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.

Тема работы: «ПРОЦЕДУРЫ, ФУНКЦИИ, ТРИГГЕРЫ В POSTGRESQL»

Обучающийся: Кошкарев Кирилл Павлович, K3239

Преподаватель: Говорова М. М.

Санкт-Петербург
2025

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

Практическое задание (min - 6 баллов, max - 10 баллов, доп. баллы - 3): 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

Создание 3-х процедур по варианту:

- Для снижения цен на книги, которые находятся на базе в количестве, превышающем 1000 штук.

```
CREATE OR REPLACE PROCEDURE decrease_price_for_books_with_overstock()
LANGUAGE plpgsql
AS $$
BEGIN
    -- Обновляем цену для книг, количество которых больше 1000
    UPDATE public.book b
    SET price = price - 50  -- Пример: уменьшаем цену на 50
    WHERE EXISTS (
        SELECT 1
        FROM public.ordered_books ob
        WHERE ob.ordered_book_id = b.book_id  -- Используем правильное имя столбца
        GROUP BY ob.ordered_book_id
        HAVING SUM(ob.quantity) > 1000  -- Условие для количества больше 1000
    );
END;
$$;
```

```
postgres=# SELECT book_id, title, price
FROM public.book
WHERE book_id IN (
    SELECT ordered_book_id
    FROM public.ordered_books
    GROUP BY ordered_book_id
    HAVING SUM(quantity) > 1000
);
 book_id |          title           | price
---------+--------------------------+--------
       2 | Сетевое программирование | 850.00
(1 строка)

postgres=# CREATE OR REPLACE PROCEDURE decrease_price_for_books_with_overstock()
LANGUAGE plpgsql
AS $$
BEGIN
    -- Обновляем цену для книг, количество которых больше 1000
    UPDATE public.book b
    SET price = price - 50  -- Пример: уменьшаем цену на 50
    WHERE EXISTS (
        SELECT 1
        FROM public.ordered_books ob
        WHERE ob.ordered_book_id = b.book_id  -- Используем правильное имя столбца
        GROUP BY ob.ordered_book_id
        HAVING SUM(ob.quantity) > 1000  -- Условие для количества больше 1000
    );
END;
$$;
CREATE PROCEDURE
postgres=# CALL decrease_price_for_books_with_overstock();
CALL
postgres=# SELECT book_id, title, price
FROM public.book
WHERE book_id IN (
    SELECT ordered_book_id
    FROM public.ordered_books
    GROUP BY ordered_book_id
    HAVING SUM(quantity) > 1000
);
 book_id |          title           | price
---------+--------------------------+--------
       2 | Сетевое программирование | 800.00
(1 строка)
```

- Для удаления заказов, имеющих статус “отменен”

```sql
CREATE OR REPLACE PROCEDURE delete_canceled_orders()
LANGUAGE plpgsql
AS $$
BEGIN
    -- Удаляем все связанные записи в таблице ordered_books
    DELETE FROM public.ordered_books
    WHERE order_id IN (
        SELECT order_id
        FROM public.orders
        WHERE status_id = 3  -- статус "отменен"
    );

    -- Удаляем все связанные записи в таблице invoice
    DELETE FROM public.invoice
    WHERE order_id IN (
        SELECT order_id
        FROM public.orders
        WHERE status_id = 3  -- статус "отменен"
    );
```

```
    -- Теперь удаляем заказы со статусом "отменен"
    DELETE FROM public.orders
    WHERE status_id = 3;  -- статус "отменен"
END;
$$;
```

```
postgres=# CREATE OR REPLACE PROCEDURE delete_canceled_orders()
LANGUAGE plpgsql
AS $$
BEGIN
    -- Удаляем все связанные записи в таблице ordered_books
    DELETE FROM public.ordered_books
    WHERE order_id IN (
        SELECT order_id
        FROM public.orders
        WHERE status_id = 3  -- статус "отменен"
    );

    -- Удаляем все связанные записи в таблице invoice
    DELETE FROM public.invoice
    WHERE order_id IN (
        SELECT order_id
        FROM public.orders
        WHERE status_id = 3  -- статус "отменен"
    );

    -- Теперь удаляем заказы со статусом "отменен"
    DELETE FROM public.orders
    WHERE status_id = 3;  -- статус "отменен"
END;
$$;
CREATE PROCEDURE
postgres=# CALL delete_canceled_orders();
CALL
postgres=# SELECT *
FROM public.orders
WHERE status_id = 3;
 order_id | order_date | deadline | completion_date | status_id | client_id | manager_id | act_id
----------+------------+----------+-----------------+-----------+-----------+------------+--------
(0 строк)
```

- Для ввода нового заказа.

```
CREATE OR REPLACE PROCEDURE insert_new_order(
    p_client_id integer,
    p_manager_id integer,
    p_status_id integer,
    p_order_date date,
    p_deadline date
)
LANGUAGE plpgsql
AS $$
BEGIN
    -- Вставляем новый заказ в таблицу orders
    INSERT INTO public.orders (client_id, manager_id, status_id, order_date,
deadline)
    VALUES (p_client_id, p_manager_id, p_status_id, p_order_date, p_deadline);
END;
$$;
```

```
postgres=# CREATE OR REPLACE PROCEDURE insert_new_order(
    p_client_id integer,
    p_manager_id integer,
    p_status_id integer,
    p_order_date date,
    p_deadline date
)
LANGUAGE plpgsql
AS $$
BEGIN
    -- Вставляем новый заказ в таблицу orders
    INSERT INTO public.orders (client_id, manager_id, status_id, order_date, deadline)
    VALUES (p_client_id, p_manager_id, p_status_id, p_order_date, p_deadline);
END;
$$;
CREATE PROCEDURE
postgres=# CALL insert_new_order(1, 2, 3, '2025-06-22', '2025-07-01');
CALL
postgres=# SELECT * FROM public.orders
WHERE client_id = 1  -- или другой идентификатор клиента, который был использован
ORDER BY order_date DESC
LIMIT 10;
 order_id | order_date |  deadline  | completion_date | status_id | client_id | manager_id | act
----------+------------+------------+-----------------+-----------+-----------+------------+-----
      308 | 2025-06-22 | 2025-07-01 |                 |         3 |         1 |          2 |
        5 | 2025-06-10 | 2025-06-15 |                 |         3 |         1 |          1 |
        6 | 2025-06-05 | 2025-06-13 | 2025-06-17      |         3 |         1 |          1 |
        1 | 2025-04-01 | 2025-04-11 | 2025-04-10      |         3 |         1 |          1 |
(4 строки)

postgres=# CALL insert_new_order(1, 2, 3, '2025-06-22', '2025-07-01');
CALL
postgres=# SELECT * FROM public.orders
WHERE client_id = 1  -- или другой идентификатор клиента, который был использован
ORDER BY order_date DESC
LIMIT 10;
 order_id | order_date |  deadline  | completion_date | status_id | client_id | manager_id | act
----------+------------+------------+-----------------+-----------+-----------+------------+-----
      308 | 2025-06-22 | 2025-07-01 |                 |         3 |         1 |          2 |
      309 | 2025-06-22 | 2025-07-01 |                 |         3 |         1 |          2 |
        5 | 2025-06-10 | 2025-06-15 |                 |         3 |         1 |          1 |
        6 | 2025-06-05 | 2025-06-13 | 2025-06-17      |         3 |         1 |          1 |
        1 | 2025-04-01 | 2025-04-11 | 2025-04-10      |         3 |         1 |          1 |
(5 строк)
```

Создание триггеров:

`1. set_invoice_date` — **автоматически устанавливает дату счёта**

Зачем нужен: чтобы дата выставления счёта (`invoice_date`) не оставалась пустой, если пользователь её не указал вручную.

```
CREATE OR REPLACE FUNCTION set_invoice_date()
RETURNS trigger AS $$
BEGIN
    IF NEW.invoice_date IS NULL THEN
        NEW.invoice_date := CURRENT_DATE;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_set_invoice_date
BEFORE INSERT ON invoice
FOR EACH ROW
EXECUTE FUNCTION set_invoice_date();
```

Тест:

```
INSERT INTO invoice (order_id, prepayment, balance) VALUES (1, 100.00, 200.00);
```

614	2248	1	2025-06-23	100.00	200.00

2. `check_book_price` — **запрещает вставку книг с отрицательной ценой.**

Зачем нужен: не допускает логических ошибок и защищает от некорректных значений (например, из UI-формы).

```
CREATE OR REPLACE FUNCTION check_book_price()
RETURNS trigger AS $$
BEGIN
    IF NEW.price < 0 THEN
        RAISE EXCEPTION 'Цена книги не может быть отрицательной!';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_check_book_price
BEFORE INSERT OR UPDATE ON book
FOR EACH ROW
EXECUTE FUNCTION check_book_price();
```

Проверка работы:

```
postgres=# INSERT INTO book (isbn, title, publish_year, page_count, category_id, has_illustrations, price
, print_run_id)
VALUES ('978-0-00-000000-0', 'Книга с ошибкой', 2025, 100, 1, TRUE, -100.00, 1);
ERROR:  Цена книги не может быть отрицательной!
КОНТЕКСТ:  PL/pgSQL function check_book_price() line 4 at RAISE
postgres=# INSERT INTO book (isbn, title, publish_year, page_count, category_id, has_illustrations, price
, print_run_id)
VALUES ('978-0-00-000000-0', 'Книга с ошибкой', 2025, 100, 1, TRUE, 100.00, 1);
INSERT 0 1
```

3. `set_default_illustration_flag` **подставляет FALSE, если не указано наличие иллюстраций**

Зачем нужен: чтобы в базе не было `NULL`, а было чёткое значение `true`/`false` — это важно для фильтрации и визуализации данных.

```
CREATE OR REPLACE FUNCTION set_default_illustration_flag()
RETURNS trigger AS $$
BEGIN
    IF NEW.has_illustrations IS NULL THEN
        NEW.has_illustrations := FALSE;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_set_default_illustration
BEFORE INSERT ON book
FOR EACH ROW
EXECUTE FUNCTION set_default_illustration_flag();
```

Проверка работы:

```
postgres=# INSERT INTO book (isbn, title, publish_year, page_count, category_id, price, print_run_id)
VALUES ('978-0-00-000000-1', 'Книга без флага', 2025, 100, 1, 100.00, 1);
INSERT 0 1
```

6	7	978-0-00-000000-1	Книга без флага	2025	100	1	false	100.00	1

4. `log_author_changes` — **сообщает об изменениях в терминал.**

Зачем нужен: чтобы при любом изменении авторов было видно, **что и у кого изменилось**, без создания лишней таблицы.

Триггер функция:

```
CREATE OR REPLACE FUNCTION log_author_changes()
RETURNS trigger AS $$
BEGIN
    RAISE NOTICE 'Изменён автор (ID = %): Фамилия: % → %, Имя: % → %, Телефон: % → %',
        COALESCE(NEW.author_id, OLD.author_id),
        OLD.last_name, NEW.last_name,
        OLD.first_name, NEW.first_name,
        OLD.phone, NEW.phone;

    RETURN COALESCE(NEW, OLD);
END;
$$ LANGUAGE plpgsql;
```

Триггер:

```
CREATE TRIGGER trg_log_author_changes
AFTER INSERT OR UPDATE OR DELETE ON author
FOR EACH ROW
EXECUTE FUNCTION log_author_changes();
```

Проверка работы:

```
postgres=# UPDATE author
SET phone = '8888888888'
WHERE author_id = 1;
NOTICE:  Изменён автор (ID = 1): Фамилия: Смирнов → Смирнов, Имя: Иван → Иван, Телефон: 9010000005 → 8888888888
NOTICE:  Изменён автор (ID = 1): Фамилия: Смирнов → Смирнов, Имя: Иван → Иван, Телефон: 9010000005 → 8888888888
UPDATE 1
postgres=#
```

5. `notify_order_delete` — **выводит сообщение при удалении заказа**

Зачем нужен: чтобы при удалении заказа пользователь увидел уведомление (например, в логах или терминале).

```
 CREATE OR REPLACE FUNCTION notify_order_delete()
RETURNS trigger AS $$
BEGIN
    RAISE NOTICE 'Удалён заказ с ID = %', OLD.order_id;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_notify_order_delete
BEFORE DELETE ON orders
```

```
FOR EACH ROW
EXECUTE FUNCTION notify_order_delete();
```

Проверка работы:

```
postgres=# DELETE FROM orders WHERE order_id = 1;
NOTICE:  Удалён заказ с ID = 1
```

6. `prevent_past_deadline` — **запрещает устанавливать дедлайн в прошлом**

   Зачем нужен: чтобы никто не мог создать или обновить заказ с дедлайном в **прошлом** — это логическая ошибка, которая может испортить логику планирования, отчётов и т.д.

```
CREATE OR REPLACE FUNCTION prevent_past_deadline()
RETURNS trigger AS $$
BEGIN
    IF NEW.deadline < CURRENT_DATE THEN
        RAISE EXCEPTION 'Нельзя устанавливать дедлайн в прошлом!';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER trg_prevent_past_deadline
BEFORE INSERT OR UPDATE ON orders
FOR EACH ROW
EXECUTE FUNCTION prevent_past_deadline();
```

Проверка работы:

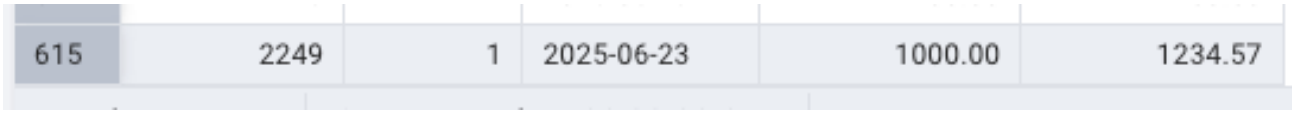
```
INSERT INTO orders (order_date, deadline, client_id, manager_id)
VALUES (CURRENT_DATE, CURRENT_DATE - INTERVAL '1 day', 1, 1);
ERROR:  Нельзя устанавливать дедлайн в прошлом!
```

7. `round_invoice_balance` — **округляет баланс в счёте до двух знаков**

   Часто после расчётов (особенно при делении или применении скидок) в поле `balance` остаются длинные дроби (например, `1234.567890`). Этот триггер автоматически округляет значение `balance` до 2 знаков после запятой.

```
CREATE OR REPLACE FUNCTION round_invoice_balance()
RETURNS trigger AS $$
BEGIN
    IF NEW.balance IS NOT NULL THEN
        NEW.balance := ROUND(NEW.balance, 2);
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER trg_round_invoice_balance
BEFORE INSERT OR UPDATE ON invoice
FOR EACH ROW
```

```
EXECUTE FUNCTION round_invoice_balance();
```



615	2249	1	2025-06-23	1000.00	1234.57

Вывод:

В данной лабораторной работе я овладел практическими навыками создания и использования процедур, функций и триггеров в базе данных PostgreSQL.