Revista Cubana de Ciencias Informáticas
Vol. 11, No. 4, Octubre-Diciembre, 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 12-22

Tipo de artículo: Artículos originales
Temática: Tecnologías de bases de datos
Recibido: dd/mm/aa | Aceptado: dd/mm/aa

Control de acceso multidimensional en PostgreSQL

Multi Dimensional Access Control In PostgreSQL

Charles Clavadetscher[1*]

[1]Centro de investigaciones económicas, Instituto tecnológico federal, Zurich, Suiza

[*]Autor para correspondencia: clavadetscher@kof.ethz.ch

Resumen
Una base de datos tiene informaciones para apoyar todos los procesos productivos de una empresa, independentiemente de su estructura laboral. Así se encontrarán, p.ej. catálogos de venta, descripciones de productos, datos contables, listas de empleados, etc. Siendo claro que no todos los empleados de una empresa necesitan acceso a todas las informaciones se pone la cuestión de cómo gestionar este acceso de forma confiable. Este proceso de selección se llama autorización y es disponible en todos los sistemas de base de datos. En PostgreSQL el control de acceso se organiza mediante el empleo de roles. El sistema clásico de autorización es vertical, es decir que permite elegir cuáles tablas o columnas de una tabla son accesibles para un usuario. Desde la versión 9.5, PostgreSQL introdujo la posibilidad de un control de acceso horizontal. El control de acceso a nivel de filas (row level security) permite elegir en base a unos criterios configurables, cuáles filas son visibles y por lo tanto gestionables para un usuario. La combinación de técnicas de control vertical y horizontal permite obtener una granularidad en el acceso alcanzable en el pasado solo por medio de soluciones alternativas difíciles de gestionar y, por ende, inseguras.

Palabras claves: PostgreSQL, bases de datos, seguridad, autorización, control de acceso, row level security

Abstract
A database contains the information required to support all the business processes of a company, independently of its personnel structure. Therefore you will have, e.g. sales catalogues, product descriptions, accounting information, lists of employees, etc. Obviously not all employees are supposed to have access to all data, thus posing the question on how to manage their access to them in a secure way. This selection process is called authorization and is available in all database systems. In PostgreSQL, access control is organized around roles. The classic authorization system is vertical. This means that it allows to choose which tables or columns thereof are accessible to a user. Since version 9.5, PostgreSQL introduced the possibility of a horizontal access control. This type of access (row level security) allows to choose based on a configurable set of crieteria which rows are visible, and therefore modifiable, by a user. The combination of vertical and horizontal access control techniques enables a granularity in the configuration that, in the past, could only be achieved through workarounds difficult to maintain and, therefore, insecure.

Keywords: *PostgreSQL, databases, security, authorization, access control, row level security*

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 4, Octubre-Diciembre, 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 12-22

Introducción

La necesidad de seleccionar filas de una tabla en base a unos criterios predefinidos siempre ha existido. Las soluciones tradicionales consistían en crear vistas sobre la tabla y revocar y otorgar privilegios para cumplir con los requisitos de acceso.

Se suponga que en la base de datos haya una tabla `library.catalogue` con la estructura que se indica en la tabla correspondiente.

item_id	name	location	price	responsible	quantity
1	Hemingway, El viejo y el mar	La Habana	10.00	ventas_la_habana	4
2	Padura, Pasado perfecto	La Habana	7.60	ventas_la_habana	2
3	Brecht, Poemas	La Habana	5.50	ventas_la_habana	0
4	Dante, La divina comedia	Santiago	4.20	ventas_santiago	5
5	Homero, Odisea	Santiago	5.30	ventas_santiago	4

Además existen unos grupos y usuarios que la utilizan.

Role name	Attributes	Member of
gestion	Cannot login	{}
ventas_la_habana	Cannot login	{}
ventas_santiago	Cannot login	{}
angela		{gestion}
juan		{ventas_la_habana}
felipe		{ventas_santiago}

En este ejemplo, que se utilizará para seguir ilustrando las nuevas posibilidades de seguridad en PostgreSQL, se presenta una situación sumamente simplificada. Una pequeña empresa con tres empleados que se dedica a la compraventa de libros. Uno de ellos, Ángela actúa como gestora de la empresa, mientras que Juan y Felipe están al frente de ventas en La Habana y en Santiago respectivamente.

Por razones de brevedad y con el fin de poner el enfoque sobre el tema específico se omiten aquí los detalles sobre la creación de la tabla, de los usuarios y de los privilegios básicos para acceder a los objetos.

Los requisitos de control de acceso de esta pequeña empresa son:

1. Los miembros del grupo de gestión pueden ver y manipular todas las entradas de la tabla.
2. Los miembros de los grupos de ventas en las regiones solo pueden ver las entradas de su región y solo cuando el producto está disponible para la venta.
3. Los miembros de los grupos de ventas en las regiones solo pueden cambiar el campo `quantity` de los productos en su región.

Los primeros dos requisitos requieren un control sobre las filas retornadas por una consulta sobre la tabla, en este contexto una selección horizontal, mientras que el tercero implica un control de acceso tanto horizontal como vertical.

En las versiones anteriores a 9.5 de PostgreSQL, se podía cumplir con los requisitos utilizando vistas sobre la tabla.

```
CREATE VIEW library.v_catalogue AS
SELECT * FROM library.catalogue
WHERE pg_has_role(CURRENT_USER,responsible,'USAGE')
AND quantity > 0;
```

La función `pg_has_role` se usa para averiguar si un usuario tiene un rol, es decir en este contexto, si es miembro de un grupo y hereda sus privilegios (PostgreSQL Global Development Group, 2015b). Para limitar el acceso de los sitios de venta se necesita quitar todos los privilegios sobre la tabla de base y otorgar los necesarios sobre la vista a los grupos respectivos.

```
REVOKE ALL ON library.catalogue FROM ventas_la_habana, ventas_santiago;
GRANT SELECT ON library.v_catalogue TO ventas_la_habana, ventas_santiago;
GRANT UPDATE (quantity) ON library.v_catalogue TO ventas_la_habana,
ventas_santiago;
```

Cabe notar que una vista es ejecutada en PostgreSQL con los privilegios del usuario que la creó. Por esta razón es posible que un usuario acceda a una tabla sobre la que no tiene privilegios directos, por medio de una vista.

Finalmente el acceso del grupo de gestores puede definirse sobre la tabla de base. En este caso también sería posible otorgar los privilegios sobra la vista. La ventaja de otorgar los privilegios directamente sobre la tabla es que cambios

eventuales en la estructura de la tabla (p.ej. al añadir una nueva columna) son visibles enseguida, cuando menos por los gestores. El resultado de una consulta del tipo:

```
SELECT * FROM library.v_catalogue;
```

no mostraría la nueva columna, porque cuando se creó la vista la columna no existía. Para esto se debería volver a crear la vista. Los gestores tienen que gozar de todos los privilegios de lectura y escritura sobre la tabla de base.

```
GRANT SELECT, INSERT, UPDATE, DELETE ON library.catalogue TO gestion;
```

Esta configuración cumple todos los requisitos alistados arriba. En resumen fue necesario:

- Crear una vista para filtrar las filas retornadas.
- Revocar los privilegios al pseudo grupo PUBLIC (todos los usuarios) sobre la tabla de base.
- Otorgar privilegios de lectura y de actualización sobre la vista para los puestos de venta regionales.
- Otorgar los privilegios de lectura y escritura sobre la tabla de base para los gestores.

En la siguiente sección se mostrará cómo es posible conseguir los mismos objetivos con las nuevas técnicas de control de acceso a nivel de fila. También se mostrarán las diferencias entre los enfoques para realizar una evaluación de los dos.

Hacia la seguridad horizontal

Para implementar un sistema de seguridad a nivel de filas se requieren por lo menos dos cosas:

1. Una o más normas (en inglés policy) que definen las modalidades del acceso.
2. Habilitar la tabla para el uso de las normas.

Anatomía de una norma

```
CREATE POLICY name ON table_name
```

```
[ FOR { ALL | SELECT | INSERT | UPDATE | DELETE } ]
[ TO { role_name | PUBLIC | CURRENT_USER | SESSION_USER } [, ...] ]
[ USING ( using_expression ) ]
[ WITH CHECK ( check_expression ) ]
```

Desde un punto de vista general una norma funciona como un filtro. Se definen:

- El nombre de la norma.
- La tabla por la que esta norma es válida (`ON`).
- Las operaciones por las que esta norma es válida (`FOR`).
- Los usuarios (o grupos) sometidos a la norma (`TO`).
- Las reglas de acceso al leer filas (`USING`).
- Las reglas de acceso al insertar o modificar filas (`WITH CHECK`).

La definición de una norma no es suficiente para que esa sea usada. Es necesario habilitar la tabla para su uso. Eso se hace con el comando siguiente:

```
ALTER TABLE tablename ENABLE ROW LEVEL SECURITY;
```

Para averiguar la configuración de acceso de una tabla, se puede utilizar el comando `\dp` en psql. En la sección sobre la implementación del ejemplo se encuentran más detalles sobre cómo interpretrar la salida del comando.

Implementación

Para el ejemplo se necesitan dos normas. Una para los usuarios en ventas y otra para los gestores. La primera norma se puede crear de la forma siguiente:

```
CREATE POLICY policy_ventas
ON library.catalogue
FOR ALL
```

```
TO PUBLIC
USING (pg_has_role(CURRENT_USER,responsible,'USAGE')
       AND quantity > 0);
```

En prosa este comando dice: Cuando un usuario cualquiera (`TO PUBLIC`) ejecuta un comando cualquiera (`FOR ALL`) sobre la tabla library.catalogue (`ON`) solo retorna filas de los grupos (`responsible`) en los que el usuario específico que ejecuta el comando (`CURRENT_USER`) es miembro y hereda sus privilegios. Además solo retorna filas cuyo valor en el campo quantity sea mayor que cero.

Es importante no olvidar que para activar la norma es necesario habilitar la tabla para su uso:

```
ALTER TABLE library.catalogue ENABLE ROW LEVEL SECURITY;
```

Esto corresponde a la parte horizontal de los requisitos números 2 y 3 que se definieron previamente. Falta todavía la parte vertical:

```
GRANT SELECT ON library.catalogue TO ventas_la_habana, ventas_santiago;
GRANT UPDATE (quantity) ON library.catalogue TO ventas_la_habana,
ventas_santiago;
GRANT SELECT, INSERT, UPDATE, DELETE ON library.catalogue TO gestion;
```

De esta forma el usuario Juan podrá ver solo las filas cuyo `responsible` es `ventas_la_habana` (Figura 3).

```
You are now connected to database "library" as user "juan".
juan@library=> SELECT * FROM library.catalogue ORDER BY item_id;

 item_id |            name              | location  | price |   responsible    | quantity
---------+------------------------------+-----------+-------+------------------+----------
       1 | Hemingway, El viejo y el mar | La Habana | 10.00 | ventas_la_habana |        4
       2 | Padura, Pasado perfecto      | La Habana |  7.60 | ventas_la_habana |        2
(2 rows)
```

Asimismo podrá modificar el campo `quantity` de estas filas.

```
juan@library=> UPDATE library.catalogue SET quantity = quantity - 1
WHERE item_id = 1;
UPDATE 1
```

```
juan@library=> SELECT * FROM library.catalogue ORDER BY item_id;
```

item_id	name	location	price	responsible	quantity
1	Hemingway, El viejo y el mar	La Habana	10.00	ventas_la_habana	3
2	Padura, Pasado perfecto	La Habana	7.60	ventas_la_habana	2

(2 rows)

Intentos de modificar otros campos o el campo `quantity` de filas que no son visibles no se ejecutan.

```
juan@library=> UPDATE library.catalogue
               SET responsible = 'ventas_santiago'
               WHERE item_id = 1;
ERROR:  permission denied for relation catalogue

juan@library=> UPDATE library.catalogue
               SET quantity = quantity - 1
               WHERE item_id = 4;
UPDATE 0
```

Para mostrar que Juan realmente no ha modificado el campo quantity, cuya responsabilidad es del grupo `ventas_santiago`, tenemos que anunciarnos a la base de datos con un usuario que no está sujeto a la seguridad a nivel de filas. Por definición estos son el dueño de la tabla y los superusuarios (PostgreSQL Global Development Group, 2015a; Clavadetscher, 2015).

```
You are now connected to database "library" as user "charles".
charles@library=# SELECT * FROM library.catalogue ORDER BY item_id;
```

```
 item_id |            name              | location  | price |   responsible    | quantity
---------+------------------------------+-----------+-------+------------------+----------
[...]
       4 | Dante, La divina comedia     | Santiago  |  4.20 | ventas_santiago  |        5
[...]
```

También por definición, si una tabla está sujeta a la seguridad a nivel de filas y no existe una norma para el usuario que ejecuta un comando, se aplica una norma de denegación total, o sea que aunque este usuario tenga p.ej. privilegios de lectura, la norma no retornará ninguna fila.

El usuario Ángela está sujeto a la norma `policy_ventas` y como no está en ninguno de los grupos presentes en `responsible` el intento de leer datos, aunque tenga el privilegio necesario retornará cero filas.

```
You are now connected to database "library" as user "angela".
angela@library=> SELECT * FROM library.catalogue ORDER BY item_id;

 item_id | name | location | price | responsible | quantity
---------+------+----------+-------+-------------+----------
(0 rows)
```

Para cumplir con el requisito número 1 necesitamos definir una norma para el grupo `gestion` al que Ángela pertenece:

```
CREATE POLICY policy_gestion
ON library.catalogue
FOR ALL
TO gestion
USING (true);
```

De esta forma miembros de este grupo tienen acceso a todas las filas de la tabla.

```
You are now connected to database "library" as user "angela".
angela@library=> SELECT * FROM library.catalogue ORDER BY item_id;
```

Todas las reglas de acceso, tanto verticales como horizontales, pueden visualizarse en un cliente (en este caso `psql`) con un simple comando `\dp`. El resultado del comando está dividido en 6 secciones, tres identifican el objeto de base (`Schema`, `Name` y `Type`) y tres muestran en forma compacta los privilegios definidos sobre el objeto:

```
 item_id |            name              | location  | price |   responsible    | quantity
---------+------------------------------+-----------+-------+------------------+----------
       1 | Hemingway, El viejo y el mar | La Habana | 10.00 | ventas_la_habana |        3
[...]
(5 rows)
```

- `Access privileges`: Son los privilegios verticales a nivel de tabla, privilegios que aparecen en esta sección son válidos para todas las columnas.
- `Column privileges`: Son los privilegios verticales sobre columnas de la tabla.
- `Policies`: Son los privilegios horizontales y definen bajo cuáles condiciones se deben retornar filas.

```
charles@library=# \dp library.catalogue
Access privileges
-[ RECORD 1 ]-----+-----------------------------------------
Schema            | library
Name              | catalogue
Type              | table
Access privileges | charles=arwdDxt/charles
                  | ventas_la_habana=r/charles
                  | ventas_santiago=r/charles
                  | gestion=arwd/charles
Column privileges | quantity:
                  |   ventas_la_habana=w/charles
                  |   ventas_santiago=w/charles
Policies          | policy_ventas:
                  |   (u): (pg_has_role("current_user"(),
                  |                     (responsible)::name,
                  |                     'USAGE'::text)
                  |         AND (quantity > 0))
                  | policy_gestion:
                  |   (u): true
                  |   to: gestion
```

Lo primero que se ve es que a nivel de tabla el usuario `charles` tiene todos los privilegios. El grupo `gestion` puede leer (r), añadir (a), modificar (w) y borrar (d) filas de la tabla. Los privilegios cubren todos los campos. Los dos grupos de ventas, en cambio, pueden leer todos los campos de la tabla y solo modificar el campo `quantity`, como se indica en la sección `Column privileges`. Las indicaciones en `Policies` son las que se explicaron arriba y no requieren más aclaraciones.

Las Listas completas de todas las abreviaciones usadas en la salida del comando se encuentran en la documentación de PostgreSQL (PostgreSQL Global Development Group, 2016), en varios libros de introducción general al gestor de bases de datos (Eisentraut and Helmle, 2011; Schönig, 2015; Riggs et al., 2015; Ahmed et al., 2015; Schönig, 2014) y en el compendio sobre autorización en PostgreSQL (Clavadetscher, 2015).

Conclusiones

La introducción de técnicas de seguridad a nivel de filas soluciona de forma simple y elegante un problema que al crecer el número de tablas en una base de datos se agudiza fuertemente. Las principales ventajas en comparación con el modelo de solución antiguo utilizando vistas como alternativa para el filtrado de filas, se resumen en unos pocos puntos:

- Todo el control de acceso a la tabla se define directamente sobre la tabla misma.
- No se necesitan vistas u otros objetos adicionales con privilegios propios.
- La implementación en la base de datos de las normas es más eficiente.
- El mantenimiento del sistema es más sencillo, no se deben de gestionar diferentes objetos con diferentes privilegios.
- Cambios en la tabla de base son visibles inmediatamente.

Referencias

Ibrar Ahmed, Asif Fayyaz, and Amjad Shahzad. *PostgreSQL Developer's Guide*. Packt Publishing, 1 edition, 2 2015. ISBN 9781783989027.

Charles Clavadetscher. *Autorización en PostgreSQL*. lulu.com, 1 edition, 2015. ISBN 9781326933241. http://www.lulu.com.

Paolo Corti, Thomas J. Kraft, Stephen V. Mather, and Bborie Park. *PostGIS Cookbook*. Packt Publishing, 1 edition, 2 2014. ISBN 9781849518666.

Peter Eisentraut and Bernd Helmle. *PostgreSQL Administration*. O'Really, 2 edition, 2011. ISBN 9783897216617.

Hannu Krosing, Jim Mlodgenski, and Kirk Roybal. *PostgreSQL Server Programming*. Packt Publishing, 1 edition, 6 2013. ISBN 9781849516983.

Regina Obe and Leo S. Hsu. *PostGIS In Action*. Packt Publishing, 2 edition, 2015. ISBN 9781617291395.

PostgreSQL Global Development Group. *Row Security Policies*. 2015a. http://www.postgresql.org/docs/9.5/static/ddl-rowsecurity.html.

PostgreSQL Global Development Group. *System Information Functions*. 2015b. https://www.postgresql.org/docs/9.5/static/functions-info.html.

PostgreSQL Global Development Group. *PostgreSQL 9.5 Documentation*. 2016. https://www.postgresql.org/files/documentation/pdf/9.5/postgresql-9.5-A4.pdf.

Simon Riggs, Hannu Krosing, Gianni Ciolli, and Gabriele Bartolini. *PostgreSQL 9 Administration Cookbook*. Packt Publishing, 2 edition, 4 2015. ISBN 9781849519069.

A. Rubinos and H. A. Nuevo. Seguridad en bases de datos. *Revista Cubana de Ciencias Informáticas*, 5(1), January-March 2011.

Hans-Jürgen Schönig. *PostgreSQL Administration Essentials*. Packt Publishing, 1 edition, 10 2014. ISBN 9781783988983.

Hans-Jürgen Schönig. *Troubleshooting PostgreSQL*. Packt Publishing, 1 edition, 3 2015. ISBN 9781783555314.

Anthony R. Sotolongo León and Yudisney Vazquez Ortíz. *PL/pgSQL y otros lenguajes procedurales en PostgreSQL*. lulu.com, 2 edition, 1 2017. ISBN 9781365689147. http://www.lulu.com.

Yudisney Vazquez Ortíz, Lisleydi Mier Pierre, and Anthony R. Sotolongo León. Características no relacionales de postgresql: incremento del rendimiento en el uso de datos json. *Revista Cubana de Ciencias Informáticas*, 10(Especial Informática 2016):70–81, January-March 2016.