PostGIS - регулирование доступа на уровне объектов

Настройка доступа БД PostrgreSQL

Обсудить в форуме Комментариев — 8

Часто требуется настроить какую-либо таблицу базы данных PostgreSQL для работы с пользователями таким образом, чтобы каждый пользователь мог видеть ВСЕ объекты (записи) в таблице, а редактировать (в том числе удалять) мог только те объекты, которые добавил он сам.

Задача разграничения доступа к данным для разных пользователей встречается практически во всех реально работающих ситемах, и различные СУБД имеют различные реализации системы безопасности. Решение этой задачи в СУБД PostgreSQL описано здесь. В данной статье дается пример реализации и делаются некоторые замечания, связанные с особенностью хранения пространственных данных в PostGIS.

Оглавление

1. Реализация: общая идея

Чтобы не повторять уже описаное, отметим только основную идею и перейдем к ее реализации на примере.

Главное, что нужно сделать - запретить пользователям работу с исходной таблицей. Все операции, которые должен производить пользователь над данными, могут быть доступны ему через представления (VIEW). Представление - нечто вроде вирутальной таблицы, являющейся результатом запроса, причем, с точки зрения пользователя, представление и выглядит как таблица. Данные, полученные посредством представления, можно просмотреть и, при наличии соответствующих прав, модифицировать. Запрет модификации чужих записей тем или иным пользователем реализуется настройкой правил работы с представлениями.

Рассмотрим процедуру предоставления прав на конкретном примере. Для этого: возьмем какой-либо shp-файл и поместим его в базу данных; потом создадим двух пользователей и настроим для них желаемые права доступа. В качестве примера shp-файла можно взять, к примеру, файл из набора vmap0, например файл с населенными пунктами.

Создадим новую БД в PostgreSQL, в которой мы будем производить эксперименты:

```
CREATE DATABASE vmap0 OWNER postgres;
```

или

```
createdb -U postgres vmap0
```

Добавим в БД возможность манипулировать географическими объектами (расширение PostGIS):

```
createlang plpgsql vmap0
psql -d vmap0 -f /usr/share/pgsql/contrib/lwpostgis.sql
psql -d vmap0 -f /usr/share/pgsql/contrib/spatial_ref_sys.sql
```

Создаем двух пользователей для тестирования работы:

```
CREATE USER gis1 PASSWORD 'gis1';
CREATE USER gis2 PASSWORD 'gis2';
```

Экспортируем в базу данных слой населенных пунктов (для примера возьмем только один слой из набора):

```
shp2pgsql pop-built-up-a.shp -s 4326 goroda > goroda.sql
psql -d vmap0 -f goroda.sql
```

В результате в БД появится таблица со следующей структурой:

```
   Column   |          Type          |                      Modifiers
------------+------------------------+-----------------------------------------------
-----
 gid        | integer                | not null default
nextval('goroda_gid_seq'::regclass)
 id         | bigint                 |
 f_code     | character varying(5)   |
 f_code_des | character varying(254) |
 nam        | character varying(254) |
 nam_descri | character varying(254) |
 tile_id    | integer                |
 fac_id     | bigint                 |
 the_geom   | geometry               |
Indexes:
    "goroda_pkey" PRIMARY KEY, btree (gid)
Check constraints:
    "enforce_dims_the_geom" CHECK (ndims(the_geom) = 2)
    "enforce_geotype_the_geom" CHECK (geometrytype(the_geom) = 'MULTIPOLYGON'::text OR
the_geom IS NULL)
    "enforce_srid_the_geom" CHECK (srid(the_geom) = 4326)
```

К существующей таблице добавим поле login для хранения имени пользователя - хозяина записи:

```
ALTER TABLE goroda ADD login text;
```

Назначим хозяина таблице:

```
ALTER TABLE goroda OWNER TO postgres;
```

Действуя, как указано в исходной ссылке, создаем VIEW и назначаем права доступа пользователям (не на все поля):

```
CREATE OR REPLACE VIEW goroda_view AS
        SELECT gid,nam,the_geom from goroda;

ALTER TABLE goroda_view OWNER TO postgres;
GRANT SELECT, UPDATE, INSERT, DELETE ON TABLE goroda_view TO gis1;
GRANT SELECT, UPDATE, INSERT, DELETE ON TABLE goroda_view TO gis2;

CREATE OR REPLACE RULE add AS
        ON INSERT TO goroda_view DO INSTEAD
                INSERT INTO goroda (nam,the_geom, login)
                values (new.nam, new.the_geom, user);

CREATE OR REPLACE RULE upd AS
        ON UPDATE TO goroda_view DO INSTEAD
                UPDATE goroda
                        SET nam=new.nam,the_geom=new.the_geom,login=user
                WHERE goroda.login=user AND gid=new.gid;

CREATE OR REPLACE RULE del AS
        ON DELETE TO goroda_view DO INSTEAD
                DELETE FROM goroda
                WHERE login=user AND gid=old.gid;
```

2. Особенности реализации

Приведенного выше примера должно быть достаточно, чтобы понять принципы работы. Однако, возникают небольшие тонкости, связанные с тем, что таблица, содержащая геометрические поля, связана с другими таблицами. В частности, таблица geometry_columns содержит информацию о геометрических полях и таблицах, которым принадлежат эти поля. Соответственно, необходимо дать возможность пользователям делать выборки из этой таблицы:

```
GRANT SELECT ON geometry_columns TO gis1;
GRANT SELECT ON geometry_columns TO gis2;
```

Далее нужно занести информацию о вновь созданном представлении (которое является виртуальной таблицей, содержащей пространственные данные) в таблицу geometry_columns. Поскольку представление goroda_view создано на основе таблицы goroda, естественно, что информация, описывающая геометрию, у таблиц goroda_view и goroda должна быть одинаковой:

```
INSERT INTO geometry_columns
        (f_table_catalog,f_table_schema, f_table_name, f_geometry_column,
coord_dimension, srid, type)
values
        ('','public', 'goroda_view', 'the_geom', 2, 4326, 'MULTIPOLYGON');
```

И, наконец, разрешим использование последовательности goroda_gid_seq (необходима при создании новой записи):

```
GRANT USAGE ON SEQUENCE goroda_gid_seq TO gis1;
GRANT USAGE ON SEQUENCE goroda_gid_seq TO gis2;
```

Проверяем, что получилось в результате. Создадим, например, в QGIS, соединения для пользователей gis1 и gis2. Пользователем gis1 попытаемся отредактировать какой-либо населенный пункт. При попытке сохранения результатов редактирования мы получим, что gis1 не может изменять уже существующие в БД данные по населенным пунктам (т.к. пользователи gis1 и gis2 не являются хозяевами импортированных записей); редактирование пользователем gis1 объектов, принадлежащих gis2 (и наоборот) также не может быть произведено. При этом создавать и редактировать собственные объекты пользователи gis1 и gis2 могут.

3. Реализация доступа для чтения для всех таблиц базы

Для того, чтобы открыть доступ на чтение к таблице базы нужно выполнить следующую команду:

```
GRANT SELECT ON table_name TO read_only_user
```

Выполнить из командной строки его можно так:

```
psql -U pgsql -d db_name -c "GRANT SELECT ON table_name TO read_only_user"
```

Если таких таблиц в базе много, то создадим небольшой скрипт такого содежания (назовем его grant.sql):

```
SELECT 'GRANT SELECT ON ' || relname || ' TO xxx;'
FROM pg_class JOIN pg_namespace ON pg_namespace.oid = pg_class.relnamespace
WHERE nspname = 'public' AND relkind IN ('r', 'v')
```

Дальше выполним его, задав имя базы данных, скрипт создаст еще один файл - res.sql, который содержит инструкции для всех таблиц:

```
psql -t -U pgsql -d mydatabase -f grant.sql > res.sql
psql -U pgsql -d mydatabase -f res.sql
```

Обсудить в форуме Комментариев — 8

Ссылки по теме

- Основы работы с PostGIS
- Руководство пользователя PostGIS

Последнее обновление: August 26 2010

Дата создания: 13.02.2009
Автор(ы): Дмитрий Колесов