Авторы: Егор Рогов, Павел Лузанов, Илья Баштанов

Обратная связь
Отзывы, замечания и предложения направляйте по адресу:
edu@postgrespro.ru

Темы

Что такое политики защиты строк

Условия применения политик

Несколько политик на одной таблице

Политика защиты строк

Определяет видимость и изменяемость строк таблицы

- предикат вычисляется *для каждой строки* с правами вызывающего
- клиенту доступны только те строки, для которых предикат истинен

Предикат для существующих строк (USING)

- используется операторами SELECT, UPDATE, DELETE
- при нарушении политики не возникает ошибка (если только не сброшен параметр *row_security*)

Предикат для новых строк (WITH CHECK)

- используется операторами INSERT, UPDATE
- если не задан, используется первый предикат
- при нарушении политики возникает ошибка

Политики защиты строк (row-level security, RLS) позволяют управлять доступом к таблице на уровне отдельных строк. Другое название этого инструмента — Fine Grained Access Control.

Политики являются дополнительным механизмом: роль по-прежнему должна иметь доступ к таблице, предоставленный привилегиями.

Политика определяет возможность выборки или изменения строк таблицы с помощью двух предикатов (логических выражений), которые вычисляются для каждой строки запроса. Результаты говорят о том, может ли пользователь видеть или менять строку.

Первый предикат вычисляется для *существующих* строк и используется операторами SELECT, UPDATE, DELETE. Если для какой-то строки предикат не принимает истинное значение (то есть дает ложь или NULL), строка не попадает в итоговую выборку. Упрощенно можно считать, что предикат просто «дописывается» к условию WHERE — хотя на самом деле все значительно сложнее.

Если установлено *row_security* = off, при ложном значении предиката хотя бы для одной строки будет зафиксирована ошибка. Это полезно при изготовлении логической резервной копии, чтобы гарантировать, что в нее попали все строки всех таблиц.

Второй предикат определяет видимость *новых* строк. Он проверяется командами INSERT и UPDATE; при нарушении политики возникает ошибка.

https://postgrespro.ru/docs/postgresql/13/ddl-rowsecurity

Условия применения

Политика применяется

- к таблице, для которой включена защита
- для указанных ролей и для указанных операторов (SELECT, INSERT, UPDATE, DELETE)

Политика не применяется

- при проверке ограничений целостности
- для суперпользователей и ролей с атрибутом BYPASSRLS
- для владельца (если не включить принудительно)

Чтобы политики защиты строк начали работать, нужно явно включить этот механизм для каждой таблицы.

При создании политики можно также задать, для каких ролей она будет работать (по умолчанию — для всех) и для каких операторов (по умолчанию — также для всех).

Политики не применяются при проверке ограничений целостности — независимо от настроенных политик СУБД гарантирует целостность данных.

Политики не применяются для суперпользователей (для них, как обычно, никакие проверки безопасности не выполняются) и для ролей с атрибутом BYPASSRLS.

Для владельца таблицы политики не работают по умолчанию, но специальной командой можно включить защиту и для владельца.

Пример политики защиты строк

```
=> CREATE DATABASE access_rls;
```

```
CREATE DATABASE
```

```
=> \c access_rls
```

```
You are now connected to database "access_rls" as user "student".
```

Алиса и Боб работают в разных отделах одной компании.

```
student=# CREATE ROLE alice LOGIN;
```

```
CREATE ROLE
```

```
student=# CREATE ROLE bob LOGIN;
```

```
CREATE ROLE
```

```
student=# CREATE TABLE users_depts(
  login text,
  department text
);
```

```
CREATE TABLE
```

```
student=# INSERT INTO users_depts VALUES ('alice','PR'), ('bob','Sales');
```

```
INSERT 0 2
```

Они обращаются к одной таблице, содержащей информацию обо всех отделах. При этом и Алиса, и Боб должны видеть данные только своего отдела.

```
student=# CREATE TABLE revenue(
  department text,
  amount numeric(10,2)
);
```

```
CREATE TABLE
```

```
student=# INSERT INTO revenue SELECT 'PR',   -random()* 100.00 FROM generate_series(1,100000);
```

```
INSERT 0 100000
```

```
student=# INSERT INTO revenue SELECT 'Sales', random()*1000.00 FROM generate_series(1,10000);
```

```
INSERT 0 10000
```

Определим соответствующую политику и включим ее:

```
student=# CREATE POLICY departments ON revenue
  USING (department = (SELECT department FROM users_depts WHERE login = current_user));
```

```
CREATE POLICY
```

```
student=# ALTER TABLE revenue ENABLE ROW LEVEL SECURITY;
```

```
ALTER TABLE
```

И нужно выдать Алисе и Бобу привилегии:

```
student=# GRANT SELECT ON users_depts, revenue TO alice, bob;
```

```
GRANT
```

Суперпользователь (он же владелец в данном случае) видит все строки независимо от политики:

```
student=# SELECT department, sum(amount) FROM revenue GROUP BY department;
```

```
 department |     sum
------------+-------------
 PR         | -5011384.62
 Sales      |  5014871.94
(2 rows)
```

А что увидят Алиса и Боб?

```
student=# \c - alice
```

```
You are now connected to database "access_rls" as user "alice".
```

```
alice=> SELECT department, sum(amount) FROM revenue GROUP BY department;

 department |     sum
------------+-------------
 PR         | -5011384.62
(1 row)
```

```
=> \c access_rls bob

You are now connected to database "access_rls" as user "bob".

bob=> SELECT department, sum(amount) FROM revenue GROUP BY department;

 department |    sum
------------+------------
 Sales      | 5014871.94
(1 row)
```

Несколько политик

Разрешительные политики

видимость должна предоставить хотя бы одна разрешительная политика

если не определена ни одна политика, строка не видима

Ограничительные политики

видимость должны предоставить все ограничительные политики, если они определены

На одной таблице можно определить несколько политик. В этом случае будут учитываться все предикаты.

По умолчанию создаются *разрешительные (permissive)* политики. Чтобы строка была доступна, достаточно, чтобы *хотя бы один* из предикатов этих политик был истинен.

Но если на таблице включена защита строк, и при этом не определено ни одной разрешительной политики, не будет доступна ни одна строка.

Дополнительно можно создать и *ограничительные (restricted)* политики. Если они заданы, то *все* их предикаты должны быть истинны.

Иными словами, если определены только разрешительные политики с предикатами P_1,...,P_N, то для каждой строки вычисляется выражение

$$P_1 \text{ OR } \ldots \text{ OR } P_N.$$

А если к тому же определены ограничительные политики с предикатами R_1,…, R_M, то для каждой строки будет вычисляться выражение

$$(P_1 \text{ OR } \ldots \text{ OR } P_N) \text{ AND } R_1 \text{ AND } \ldots \text{ AND } R_M.$$

То есть доступ должны предоставить *хотя бы одна разрешительная* и *все ограничительные* политики.

Несколько политик

Разрешим теперь Бобу добавлять строки в таблицу, но только для своего отдела и только в пределах 100 рублей:

- первое требование будет выполнено автоматически (единственный предикат работает и для существующих, и для новых строк);
- для второго создадим новую ограничительную политику.

```
alice=> \c - student
```

```
You are now connected to database "access_rls" as user "student".
```

```
student=# CREATE POLICY amount ON revenue AS RESTRICTIVE
  USING (true)                        -- видны все существующие строки
  WITH CHECK (abs(amount) <= 100.00); -- новые строки должны удовлетворять
```

```
CREATE POLICY
```

```
student=# GRANT INSERT ON revenue TO bob;
```

```
GRANT
```

Проверим:

```
bob=> INSERT INTO revenue VALUES ('Sales', 42.00);
```

```
INSERT 0 1
```

```
bob=> INSERT INTO revenue VALUES ('PR', 42.00);
```

```
ERROR:  new row violates row-level security policy for table "revenue"
```

```
bob=> INSERT INTO revenue VALUES ('Sales', 1000.00);
```

```
ERROR:  new row violates row-level security policy "amount" for table "revenue"
```

Политики, созданные для таблицы, показывают команды psql \d (описание объекта) и \dp (описание привилегий), например:

```
student=# \d revenue
```

```
                  Table "public.revenue"
   Column   |     Type      | Collation | Nullable | Default
------------+---------------+-----------+----------+---------
 department | text          |           |          |
 amount     | numeric(10,2) |           |          |
Policies:
    POLICY "amount" AS RESTRICTIVE
      USING (true)
      WITH CHECK ((abs(amount) <= 100.00))
    POLICY "departments"
      USING ((department = ( SELECT users_depts.department
   FROM users_depts
  WHERE (users_depts.login = CURRENT_USER))))
```

Эту информацию можно получить и из представления pg_policies системного каталога.

Итоги

Привилегии управляют доступом к таблицам и столбцам, политики защиты строк — к строкам

Политики настраиваются проще и работают эффективнее, чем реализация с помощью представлений и триггеров

Практика

1. Продолжая пример из демонстрации, создайте роль для Чарли и назначьте ему *два* отдела в таблице пользователей.
2. Определите политики защиты строк таким образом, чтобы:
   - роли видели строки только тех отделов, с которыми связаны;
   - роли, связанные с одним отделом, могли добавлять строки с суммой в пределах 100 рублей;
   - роли, связанные с несколькими отделами, могли добавлять строки с любой суммой.
3. Проверьте корректность настроенных политик.
4. Оцените накладные расходы на политики защиты строк, выполнив один и тот же запрос от имени обычной и суперпользовательской ролей.

1. Роли и таблицы

```
=> CREATE DATABASE access_rls;
```

```
CREATE DATABASE
```

```
=> \c access_rls
```

```
You are now connected to database "access_rls" as user "student".
```

```
student=# CREATE ROLE alice LOGIN;
```

```
CREATE ROLE
```

```
student=# CREATE ROLE bob LOGIN;
```

```
CREATE ROLE
```

```
student=# CREATE ROLE charlie LOGIN;
```

```
CREATE ROLE
```

```
student=# CREATE TABLE users_depts(
  login text,
  department text
);
```

```
CREATE TABLE
```

```
student=# INSERT INTO users_depts VALUES
  ('alice',  'PR'),
  ('bob',    'Sales'),
  ('charlie','PR'),
  ('charlie','Sales');
```

```
INSERT 0 4
```

```
student=# CREATE TABLE revenue(
  department text,
  amount numeric(10,2)
);
```

```
CREATE TABLE
```

```
student=# INSERT INTO revenue SELECT 'PR',   -random()* 100.00 FROM generate_series(1,100000);
```

```
INSERT 0 100000
```

```
student=# INSERT INTO revenue SELECT 'Sales', random()*1000.00 FROM generate_series(1,10000);
```

```
INSERT 0 10000
```

2. Политики и привилегии

```
student=# CREATE POLICY departments ON revenue
  USING (department IN (SELECT department FROM users_depts WHERE login = current_user));
```

```
CREATE POLICY
```

```
student=# CREATE POLICY amount ON revenue AS RESTRICTIVE
  USING (true)
  WITH CHECK (
    (SELECT count(*) FROM users_depts WHERE login = current_user) > 1
    OR abs(amount) <= 100.00
  );
```

```
CREATE POLICY
```

```
student=# ALTER TABLE revenue ENABLE ROW LEVEL SECURITY;
```

```
ALTER TABLE
```

```
student=# GRANT SELECT ON users_depts TO alice, bob, charlie;
```

```
GRANT
```

```
student=# GRANT SELECT, INSERT ON revenue TO alice, bob, charlie;
```

```
GRANT
```

3. Проверка

Алиса:

```
student=# \c - alice
```

```
You are now connected to database "access_rls" as user "alice".
```

```
alice=> SELECT department, sum(amount) FROM revenue GROUP BY department;
```

```
 department |     sum
------------+-------------
 PR         | -5016536.57
(1 row)
```

```
alice=> INSERT INTO revenue VALUES ('PR', 100.00);
```

```
INSERT 0 1
```

```
alice=> INSERT INTO revenue VALUES ('PR', 101.00);
```

```
ERROR:  new row violates row-level security policy "amount" for table "revenue"
```

Боб:

```
alice=> \c - bob
```

```
You are now connected to database "access_rls" as user "bob".
```

```
bob=> SELECT department, sum(amount) FROM revenue GROUP BY department;
```

```
 department |    sum
------------+------------
 Sales      | 4969640.15
(1 row)
```

```
bob=> INSERT INTO revenue VALUES ('Sales', 100.00);
```

```
INSERT 0 1
```

```
bob=> INSERT INTO revenue VALUES ('Sales', 101.00);
```

```
ERROR:  new row violates row-level security policy "amount" for table "revenue"
```

Чарли:

```
bob=> \c - charlie
```

```
You are now connected to database "access_rls" as user "charlie".
```

```
charlie=> SELECT department, sum(amount) FROM revenue GROUP BY department;
```

```
 department |     sum
------------+-------------
 PR         | -5016436.57
 Sales      |  4969740.15
(2 rows)
```

```
charlie=> INSERT INTO revenue VALUES ('PR', 1000.00);
```

```
INSERT 0 1
```

```
charlie=> INSERT INTO revenue VALUES ('Sales', 1000.00);
```

```
INSERT 0 1
```

4. Накладные расходы

Выполним запрос несколько раз, чтобы оценить среднее значение времени выполнения.

```
charlie=> \timing on
```

```
Timing is on.
```

Сначала от имени charlie:

```
charlie=> SELECT department, sum(amount) FROM revenue GROUP BY department;
```

```
 department |     sum
------------+-------------
 PR         | -5015436.57
 Sales      |  4970740.15
(2 rows)

Time: 37,630 ms
```

```
charlie=> SELECT department, sum(amount) FROM revenue GROUP BY department;
```

```
 department |     sum
------------+-------------
 PR         | -5015436.57
 Sales      |  4970740.15
(2 rows)

Time: 39,893 ms
```

```
charlie=> SELECT department, sum(amount) FROM revenue GROUP BY department;
```

```
 department |     sum
------------+-------------
 PR         | -5015436.57
 Sales      |  4970740.15
(2 rows)

Time: 39,495 ms
```

А теперь от имени владельца таблицы, на которого по умолчанию политики не действуют:

```
charlie=> \c - student
```

```
You are now connected to database "access_rls" as user "student".
```

```
student=# SELECT department, sum(amount) FROM revenue GROUP BY department;
```

```
 department |     sum
------------+-------------
 PR         | -5015436.57
 Sales      |  4970740.15
(2 rows)

Time: 29,704 ms
```

```
student=# SELECT department, sum(amount) FROM revenue GROUP BY department;
```

```
 department |     sum
------------+-------------
 PR         | -5015436.57
 Sales      |  4970740.15
(2 rows)

Time: 25,989 ms
```

```
student=# SELECT department, sum(amount) FROM revenue GROUP BY department;
```

```
 department |     sum
------------+-------------
 PR         | -5015436.57
 Sales      |  4970740.15
(2 rows)

Time: 28,573 ms
```

В данном конкретном случае накладные расходы не драматичны, хотя и вполне ощутимы.