Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчет

по лабораторной работе №5 «Процедуры, функции, триггеры в PostgreSQL»

по дисциплине **«Проектирование и реализация баз данных»**

Автор: Цатинян А.А.

Факультет: ПИН

Группа: К3239

Преподаватель: Говорова М.М.

Санкт-Петербург 2025

Оглавление

Цель работы

Овладеть практическими создания и использования процедур, функций и триггеров в базе данных PostgreSQL.

Практическое задание

1. Создать 3 процедуры для индивидуальной БД согласно варианту (часть 4 ЛР 2). Допустимо использование IN/OUT параметров. Допустимо создать авторские процедуры. (3 балла)
2. Создать триггеры для индивидуальной БД согласно варианту:
   Вариант 2.1. 3 триггера - 3 балла (min). Допустимо использовать триггеры логирования из практического занятия по функциям и триггерам.
   Вариант 2.2. 7 оригинальных триггеров - 7 баллов (max).

Выполнение

Процедуры и функции

Для снижения цены на заданный процент для товаров, у которых срок пребывания на складе превысил заданный норматив.

Код

```
CREATE
OR REPLACE PROCEDURE opt.reduce_stale_product_price_in_pending_orders(
    discount_percentage INTEGER,
    days_on_warehouse_limit INTEGER
)
LANGUAGE plpgsql
AS $$
DECLARE
stale_product RECORD;
    order_item_to_update
RECORD;
    v_discount_multiplier
NUMERIC;
BEGIN
    v_discount_multiplier
:= (100.0 - discount_percentage) / 100.0;
UPDATE opt.order_item oi
SET unit_price = ROUND(oi.unit_price * v_discount_multiplier)
WHERE oi.order_id IN (SELECT co.order_id FROM opt.customer_order co WHERE co.payment_status =
'Pending')
  AND EXISTS (SELECT 1
              FROM opt.delivery_item di
                       JOIN opt.delivery d ON di.delivery_id = d.delivery_id
              WHERE di.product_id = oi.product_id
                AND d.delivery_date < (CURRENT_DATE - (days_on_warehouse_limit || '
days')::INTERVAL));

RAISE
NOTICE 'Цены на залежавшиеся товары в неоплаченных заказах обновлены с % скидкой.',
discount_percentage;
END;
$$;
```

Запрос на получение данных

```sql
SELECT oi.order_item_id, oi.order_id, co.payment_status, oi.product_id, oi.unit_price,
d.delivery_date
FROM opt.order_item oi
        JOIN opt.customer_order co ON oi.order_id = co.order_id
        JOIN opt.product p ON oi.product_id = p.product_id
        JOIN opt.delivery_item di ON p.product_id = di.product_id
        JOIN opt.delivery d ON di.delivery_id = d.delivery_id
WHERE oi.order_id = 'O001'
  AND oi.product_id = 'P001'
ORDER BY d.delivery_date ASC LIMIT 1;
```

До применения

```
 order_item_id | order_id | payment_status | product_id | unit_price | delivery_date
---------------+----------+----------------+------------+------------+---------------
 OI001         | O001     | Pending        | P001       |        403 | 2023-08-10
(1 строка)
```

После применения

```
sales_base=# CALL opt.reduce_stale_product_price_in_pending_orders(10, 90);
ЗАМЕЧАНИЕ:  Цены на залежавшиеся товары в неоплаченных заказах обновлены с 10 скидкой.
CALL
sales_base=# SELECT order_item_id, order_id, product_id, unit_price
sales_base-# FROM opt.order_item
sales_base-# WHERE order_id = 'O001' AND product_id = 'P001';
 order_item_id | order_id | product_id | unit_price
---------------+----------+------------+------------
 OI001         | O001     | P001       |        363
(1 строка)
```

Для расчета стоимости всех партий товаров, проданных за прошедшие сутки

Код

```sql
CREATE
OR REPLACE FUNCTION opt.calculate_sales_yesterday()
RETURNS TABLE(total_sales_yesterday NUMERIC)
LANGUAGE sql
AS $$
SELECT COALESCE(SUM(oi.quantity_in_order * oi.unit_price), 0)
FROM opt.order_item oi
        JOIN opt.customer_order co ON oi.order_id = co.order_id
WHERE co.order_date = (CURRENT_DATE - INTERVAL '1 day');
$$;
```

Проверим наличие данных

```sql
SELECT
    co.order_id,
    co.order_date,
    (oi.quantity_in_order * oi.unit_price) AS item_total
FROM opt.customer_order co
        JOIN opt.order_item oi ON co.order_id = oi.order_id
WHERE co.order_date = (CURRENT_DATE - INTERVAL '1 day');
```

```
 order_id   | order_date | item_total
------------+------------+------------
 O_13MAY_1 | 2025-05-13 |        800
 O_13MAY_2 | 2025-05-13 |       1200
(2 строки)
```

Применение

```
SELECT * FROM opt.calculate_sales_yesterday();
```

```
sales_base=# SELECT * FROM opt.calculate_sales_yesterday();
 total_sales_yesterday
-----------------------
                  2000
(1 строка)
```

Триггеры

Логирование изменений цены в order_item

Триггер 1: Создание таблицы для логов

```
CREATE TABLE opt.price_change_log
(
    log_id            SERIAL PRIMARY KEY,
    order_item_id_ref CHARACTER VARYING(10) NOT NULL,
    product_id_ref    CHARACTER VARYING(10) NOT NULL,
    old_price         INTEGER,
    new_price         INTEGER,
    change_timestamp  TIMESTAMP WITHOUT TIME ZONE DEFAULT CURRENT_TIMESTAMP
);
```

```
sales_base=# CREATE TABLE opt.price_change_log
sales_base-# (
sales_base(#     log_id            SERIAL PRIMARY KEY,
sales_base(#     order_item_id_ref CHARACTER VARYING(10) NOT NULL,
sales_base(#     product_id_ref    CHARACTER VARYING(10) NOT NULL,
sales_base(#     old_price         INTEGER,
sales_base(#     new_price         INTEGER,
sales_base(#     change_timestamp  TIMESTAMP WITHOUT TIME ZONE DEFAULT CURRENT_TIMESTAMP
sales_base(# );
CREATE TABLE
sales_base=#
```

Создание триггерной функции

```
CREATE
OR REPLACE FUNCTION opt.log_order_item_price_change()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
    IF
NEW.unit_price IS DISTINCT FROM OLD.unit_price THEN
        INSERT INTO opt.price_change_log (order_item_id_ref, product_id_ref, old_price,
new_price)
```

```
        VALUES (OLD.order_item_id, OLD.product_id, OLD.unit_price, NEW.unit_price);
END IF;
RETURN NEW;
END;
$$;
```

```
sales_base$# $$;
CREATE FUNCTION
sales_base=#
```

Создание триггера

```
CREATE TRIGGER trg_log_price_after_update_order_item
    AFTER UPDATE OF unit_price ON opt.order_item
    FOR EACH ROW
    WHEN (OLD.unit_price IS DISTINCT FROM NEW.unit_price)
EXECUTE FUNCTION opt.log_order_item_price_change();
```

```
CREATE TRIGGER
sales_base=#
```

Тестирование

```
SELECT * FROM opt.price_change_log;
```

До

```
sales_base=# SELECT * FROM opt.price_change_log;
 log_id | order_item_id_ref | product_id_ref | old_price | new_price | change_timestamp
--------+-------------------+----------------+-----------+-----------+------------------
(0 строк)


sales_base=#
```

Изменим цену

```
UPDATE opt.order_item SET unit_price = 420 WHERE order_item_id = 'OI_13MA_1';
```

После

```
UPDATE 1
sales_base=# SELECT * FROM opt.price_change_log;
 log_id | order_item_id_ref | product_id_ref | old_price | new_price |      change_timestamp
--------+-------------------+----------------+-----------+-----------+----------------------------
      1 | OI_13MA_1         | P001           |       400 |       420 | 2025-05-14 18:06:53.282068
(1 строка)


sales_base=#
```

Триггер 2: Проверка остатка товара перед добавлением в order_item

Триггерная функция будет вызываться перед INSERT в order_item.

Она посчитает актуальный остаток товара (приход - расход).

Если запрашиваемое количество (NEW.quantity_in_order) больше остатка, она вызовет ошибку и не даст вставить строку

Создание триггерной функции

```
CREATE
OR REPLACE FUNCTION opt.check_stock_before_order_item_insert()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
DECLARE
v_current_stock INTEGER;
    v_product_name
CHARACTER VARYING;
BEGIN
SELECT COALESCE(SUM(CASE WHEN type = 'delivery' THEN qty ELSE -qty END), 0)
INTO v_current_stock
FROM (SELECT product_id, quantity_in_delivery as qty, 'delivery' as type
      FROM opt.delivery_item
      WHERE product_id = NEW.product_id
      UNION ALL
      SELECT product_id, quantity_in_order as qty, 'order' as type
      FROM opt.order_item
      WHERE product_id = NEW.product_id) AS stock_movements;

IF
NEW.quantity_in_order > v_current_stock THEN
SELECT name
INTO v_product_name
FROM opt.product
WHERE product_id = NEW.product_id;
RAISE
EXCEPTION 'Недостаточно товара "%" (ID: %) на складе. Доступно: %, Запрошено: %',
                      v_product_name, NEW.product_id, v_current_stock, NEW.quantity_in_order;
END IF;

RETURN NEW;
END;
$$;
```

Создание триггера

```
CREATE TRIGGER trg_check_stock_before_insert_order_item
    BEFORE INSERT ON opt.order_item
    FOR EACH ROW
    EXECUTE FUNCTION opt.check_stock_before_order_item_insert();
```

Тестирование триггера

Проверяем актуальный остаток

```
SELECT product_id, SUM(quantity_change) AS calculated_stock
FROM (SELECT product_id, quantity_in_delivery AS quantity_change
      FROM opt.delivery_item
      WHERE product_id = 'P001'
      UNION ALL
      SELECT product_id, -quantity_in_order AS quantity_change
      FROM opt.order_item
      WHERE product_id = 'P001') AS movements
GROUP BY product_id;
```

```
 product_id | calculated_stock
------------+------------------
 P001       |              494
(1 строка)
```

Попытка вставить больше

```
INSERT INTO opt.order_item (order_item_id, product_id, order_id, unit_price, quantity_in_order)
VALUES ('OI_TEST_ST', 'P001', 'O_13MAY_1', 500, 504);
```

```
sales_base=# INSERT INTO opt.order_item (order_item_id, product_id, order_id, unit_price, quantity_in_order)
sales_base-# VALUES ('OI_TEST_ST', 'P001', 'O_13MAY_1', 500, 504);
ОШИБКА:  Недостаточно товара "Galaxy S20" (ID: P001) на складе. Доступно: 494, Запрошено: 504
КОНТЕКСТ:  функция PL/pgSQL opt.check_stock_before_order_item_insert(), строка 23, оператор RAISE
sales_base=#
```

Попытка вставить меньше остатка

```
INSERT INTO opt.order_item (order_item_id, product_id, order_id, unit_price, quantity_in_order)
VALUES ('OI_TEST_SK', 'P001', 'O_13MAY_1', 500, 5);
```

```
sales_base=# INSERT INTO opt.order_item (order_item_id, product_id, order_id, unit_price, quantity_in_order)
sales_base-# VALUES ('OI_TEST_SK', 'P001', 'O_13MAY_1', 500, 5);
INSERT 0 1
sales_base=#
```

Триггер 3: Запрет удаления менеджера, если за ним числятся активные заказы.

Триггерная функция будет вызываться перед DELETE из таблицы opt.manager.

Она проверит, есть ли в opt.customer_order заказы, у которых manager_id совпадает с ID удаляемого менеджера, и статус которых не является "Completed" или "Delivered" .

Если такие заказы есть, триггер вызовет ошибку.

Создание триггерной функции

```
CREATE
OR REPLACE FUNCTION opt.prevent_delete_manager_with_active_orders()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
DECLARE
active_order_count INTEGER;
BEGIN
SELECT COUNT(*)
INTO active_order_count
FROM opt.customer_order
WHERE manager_id = OLD.manager_id
  AND order_status NOT IN ('Completed', 'Delivered', 'Cancelled');

IF
active_order_count > 0 THEN
        RAISE EXCEPTION 'Нельзя удалить менеджера (ID: %), так как за ним числятся активные
заказы (кол-во: %). Сначала переназначьте или завершите эти заказы.',
                        OLD.manager_id, active_order_count;
END IF;
```

```
RETURN OLD;
END;
$$;
```

Создание триггера

```
CREATE TRIGGER trg_prevent_delete_manager_active_orders
    BEFORE DELETE
    ON opt.manager
    FOR EACH ROW
    EXECUTE FUNCTION opt.prevent_delete_manager_with_active_orders();
```

Тестирование триггера

Попытка удалить этого менеджера

```
DELETE FROM opt.manager WHERE manager_id = 'MGR1';
ОШИБКА:  Нельзя удалить менеджера (ID: MGR1), так как за ним числятся активные заказы (кол-во: 9). Сначала переназначьте или завершите э
ти заказы.
КОНТЕКСТ:  функция PL/pgSQL opt.prevent_delete_manager_with_active_orders(), строка 13, оператор RAISE
sales_base=#
```

Триггер 4: Автоматическое обновление даты последнего заказа клиента.

Добавим в таблицу opt.customer новый столбец last_order_date DATE.

Триггерная функция будет вызываться после INSERT в opt.customer_order.

Она будет обновлять поле last_order_date в таблице opt.customer на дату нового заказа

Добавление столбца в таблицу

```
ALTER TABLE opt.customer ADD COLUMN last_order_date DATE;
```

Создание триггерной функции

```
CREATE
OR REPLACE FUNCTION opt.update_customer_last_order_date()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
UPDATE opt.customer
SET last_order_date = NEW.order_date
WHERE customer_id = NEW.customer_id;
RETURN NEW;
END;
$$;
```

Создание триггера

```
CREATE TRIGGER trg_update_last_order_date_after_insert_order
    AFTER INSERT
    ON opt.customer_order
    FOR EACH ROW
    EXECUTE FUNCTION opt.update_customer_last_order_date();
```

Тестирование триггера

Проверка последнего заказа

```
sales_base=# SELECT name, last_order_date FROM opt.customer WHERE customer_id = 'CUST5';
    name     | last_order_date
-------------+-----------------
 ITSolutions |
(1 строка)
```

Добавим заказ

```
INSERT INTO opt.customer_order (order_id, order_status, payment_status, order_date, address,
manager_id, customer_id)
VALUES ('O_CUST5_NW', 'Created', 'Pending', CURRENT_DATE, 'San Francisco, Market St 200', 'MGR5',
'CUST5');
```

Проверка последнего заказа

```
sales_base=# SELECT name, last_order_date FROM opt.customer WHERE customer_id = 'CUST5';
    name     | last_order_date
-------------+-----------------
 ITSolutions | 2025-05-14
(1 строка)


sales_base=#
```

Триггер 5: Запрет на изменение product_id в уже существующей позиции заказа

Часто бывает важно, чтобы после того, как позиция заказа создана, сам товар в ней не менялся.

Триггерная функция будет вызываться перед UPDATE в opt.order_item.

Если новое значение product_id отличается от старого, триггер вызовет ошибку

Создание триггерной функции

```
CREATE
OR REPLACE FUNCTION opt.prevent_order_item_product_change()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
    IF
NEW.product_id IS DISTINCT FROM OLD.product_id THEN
        RAISE EXCEPTION 'Изменение товара (product_id) в существующей позиции заказа (ID: %)
запрещено. Создайте новую позицию.',
                        OLD.order_item_id;
END IF;
RETURN NEW;
END;
$$;
```

Создание триггера

```
CREATE TRIGGER trg_prevent_product_change_in_order_item
    BEFORE UPDATE
    ON opt.order_item
    FOR EACH ROW
    EXECUTE FUNCTION opt.prevent_order_item_product_change();
```

Тестирование триггера

Проверка до

```
SELECT order_item_id, product_id, unit_price FROM opt.order_item WHERE order_item_id =
'OI_13MA_1';
```

```
sales_base=# SELECT order_item_id, product_id, unit_price FROM opt.order_item WHERE order_item_id = 'OI_13MA_1';
 order_item_id | product_id | unit_price
---------------+------------+------------
 OI_13MA_1     | P001       |        420
(1 строка)
```

Попытка изменить

```
UPDATE opt.order_item SET product_id = 'P002' WHERE order_item_id = 'OI_13MA_1';
```

```
sales_base=# UPDATE opt.order_item SET product_id = 'P002' WHERE order_item_id = 'OI_13MA_1';
ОШИБКА:  Изменение товара (product_id) в существующей позиции заказа (ID: OI_13MA_1) запрещено. Создайте новую позицию.
КОНТЕКСТ:  функция PL/pgSQL opt.prevent_order_item_product_change(), строка 5, оператор RAISE
sales_base=#
```

Попытка изменить другое поле, не трогая product_id

```
UPDATE opt.order_item SET unit_price = 425 WHERE order_item_id = 'OI_13MA_1';
```

```
sales_base=# UPDATE opt.order_item SET unit_price = 425 WHERE order_item_id = 'OI_13MA_1';
UPDATE 1
```

Проверка

```
SELECT order_item_id, product_id, unit_price FROM opt.order_item WHERE order_item_id =
'OI_13MA_1';
```

```
sales_base=# SELECT order_item_id, product_id, unit_price FROM opt.order_item WHERE order_item_id = 'OI_13MA_1';
 order_item_id | product_id | unit_price
---------------+------------+------------
 OI_13MA_1     | P001       |        425
(1 строка)
```

Триггер 6: Автоматическое выставление статуса заказа 'Processing', если добавляется первая позиция в заказ со статусом 'Created'

Когда заказ только создан, он может иметь статус 'Created'. Как только в него добавляют первый товар, логично перевести его в статус 'Processing'.

Триггерная функция будет вызываться после INSERT в opt.order_item.

Она проверит статус связанного заказа. Если он 'Created', изменит его на 'Processing'.

Создание триггерной функции

```
CREATE OR REPLACE FUNCTION opt.set_order_to_processing_on_first_item()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
UPDATE opt.customer_order
```

```
SET order_status = 'Processing'
WHERE order_id = NEW.order_id
  AND order_status = 'Created';
RETURN NEW;
END;
$$;
```

Создание триггера

```
CREATE TRIGGER trg_set_processing_after_first_item_insert
    AFTER INSERT
    ON opt.order_item
    FOR EACH ROW
    EXECUTE FUNCTION opt.set_order_to_processing_on_first_item();
```

Тестирование триггера

Создадим новый заказ со статусом 'Created'

```
INSERT INTO opt.customer_order (order_id, order_status, payment_status, order_date, address,
manager_id, customer_id)
VALUES ('O_NEW_CRTD', 'Created', 'Pending', CURRENT_DATE, 'Test Address 123', 'MGR1', 'CUST1');
```

Проверка статуса до добавления позиции

```
SELECT order_id, order_status FROM opt.customer_order WHERE order_id = 'O_NEW_CRTD';
```

```
INSERT 0 1
sales_base=# SELECT order_id, order_status FROM opt.customer_order WHERE order_id = 'O_NEW_CRTD';
  order_id  | order_status
------------+--------------
 O_NEW_CRTD | Created
(1 строка)
```

Добавим позицию

```
INSERT INTO opt.order_item (order_item_id, product_id, order_id, unit_price, quantity_in_order)
VALUES ('OI_NEW_CD1', 'P001', 'O_NEW_CRTD', 300, 1);
```

Проверка статуса

```
SELECT order_id, order_status FROM opt.customer_order WHERE order_id = 'O_NEW_CRTD';
```

```
sales_base=# SELECT order_id, order_status FROM opt.customer_order WHERE order_id = 'O_NEW_CRTD';
  order_id  | order_status
------------+--------------
 O_NEW_CRTD | Processing
(1 строка)
```

Триггер 7: Запрет устанавливать дату поставки в прошлом относительно текущей даты.

Предотвращает ошибки ввода, когда дата поставки указывается уже прошедшей.

Триггерная функция будет вызываться перед INSERT или UPDATE в opt.delivery.

Если NEW.delivery_date меньше CURRENT_DATE, вызывается ошибка

Создание триггерной функции

```
CREATE
OR REPLACE FUNCTION opt.prevent_past_delivery_date()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
    IF
NEW.delivery_date < CURRENT_DATE THEN
        RAISE EXCEPTION 'Дата поставки (delivery_date: %) не может быть в прошлом.',
                        NEW.delivery_date;
END IF;
RETURN NEW;
END;
$$;
```

Создание триггера

```
CREATE TRIGGER trg_prevent_past_delivery_date_on_insert_update
    BEFORE INSERT OR
UPDATE OF delivery_date
ON opt.delivery
    FOR EACH ROW
    EXECUTE FUNCTION opt.prevent_past_delivery_date();
```

Тестирование триггера

Попытка вставить поставку с датой в прошлом

```
INSERT INTO opt.delivery (delivery_id, delivery_date, warehouse_id, supplier_id, manager_id)
VALUES ('D_PAST_01', '2020-01-01', 'W001', 'SUP1', 'MGR1');
```

```
sales_base=# INSERT INTO opt.delivery (delivery_id, delivery_date, warehouse_id, supplie
sales_base-# VALUES ('D_PAST_01', '2020-01-01', 'W001', 'SUP1', 'MGR1');
ОШИБКА:  Дата поставки (delivery_date: 2020-01-01) не может быть в прошлом.
КОНТЕКСТ:  функция PL/pgSQL opt.prevent_past_delivery_date(), строка 5, оператор RAISE
sales_base=#
```

Попытка вставить поставку с сегодняшней или будущей датой

```
INSERT INTO opt.delivery (delivery_id, delivery_date, warehouse_id, supplier_id, manager_id)
VALUES ('D_VALID_01', CURRENT_DATE + INTERVAL '1 day', 'W001', 'SUP1', 'MGR1');
```

```
sales_base=# INSERT INTO opt.delivery (delivery_id, delivery_date, warehouse_id, supplier_id, manager_id)
sales_base-# VALUES ('D_VALID_01', CURRENT_DATE + INTERVAL '1 day', 'W001', 'SUP1', 'MGR1');
INSERT 0 1
sales_base=#
```

Вывод

В ходе выполнения лабораторной работы были успешно освоены практические навыки создания и использования хранимых процедур и триггеров в СУБД PostgreSQL. Были разработаны и протестированы процедуры для модификации цен на товары и расчета суточных продаж, а также созданы и проверены в действии семь оригинальных триггеров,

направленных на автоматизацию бизнес-логики, логирование изменений и поддержание целостности данных в базе "Оптовая база", включая проверку остатков, обновление статусов и запрет некорректных операций. Все разработанные объекты продемонстрировали ожидаемое поведение при тестировании в консольном клиенте psql.