjunto de páginas HTML que se puede ver de forma local   
con cualquier navegador web. Select file :/ / usr / local / pgsql / doc / html / index.html o visite el sitio <http://www.postgresql.org>.

Todos los operadores se listan en la tabla pg\_operator de la base de datos, y psql puede enumerar todos de los operadores y funciones con la \ hacer y comandos internos df \

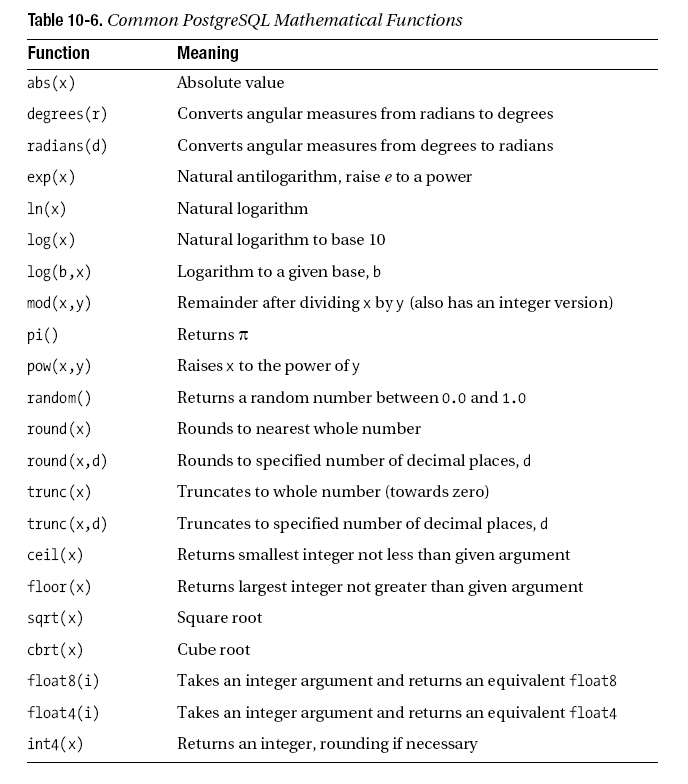
**Funciones Integradas**

PostgreSQL disfruta de una larga lista de funciones integradas que podemos utilizar en las expresiones SELECT. Un resumen por categorías de las funciones disponibles del siguiente modo:   
• Los equivalentes funcionales a los operadores que se presentan en la sección anterior   
• Otras funciones matemáticas   
• Otras funciones para el manejo de cadenas de caracteres   
• Funciones para el manejo de fechas y horas   
• Las funciones de formato de texto   
• Funciones para los tipos geométricos PostgreSQL como punto y el círculo   
• Las funciones de la dirección IP

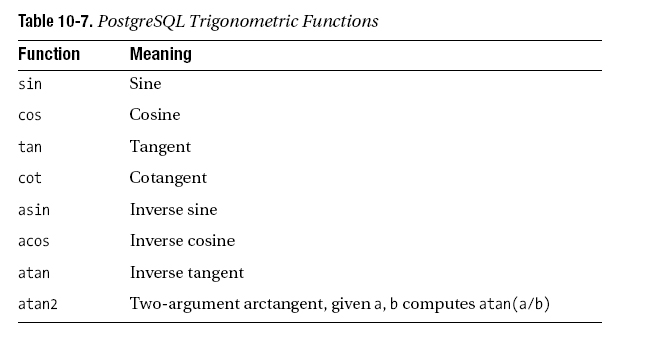
Las funciones integradas (y los definidos por el usuario) se registran en una tabla del sistema de la Base de datos PostgreSQL, pg\_proc. Al PostgreSQL versión 8.0 y posteriores, esta tabla tiene más de 1.700 entradas

En psql, puede hacer una lista de todas las funciones y sus argumentos con el comando \ df. además, comentarios acerca de cualquiera de las funciones o grupo de funciones se pueden visualizar utilizando la \ comando dd

Muchas funciones integradas proporcionan una función equivalente para cada matemática y lógica operador. Los ejemplos incluyen int4shl y float8mul como el equivalente del operador de desplazamiento a la izquierda (<<) Para los números enteros y el operador de multiplicación (\*) para los valores de punto flotante. adicional matemática funciones se enumeran en la Tabla 10-6, todas las cuales operan con números en coma flotante y devolver un número de punto flotante, a menos que se indique lo contrario

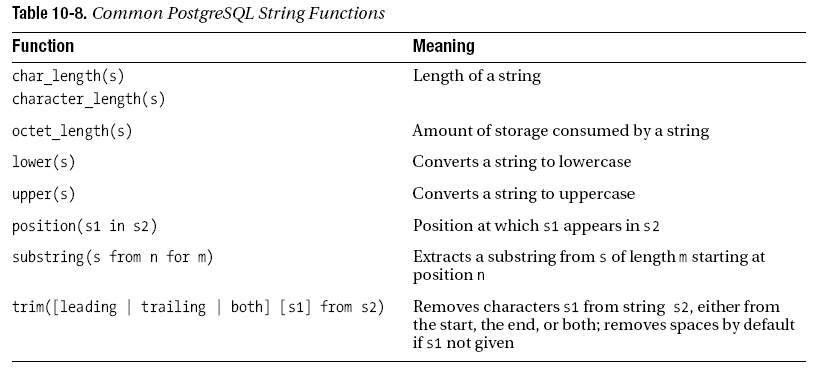
****

Las funciones trigonométricas son compatibles, como se indica en la Tabla 10-7. Todos los argumentos angulares y los resultados son en radianes

****

PostgreSQL incluye las funciones de cadena SQL estándar, con su propia sintaxis. Estas funciones se presentan en la Tabla 10-8. Para estas funciones, una cadena puede ser de tipo char, varchar o text.

PostgreSQL extiende las características de manipulación de cadenas con funciones adicionales propios. referirse a la documentación de PostgreSQL para obtener más información

****

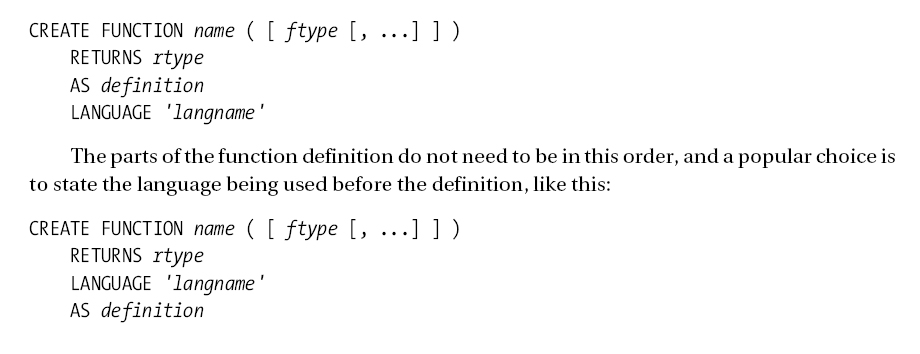
Una función importante de formato pena mencionar aquí es la función to\_char. desempeña el mismo papel en PostgreSQL que printf hace en C, la manipulación de todo tipo de valores de formato para la impresión o la visualización. Se dará formato a un valor de fecha y hora de acuerdo a una plantilla de la fecha, y puede formatear los valores numéricos de muchas maneras diferentes (incluyendo los números romanos).

Nota Para obtener más información sobre todas las funciones de PostgreSQL, consulte la documentación de PostgreSQL o navegar por la pruebas de regresión distribuidos con el código fuente de PostgreSQL.

**Idiomas del procedimiento**

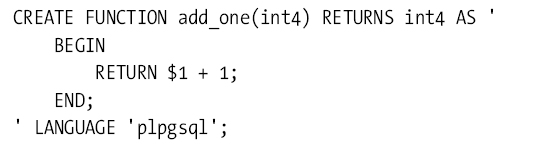
Como se mencionó en la introducción capítulo, es posible definir nuestras propias funciones para uso dentro de una base de datos PostgreSQL. Esto es útil cuando queremos capturar un cálculo en particular o consultar y volver a utilizarlo en un número de lugares.

El SQL necesario para crear una nueva función es CREATE FUNCTION, la cual tendrá las siguientes sintaxis

****

Nota: no hay otra forma de CREATE FUNCTION que permite que el código objeto compilado que se incorpore en el servidor PostgreSQL, por lo general creado a partir de código fuente C.

Una función muy simple que incrementa su único argumento podría escribirse así

****

La definición de la función se administra en una sola cadena que puede abarcar varias líneas y puede estar escritos en cualquier lenguaje soportado por PostgreSQL como lengua de procedimiento que puede cargarse. En este caso, PL / pgSQL se utiliza, como se indica por la cláusula LANGUAGE especificando plpgsql. PL / pgSQL es un lenguaje de programación desarrollado específicamente para procedimientos almacenados de programación para   
PostgreSQL

Nota Los lenguajes de procedimientos que se pueden cargar PL / Tcl, PL / Perl y PL / Python permiten crear PostgreSQL extensiones en el Tcl, Perl, Python y los lenguajes de programación, respectivamente.

Para manejar un lenguaje de procedimientos, PostgreSQL primero debe ampliarse mediante la inclusión de un controlador función normalmente escrito en C. Por PL / pgSQL, un controlador se incluye en la distribución como biblioteca compartida.   
 Cuando se crea una función, su definición se almacena dentro de la base de datos. Cuando la función es llamado por primera vez, la definición es compilada por el controlador en un formato ejecutable y entonces ejecutado. Esto significa que no se le puede aconsejar a un error en nuestra función hasta que lo intentemos para usarlo.   
 Antes de escribir nuestras propias funciones en nuestro idioma se puede cargar de elección, primero tenemos que arreglar para PostgreSQL que soporta el idioma. Esto es lo que vamos a hacer a continuación para PL / pgSQL

**Introducción a PL / pgSQL**

Vamos a utilizar PL / pgSQL como lengua de procedimiento que puede cargarse en los procedimientos almacenados de la muestra en este capítulo. En una instalación de PostgreSQL estándar, la función de controlador de PL / pgSQL es incluido en el plpgsql.so biblioteca compartida en el directorio / usr / local / pgsql / lib para UNIX y Linux o plpgsql.dll en la carpeta \ lib de la instalación en Microsoft Windows

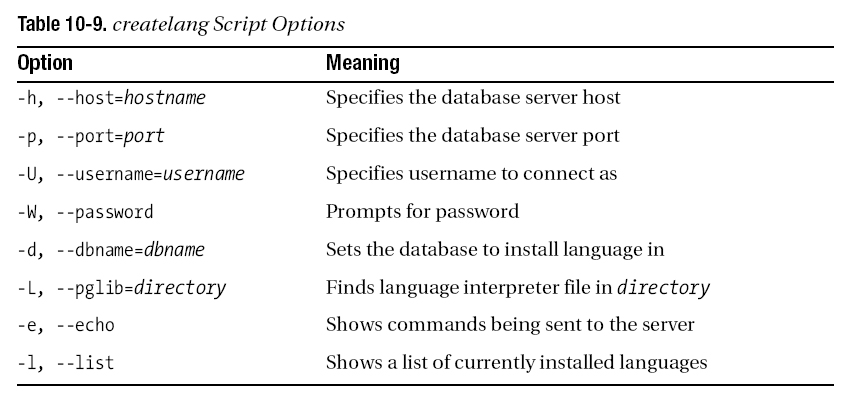
Cada base de datos PostgreSQL en un servidor tiene su propia lista de lenguas de procedimiento. Cuando nos instalar un lenguaje de procedimientos, podemos elegir cuál de las bases de datos se desarrollará procedimientos almacenados en ese idioma. Esto es en parte para la seguridad, ya que es posible crear funciones que, o bien consumen accidentalmente o maliciosamente recursos de la CPU, por ejemplo, un bucle indefinidamente. este podría constituir la base de un (DoS) ataque de denegación de servicio, haciendo que el servidor tan ocupado que no puede responder a las nuevas solicitudes. Por lo tanto, de manera predeterminada, las bases de datos PostgreSQL no tienen procesal idiomas instalados. Para utilizar PL / pgSQL, tenemos que instalar el controlador de nosotros mismos.

Nota El administrador de base de datos puede agregar idiomas a la base de datos template1, en cuyo caso, todo nuevo bases de datos tendrán esos idiomas por defecto.

Para instalar PL / pgSQL para nuestra base de datos bpfinal, podríamos utilizar el comando CREATE LANGUAGE dentro de psql y cargar la biblioteca compartida, la creación de la función de controlador de forma explícita. Esto es complejo lo suficiente como para justificar un script de ayuda, y el otro está dentro de la instalación de PostgreSQL. el comando que necesitamos es createlang:

createlang [*options*] *langname dbname*

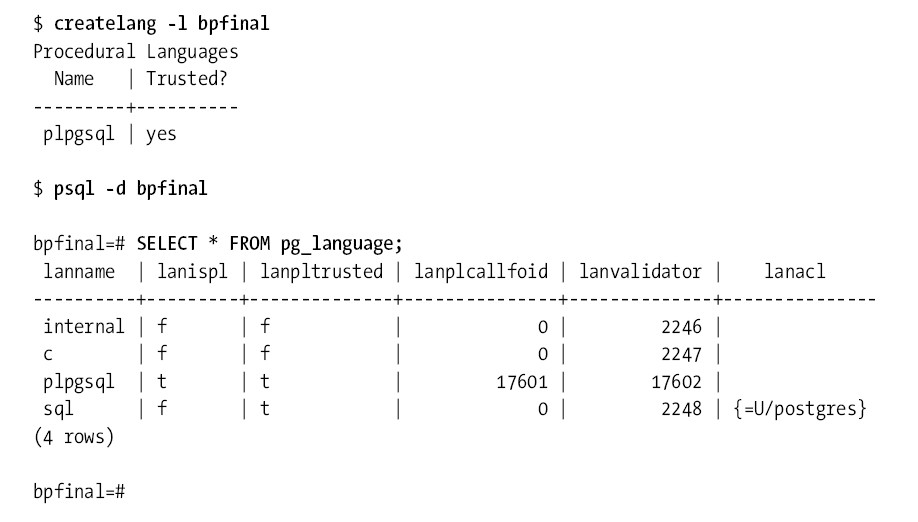
**Las opciones están listadas en la Tabla 10-9.**

****

Los usuarios normales no pueden añadir idiomas a bases de datos, por lo que por lo general se conectará como los postgres superusuario

$ createlang -U postgres plpgsql bpfinal

Podemos comprobar que el lenguaje está presente haciendo una lista de idiomas con createlang, o por la consulta de la tabla de sistema pg\_language con psql o pgAdmin:

****

Nota createlang se puede utilizar para instalar otros lenguajes de procedimiento. El comando para instalar PL / Tcl, por ejemplo, es createlang pltcl

Los superusuarios pueden eliminar el soporte para lenguajes de bases de datos mediante la ejecución de DROP LANGUAGE desde dentro psql:

bpfinal=# **DROP LANGUAGE 'plpgsql';**

DROP LANGUAGE

bpfinal=#

**Pruébelo: Crear un primer procedimiento almacenado**

Ahora estamos listos para comenzar a trabajar con procedimientos almacenados PL / pgSQL escribiendo nuestra propia funciones. Vamos a empezar por comprobar que todo funciona mediante la aplicación de la función add\_one mostrado anteriormente:

bpfinal=# **CREATE FUNCTION add\_one (int4) RETURNS int4 as '**

bpfinal'# **BEGIN return $1 + 1; end;' language 'plpgsql';**

CREATE FUNCTION

bpfinal=# **SELECT add\_one(2) AS answer;**

answer

--------

3

(1 row)

bpfinal=#

**¿Cómo funciona?**   
 La función Crear comando almacena la definición de la add\_one función escrita en PL / pgSQL a la base de datos. Se ejecuta cuando se evalúa la expresión SELECT. Tenga en cuenta que PL / pgSQL palabras claves tales como BEGIN y IDIOMA no distinguen entre mayúsculas y minúsculas y el diseño de la orden no importa. Podríamos haber definido la función como esta:

bpfinal=# **CREATE function**

bpfinal-# **add\_one(int4) RETURNS int4**

bpfinal-# **AS '**

bpfinal'# **BEGIN**

bpfinal'# **RETURN $1 + 1;**

bpfinal'# **END;**

bpfinal'# **'**

bpfinal-# **LANGUAGE 'plpgsql';**

CREATE FUNCTION

bpfinal=#

**sobrecarga de funciones**

PostgreSQL considera funciones para ser distinto si tienen nombres diferentes, si tienen un diferente número de parámetros, o si sus parámetros tienen diferentes tipos. Podemos crear   
otras funciones add\_one que se ocupan de diferentes tipos, si queremos. Considere lo que sucede cuando usamos nuestra función add\_one en un valor de coma flotante:

bpfinal=# **SELECT add\_one(3.1);**

ERROR: function add\_one(numeric) does not exist

HINT: No function matches the given name and argument types. You may need to add

explicit type casts.

bpfinal=#

PostgreSQL devuelve un error, porque no encontró una versión de la función add\_one   
que toma un valor de punto flotante como parámetro

Nota Las versiones anteriores de PostgreSQL habrían ejecutado esta función automáticamente por add\_one convertir el valor de punto flotante a un entero. El valor 3,1 se habría redondeado a 3, y el función hubiera devuelto 4.

Si queremos tener una función de incremento para los números de punto flotante, sólo tenemos que crear otra definición de add\_one:

bpfinal=# **CREATE FUNCTION**

bpfinal-# **add\_one(float8) RETURNS float8**

bpfinal-# **AS '**

bpfinal'# **BEGIN**

bpfinal'# **RETURN $1 + 1;**

bpfinal'# **END;**

bpfinal'# **'**

bpfinal-# **LANGUAGE 'plpgsql';**

CREATE FUNCTION

bpfinal=# **SELECT add\_one(3.1);**

add\_one

---------

4.1

(1 row)

bpfinal=#

Esta vez, obtenemos el resultado que queremos, porque PostgreSQL puede encontrar y ejecutar una adecuada versión de la función add\_one. Este comportamiento, conocido como la sobrecarga de funciones, puede ser bastante útil, pero también es bastante confuso. Para mantener las funciones distintas, hay que referirse a ellos en de una manera que indica sus parámetros. En este caso, tenemos dos funciones que podemos hacer referencia a como add\_one (int4) y add\_one (float8).

Sugerencia Una forma más conveniente para crear funciones es editar archivos de script que contiene la definición de funciones y para utilizar el psql \ i mando para leerlos.

**Listado Funciones**

Si tenemos que ver el código fuente de nuestras funciones una vez que se han cargado en la base de datos, podemos utilizar el comando interno psql \ df + o consultar la tabla que se utiliza para el almacenamiento de los procedimientos, que es la tabla pg\_proc

bpfinal=# **SELECT prosrc FROM pg\_proc WHERE proname = 'add\_one';**

prosrc

--------

begin

return $1 + 1;

end;

begin

return $1 + 1;

end;

(2 rows)

bpfinal=#

**Eliminación de Funciones**

Las funciones pueden ser lanzados desde una base de datos con DROP FUNCTION. Debemos estar seguros de que especificar el versión correcta de una función para las funciones sobrecargadas, y el abandono de todas las versiones de una función si queremos eliminar por completo la función:

bpfinal=# **DROP FUNCTION add\_one(int4);**

DROP FUNCTION

bpfinal=# **DROP FUNCTION add\_one(float8);**

DROP FUNCTION

bpfinal=#

**Citando (Quoting)**

Una ligera complicación que surge al utilizar PL / pgSQL para procedimientos almacenados preocupaciones citando. La definición de una función entera se le da al comando CREATE FUNCTION como una sola cadena entre comillas. Esto significa que si tenemos comillas simples dentro de nuestra definición de la función, que debe ser filtrado. Hacemos esto mediante el uso de dos citas junto a presentarse a una cita dentro de una cadena. Si nuestro procedimiento utiliza una cadena entrecomillada con comillas incrustado y ya se escapó, debemos escapar de esos también. Se puede acabar necesitando una fila de cuatro comillas consecutivas (o más). He aquí un ejemplo:

create function ... as ' ... return ''string with a single '''' in it ''; ...' ...

Afortunadamente, en PostgreSQL versión 8.0 y posteriores, la función de llamada en dólares citando disponible. Similar a la forma en que Perl y el UNIX / Linux conchas trabajan, dólar citando nos permite elegir una cadena que se utiliza en lugar de una apertura y cierre de cotización. Al elegir una cadena adecuada eso no ocurre en nuestro procedimiento, no es necesario utilizar ningún escapes. Una cotización del dólar es un cadena de cero o más caracteres entre $ caracteres. Así, en el ejemplo anterior se ve así:

create function ... as $$ ... return 'string with a single '' in it '; ... $$ ...

Si tenemos la cadena $ $ en nuestro procedimiento, podemos optar por una cotización del dólar diferente:

create function ... as $WHAT$ ... return 'string with a $$ in it '; ... $WHAT$ ...

Los ejemplos de este capítulo a partir de ahora utilizarán dólar citando. Si está utilizando una versión del PostgreSQL 8.0 antes, le sugerimos que actualice a la última versión. De lo contrario, se quiere necesitará usar tradicional cita.

**Anatomía de un procedimiento almacenado**

Ahora que hemos visto cómo crear, ejecutar y eliminar un procedimiento almacenado de ejemplo, es el momento para pasar a considerar la construcción de procedimientos almacenados PL / pgSQL con más detalle.

PL / pgSQL es un lenguaje con estructura de bloques, como Pascal o C, con variables con declaraciones y bloquear el ámbito de aplicación. Cada bloque tiene una etiqueta opcional; puede tener algunas declaraciones de variables, y   
incluye sentencias que conforman el bloque entre BEGIN y END palabras clave. La sintaxis de una bloque es de la siguiente manera:

[<<*label*>>]

[DECLARE *declarations*]

BEGIN

statements

END;

Nota PL / pgSQL distingue entre mayúsculas y minúsculas. Todas las palabras clave y los nombres de las variables pueden estar escritos en cualquiera de los casos.

Una función PL / pgSQL se define con una instrucción CREATE FUNCTION con un bloque como la definición parte, entre comillas (comillas simples o comillas dólar).

-- For all PostgreSQL versions

CREATE FUNCTION *name* ( [ *ftype* [, ...] ] )

RETURNS *rtype*

AS '*block definition*'

LANGUAGE 'plpgsql';

-- For PostgreSQL 8.0 and later

CREATE FUNCTION *name* ( [ [*arg*] *ftype* [, ...] ] )

RETURNS *rtype*

AS $$

*block definition*

$$ LANGUAGE plpgsql;

Dentro de la definición, como se muestra en los ejemplos, un guión doble (-) introduce un comentario que se extiende hasta el final de la línea y se ignorará. Pronto nos ocuparemos de utilizar comentarios en breve

**Argumentos de función**

Una función PL / pgSQL puede tomar cero o más parámetros, y los tipos de los parámetros son   
dado entre paréntesis después del nombre de la función. Los tipos están incorporados en los tipos de PostgreSQL, como int4 o float8. Todos los procedimientos almacenados deben devolver un valor, y el tipo de retorno se especifica en la DEVOLUCIONES cláusula de la definición de función.

Dentro de un cuerpo de la función, los parámetros de la función se conocen como $ 1, $ 2, y así sucesivamente, en el orden en que se definen. Veremos más adelante que es posible dar los parámetros de nombres mediante una declaración de ALIAS. En PostgreSQL 8.0, se hizo posible para dar nombres a los parámetros siempre que se declaren.

Considere este procedimiento almacenado simple que proporciona una media geométrica de punto flotante de dos números enteros:

-- geom\_avg

-- get a geometric average of two integers

create function geom\_avg(int4, int4) returns float8 as $$

begin

return sqrt($1 \* $2::float8);

end;

$$ language plpgsql;

Con el parámetro de nomenclatura en PostgreSQL 8.0, se puede declarar la misma función como esta:

-- geom\_avg

-- get a geometric average of two integers

create function geom\_avg(a int4, b int4) returns float8 as $$

begin

return sqrt(a \* b::float8);

end;

$$ language plpgsql;

Nótese que hemos echado uno de los valores enteros de un valor de punto flotante por lo que somos seguro de pasar un resultado de coma flotante de la multiplicación a la función sqrt. Si no lo hacemos esto, vamos a correr el riesgo de contraer un error que dice que no existe la función sqrt (int4), como era el caso antes de la versión 8.0 de PostgreSQL. Cuando las funciones están sobrecargados, el uso de un elenco asegurará que llamamos la versión deseada de la función.

**Comentarios**

Como se puede ver en los ejemplos hasta ahora, PL / pgSQL permite comentarios que se incluirán en la función definiciones. De hecho, hay dos tipos de comentarios: comentarios de una sola línea y comentarios en bloque. Una sola línea de SQL comentario estándar es introducido por dos guiones (-). Todo lo siguiente los dos guiones hasta el final de la línea se ignora:

-- This is a single line comment

create -- comments can

function -- come anywhere, and extend to the end of the line

Bloquea los comentarios se utilizan para introducir grandes bloques de texto como comentarios o de forma temporal   
la eliminación de secciones de código no es obligatorio. La sintaxis es la misma que C y C + +, con bloques de   
comentarios están rodeados por / \* y \* /:

/\*

Este es un comentario de bloque utilizado para describir el uso y el comportamiento de la función siguiente

\*/

create function blah() returns integer as $$

begin

/\* comment out call to func

func();

\*/

return 1;

end;

$$ language plpgsql;

Bloquea los comentarios no pueden anidarse, pero puede utilizar los comentarios de una sola línea para evitar que el bloquear delimitadores de comentarios sea interpretado como un comentario de bloque.

**Declaraciones**

PL / pgSQL funciones pueden declarar variables locales para el uso dentro de la función. Cada variable tiene un tipo que puede ser uno de los PostgreSQL incorporadas en tipos, un tipo definido por el usuario, o un tipo que corresponde a una fila en una tabla. Las declaraciones de variables de una función se escriben en la sección DECLARE de una definición de función o un bloque dentro de una función. Como es habitual en las lenguas de bloques estructurados como C y C + +, variables declaradas para un bloque sólo son visibles dentro de ese bloque o bloques dentro de ese bloque. Variables declarado en un bloque interno con el mismo nombre que una variable fuera de ese bloque de ocultar el exterior variables del bloque:

DECLARE

n1 integer;

n2 integer;

BEGIN

-- can use n1 and n2 in here

n2 := 1;

DECLARE

n2 integer; -- hides the earlier n2

n3 integer;

BEGIN

-- can use n1, n2 and n3 in here

n2 := 2;

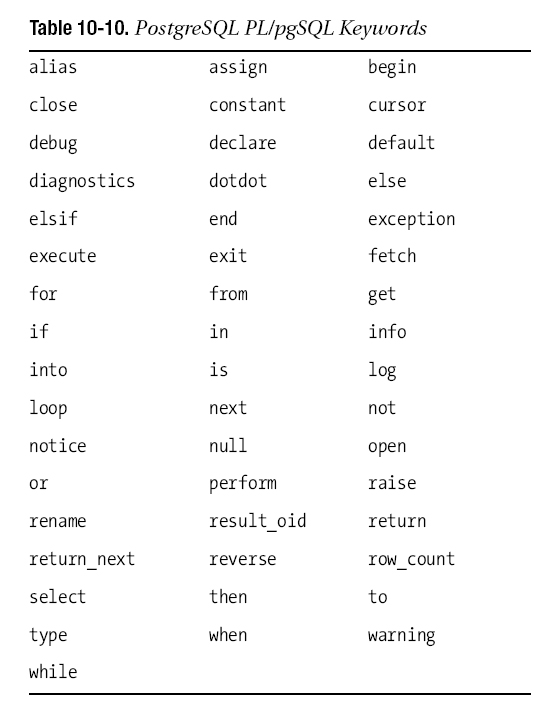
END;

-- n3 no longer available here

-- n2 still has value 1 here

END;

Todas las variables que se utilizan en una función se deben declarar antes de que puedan ser utilizados, a excepción de bucle variables de control que nos reuniremos más tarde. Una variable no puede tener el mismo nombre que un PL / pgSQL palabra clave, que están reservados. Tabla 10-10 enumera las palabras clave PostgreSQL PL / pgSQL.

****

Hay varias formas de declarar una variable, en función de su uso previsto. Estos declaración   
variaciones se discuten a continuación.

**ALIAS.**

La declaración más simple es una declaración ALIAS que le da un nombre a un parámetro de posición de una función. Esto nos ayuda a escribir código ligeramente más significativo, y el código que es más robusto contra los cambios en los números de parámetros y pedidos. La declaración ALIAS tiene esta sintaxis:

*name* ALIAS FOR $n;

A new variable called name is made available, which acts as another name for the specified

positional parameter. For example, we might have written the geom\_avg function as follows:

create function geom\_avg(integer, integer) returns float8 as '

declare

first alias for $1;

second alias for $2;

begin

return sqrt(first \* second::float8);

end;

' language 'plpgsql';

Como se señaló anteriormente, en PostgreSQL 8.0 podemos crear alias para los parámetros posicionales automáticamente al nombrarlos en la declaración funcional. La siguiente definición es equivalente a la anterior:

create function geom\_avg(first integer, second integer) returns float8 as $$

begin

return sqrt(first \* second::float8);

end;

$$ language plpgsql;

**RENAME**

También es posible cambiar el nombre de las variables con una declaración RENAME. Esto puede ser útil en el interior funciones de disparo que veremos más adelante, pero por lo general no se recomienda, ya que puede hacer que el código difícil de leer. La sintaxis de una declaración RENAME es el siguiente:

RENAME *original* TO *new*;

**Una simple declaración de variables**

Una variable simple es declarado por darle un nombre, un tipo y, opcionalmente, un valor inicial. Aquí está la sintaxis:

*name* [CONSTANT] *type* [NOT NULL] [{ DEFAULT | := } *value*];

El modificador CONSTANTE declara que la variable no puede ser cambiado. Debe, por lo tanto, incluir un valor inicial en su declaración.

La cláusula NOT NULL indica a PostgreSQL para criar a un error de ejecución si la variable es cada vez dado un valor NULL.

El valor inicial no tiene que ser una constante, y se evalúa y se asigna cada vez   
la función se llama o cuando se entra en el bloque en el que se declara la variable. Por ejemplo, dando un valor inicial de ahora para una variable de marca de tiempo se traduciría en la variable de tomar la la hora actual cuando se ejecuta, no cuando se compiló.

El tipo puede ser un PostgreSQL tipo incorporado; es decir, puede declarar variables con el mismo tipo de datos o la estructura de otro elemento de la base de datos. La ventaja de especificar el tipo de la variable de esta manera indirecta es que el código de procedimiento almacenado sigue siendo correcta, aun cuando los cambios   
se realizan en la base de datos. La sintaxis es la siguiente:

*builtintype*

*variable*%TYPE

*table.column*%TYPE

Estos son algunos ejemplos de declaraciones de variables:

n integer := 1;

mypi constant float8 := pi();

pizza\_pi mypi%TYPE;

mydesc item.description%type := 'extra large size pizza';

En el ejemplo, la declaración de mydesc dará lugar a una adecuada variable para el manejo de la descripción de la columna en la tabla de artículos de nuestra base de datos de la muestra. Si esa columna se define como char (64), cuando declaramos la variable, pero más tarde cambió a un char (80), el código usando mydesc seguiría trabajando, y PostgreSQL crearía el tipo correcto de variable.

Nota Nosotros hemos asumido dólar citando los ejemplos aquí. Usando solo citar lugar, necesitaríamos tener entre comillas dobles la cadena inicializamos mydesc con, ya que esta declaración se aparecen en el interior de un string-citado solo como parte de una definición de función.

Una declaración de variables Composite

Una variable de material compuesto es uno que corresponde a una fila completa en una tabla en particular. tiene campos que corresponden a cada columna de la tabla. Podemos declarar y utilizar variables compuestas en nuestros procedimientos almacenados, ya sea como tipo de fila o registro.

Para declarar una variable compuesta, usamos la sintaxis de declaración de tipo de fila de la siguiente manera:

*name table*%rowtype;

El resultado de esta declaración será una variable que sí tiene campos, uno para cada columna en la mesa sobre la que se basa. Considere lo siguiente:

contact customer%rowtype;

Esto creará una variable llamada contacto con campos correspondientes a las columnas en el cliente tabla. Para utilizar los campos, se utiliza el variable.field sintaxis. He aquí un fragmento de código de ejemplo:

DECLARE

contact customer%rowtype;

address text;

BEGIN

contact.zipcode := 'XY1 6ZZ';

contact.fname := NULL;

address := contact.addressline || contact.town;

...

END;

Un segundo tipo de tipo compuesto es el tipo de registro. Este es un tipo que actúa como un   
RowType, pero no se basa en una tabla en particular cuando se define. El registro tendrá los campos que coincidir con lo que se le asigna al expediente en tiempo de ejecución. Los registros son útiles en el código que se debe trabajar como factores desencadenantes que se llaman de diferentes tablas, tales como en un procedimiento de propósito general usado para   
iniciar eliminaciones de filas. También pueden ser utilizados para almacenar los resultados de los comandos SELECT en un general manera. Una declaración de registro es muy sencillo:

*name* record;

Vamos a ver más de los registros cuando cubramos la asignación a través de selecciones y de los factores desencadenantes.

Asignaciones

PL / pgSQL las variables se asignan nuevos valores en instrucciones de asignación. La sintaxis de una asignación es la siguiente:

*reference* := *expression*;

de referencia es un nombre de variable o campo en un tipo compuesto, tal como un tipo de fila o registro. la expresión puede ser una constante; otra variable; o una referencia de campo de una expresión compleja construido con los operadores, yesos y llamadas a funciones. Estos son algunos ejemplos de las tareas:

n1: = 23;

long\_variable\_names\_are\_OK := (n1 + 45)/2;

f2 := add\_one(n1)::float8 \* sqrt(2.0);

/ \* Los tipos compuestos se pueden asignar un campo a la vez   
haciendo referencia a los campos individuales: \* /

contact.zipcode := 'AB12 3CD';

**SELECT INTO Statement**

Un mecanismo de asignación alternativa es una extensión a SELECT. Podemos asignar una variable, una lista de variables, una variable de tipo fila entera, o un registro con una instrucción SELECT INTO. La sintaxis es una extensión de la normal de SQL SELECT:

SELECT *expressions* INTO *target* [FROM ...];

Algunos ejemplos sencillos de usar SELECT en lugar de un seguimiento operador de asignación:

SELECT sqrt(2.0) INTO sqrt2;

SELECT add\_one(n1) INTO n1;

SELECT 1,2,3,4 INTO n1, n2, n3, n4;

SELECT 'Mole', 'Adrian' INTO contact.lname, contact.fname;

Podemos asignar todo un tipo de fila a la vez, si especificamos valores para cada columna de la correcta orden:

DECLARE

product item%rowtype;

BEGIN

select NULL, 'Widget', '1.45', '1.99' into product;

END;

Donde los valores y variables Asignación son de diferentes tipos, PostgreSQL se aplicará Apropiada y esos donde puede. En el ejemplo anterior, los precios de las tarifas y vender son de tipo numérico (7,2)  
pero les asigna con éxito de valores de cadena