Пуленепробиваемый бэкенд на PostgreSQL

Денис Милованов, 404 Group

С чего все начинается

- 1 база (CREATE DATABASE),
- 1 пользователь базы (CREATE USER).

Зачастую этим и заканчивается.

В чем проблема?

1)
```
SELECT *
    FROM main.users
    WHERE id = '1'; DROP SCHEMA public; SELECT '1';
```

   Ок, у всех сейчас ORM или DB Access Level, который заэкранирует или приведет к числу.

2) Злоумышленник закачал скрипт на сервер и может его выполнить. Ему доступны все коннекты, которые есть у приложения.

Как решать?

- 1 привилегированный пользователь db_owner, от которого ведется раскатка и раздаются гранты:

  CREATE SCHEMA / SEQUENCE / TABLE / INDEX / FUNCTION etc

- доступ только через сокеты:

```
local   db         db_owner                                md5
(psql -U db_owner -p 5432)

host    db         db_owner      0.0.0.0/0                 reject
```

Как решать?

1. непривилегированные пользователи баз:
   a. db_user_front (для интерфейса),
   b. db_user_admin (для интерфейса администратора),
   c. db_user_daemons (для фоновых служб),

   и так далее:

```
local/host   db      db_user_front      md5
local/host   db      db_user_admin      md5
local/host   db      db_user_daemons    md5
...
local/host   db        all              reject
```

2. изолированные компоненты приложения ходят в базы от имени соответствующих пользователей.

Как решать?

Грантуются свои права на таблицы:

```
GRANT SELECT
    ON TABLE main.users TO db_front;

GRANT SELECT, INSERT, UPDATE
    ON TABLE main.users TO db_user_daemons;
```

В чем проблема по-прежнему?

Гранты “крупные” – сразу на всю таблицу:

1. критически важные данные могут утечь (SELECT),
2. вставить можно что угодно и в каком-угодно количестве (INSERT).

Нужно ограничить запросы некоторой логикой, соответствующей тому, что может делать приложение.

Какие есть варианты?

Очевидно, что надо воспользоваться хранимыми процедурами.

```
CREATE FUNCTION name (args)
RETURNS rettype
LANGUAGE lang_name
SECURITY DEFINER
AS <definition>
```

“SECURITY DEFINER specifies that the function is to be executed with the privileges of the user that created it”

Практика

Привилегия на EXECUTE процедур дается всем пользователям по умолчанию (в отличие от привилегий на таблицы).

Изменяем это (от имени db_owner):

```
ALTER DEFAULT PRIVILEGES
    REVOKE EXECUTE ON FUNCTIONS FROM PUBLIC;
```

“The key word PUBLIC indicates that the privileges are to be **granted to all roles**, including those that might be created later”

Практика

Запрещаем кому бы то ни было читать код процедур:

```
REVOKE SELECT
    ON pg_catalog.pg_proc, information_schema.routines FROM PUBLIC;
REVOKE EXECUTE
    ON FUNCTION pg_catalog.pg_get_functiondef(oid) FROM PUBLIC;
```

Или даже вообще видеть код и структуру данных:

```
REVOKE USAGE
    ON SCHEMA pg_catalog, information_schema FROM PUBLIC;
```

Практика

Раскатка функций от имени db_owner:

```
CREATE FUNCTION main.foo();
```

плюс **явный** грант:

```
GRANT EXECUTE ON FUNCTION main.foo() TO db_user_front;
```

Практика

```
CREATE OR REPLACE FUNCTION main.get_user(
     s_uuid varchar
)
RETURNS main.t_user AS $BODY$
DECLARE
     r_user main.t_user;
BEGIN
     SELECT name, important_data INTO r_user
          FROM main.users
          WHERE uuid = s_uuid;

     RETURN r_user;
END;
$BODY$ LANGUAGE plpgsql SECURITY DEFINER;
```

```
GRANT USAGE
     ON SCHEMA main
     TO db_front_user;

GRANT EXECUTE
     ON FUNCTION main.get_user(
          s_uuid varchar
     )
     TO db_front_user;
```

Практика

```
db=> TABLE main.users;
ОШИБКА:  нет доступа к отношению users
```

```
db=> SELECT main.get_user('1234-5678');
          get_user
-----------------------------
 (Denis,"{""params"": [1]}")
(1 строка)
```

Практика

Вставим в код main.get_user:

```
RAISE NOTICE '%', session_user;
RAISE NOTICE '%', current_user;
```

И увидим:

```
NOTICE: db_front_user
NOTICE: db_owner
```

“The current user identifier is normally equal to the session user identifier, but might change temporarily in the context of SECURITY DEFINER functions”.

Практика

1) Приложение работает с пользователями не по айди, а по **строковым хешам** (сложно перебрать).

2) В логику процедур монтируется код, который делает оповещения при нарушении логики (напр.: трижды запрошен несуществующий пользователь и т.п.).

3) Приложения общаются друг с другом посредством очередей, которые, по тем же соображениям, лучше иметь родными (постгресовыми). Нам вполне подошел mbus.

Как раскатывать?

1. В текущем проекте набежало порядка 100 процедур.

2. На PGConf 2015 когда речь заходила про хранимые процедуры, то все сходились на мысли, что их надо хранить в гите.

PostgreSQLDeployerGUI

Идея:

1. Храним в базе актуальное описание объектов схемы.
2. Показываем те объекты в гите, код которых отличается от сохраненного.
3. Делаем возможным раскатать выбранные изменения в одной транзакции: DDL транзакционен.
4. По факту раскатки актуализируем описание в базе.

Организация репозитория

```
database-name.git/schemas/
    /main — схема
        /seeds — сиды (константы, словари и т.п.)
        /types — типы
        /functions — хранимые процедуры
        /tables — таблицы + индексы + констрейнты
        /sequences — сиквенсы
        /triggers — триггеры
        /queries* — произвольные запросы
    /other_schema — другая схема
        ...
```

Пример

database-name.git/schemas/main/users.sql

```
CREATE TABLE main.users ( … );
```

1 коммит

```
ALTER TABLE main.users
     ADD COLUMN deleted_at timestamptz NULL;
```

2 коммит

```
CREATE INDEX
     ON main.users USING btree(...);
```

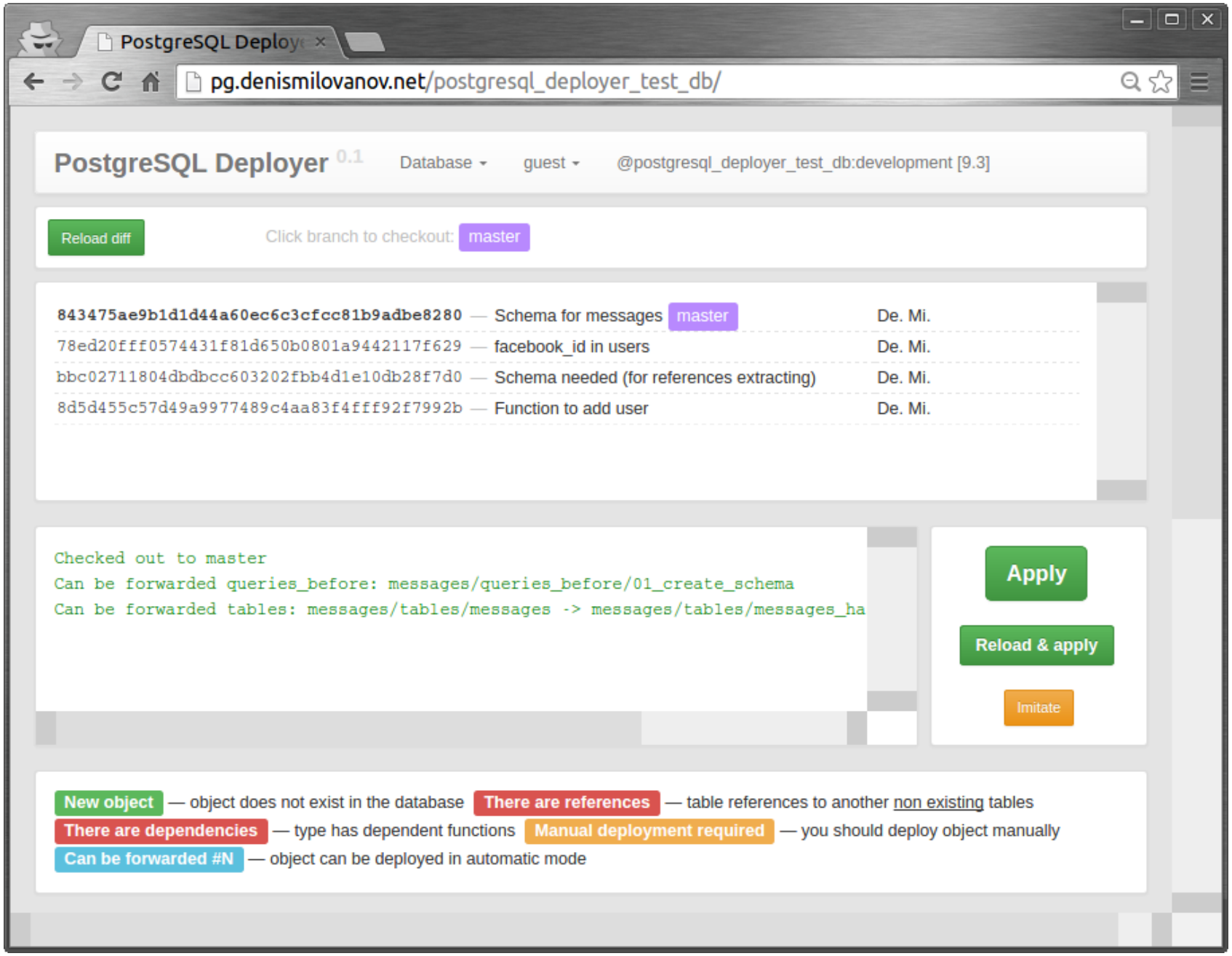
3 коммит

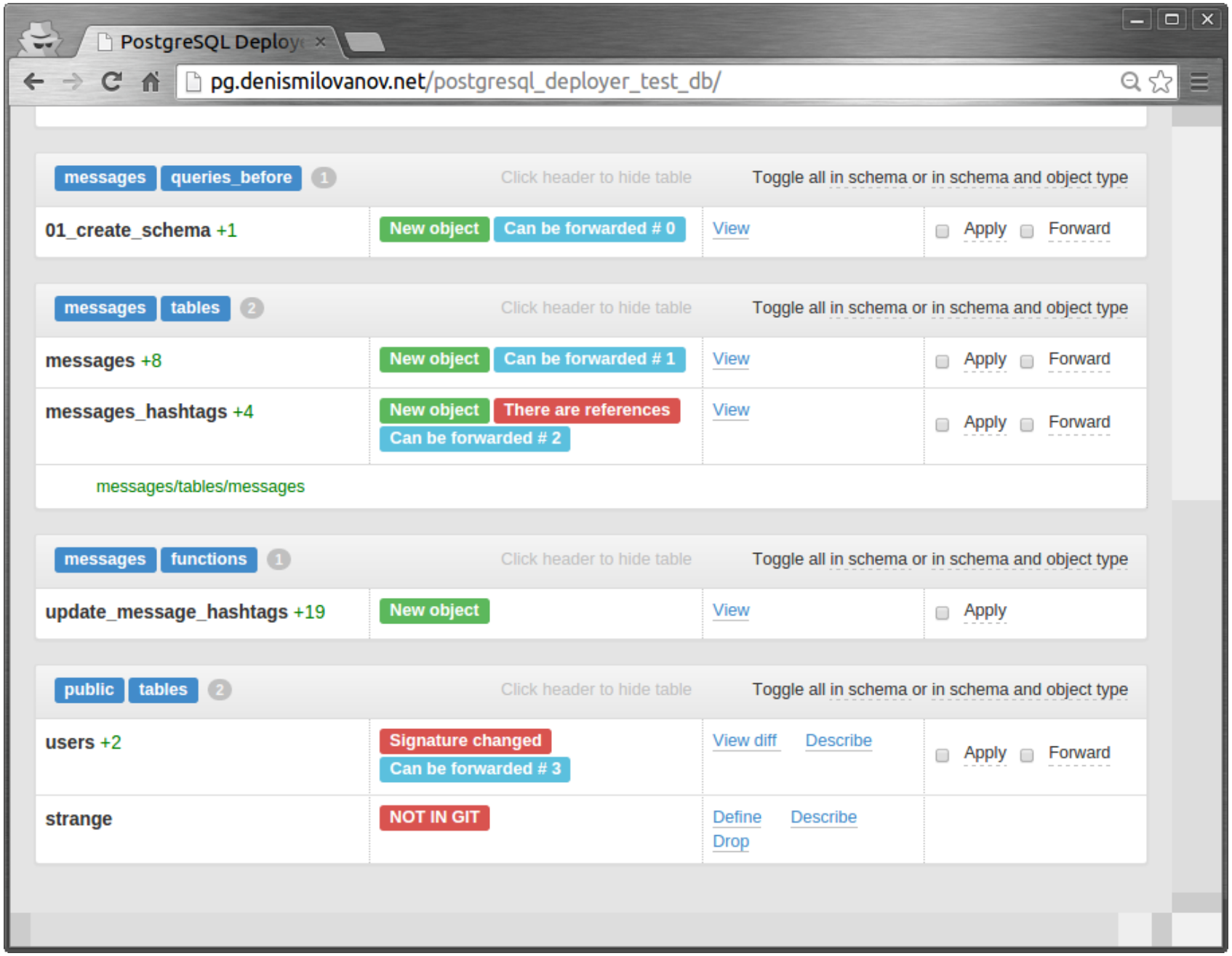
Раскатка отдельной процедуры

1. Удалить процедуру с таким именем, если поменялась сигнатура,
2. Выполнить запросы из файла (код + гранты).

```
main/functions/get_id.sql >
```

```
CREATE OR REPLACE FUNCTION main.get_id (
    s_dict_name varchar,
    s_index varchar
)
+RETURNS integer AS
-RETURNS bigint AS
$BODY$
DECLARE
    i_id integer;
BEGIN
    ...
```


PostgreSQL Deploy
pg.denismilovanov.net/postgresql_deployer_test_db/
PostgreSQL Deployer 0.1
Database
guest
@postgresql_deployer_test_db:development [9.3]
Reload diff
Click branch to checkout: master
843475ae9b1d1d44a60ec6c3cfcc81b9adbe8280 — Schema for messages master De. Mi.
78ed20fff0574431f81d650b0801a9442117f629 — facebook_id in users De. Mi.
bbc02711804dbdbcc603202fbb4d1e10db28f7d0 — Schema needed (for references extracting) De. Mi.
8d5d455c57d49a9977489c4aa83f4fff92f7992b — Function to add user De. Mi.
Checked out to master
Can be forwarded queries_before: messages/queries_before/01_create_schema
Can be forwarded tables: messages/tables/messages -> messages/tables/messages_ha
Apply
Reload & apply
Imitate
New object — object does not exist in the database
There are references — table references to another non existing tables
There are dependencies — type has dependent functions
Manual deployment required — you should deploy object manually
Can be forwarded #N — object can be deployed in automatic mode


PostgreSQL Deploy
pg.denismilovanov.net/postgresql_deployer_test_db/
messages queries_before 1
Click header to hide table
Toggle all in schema or in schema and object type
01_create_schema +1
New object
Can be forwarded # 0
View
Apply
Forward
messages tables 2
Click header to hide table
Toggle all in schema or in schema and object type
messages +8
New object
Can be forwarded # 1
View
Apply
Forward
messages_hashtags +4
New object
There are references
Can be forwarded # 2
View
Apply
Forward
messages/tables/messages
messages functions 1
Click header to hide table
Toggle all in schema or in schema and object type
update_message_hashtags +19
New object
View
Apply
public tables 2
Click header to hide table
Toggle all in schema or in schema and object type
users +2
Signature changed
Can be forwarded # 3
View diff
Describe
Apply
Forward
strange
NOT IN GIT
Define
Describe
Drop

Полезные возможности

1. Просмотр diff'ов.

2. Возможность показать то, что есть в базе, но нет в гите.

   Плюс можно получить описание через psql (\dt+, \sf), схему через pg_dump (--schema-only) или удалить.

Полезные возможности

3. Проверка на наличие логических ошибок в процедурах с помощью расширения plpgsql_check.
   https://github.com/okbob/plpgsql_check

4. “Reload and deploy” – перечитать диффы и раскатать все изменения сразу. Полезно для разработки.
   https://github.com/denismilovanov/PostgresqlDeployerGUI

Вопросы?

Денис Милованов

me@denismilovanov.net