Министерство науки и высшего образования РФ

федеральное государственное автономное  
образовательное учреждение высшего образования

«Омский государственный технический университет»

Факультет информационных технологий и компьютерных систем

Кафедра «Прикладная математика и фундаментальная информатика»

|  |
| --- |
| **Лабораторная работа №7** |
| по дисциплине **Системы управления базами данных**  **Тема: ФУНКЦИИ И ПРОЦЕДУРЫ** |

|  |  |  |  |
| --- | --- | --- | --- |
| Студента | Кузиной Варвары Михайловны | | |
| Курс | 3 | Группа | МО-231 |
| Направление | 02.03.03 Математическое обеспечение и администрирование информационных систем | | |
| Руководитель | доц., к.н. | | |
|  | Моисеева Н.А. | | |
| Выполнил | 31.10 | | |
| Проверил |  | | |

Омск 2025

**Цель работы:**

Создать в БД rgr\_store три хранимые процедуры и две функции на языке PL/pgSQL, включить их в базу данных, протестировать вызовы и возврат результатов. Результаты подтвердить SQL-скриптами и нумерованными скриншотами.

**Исходные данные:**

БД rgr\_store: employees, suppliers, product\_catalog, orders, order\_composition, receipts, check\_composition, invoices, invoice\_composition, а также расширенные таблицы product\_media (массивы) и receipt\_payments (JSONB). Все связи реализованы внешними ключами с каскадами; заданы ограничения CHECK/NOT NULL/PK.

**1. Процедура add\_receipt\_with\_items**

Смысл: Безопасно добавить чек и его состав за один вызов с проверкой существования сотрудника и товаров.

SQL:

/\*Процедура обновления,ее вызов и проверка результата\*/

CREATE OR REPLACE PROCEDURE public.add\_receipt\_with\_items(

p\_receipt\_number INTEGER,

p\_receipt\_date TIMESTAMP,

p\_cash\_register\_number INTEGER,

p\_employee\_id INTEGER,

p\_items JSONB -- массив объектов: [{"product\_code": 100, "quantity": 2}, ...]

)

LANGUAGE plpgsql AS $$

DECLARE

item JSONB;

v\_product\_code INTEGER;

v\_quantity INTEGER;

BEGIN

-- Проверка существования сотрудника

IF NOT EXISTS (SELECT 1 FROM public.employees WHERE employee\_id = p\_employee\_id) THEN

RAISE EXCEPTION 'Сотрудник с ID % не найден', p\_employee\_id;

END IF;

-- Проверка даты (не из будущего)

IF p\_receipt\_date > CURRENT\_TIMESTAMP THEN

RAISE EXCEPTION 'Дата чека не может быть из будущего';

END IF;

-- Вставка чека

INSERT INTO public.receipts (

receipt\_number, receipt\_date, cash\_register\_number, employee\_id

) VALUES (

p\_receipt\_number, p\_receipt\_date, p\_cash\_register\_number, p\_employee\_id

);

-- Вставка состава чека

FOR item IN SELECT \* FROM jsonb\_array\_elements(p\_items)

LOOP

v\_product\_code := (item->>'product\_code')::INTEGER;

v\_quantity := (item->>'quantity')::INTEGER;

-- Проверка товара

IF NOT EXISTS (SELECT 1 FROM public.product\_catalog WHERE product\_code = v\_product\_code) THEN

RAISE EXCEPTION 'Товар с кодом % не найден в каталоге', v\_product\_code;

END IF;

-- Проверка количества

IF v\_quantity <= 0 THEN

RAISE EXCEPTION 'Количество товара должно быть > 0';

END IF;

INSERT INTO public.check\_composition (receipt\_number, product\_code, quantity\_in\_check)

VALUES (p\_receipt\_number, v\_product\_code, v\_quantity);

END LOOP;

RAISE NOTICE 'Чек % успешно добавлен с % позициями', p\_receipt\_number, jsonb\_array\_length(p\_items);

END;

$$;

CALL public.add\_receipt\_with\_items(

70003,

'2025-10-31 14:30:00'::TIMESTAMP,

1,

1,

'[{"product\_code": 100, "quantity": 5}, {"product\_code": 200, "quantity": 2}]'::JSONB

);

-- Чек создан?

SELECT \* FROM public.receipts WHERE receipt\_number = 70003;

-- Состав чека добавлен?

SELECT \* FROM public.check\_composition WHERE receipt\_number = 70003;

Пояснение:

Процедура принимает чек и массив позиций в формате JSONB, проверяет данные и вставляет записи атомарно (рисунок 1—2).

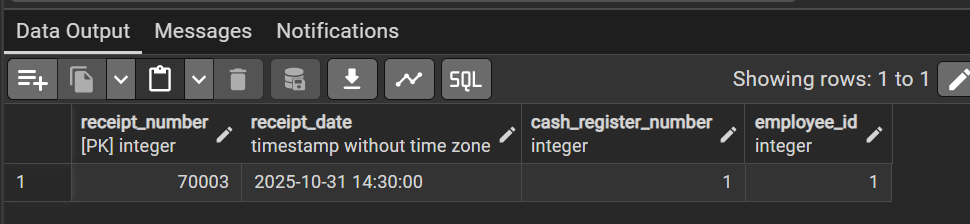


Рисунок 1—Чек создан

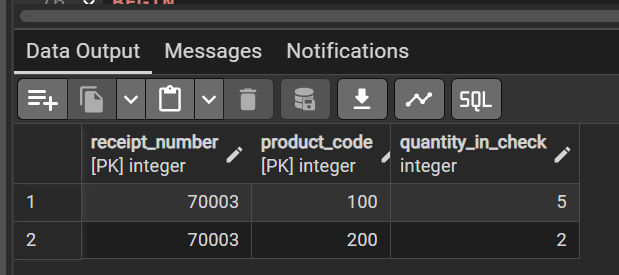


Рисунок 2—Что находится в составе этого чека

**2. Процедура update\_product\_price**

Смысл: Обновить цену товара с логированием и проверкой на неотрицательность.

SQL:

/\*Процедура обновления, ее вызов и проверка результата\*/

CREATE OR REPLACE PROCEDURE public.update\_product\_price(

p\_product\_code INTEGER,

p\_new\_price NUMERIC(10,2)

)

LANGUAGE plpgsql AS $$

BEGIN

IF p\_new\_price < 0 THEN

RAISE EXCEPTION 'Цена не может быть отрицательной';

END IF;

UPDATE public.product\_catalog

SET product\_price = p\_new\_price

WHERE product\_code = p\_product\_code;

IF NOT FOUND THEN

RAISE EXCEPTION 'Товар с кодом % не найден', p\_product\_code;

END IF;

RAISE NOTICE 'Цена товара % обновлена на %', p\_product\_code, p\_new\_price;

END;

$$;

-- Вызов процедуры

CALL public.update\_product\_price(100, 40.00);

-- Проверка результата

SELECT product\_code, product\_name, product\_price

FROM public.product\_catalog

WHERE product\_code = 100;

Пояснение:

Проверяет корректность цены и существование товара (рисунок 3).

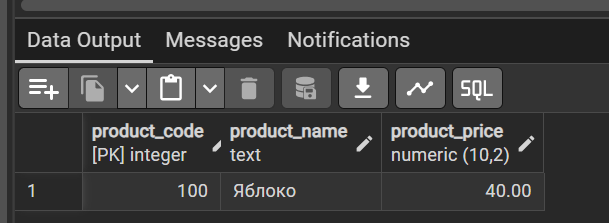


Рисунок 3— Результат обновления цены товара

**3. Процедура delete\_old\_orders**

Смысл: Удалить заявки старше N дней (по умолчанию — 365).

SQL:

/\*Процедура удаления\*/

CREATE OR REPLACE PROCEDURE public.delete\_old\_orders(

p\_older\_than\_days INTEGER DEFAULT 365

)

LANGUAGE plpgsql AS $$

DECLARE

v\_count INTEGER;

BEGIN

SELECT COUNT(\*) INTO v\_count

FROM public.orders

WHERE order\_date <= CURRENT\_DATE - p\_older\_than\_days::INTEGER;

DELETE FROM public.orders

WHERE order\_date <= CURRENT\_DATE - p\_older\_than\_days::INTEGER;

RAISE NOTICE 'Удалено % старых заявок (старше % дней)', v\_count, p\_older\_than\_days;

END;

$$;

CALL public.delete\_old\_orders(1);

-- Сколько заявок осталось?

SELECT \* FROM public.orders;

Пояснение:

Подсчитывает и удаляет старые заявки с каскадным удалением состава. Выводит количество удалённых записей (рисунок 4).

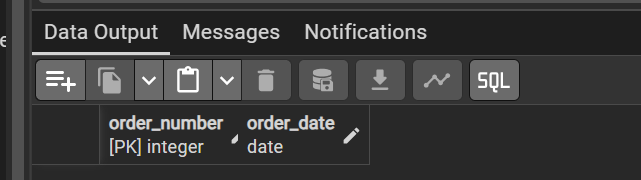


Рисунок 4— Сколько записей осталось

**4. Функция get\_total\_receipt\_amount (скалярная)**

Смысл: Вернуть общую сумму чека по его номеру.

SQL:

/\*Функция получения суммы по чеку\*/

CREATE OR REPLACE FUNCTION public.get\_total\_receipt\_amount(

p\_receipt\_number INTEGER

) RETURNS NUMERIC(10,2)

LANGUAGE plpgsql AS $$

DECLARE

v\_total NUMERIC(10,2);

BEGIN

SELECT COALESCE(SUM(cc.quantity\_in\_check \* pc.product\_price), 0)

INTO v\_total

FROM public.check\_composition cc

JOIN public.product\_catalog pc ON pc.product\_code = cc.product\_code

WHERE cc.receipt\_number = p\_receipt\_number;

RETURN v\_total;

END;

$$;

SELECT public.get\_total\_receipt\_amount(70001) AS total\_amount;

Пояснение:

Функция возвращает одно значение — сумму чека. Вызывается в SELECT и возвращает таблицу 1×1 (рисунок 5).

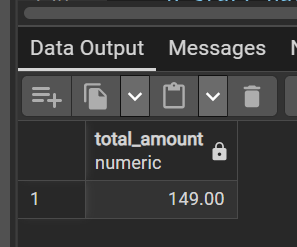


Рисунок 5— Результат работы функции суммирования по чеку

**5. Функция get\_employee\_sales\_report (табличная)**

Смысл: Вернуть отчёт по продажам сотрудника за период: товар, количество, выручка.

SQL:

/\*Функция отчета по товарам по сотрудника\*/

CREATE OR REPLACE FUNCTION public.get\_employee\_sales\_report(

p\_employee\_id INTEGER,

p\_start\_date DATE,

p\_end\_date DATE

)

RETURNS TABLE (

product\_name TEXT,

total\_quantity INTEGER,

total\_revenue NUMERIC(10,2)

)

LANGUAGE plpgsql AS $$

BEGIN

RETURN QUERY

SELECT

pc.product\_name,

SUM(cc.quantity\_in\_check)::INTEGER AS total\_quantity,

SUM(cc.quantity\_in\_check \* pc.product\_price)::NUMERIC(10,2) AS total\_revenue

FROM public.receipts r

JOIN public.check\_composition cc ON cc.receipt\_number = r.receipt\_number

JOIN public.product\_catalog pc ON pc.product\_code = cc.product\_code

WHERE r.employee\_id = p\_employee\_id

AND r.receipt\_date::DATE BETWEEN p\_start\_date AND p\_end\_date

GROUP BY pc.product\_name

ORDER BY total\_revenue DESC;

END;

$$;

SELECT \*

FROM public.get\_employee\_sales\_report(

p\_employee\_id := 1,

p\_start\_date := '2025-03-01',

p\_end\_date := '2025-03-31'

);

Пояснение:

Функция возвращает набор строк. Используется в FROM или напрямую в SELECT \* FROM ... (рисунок 6).

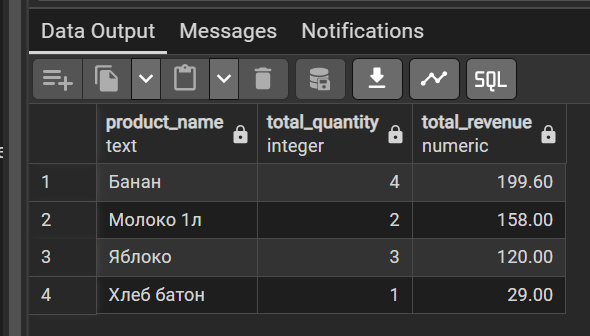


Рисунок 6— Отчет по продажам сотрудника

**Заключение:**

Для базы rgr\_store созданы три хранимые процедуры (add\_receipt\_with\_items, update\_product\_price, delete\_old\_orders) и две функции (get\_total\_receipt\_amount, get\_employee\_sales\_report). Все объекты написаны на PL/pgSQL, используют проверки, обработку ошибок и возврат данных. Выполнены тестовые вызовы через CALL и SELECT. Результаты подтверждены нумерованными скриншотами в pgAdmin.