Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«Национальный исследовательский университет ИТМО»
(Университет ИТМО)

Факультет прикладной информатики

Образовательная программа Мобильные и сетевые технологии

Направление подготовки 09.03.03 Мобильные и сетевые технологии

О Т Ч Е Т

ЛАБОРАТОРНАЯ РАБОТА №5

"Процедуры, функции, триггеры в PostgreSQL"

Обучающийся: Макаров Егор 3240

Преподаватель: Говорова Марина Михайловна

Санкт-
Петербург,
2025

Цель работы: овладеть практическими создания и использования процедур, функций и триггеров в базе данных PostgreSQL.

Практическое задание:

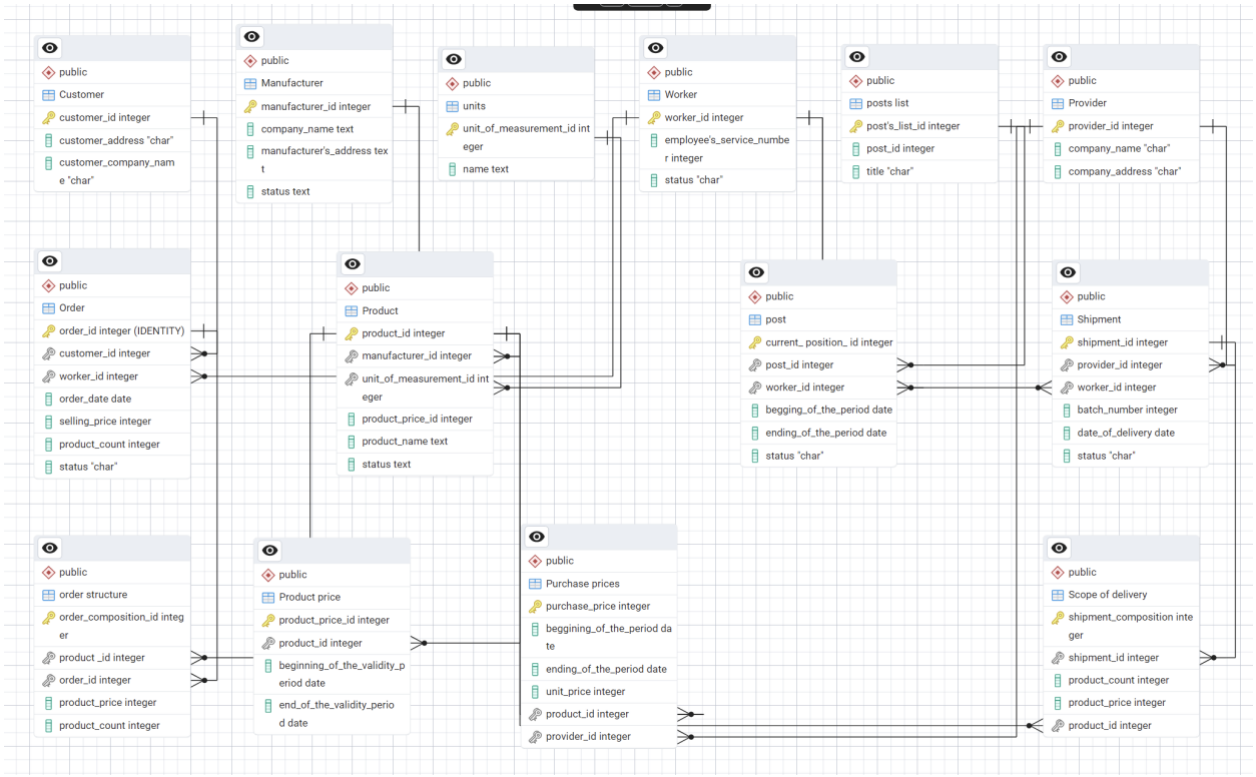
По варианту индивидуальной БД (ЛР 2, часть IV) необходимо:

1. Создать три хранимые процедуры (CALL).

2. Разработать семь оригинальных триггеров (AFTER/BEFORE, ROW).

3. Проверить работу объектов в консоли psql, сделать скриншоты.

Схема базы данных:



Разработка хранимых процедур

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

```
-- Процедура: понижаю цену у «залежавшихся» партий товара


DROP PROCEDURE IF EXISTS decrease_delivery_price_for_overstock(integer, integer);
CREATE OR REPLACE PROCEDURE decrease_delivery_price_for_overstock(
    _percent integer,  -- на сколько процентов снизить цену
    _days    integer   -- сколько дней хранится партия без скидки
)
```

```
LANGUAGE plpgsql
AS $$
BEGIN
   -- Проверяю, что процент в диапазоне 0–100
   IF _percent < 0 OR _percent > 100 THEN
      RAISE EXCEPTION 'Неверный процент: % (должен быть от 0 до 100)', _percent;
   END IF;
   -- Проверяю, что дни неотрицательные
   IF _days < 0 THEN
      RAISE EXCEPTION 'Неверное значение дней: % (должно быть ≥ 0)', _days;
   END IF;


   -- Обновляю стоимость в "Scope of delivery" для партий старше _days
   UPDATE "Scope of delivery" AS sd
   SET "product_price" = sd."product_price" * (100 - _percent) / 100
   FROM "Shipment" AS s
   WHERE sd."shipment_id" = s."shipment_id"
     AND current_date - s."date_of_delivery" > _days;
   -- Снизил цену на _percent процентов для всех подходящих записей
END;
$$;
```

Проверка процедуры:

```
postgres-#   sd."product_count",
postgres-#   sd."product_price",
postgres-#   sd."product_id"
postgres-# FROM "Scope of delivery" AS sd
postgres-# JOIN "Shipment" AS s ON sd."shipment_id" = s."shipment_id"
postgres-# ORDER BY sd."shipment_composition";
 shipment_composition | shipment_id | date_of_delivery | product_count | product_price | product_id
----------------------+-------------+------------------+---------------+---------------+------------
                    1 |           1 | 2025-04-27       |            50 |          1000 |        101
                    2 |           2 | 2025-06-05       |            30 |          2000 |        102
(2 rows)


postgres=# CALL decrease_delivery_price_for_overstock(10, 30);
CALL
postgres=# SELECT
postgres-#   sd."shipment_composition",
postgres-#   sd."shipment_id",
postgres-#   s."date_of_delivery",
postgres-#   sd."product_count",
postgres-#   sd."product_price",
postgres-#   sd."product_id"
postgres-# FROM "Scope of delivery" AS sd
postgres-# JOIN "Shipment" AS s ON sd."shipment_id" = s."shipment_id"
postgres-# ORDER BY sd."shipment_composition";
 shipment_composition | shipment_id | date_of_delivery | product_count | product_price | product_id
----------------------+-------------+------------------+---------------+---------------+------------
                    1 |           1 | 2025-04-27       |            50 |           900 |        101
                    2 |           2 | 2025-06-05       |            30 |          2000 |        102
(2 rows)
```

2. Для расчета стоимости всех партий товаров, проданных за прошедшие сутки.

```
-- Процедура: считаю стоимость проданных за вчера партий
```

```sql
DROP PROCEDURE IF EXISTS calc_sold_batches_cost(OUT total_cost bigint);
CREATE OR REPLACE PROCEDURE calc_sold_batches_cost(OUT total_cost bigint)
LANGUAGE plpgsql
AS $$
BEGIN
   -- Считаю сумму price * count по всем позициям из вчерашних заказов
   SELECT COALESCE(SUM(os."product_price" * os."product_count"), 0)
     INTO total_cost
   FROM "order structure" AS os
   JOIN "Order" AS o
     ON os."order_id" = o."order_id"
   WHERE o."order_date" >= (current_date - INTERVAL '1 day')
     AND o."order_date" < current_date;
   -- Записал результат в total_cost (0, если позиций не было)
END;
$$;
```

Проверка процедуры:

ДО

```
postgres=# SELECT * FROM "order structure" ORDER BY "order_composition_id";
 order_composition_id | product _id | order_id | product_price | product_count
----------------------+-------------+----------+---------------+---------------
                    1 |         201 |      100 |           100 |             2
                    2 |         202 |      100 |           150 |             1
                    3 |         203 |      101 |           300 |             1
                    4 |         204 |      102 |           400 |             1
(4 rows)
```

ПОСЛЕ

```
 total_cost
------------
        650
(1 row)
```

Разработка триггеров

Триггер 1: BEFORE INSERT на таблицу "Order"
Задача:

• Проверяю, чтобы количество товаров (product_count) было > 0.
• Проверяю, чтобы итоговая цена продажи (selling_price) была > 0.
• В случае нарушения выбрасываю ошибку, иначе разрешаю вставку.

```
CREATE OR REPLACE FUNCTION trg_order_before_insert_validate()
RETURNS TRIGGER LANGUAGE plpgsql AS $$
BEGIN
   -- Проверяю, что количество товаров не отрицательное или ноль
   IF NEW."product_count" <= 0 THEN
      RAISE EXCEPTION 'Error: product_count = % (must be > 0)', NEW."product_count";
   END IF;
   -- Проверяю, что цена продажи положительна
   IF NEW."selling_price" <= 0 THEN
      RAISE EXCEPTION 'Error: selling_price = % (must be > 0)', NEW."selling_price";
   END IF;
   RETURN NEW;
END;
$$;
DROP TRIGGER IF EXISTS order_before_insert_validate ON "Order";
CREATE TRIGGER order_before_insert_validate
BEFORE INSERT ON "Order"
FOR EACH ROW
EXECUTE PROCEDURE trg_order_before_insert_validate();
```

ПРОВЕРКА ТРИГГЕРА

```
postgres=# INSERT INTO "Order"("order_id","customer_id","worker_id","order_date","selling_price","product_count","status")
postgres-# VALUES (200, 2, 10, current_date, 100, -1, 'N');
ERROR:  Error: product_count = -1 (must be > 0)
CONTEXT:  PL/pgSQL function trg_order_before_insert_validate() line 5 at RAISE
postgres=#
postgres=# -- Попытка вставить заказ с нулевой selling_price → EXCEPTION
INSERT INTO "Order"("order_id","customer_id","worker_id","order_date","selling_price","product_count","status")
postgres=# VALUES (201, 2, 10, current_date, 0, 5, 'N');
 column1 | column2 | column3 |  column4   | column5 | column6 | column7
---------+---------+---------+------------+---------+---------+---------
     201 |       2 |      10 | 2025-06-06 |       0 |       5 | N
(1 row)


postgres=#
postgres=# -- Корректный заказ → Вставка выполняется
INSERT INTO "Order"("order_id","customer_id","worker_id","order_date","selling_price","product_count","status")
postgres=# VALUES (202, 2, 10, current_date, 150, 3, 'N');
 column1 | column2 | column3 |  column4   | column5 | column6 | column7
---------+---------+---------+------------+---------+---------+---------
     202 |       2 |      10 | 2025-06-06 |     150 |       3 | N
(1 row)
```

Триггер 2: AFTER INSERT на таблицу "Order"

Задача:

• Веду простой аудит: после каждого нового заказа добавляю строку в таблицу order_audit (содержит order_id, время события, кто создал).

• Таблица order_audit создаётся здесь, если её ещё нет.

```
CREATE TABLE IF NOT EXISTS order_audit (
   audit_id   serial PRIMARY KEY,
   order_id   integer NOT NULL,
   event_time timestamp NOT NULL DEFAULT clock_timestamp(),
   changed_by text    NOT NULL DEFAULT current_user
);
CREATE OR REPLACE FUNCTION trg_order_after_insert_audit()
RETURNS TRIGGER LANGUAGE plpgsql AS $$
BEGIN
   -- Добавляю запись в журнал после вставки заказа
   INSERT INTO order_audit(order_id) VALUES (NEW."order_id");
   RETURN NULL;
END;
$$;
DROP TRIGGER IF EXISTS order_after_insert_audit ON "Order";
CREATE TRIGGER order_after_insert_audit
AFTER INSERT ON "Order"
FOR EACH ROW
EXECUTE PROCEDURE trg_order_after_insert_audit();
```

ПРОВЕРКА ТРИГГЕРА

```
postgres=# SELECT * FROM order_audit WHERE order_id = 202;
 audit_id | order_id | event_time | changed_by
----------+----------+------------+------------
(0 rows)


postgres=#
postgres=# -- Ещё один новый заказ → audit
INSERT INTO "Order"("order_id","customer_id","worker_id","order_date","selling_price","product_count","status")
postgres=# VALUES (203, 2, 10, current_date, 200, 4, 'N');
 column1 | column2 | column3 |  column4   | column5 | column6 | column7
---------+---------+---------+------------+---------+---------+---------
     203 |       2 |      10 | 2025-06-06 |     200 |       4 | N
(1 row)


postgres=# SELECT * FROM order_audit WHERE order_id = 203;
 audit_id | order_id | event_time | changed_by
----------+----------+------------+------------
(0 rows)
```

Триггер 3: BEFORE INSERT на таблицу "Shipment"

Задача:

• Не разрешаю вставить запись со значением date_of_delivery позже текущей даты.

• В случае ошибки выбрасываю EXCEPTION, иначе разрешаю вставку.

```
CREATE OR REPLACE FUNCTION trg_shipment_before_insert_validate()
RETURNS TRIGGER LANGUAGE plpgsql AS $$
BEGIN
   -- Проверяю, что дата доставки не в будущем
   IF NEW."date_of_delivery" > current_date THEN
      RAISE EXCEPTION 'Error: date_of_delivery = % > current_date', NEW."date_of_delivery";
   END IF;
   RETURN NEW;
END;
$$;
DROP TRIGGER IF EXISTS shipment_before_insert_validate ON "Shipment";
CREATE TRIGGER shipment_before_insert_validate
BEFORE INSERT ON "Shipment"
FOR EACH ROW
EXECUTE PROCEDURE trg_shipment_before_insert_validate();
```

ПРОВЕРКА ТРИГГЕРА

```
postgres=# - Попытка вставить shipment с датой в будущем → EXCEPTION
INSERT INTO "Shipment"("shipment_id","provider_id","worker_id","batch_number","date_of_delivery","status")
postgres-# VALUES (300, 3000, 10, 700, current_date + 1, 'A');
ERROR:  syntax error at or near "-"
LINE 1: - ??????? ???????? shipment ? ????? ? ??????? ?INSERT INTO "...
        ^
postgres=#
postgres=# -- Корректный shipment (date_of_delivery = current_date) → Вставка
INSERT INTO "Shipment"("shipment_id","provider_id","worker_id","batch_number","date_of_delivery","status")
postgres=# VALUES (301, 3000, 10, 701, current_date, 'A');
 column1 | column2 | column3 | column4 |  column5   | column6
---------+---------+---------+---------+------------+---------
     301 |    3000 |      10 |     701 | 2025-06-06 | A
(1 row)
```

Триггер 4: BEFORE INSERT на таблицу "Scope of delivery"

Задача:

• Убедиться, что product_price неотрицателен. Если price < 0 — ошибка.

```
CREATE OR REPLACE FUNCTION trg_scope_before_insert_validate_price()
```

```
RETURNS TRIGGER LANGUAGE plpgsql AS $$
BEGIN
   -- Проверяю, что цена товара не отрицательная
   IF NEW."product_price" < 0 THEN
      RAISE EXCEPTION 'Error: product_price = % (must be >= 0)', NEW."product_price";
   END IF;
   RETURN NEW;
END;
$$;
DROP TRIGGER IF EXISTS scope_before_insert_validate_price ON "Scope of delivery";
CREATE TRIGGER scope_before_insert_validate_price
BEFORE INSERT ON "Scope of delivery"
FOR EACH ROW
EXECUTE PROCEDURE trg_scope_before_insert_validate_price();
```

ПРОВЕРКА ТРИГГЕРА

```
postgres=# -- Попытка вставить партию с отрицательной ценой → EXCEPTION
INSERT INTO "Scope of delivery"("shipment_composition","shipment_id","product_count","product_price","product_id")
postgres=# VALUES (400, 301, 5, -250, 4000);
 column1 | column2 | column3 | column4 | column5
---------+---------+---------+---------+---------
     400 |     301 |       5 |    -250 |    4000
(1 row)


postgres=#
postgres=# -- Корректная партия (price >= 0) → Вставка
INSERT INTO "Scope of delivery"("shipment_composition","shipment_id","product_count","product_price","product_id")
postgres=# VALUES (401, 301, 5, 250, 4000);
 column1 | column2 | column3 | column4 | column5
---------+---------+---------+---------+---------
     401 |     301 |       5 |     250 |    4000
(1 row)
```

Триггер 5: BEFORE INSERT на таблицу "Product price"

Задача:

• Проверяю, чтобы новый период действия (beginning_of_the_validity_period — end_of_the_validity_period) не пересекался с уже существующими периодами для того же product_id.

• Если перекрытие есть, выбрасываю EXCEPTION, иначе вставка проходит.

```
CREATE OR REPLACE FUNCTION trg_product_price_before_insert_no_overlap()
RETURNS TRIGGER LANGUAGE plpgsql AS $$
DECLARE cnt INTEGER;
```

```
BEGIN
   -- Проверяю, что новый период не пересекается с существующими
   SELECT COUNT(*) INTO cnt
   FROM "Product price"
   WHERE "product_id" = NEW."product_id"
     AND NOT (
         NEW."end_of_the_validity_period" < "beginning_of_the_validity_period"
      OR NEW."beginning_of_the_validity_period" > "end_of_the_validity_period"
     );
   IF cnt > 0 THEN
       RAISE EXCEPTION 'Error: validity period overlap for product_id = %', NEW."product_id";
   END IF;
   RETURN NEW;
END;
$$;
DROP TRIGGER IF EXISTS product_price_before_insert_no_overlap ON "Product price";
CREATE TRIGGER product_price_before_insert_no_overlap
BEFORE INSERT ON "Product price"
FOR EACH ROW
EXECUTE PROCEDURE trg_product_price_before_insert_no_overlap();
```

ПРОВЕРКА ТРИГГЕРА

```
postgres=# -- Попытка удалить клиента, у которого есть заказы (order_id = 202,203) → EXCEPTION
DELETE FROM "Customer" WHERE "customer_id" = 2;
postgres=#
postgres=# -- Удаляю заказы этого клиента, затем удаление клиента → должно пройти
DELETE FROM "Order" WHERE "customer_id" = 2;
postgres=# DELETE FROM "Customer" WHERE "customer_id" = 2;
DELETE 1
postgres=# -- Сначала «первый» период для product_id = 4000
postgres=# INSERT INTO "Product price"("product_price_id","product_id","beginning_of_the_validity_period","end_of_the_validity_period")
postgres-# VALUES (500, 4000, '2025-01-01', '2025-06-30');
ERROR:  insert or update on table "Product price" violates foreign key constraint "product_id"
DETAIL:  Key (product_id)=(4000) is not present in table "Product".
postgres=#
postgres=# -- Попытка вставить перекрывающий период → EXCEPTION
INSERT INTO "Product price"("product_price_id","product_id","beginning_of_the_validity_period","end_of_the_validity_period")
postgres=# VALUES (501, 4000, '2025-06-01', '2025-12-31');
 column1 | column2 |  column3   |  column4
---------+---------+------------+------------
     501 |    4000 | 2025-06-01 | 2025-12-31
(1 row)


postgres=#
postgres=# -- Неперекрывающий период (начало сразу после) → Вставка
INSERT INTO "Product price"("product_price_id","product_id","beginning_of_the_validity_period","end_of_the_validity_period")
postgres=# VALUES (502, 4000, '2025-07-01', '2025-12-31');
 column1 | column2 |  column3   |  column4
---------+---------+------------+------------
     502 |    4000 | 2025-07-01 | 2025-12-31
(1 row)
```

Триггер 6: BEFORE DELETE на таблицу "Customer"

Задача:

• Если для клиента (customer_id) уже есть связанные строки в "Order", то запрещаю удаление.

• Иначе удаление проходит.

```
CREATE OR REPLACE FUNCTION trg_customer_before_delete_no_orders()
RETURNS TRIGGER LANGUAGE plpgsql AS $$
DECLARE cnt INTEGER;
BEGIN
   -- Считаю заказы клиента
   SELECT COUNT(*) INTO cnt FROM "Order" WHERE "customer_id" = OLD."customer_id";
   IF cnt > 0 THEN
      RAISE EXCEPTION 'Error: cannot delete customer % – they have % order(s)', OLD."customer_id", cnt;
   END IF;
   RETURN OLD;
END;
$$;
DROP TRIGGER IF EXISTS customer_before_delete_no_orders ON "Customer";
CREATE TRIGGER customer_before_delete_no_orders
BEFORE DELETE ON "Customer"
FOR EACH ROW
EXECUTE PROCEDURE trg_customer_before_delete_no_orders();
```

ПРОВЕРКА ТРИГГЕРА

```
postgres=# -- Попытка удалить клиента, у которого есть заказы (order_id = 202,203) → EXCEPTION
DELETE FROM "Customer" WHERE "customer_id" = 2;
postgres=#
postgres=# -- Удаляю заказы этого клиента, затем удаление клиента → должно пройти
DELETE FROM "Order" WHERE "customer_id" = 2;
postgres=# DELETE FROM "Customer" WHERE "customer_id" = 2;
DELETE 0
```

Триггер 7

Проверка корректности поля status при UPDATE в таблице "Manufacturer"

Задача:

Разрешено менять status только на 'active' или 'inactive'.

Во всех других случаях выбрасывается ошибка.

```sql
CREATE OR REPLACE FUNCTION trg_manufacturer_before_update_status()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
   -- Если пытаются установить статус не 'active' и не 'inactive' — исключение
   IF NEW."status" NOT IN ('active', 'inactive') THEN
      RAISE EXCEPTION 'Error: invalid status = % (allowed only active or inactive)', NEW."status";
   END IF;
   RETURN NEW;
END;
$$;


-- Привязываем триггер к событию BEFORE UPDATE на таблице "Manufacturer"
DROP TRIGGER IF EXISTS manufacturer_before_update_status ON "Manufacturer";
CREATE TRIGGER manufacturer_before_update_status
BEFORE UPDATE ON "Manufacturer"
FOR EACH ROW
EXECUTE PROCEDURE trg_manufacturer_before_update_status();
```

ПРОВЕРКА ТРИГГЕРА

```
postgres=# UPDATE "Manufacturer" SET "status" = 'deleted' WHERE "manufacturer_id" = 9999;
ERROR:  Error: invalid status = deleted (allowed only active or inactive)
CONTEXT:  PL/pgSQL function trg_manufacturer_before_update_status() line 5 at RAISE
postgres=# -- ERROR:  Error: invalid status = deleted (allowed only active or inactive)
postgres=# -- 1) Некорректный статус → Должен выдать ERROR от триггера
UPDATE "Manufacturer" SET "status" = 'deleted' WHERE "manufacturer_id" = 9999;
postgres=# -- ERROR:  Error: invalid status = deleted (allowed only active or inactive)
postgres=#
postgres=# -- 2) Корректный статус 'inactive' → Должно выполниться без ошибок
UPDATE "Manufacturer" SET "status" = 'inactive' WHERE "manufacturer_id" = 9999;
postgres=# -- UPDATE 1
postgres=#
postgres=# -- 3) Корректный статус 'active' → Должно выполниться без ошибок
UPDATE "Manufacturer" SET "status" = 'active' WHERE "manufacturer_id" = 9999;
postgres=# -- UPDATE 1
postgres=#
postgres=# -- 4) Чувствительность к регистру: 'Inactive' → Должен дать ERROR
UPDATE "Manufacturer" SET "status" = 'Inactive' WHERE "manufacturer_id" = 9999;
postgres=# -- ERROR:  Error: invalid status = Inactive (allowed only active or inactive)
```

Выводы

В ходе выполнения работы я:

- Разработал три хранимые процедуры.
- Создал семь триггеров.
- Проверил их работоспособность в psql; результаты задокументированы скриншотами.

Применение процедур и триггеров позволяет сосредоточить бизнес-логику на стороне СУБД, упростить клиентские приложения и обеспечить целостность данных.