

Практика №26

1. Создайте триггер, обрабатывающий обновление поля `onhand_qty` представления `catalog_v`.

Проверьте, что в «Каталоге» появилась возможность заказывать книги.

2. Обеспечьте выполнение требования согласованности: количество книг на складе не может быть отрицательным (нельзя купить книгу, которой нет в наличии).

Внимательно проверьте правильность реализации, учитывая, что с приложением могут одновременно работать несколько пользователей.

Решение

2. Может показаться, что достаточно создать AFTER-триггер на таблице `operations`, подсчитывающий сумму `qty_change`. Однако на уровне изоляции `Read Committed`, с которым работает приложение «Книжный магазин», нам придется блокировать таблицу `operations` в эксклюзивном режиме — иначе возможны сценарии, при которых такая проверка не работает.

Лучше поступить следующим образом: добавить в таблицу `books` поле `onhand_qty` и создать триггер, изменяющий это поле при изменении таблицы `operations` (то есть, фактически, выполнить денормализацию данных). На поле `onhand_qty` теперь можно наложить ограничение `CHECK`, реализующее требование согласованности. А функция `onhand_qty()`, которую мы создавали ранее, больше не нужна.

Особое внимание надо уделить начальной установке значения, учитывая, что одновременно с выполнением наших операций в системе могут работать пользователи.

1. Триггер для обновления каталога

```
=> CREATE OR REPLACE FUNCTION update_catalog() RETURNS trigger
AS $$
BEGIN
    INSERT INTO operations(book_id, qty_change) VALUES
        (OLD.book_id, NEW.onhand_qty - coalesce(OLD.onhand_qty,0));
    RETURN NEW;
END;
$$ VOLATILE LANGUAGE plpgsql;CREATE FUNCTION
```

```
=> CREATE TRIGGER update_catalog_trigger  
INSTEAD OF UPDATE ON catalog_v  
FOR EACH ROW  
EXECUTE FUNCTION update_catalog();
```

CREATE TRIGGER

2. Проверка количества книг

Добавляем к таблице книг поле наличного количества. (До версии 11 важно было учитывать, что указание предложения DEFAULT вызывало перезапись всех строк таблицы, удерживая блокировку.)

```
=> ALTER TABLE books ADD COLUMN onhand_qty integer;
```

ALTER TABLE

Триггерная функция для AFTER-триггера на вставку для обновления количества (предполагаем, что поле onhand_qty не может быть пустым):

```
=> CREATE OR REPLACE FUNCTION update_onhand_qty() RETURNS trigger  
AS $$  
BEGIN  
    UPDATE books  
    SET onhand_qty = onhand_qty + NEW.qty_change  
    WHERE book_id = NEW.book_id;  
    RETURN NULL;  
END;  
$$ VOLATILE LANGUAGE plpgsql;
```

CREATE FUNCTION

Дальше все происходит внутри транзакции.

```
=> BEGIN;
```

BEGIN

Блокируем операции на время транзакции:

```
=> LOCK TABLE operations;
```

LOCK TABLE

Начальное заполнение:

```
=> UPDATE books b
```

```
SET onhand_qty = (  
    SELECT coalesce(sum(qty_change),0)  
    FROM operations o  
    WHERE o.book_id = b.book_id);
```

UPDATE 6

Теперь, когда поле заполнено, задаем ограничения:

```
=> ALTER TABLE books ALTER COLUMN onhand_qty SET DEFAULT 0;
```

ALTER TABLE

```
=> ALTER TABLE books ALTER COLUMN onhand_qty SET NOT NULL;
```

ALTER TABLE

```
=> ALTER TABLE books ADD CHECK(onhand_qty >= 0);
```

ALTER TABLE

Создаем триггер:

```
=> CREATE TRIGGER update_onhand_qty_trigger  
AFTER INSERT ON operations  
FOR EACH ROW  
EXECUTE FUNCTION update_onhand_qty();
```

CREATE TRIGGER

Готово.

```
=> COMMIT;
```

COMMIT

Теперь books.onhand_qty обновляется, но представление catalog_v по-прежнему вызывает функцию для подсчета количества. Хотя в исходном запросе обращение к функции синтаксически не отличается от обращения к полю, запрос был запомнен в другом виде:

```
=> \d+ catalog_v
```

View "bookstore.catalog_v"						
Column	Type	Collation	Nullable	Default	Storage	Description
book_id	integer				plain	
title	text				extended	
onhand_qty	integer				plain	
display_name	text				extended	
authors	text				extended	

View definition:

```
SELECT b.book_id,
       b.title,
       onhand_qty(b.*) AS onhand_qty,
       book_name(b.book_id, b.title) AS display_name,
       authors(b.*) AS authors
FROM books b
ORDER BY (book_name(b.book_id, b.title));
```

Triggers:

```
update_catalog_trigger INSTEAD OF UPDATE ON catalog_v FOR EACH ROW
EXECUTE FUNCTION update_catalog()
```

Пересоздадим представление:

```
=> CREATE OR REPLACE VIEW catalog_v AS
SELECT b.book_id,
       b.title,
       b.onhand_qty,
       book_name(b.book_id, b.title) AS display_name,
       b.authors
FROM books b
ORDER BY display_name;
```

CREATE VIEW

Теперь функцию можно удалить.

```
=> DROP FUNCTION onhand_qty(books);
```

DROP FUNCTION

Небольшая проверка:

```
=> SELECT * FROM catalog_v WHERE book_id = 1 \gx
```

```
-[ RECORD 1 ]+-----
book_id      | 1
title        | Сказка о царе Салтане
onhand_qty   | 19
display_name | Сказка о царе Салтане. Пушкин А. С.
authors      | Пушкин Александр Сергеевич
```

=> **INSERT INTO operations(book_id, qty_change) VALUES (1,+10);**

INSERT 0 1

=> **SELECT * FROM catalog_v WHERE book_id = 1 \gx**

```
-[ RECORD 1 ]+-----
book_id      | 1
title        | Сказка о царе Салтане
onhand_qty   | 29
display_name | Сказка о царе Салтане. Пушкин А. С.
authors      | Пушкин Александр Сергеевич
```

Некорректные операции обрываются:

=> **INSERT INTO operations(book_id, qty_change) VALUES (1,-100);**

ERROR: new row for relation "books" violates check constraint
"books_onhand_qty_check"

DETAIL: Failing row contains (1, Сказка о царе Салтане, -71).

CONTEXT: SQL statement "UPDATE books

SET onhand_qty = onhand_qty + NEW.qty_change

WHERE book_id = NEW.book_id"

PL/pgSQL function update_onhand_qty() line 3 at SQL statement

Практика №27

1. Напишите триггер, увеличивающий счетчик (поле version) на единицу при каждом изменении строки. При вставке новой строки счетчик должен устанавливаться в единицу. Проверьте правильность работы.

2. Даны таблицы заказов (orders) и строк заказов (lines). Требуется выполнить денормализацию: автоматически обновлять сумму заказа в таблице orders при изменении строк в заказе.

Создайте необходимые триггеры с использованием переходных таблиц для минимизации операций обновления.

Решение

2. Для создания таблиц используйте команды:

```
CREATE TABLE orders (  
    id int PRIMARY KEY,  
    total_amount numeric(20,2) NOT NULL DEFAULT 0  
);
```

```
CREATE TABLE lines (  
    id int PRIMARY KEY GENERATED ALWAYS AS IDENTITY,  
    order_id int NOT NULL REFERENCES orders(id),  
    amount numeric(20,2) NOT NULL  
);
```

Столбец orders.total_amount должен автоматически вычисляться как сумма значений столбца lines.amount всех строк, относящихся к соответствующему заказу.

1. Счетчик номера версии

```
=> CREATE DATABASE plpgsql_triggers;
```

```
CREATE DATABASE
```

```
=> \c plpgsql_triggers
```

You are now connected to database "plpgsql_triggers" as user "student". Таблица:

```
=> CREATE TABLE t(  
    id integer PRIMARY KEY GENERATED ALWAYS AS IDENTITY,  
    s text,  
    version integer  
);
```

CREATE TABLE

Триггерная функция:

```
=> CREATE OR REPLACE FUNCTION inc_version() RETURNS trigger  
AS $$  
BEGIN  
    IF TG_OP = 'INSERT' THEN  
        NEW.version := 1;  
    ELSE  
        NEW.version := OLD.version + 1;  
    END IF;  
    RETURN NEW;  
END;  
$$ LANGUAGE plpgsql;
```

CREATE FUNCTION Триггер:

```
=> CREATE TRIGGER t_inc_version  
BEFORE INSERT OR UPDATE ON t  
FOR EACH ROW EXECUTE FUNCTION inc_version();
```

CREATE TRIGGER

Проверяем:

```
=> INSERT INTO t(s) VALUES ('Pa3');
```

INSERT 0 1

```
=> SELECT * FROM t;
```

```
id| s |version  
----+-----+-----  
 1| Pa3 |      1  
(1 row)
```

Явное указание version игнорируется:

```
=> INSERT INTO t(s,version) VALUES ('Два',42);
```

INSERT 0 1

```
=> SELECT * FROM t;
```

id	s	version
1	Раз	1
2	Два	1

(2 rows)

Изменение:

```
=> UPDATE t SET s = lower(s) WHERE id = 1;
```

UPDATE 1

```
=> SELECT * FROM t;
```

id	s	version
2	Два	1
1	раз	2

(2 rows)

Явное указание также игнорируется:

```
=> UPDATE t SET s = lower(s), version = 42 WHERE id = 2;
```

UPDATE 1

```
=> SELECT * FROM t;
```

id	s	version
1	раз	2
2	два	2

(2 rows)

2. Автоматическое вычисление общей суммы заказов

Создаем таблицы упрощенной структуры, достаточной для демонстрации:

```
=> CREATE TABLE orders (  
    id integer PRIMARY KEY,
```



```
total_amount numeric(20,2) NOT NULL DEFAULT 0  
);
```

CREATE TABLE

```
=> CREATE TABLE lines (  
  id integer PRIMARY KEY GENERATED ALWAYS AS IDENTITY,  
  order_id integer NOT NULL REFERENCES orders(id),  
  amount numeric(20,2) NOT NULL  
);
```

CREATE TABLE

Создаем триггерную функцию и триггер для обработки вставки:

```
=> CREATE FUNCTION total_amount_ins() RETURNS trigger  
AS $$  
BEGIN  
  WITH I(order_id, total_amount) AS (  
    SELECT order_id, sum(amount)  
    FROM new_table  
    GROUP BY order_id  
  )  
  UPDATE orders o  
  SET total_amount = o.total_amount + I.total_amount  
  FROM I  
  WHERE o.id = I.order_id;  
  RETURN NULL;  
END;  
$$ LANGUAGE plpgsql;
```

CREATE FUNCTION

Предложение FROM в команде UPDATE позволяет соединить orders с подзапросом по переходной таблице и использовать столбцы подзапроса для вычисления значения.

```
=> CREATE TRIGGER lines_total_amount_ins  
AFTER INSERT ON lines  
REFERENCING  
  NEW TABLE AS new_table  
FOR EACH STATEMENT  
EXECUTE FUNCTION total_amount_ins();
```

CREATE TRIGGER

Функция и триггер для обработки обновления:

```
=> CREATE FUNCTION total_amount_upd() RETURNS trigger
AS $$
BEGIN
    WITH l_tmp(order_id, amount) AS (
        SELECT order_id, amount FROM new_table
        UNION ALL
        SELECT order_id, -amount FROM old_table
    ), l(order_id, total_amount) AS (
        SELECT order_id, sum(amount)
        FROM l_tmp
        GROUP BY order_id
        HAVING sum(amount) <> 0
    )
    UPDATE orders o
    SET total_amount = o.total_amount + l.total_amount
    FROM l
    WHERE o.id = l.order_id;
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;
```

CREATE FUNCTION

Условие HAVING позволяет пропускать изменения, не влияющие на общую сумму заказа.

```
=> CREATE TRIGGER lines_total_amount_upd
AFTER UPDATE ON lines
REFERENCING
    OLD TABLE AS old_table
    NEW TABLE AS new_table
FOR EACH STATEMENT
EXECUTE FUNCTION total_amount_upd();
```

CREATE TRIGGER

Функция и триггер для обработки удаления:

```
=> CREATE FUNCTION total_amount_del() RETURNS trigger
AS $$
BEGIN
    WITH l(order_id, total_amount) AS (
        SELECT order_id, -sum(amount)
```

```

        FROM old_table
        GROUP BY order_id
    )
    UPDATE orders o
    SET total_amount = o.total_amount + l.total_amount
    FROM l
    WHERE o.id = l.order_id;
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

```

CREATE FUNCTION

```

=> CREATE TRIGGER lines_total_amount_del
AFTER DELETE ON lines
REFERENCING
    OLD TABLE AS old_table
FOR EACH STATEMENT
EXECUTE FUNCTION total_amount_del();

```

CREATE TRIGGER

Остался неохваченным оператор TRUNCATE. Однако триггер для этого оператора не может использовать переходные таблицы. Но мы знаем, что после выполнения TRUNCATE в lines не останется строк, значит можно обнулить суммы всех заказов.

```

=> CREATE FUNCTION total_amount_truncate() RETURNS trigger
AS $$
BEGIN
    UPDATE orders SET total_amount = 0;
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

```

CREATE FUNCTION

```

=> CREATE TRIGGER lines_total_amount_truncate
AFTER TRUNCATE ON lines
FOR EACH STATEMENT
EXECUTE FUNCTION total_amount_truncate();

```

CREATE TRIGGER

Дополнительно нужно запретить изменять значение total_amount вручную, но это задача решается не триггерами. Проверяем работу.

Добавили два новых заказа без строк:

```
=> INSERT INTO orders VALUES (1), (2);
```

INSERT 0 2

```
=> SELECT * FROM orders ORDER BY id;
```

id	total_amount
1	0.00
2	0.00

(2 rows)

Добавили строки в заказы:

```
=> INSERT INTO lines (order_id, amount) VALUES  
    (1,100), (1,100), (2,500), (2,500);
```

INSERT 0 4

```
=> SELECT * FROM lines;
```

id	order_id	amount
1	1	100.00
2	1	100.00
3	2	500.00
4	2	500.00

(4 rows)

```
=> SELECT * FROM orders ORDER BY id;
```

id	total_amount
1	200.00
2	1000.00

(2 rows)

Удвоили суммы всех строк всех заказов:

```
=> UPDATE lines SET amount = amount * 2;
```

UPDATE 4

=> **SELECT * FROM orders ORDER BY id;**

id	total_amount
1	400.00
2	2000.00

(2 rows)

Удалим одну строку первого заказа:

=> **DELETE FROM lines WHERE id = 1;**

DELETE 1

=> **SELECT * FROM orders ORDER BY id;**

id	total_amount
1	200.00
2	2000.00

(2 rows)

Опустошим таблицу строк:

=> **TRUNCATE lines;**

TRUNCATE TABLE

=> **SELECT * FROM orders ORDER BY id;**

id	total_amount
1	0.00
2	0.00

(2 rows)

Практика №28

1. Измените функцию `get_catalog` так, чтобы динамически формируемый текст запроса записывался в журнал сообщений сервера.

В приложении выполните несколько раз поиск, заполняя разные поля, и убедитесь, что команды SQL формируются правильно.

2. Включите трассировку команд SQL на уровне сервера.

Поработайте в приложении и проверьте, какие команды попадают в журнал сообщений.

Выключите трассировку.

Решение

2. Для включения трассировки установите значение параметра `log_min_duration_statement` в 0 и перечитайте конфигурацию. В журнал будут записываться все команды и время их выполнения.

Проще всего это сделать командой `ALTER SYSTEM SET`. Другие способы рассматривались в теме «Обзор базового инструментария. Установка и управление, `psql`». Не забудьте перечитать конфигурационный файл.

После просмотра журнала следует вернуть значение параметра `log_min_duration_statement` в значение по умолчанию (-1), чтобы отключить трассировку. Удобный способ — команда `ALTER SYSTEM RESET`.

1. Функция `get_catalog`

Текст динамического запроса формируем в отдельной переменной, которую перед выполнением запишем в журнал сервера. Для более полной информации включим в сообщение значения переданных в функцию параметров.

Отладочные строки в журнале можно найти по тексту «`DEBUG get_catalog`».

После отладки команду `RAISE LOG` можно удалить или закомментировать.

```
=> CREATE OR REPLACE FUNCTION get_catalog( author_name text,  
    book_title text,  
    in_stock boolean  
)  
    RETURNS TABLE(book_id integer, display_name text, onhand_qty integer)  
AS $$  
DECLARE  
    title_cond text := " "; author_cond text := " "; qty_cond text := " "; cmd text := " ;
```

```

BEGIN
  IF book_title != '' THEN
    title_cond := format(
      ' AND cv.title ILIKE %L', '%' || book_title || '%'
    );
  END IF;
  IF author_name != '' THEN
    author_cond := format(
      ' AND cv.authors ILIKE %L', '%' || author_name || '%'
    );
  END IF;
  IF in_stock THEN
    qty_cond := ' AND cv.onhand_qty > 0';
  END IF;
  cmd := ' SELECT cv.book_id, cv.display_name, cv.onhand_qty FROM catalog_v
cv WHERE true'
    || title_cond || author_cond || qty_cond
    || ' ORDER BY display_name';
  RAISE LOG 'DEBUG get_catalog (%, %, %): %', author_name, book_title,
in_stock, cmd;
  RETURN QUERY EXECUTE cmd;
END;
$$ STABLE LANGUAGE plpgsql;

```

CREATE FUNCTION

2. Включение и выключение трассировки SQL-запросов

Чтобы включить трассировку всех запросов на уровне сервера, можно выполнить:

=> **ALTER SYSTEM SET** log_min_duration_statement = 0;

ALTER SYSTEM

=> **SELECT** pg_reload_conf();

pg_reload_conf

t

(1 row)

Чтобы выключить:

=> **ALTER SYSTEM RESET** log_min_duration_statement;

ALTER SYSTEM

=> **SELECT** pg_reload_conf();

pg_reload_conf

t

(1 row)

Последние две команды попали в журнал сообщений:

student\$ tail -n 6 /var/log/postgresql/postgresql-12-main.log

2021-10-19 23:28:26.330 MSK [88059] LOG: received SIGHUP, reloading configuration files

2021-10-19 23:28:26.331 MSK [88059] LOG: parameter "log_min_duration_statement" changed to "0"

2021-10-19 23:28:26.394 MSK [97976] student@bookstore LOG: duration: 5.749 ms statement: ALTER SYSTEM RESET log_min_duration_statement;

2021-10-19 23:28:26.426 MSK [97976] student@bookstore LOG: duration: 0.105 ms statement: SELECT pg_reload_conf();

2021-10-19 23:28:26.426 MSK [88059] LOG: received SIGHUP, reloading configuration files

2021-10-19 23:28:26.427 MSK [88059] LOG: parameter "log_min_duration_statement" removed from configuration file, reset to default

Практика №29

1. Включите трассировку PL/pgSQL-кода средствами расширения `plpgsql_check` и проверьте ее работу на примере нескольких подпрограмм, вызывающих одна другую.
2. При выводе отладочных сообщений из PL/pgSQL-кода удобно понимать, к какой подпрограмме они относятся. В демонстрации имя функции выводилось вручную. Реализуйте функционал, автоматически добавляющий к тексту сообщений имя текущей функции или процедуры.

Решение

1. Для включения трассировки загрузите расширение `plpgsql_check` в память сеанса командой `LOAD`, затем установите в сеансе оба параметра `plpgsql_check.enable_tracer` и `plpgsql_check.tracer` в значение «on».
2. Имя подпрограммы можно получить, разобрав стек вызовов. Воспользуйтесь результатами практического задания 3 к теме «Обработка ошибок».

1. Трассировка с помощью `plpgsql_check`

=> **CREATE DATABASE** `plpgsql_debug`;

CREATE DATABASE

=> **\c** `plpgsql_debug`

You are now connected to database "plpgsql_debug" as user "student".

Загрузим расширение (в данном случае устанавливать его в базу данных командой `CREATE EXTENSION` не нужно):

=> **LOAD** `'plpgsql_check'`;

LOAD

Включим трассировку:

=> **SET** `plpgsql_check.enable_tracer = on`;

SET

=> **SET** plpgsql_check.tracer = on;

SET

Несколько функций, вызывающих друг друга:

```
=> CREATE FUNCTION foo(n integer) RETURNS integer
AS $$
BEGIN
    RETURN bar(n-1);
END;
$$ LANGUAGE plpgsql;
```

CREATE FUNCTION

```
=> CREATE FUNCTION bar(n integer) RETURNS integer
AS $$
BEGIN
    RETURN baz(n-1);
END;
$$ LANGUAGE plpgsql;
```

CREATE FUNCTION

```
=> CREATE FUNCTION baz(n integer) RETURNS integer
AS $$
BEGIN
    RETURN n;
END;
$$ LANGUAGE plpgsql;
```

CREATE FUNCTION

Пример работы трассировки:

```
=> SELECT foo(3);
```

```

NOTICE: #0 -> start of function foo(integer) (oid=24851)
NOTICE: #0      "n" => '3'
NOTICE: #1 -> start of function bar(integer) (oid=24852)
NOTICE: #1      call by foo(integer) line 3 at RETURN
NOTICE: #1      "n" => '2'
NOTICE: #2 -> start of function baz(integer) (oid=24853)
NOTICE: #2      call by bar(integer) line 3 at RETURN
NOTICE: #2      "n" => '1'
NOTICE: #2      <<- end of function baz (elapsed time=0.026 ms)
NOTICE: #1      <<- end of function bar (elapsed time=0.149 ms)
NOTICE: #0      <<- end of function foo (elapsed time=0.613 ms)
foo
-----
1
(1 row)

```

Выводятся не только события начала и окончания работы функций, но и значения параметров, а также затраченное время (в расширении есть и возможность профилирования, которую мы не рассматриваем).

Выключим трассировку:

```
=> SET plpgsql_check.tracer = off;
```

SET

2. Имя функции в отладочных сообщениях

Напишем процедуру, которая выводит верхушку стека вызовов (за исключением самой процедуры трассировки). Сообщение выводится с отступом, который соответствует глубине стека.

```

=> CREATE PROCEDURE raise_msg(msg text)
AS $$
DECLARE
    ctx text;
    stack text[];
BEGIN
    GET DIAGNOSTICS ctx = pg_context;
    stack := regexp_split_to_array(ctx, E'\n');
    RAISE NOTICE '?: %',
        repeat('.', array_length(stack,1)-2) || stack[3], msg;
END;
$$ LANGUAGE plpgsql;

```

CREATE PROCEDURE

Пример работы трассировки:

```
=> CREATE TABLE t(n integer);
```

CREATE TABLE

```
=> CREATE FUNCTION on_insert() RETURNS trigger
AS $$
BEGIN
    CALL raise_msg('NEW = '||NEW::text);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

CREATE FUNCTION

```
=> CREATE TRIGGER t_before_row
BEFORE INSERT ON t
FOR EACH ROW
EXECUTE FUNCTION on_insert();
```

CREATE TRIGGER

```
=> CREATE PROCEDURE insert_into_t()
AS $$
BEGIN
    CALL raise_msg('start');
    INSERT INTO t SELECT id FROM generate_series(1,3) id;
    CALL raise_msg('end');
END;
$$ LANGUAGE plpgsql;
```

CREATE PROCEDURE

```
=> CALL insert_into_t();
```

```
NOTICE: . PL/pgSQL function insert_into_t() line 3 at CALL: start
NOTICE: . . . PL/pgSQL function on_insert() line 3 at CALL: NEW = (1)
NOTICE: . . . PL/pgSQL function on_insert() line 3 at CALL: NEW = (2)
NOTICE: . . . PL/pgSQL function on_insert() line 3 at CALL: NEW = (3)
NOTICE: . PL/pgSQL function insert_into_t() line 5 at CALL: end
CALL
```

Практика №30

1. Создайте две роли (пароль должен совпадать с именем):

– employee — сотрудник магазина,

– buyer — покупатель.

Убедитесь, что созданные роли могут подключиться к БД.

2. Отзовите у роли public права выполнения всех функций и подключения к БД.

3. Разграничьте доступ таким образом, чтобы:

– сотрудник мог только заказывать книги, а также – добавлять авторов и книги,

– покупатель мог только приобретать книги.

Проверьте выполненные настройки в приложении.

Решение

1. Сотрудник — внутренний пользователь приложения, аутентификация выполняется на уровне СУБД.

Покупатель — внешний пользователь. В реальном интернет-магазине управление такими пользователями ложится на приложение, а все запросы поступают в СУБД от одной «обобщенной» роли (buyer). Идентификатор конкретного покупателя может передаваться как параметр (но в нашем приложении мы этого не делаем).

3. Вообще говоря, разграничение доступа должно быть заложено

и в приложение. В нашем учебном приложении разграничение не сделано специально: вместо этого на веб-странице можно явно выбрать роль, от имени которой пойдет запрос в СУБД. Это позволяет посмотреть, как поведет себя серверная часть при некорректной работе приложения.

Итак, пользователям нужно выдать:

- Право подключения к БД bookstore и доступ к схеме bookstore.

- Доступ к представлениям, к которым происходит непосредственное обращение.

- Доступ к функциям, которые вызываются как часть API. Если оставить функции SECURITY INVOKER, придется выдавать доступ и ко всем «нижележащим» объектам (таблицам, другим функциям). Однако удобнее просто объявить API-функции как SECURITY DEFINER.

Разумеется, ролям нужно выдать привилегии только на те объекты, доступ к которым у них должен быть.

1. Создание ролей

```
=> CREATE ROLE employee LOGIN PASSWORD 'employee';CREATE ROLE  
=> CREATE ROLE buyer LOGIN PASSWORD 'buyer';CREATE ROLE
```

Настройки по умолчанию разрешают подключение с локального адреса по паролю. Нас это устраивает.

2. Привилегии public

У роли public надо отозвать лишние привилегии.

```
=> REVOKE EXECUTE ON ALL FUNCTIONS IN SCHEMA bookstore FROM  
public;REVOKE  
=> REVOKE CONNECT ON DATABASE bookstore FROM public;  
REVOKE
```

3. Разграничение доступа

Функции с правами владельца.

```
=> ALTER FUNCTION get_catalog(text,text,boolean) SECURITY DEFINER;
```

ALTER FUNCTION

```
=> ALTER FUNCTION update_catalog() SECURITY DEFINER;
```

ALTER FUNCTION

```
=> ALTER FUNCTION add_author(text,text,text) SECURITY DEFINER;
```

ALTER FUNCTION

```
=> ALTER FUNCTION add_book(text,integer[]) SECURITY DEFINER;
```

ALTER FUNCTION

```
=> ALTER FUNCTION buy_book(integer) SECURITY DEFINER;
```

ALTER FUNCTION

```
=> ALTER FUNCTION book_name(integer,text,integer) SECURITY DEFINER;
```

ALTER FUNCTION

=> **ALTER FUNCTION** **authors**(books) **SECURITY DEFINER**;

ALTER FUNCTION

Привилегии покупателя: покупатель должен иметь доступ к поиску книг и их покупке.

=> **GRANT CONNECT ON DATABASE** bookstore **TO** buyer;

GRANT

=> **GRANT USAGE ON SCHEMA** bookstore **TO** buyer;

GRANT

=> **GRANT EXECUTE ON FUNCTION** **get_catalog**(text,text,boolean) **TO** buyer;

GRANT

=> **GRANT EXECUTE ON FUNCTION** **buy_book**(integer) **TO** buyer;

GRANT

Привилегии сотрудника: сотрудник должен иметь доступ к просмотру и добавлению книг и авторов, а также к каталогу для заказа книг.

=> **GRANT CONNECT ON DATABASE** bookstore **TO** employee;

GRANT

=> **GRANT USAGE ON SCHEMA** bookstore **TO** employee;

GRANT

=> **GRANT SELECT,UPDATE**(onhand_qty) **ON** catalog_v **TO** employee;

GRANT

=> **GRANT SELECT ON** authors_v **TO** employee;

GRANT

=> **GRANT EXECUTE ON FUNCTION** **book_name**(integer,text,integer) **TO** employee;

GRANT

=> GRANT EXECUTE ON FUNCTION authors(books) TO employee;

GRANT

=> GRANT EXECUTE ON FUNCTION author_name(text,text,text) TO employee;

GRANT

=> GRANT EXECUTE ON FUNCTION add_book(text,integer[]) TO employee;

GRANT

=> GRANT EXECUTE ON FUNCTION add_author(text,text,text) TO employee;

GRANT

Практика №31

Подпрограммы, объявленные как выполняющиеся с правами владельца (SECURITY DEFINER), могут использоваться, чтобы предоставить обычным пользователям возможности, доступные только суперпользователю.

1. Создайте обычного непривилегированного пользователя и проверьте, что он не может изменять значение параметра `log_statement`.
2. Напишите подпрограмму для включения и выключения трассировки SQL-запросов так, чтобы созданная роль могла ей воспользоваться.

Решение

1. Вспомните демонстрацию к теме «PL/pgSQL. Отладка». В ней не возникало сложностей с установкой параметра, поскольку демонстрация выполнялась от имени роли `student`, которая является суперпользователем.

1. Создание роли и проверка

```
student=# CREATE DATABASE access_overview;
```

```
CREATE DATABASE
```

```
student=# \c access_overview
```

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

```
student=# CREATE ROLE alice LOGIN PASSWORD 'alicepass';
```

```
CREATE ROLE
```

```
student$ psql "host=localhost user=alice dbname=access_overview  
password=alicepass"
```

Алиса не может изменить значение параметра:

```
alice=> SET log_statement = 'all';
```

```
ERROR: permission denied to set parameter "log_statement"
```

2. Процедура для трассировки

От имени суперпользователя создаем процедуру для изменения параметра и объявляем ее SECURITY DEFINER:

```
student=# CREATE PROCEDURE trace(val boolean)
AS $$
SELECT set_config(
    'log_statement',
    CASE WHEN val THEN 'all' ELSE 'none' END,
    false /* is_local */
);
$$ LANGUAGE sql SECURITY DEFINER;
```

CREATE PROCEDURE

Отбираем права на выполнение у роли public...

```
student=# REVOKE EXECUTE ON PROCEDURE trace FROM public;
```

REVOKE

...и выдаем Алисе. Вместо того, чтобы выбирать между FUNCTION и PROCEDURE, почти все команды (кроме CREATE) позволяют использовать общее слово ROUTINE:

```
student=# GRANT EXECUTE ON ROUTINE trace TO alice;
```

GRANT

Теперь Алиса может включать и выключать трассировку, хотя и не имеет непосредственного доступа к параметру:

```
alice=> CALL trace(true);
```

CALL

```
alice=> SELECT 2*2;
```

```
   ?column?
-----
```

```
4
```

```
(1 row)
```

```
alice=> CALL trace(false);
```

CALL

```
student$ tail -n 2 /var/log/postgresql/postgresql-12-main.log
```

2021-10-19 23:28:34.856 MSK [99435] alice@access_overview LOG: statement:

SELECT 2*2;

2021-10-19 23:28:34.917 MSK [99435] alice@access_overview LOG: statement: CALL
trace(false);

Практика №32

1. Создайте резервную копию базы данных bookstore в формате custom.

«Случайно» удалите все записи из таблицы authorship. Проверьте, что приложение перестало отображать названия книг на вкладках «Магазин», «Книги», «Каталог».

Используйте резервную копию для восстановления потерянных данных в таблице.

Проверьте, что нормальная работа книжного магазина восстановилась.

Решение

1. При восстановлении используйте ключ --data-only, чтобы избежать ошибки при попытке создания таблицы.

1. Восстановление потерянных данных

Создание резервной копии:

```
student$ pg_dump --format=custom -d bookstore > /home/student/bookstore.custom
```

Удаляем строки:

```
=> DELETE FROM authorship;
```

```
DELETE 9
```

Восстановление:

```
student$ pg_restore -t authorship --data-only -d bookstore /home/student/bookstore.custom
```

```
=> SELECT count(*) FROM authorship;
```

```
count
```

```
-----
```

```
9
```

```
(1 row)
```

Практика №33

1. Создайте таблицу с политикой, разрешающей чтение только части строк. Создайте непривилегированного пользователя для Алисы и предоставьте ей доступ к таблице.

В обязанности Алисы входит резервное копирование таблицы. Сможет ли она выполнять их, не являясь суперпользователем? Проверьте.

2. Команда `psql \copy` позволяет направить результат на вход произвольной программы. Воспользуйтесь этим, чтобы открыть результаты какого-нибудь запроса в электронной таблице LibreOffice Calc.

Решение

1. К роли с правами суперпользователя не применяются политики защиты строк. Однако Алиса, как непривилегированный пользователь, сможет прочитать лишь часть строк таблицы и даже не узнает о том, что это не все данные.

Параметр `row_security` позволит Алисе хотя бы узнать о том, что не удалось прочитать все данные. А атрибут роли `BYPASSRLS` решит задачу.

2. Команда должна перенаправить результат в файл, а затем запустить `libreoffice`, указав этот файл в качестве параметра. Файл должен быть записан в формате CSV.

Конечно, такой способ зависит от платформы и без модификации не будет работать, например, в Windows.

1. Политики защиты строк

Создаем таблицу с политикой и роль.

```
=> CREATE DATABASE backup_logical;
```

```
CREATE DATABASE
```

```
=> \c backup_logical
```

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

```
=> CREATE TABLE t(  
    id integer PRIMARY KEY GENERATED ALWAYS AS IDENTITY,  
    s text  
);
```

CREATE TABLE

=> **INSERT INTO** t(s) **VALUES** ('foo'), ('bar'), ('baz');

INSERT 0 3

=> **CREATE POLICY** odd **ON** t **USING** (mod(id,2) = 1);

CREATE POLICY

=> **ALTER TABLE** t **ENABLE ROW LEVEL SECURITY**;

ALTER TABLE

=> **CREATE ROLE** alice **LOGIN PASSWORD** 'alicepass';

CREATE ROLE

=> **GRANT SELECT ON** t **TO** alice;

GRANT

Алиса пытается прочесть таблицу:

```
student$ psql "host=localhost user=alice dbname=backup_logical  
password=alicepass"
```

```
alice=> COPY t TO stdout; -- или SELECT * FROM t;
```

```
1 foo
```

```
3 baz
```

С выключенным параметром row_security Алиса получит ошибку, если политики запрещают видеть часть строк:

```
alice=> SET row_security = off;
```

SET

```
alice=> COPY t TO stdout;
```

ERROR: query would be affected by row-level security policy for table "t"

Для того чтобы Алиса могла обходить политики защиты строк, не являясь суперпользователем, к ее роли надо добавить атрибут BYPASSRLS:

=> **ALTER ROLE** alice **BYPASSRLS**;

ALTER ROLE

alice=> **COPY t TO** stdout;

1 foo
2 bar
3 baz

2. Открытие результата запроса в LibreOffice

Попробуйте такую команду:

\copy t TO PROGRAM 'cat > /home/student/t.csv; libreoffice /home/student/t.csv'
WITH (format csv);

Если вместо \copy использовать SQL-команду COPY, программа будет запущена на сервере СУБД, что, конечно, неправильно.