

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

О Т Ч Е Т

по лабораторной работе «Процедуры, функции, триггеры в PostgreSQL»
по дисциплине «**Проектирование и реализация баз данных**»

Автор: Бакшилова Анастасия Денисовна
Факультет: ИКТ
Группа: К33391
Преподаватель: Говорова М. М.
Дата: 23.10.2023



Санкт-Петербург
2023

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

Практическое задание:

Вариант 1

1. Создать процедуры/функции согласно индивидуальному заданию и (согласно индивидуальному заданию, часть 4).
2. Создать триггер для логирования событий вставки, удаления, редактирования данных в базе данных PostgreSQL (согласно индивидуальному заданию, часть 5). Допустимо создать универсальный триггер или отдельные триггеры на логирование действий.

Задание 1:

Создать хранимые процедуры:

- Для проверки наличия экземпляров заданной книги в библиотеке (процедура должна возвращать количество экземпляров книги).

Код:

```
CREATE FUNCTION public.check_book_availability(b_book_title text) RETURNS
integer
LANGUAGE plpgsql
AS $$
DECLARE
    book_count integer;
BEGIN
    SELECT count(bc.edition_code)
    INTO book_count
    FROM book_copy bc
    JOIN edition ed ON bc.edition_code = ed.edition_code
```

```

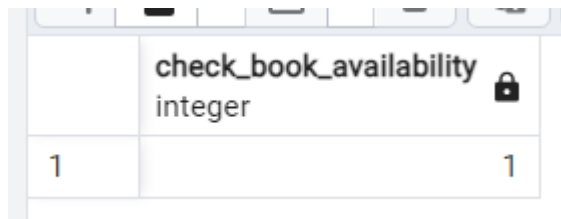
JOIN book b ON ed.book_id = b.book_id
WHERE b.book_title = b_book_title
AND bc.status = 'свободна';

RETURN book_count;
END;
$$;

```

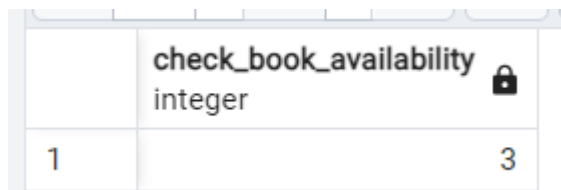
Проверка:

```
SELECT check_book_availability('Изучаем C# через разработку игр на Unity');
```



	check_book_availability integer
1	1

```
SELECT check_book_availability('Капитанская дочка');
```



	check_book_availability integer
1	3

- Для ввода в базу данных новой книги

Код:

```

CREATE PROCEDURE public.addnewbook(IN p_author_name text, IN p_book_title
text, IN p_original_language text, IN p_area_of_knowledge text, IN p_volume_num
integer, IN p_type_of_publication text)
LANGUAGE plpgsql
AS $$
DECLARE
    author_id INT;
BEGIN

```

```
INSERT INTO authors (author_name)
VALUES (p_author_name)
ON CONFLICT (id_author) DO NOTHING;
```

```
SELECT id_author INTO author_id
FROM authors
WHERE author_name = p_author_name;
```

```
INSERT INTO book (book_title, original_language, area_of_knowledge,
volume_num, type_of_publication, author_id)
VALUES (p_book_title, p_original_language, p_area_of_knowledge,
p_volume_num, p_type_of_publication, author_id);
```

```
END;
$$;
```

Проверка:

```
CALL AddNewBook('Андре Мальро', 'Удел человеческий', 'Французский',
'Художественная литература', 1, 'Книга');
```

Ответ:

```
CALL
```

```
Query returned successfully in 67 msec.
```

```
SELECT * FROM book;
```

id integer	book_title text	original_language text	area_of_knowledge text	volume_num integer	type_of_publication text	author_id integer
1	Удел человеческий	Французский	Художественная литература	1	Книга	23

- Для ввода нового читателя (необходимо проверить наличие читателя в картотеке, чтобы не назначить ему номер вторично).

Код:

```
CREATE OR REPLACE PROCEDURE AddNewReader(
    p_reader_name TEXT,
    p_address TEXT,
    p_passport_ser TEXT,
    p_passport_num TEXT,
    p_passport_place_of_issue TEXT,
    p_passport_date_of_issue DATE,
    p_police_dep_code INT,
    p_telephone TEXT,
    p_education TEXT,
    p_e_mail TEXT)
LANGUAGE plpgsql
AS $$
DECLARE
    abonement_id INT;
    reader_id INT;
BEGIN
    SELECT nextval('membership_abonement_id_seq') INTO abonement_id;

    -- Проверяем наличие читателя по имени, серии и номеру паспорта
    SELECT r.reader_id INTO reader_id
    FROM readers r
    WHERE r.reader_name = p_reader_name
        AND (r.passport_ser = p_passport_ser OR (r.passport_ser IS NULL AND
r.passport_num IS NULL))
        AND (r.passport_num = p_passport_num OR (r.passport_ser IS NULL AND
r.passport_num IS NULL))
        AND r.e_mail = p_e_mail;
```

```

IF reader_id IS NULL THEN
    -- Если читатель не найден, добавляем нового
    INSERT INTO readers (abonement_id, reader_name, address, passport_ser,
passport_num, passport_place_of_issue, passport_date_of_issue, police_dep_code,
telephone, education, e_mail)
        VALUES (abonement_id, p_reader_name, p_address, p_passport_ser,
p_passport_num,          p_passport_place_of_issue,          p_passport_date_of_issue,
p_police_dep_code, p_telephone, p_education, p_e_mail);
ELSE
    -- Если читатель найден, перезаписываем информацию
    UPDATE readers
    SET reader_name = p_reader_name,
        address = p_address,
        passport_ser = p_passport_ser,
        passport_num = p_passport_num,
        passport_place_of_issue = p_passport_place_of_issue,
        passport_date_of_issue = p_passport_date_of_issue,
        police_dep_code = p_police_dep_code,
        telephone = p_telephone,
        education = p_education
    WHERE reader_id = reader_id;
END IF;
END;
$$;

```

Проверка:

```

CALL AddNewReader('Королёва Елизавета Олеговна', 'Россия, г.
Санкт-Петербург, Ветеранов пр., д.118 кв. 119', '4017', '956632', 'ТП №29 УФМС

```

России по Санкт-Петербургу и Ленинградской области', '01-12-2013', 750029, '89994537651', 'бакалавриат', 'korolevaelizzi@gmail.com');

```
CALL
```

```
Query returned successfully in 67 msec.
```

```
SELECT * FROM readers;
```

14	16	16	Королева Елизавета Олеговна	Россия, г. Санкт-Петербург, Ветеранов пр., д.118 кв. 1...	4017	956632	ТП №29 УФМС Р
----	----	----	-----------------------------	---	------	--------	---------------

Задание 2:

Код:

```
CREATE OR REPLACE FUNCTION log_event()
```

```
RETURNS TRIGGER AS $$
```

```
BEGIN
```

```
    IF TG_OP = 'INSERT' THEN
```

```
        INSERT INTO event_log (event_type, event_timestamp, table_name,  
event_data)
```

```
        VALUES ('INSERT', now(), TG_TABLE_NAME, row_to_json(NEW));
```

```
    ELSIF TG_OP = 'UPDATE' THEN
```

```
        INSERT INTO event_log (event_type, event_timestamp, table_name,  
event_data)
```

```
        VALUES ('UPDATE', now(), TG_TABLE_NAME, row_to_json(NEW));
```

```
    ELSIF TG_OP = 'DELETE' THEN
```

```
        INSERT INTO event_log (event_type, event_timestamp, table_name,  
event_data)
```

```
        VALUES ('DELETE', now(), TG_TABLE_NAME, row_to_json(OLD));
```

```

END IF;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;

```

Триггер на каждую из таблиц:

Для таблицы workers:

```

CREATE TRIGGER log_workers
AFTER INSERT OR UPDATE OR DELETE ON workers
FOR EACH ROW
EXECUTE FUNCTION log_event();

```

Для таблицы book:

```

CREATE TRIGGER log_book
AFTER INSERT OR UPDATE OR DELETE ON book
FOR EACH ROW
EXECUTE FUNCTION log_event();

```

Проверка:

```
SELECT * FROM workers;
```

	w_id [PK] integer	job_code integer	w_name text	telephone text	seniority date
1	1025	1	Терентьева Анна Арсеньевна	89041325321	2004-06-10
2	1013	1	Тарасова Кристина Георгиевна	89307430308	2019-05-30
3	1024	4	Кузнецова Ирина Родионовна	89538995477	2008-10-24
4	1084	3	Кулагин Сергей Данилович	89224053339	2017-08-16
5	1021	2	Галкина Юлия Владимировна	89046653238	2023-10-10
6	1011	2	Лапина Людмила Вадимовна	89307430308	2023-11-18
7	2938	1	Паучихина Елена Владимировна	89654396512	2023-09-22

```
DELETE FROM workers WHERE w_id = '2938';
```



```
DELETE 1
```

```
Query returned successfully in 90 msec.
```

```
SELECT * FROM event_log;
```

6	DELETE	2023-10-29 20:55:51.040503	workers	[null]	{ "w_id": 2938, "job_code": 1, "w_name": "Паучихина Елена Владимировна" }
---	--------	----------------------------	---------	--------	---

Вывод: Мы изучили, как создавать процедуры в PostgreSQL. Процедуры позволяют упростить сложные операции, объединив их в один вызов. Мы научились создавать функции, которые могут возвращать значения. Функции полезны для выполнения вычислений и манипуляций с данными. Мы овладели созданием триггеров, которые реагируют на события в базе данных, такие как вставка, обновление или удаление данных. Триггеры могут обеспечивать автоматическую обработку данных и обеспечивать целостность данных.